Polynomial Arithmetic and Linear Systems in Symbolic Summation

Dissertation

zur Erlangung des akademischen Grades
„Doktor der technischen Wissenschaften“

verfasst von

Fabrizio Caruso

am
Institut für Symbolisches Rechnen (RISC)
Technisch-Naturwissenschaftliche Fakultät
Johannes Kepler Universität Linz

Dezember 2002

Erster Begutachter: a.Univ.-Prof. Dr. Peter Paule
Zweiter Begutachter: o.Univ.-Prof. Dr. Franz Winkler

Eidesstattliche Erklärung

Ich versichere, dass ich die Dissertation selbständig verfasst habe, andere als die angegebenen Quellen und Hilfsmittel nicht verwendet und mich auch sonst keiner unerlaubten Hilfe bedient habe.

Linz, Dezember 2002
Fabrizio Caruso

Abstract

The object of the thesis is twofold. Firstly we improve Zeilberger's algorithm for symbolic definite hypergeometric summation by interpolating the solutions of the related system of linear equations with rational function coefficients. Secondly we develop a C++/Mathematica program library for polynomial arithmetic that implements fast multiplication of sparse polynomials and some of the operations related to homogeneous linear diophantine inequalities and equations as studied in MacMahon's partition analysis and in its algorithmic version (Ω-calculus), namely exponentiation of low degree polynomials and division by powers of low degree polynomials.

Zusammenfassung

Diese Dissertation setzt sich aus zwei Teilen zusammen. Zum einen beschäftigen wir uns mit der Verbesserung des Algorithmus von Zeilberger zur symbolischen definiten hypergeometrischen Summation mittels Interpolation der Lösungen des korrespondierenden Systems von linearen Gleichungen mit rationalen Funktionen als Koeffizienten. Zum anderen entwickeln wir eine C++/Mathematica Programm-Bibliothek für polynomiale Arithmetik, die die schnelle Multiplikation von dünn besetzen Polynomen sowie einige der Operationen, die in Systemen von homogenen linearen diophantischen Ungleichungen und Gleichungen auftreten, implementiert, die in der Partitionsanalyse von MacMahon und ihrer algorithmischen Fassung (Ω-Kalkül) vorkommen, nämlich Exponenzierung von Polynomen niedrigen Grades und Division durch Potenzen von Polynomen niedrigen Grades.

I would like to dedicate my dissertation to my parents Giuseppe Caruso and Margherita Mirabella and to my dear grandmothers Vincenza and Margherita that have supported and encouraged me during my studies.

Dedico la mia tesi ai miei genitori Giuseppe Caruso e Margherita Mirabella e alle mie care nonne Vincenza e Margherita che tanto mi hanno aiutato e incoraggiato durante i miei studi.

Acknowledgments

I would like to thank my adviser Prof. P. Paule who has helped me during my studies at the RISC institute with his contagious enthusiasm and with many suggestions.

I would not have survived at RISC without the help and friendship of the other members or the institute and in particular of the members of the Combinatorics Group: Cleopatra Pau, Axel Riese, Carsten Schneider and Burkhard Zimmermann with whom I have had pleasant discussions on both mathematical and profane topics.

I also thank Erhard Aichinger of the Universal Algebra group at the University of Linz, who has given me useful suggestions.

A thank goes to my colleagues Ralf Hemmecke and Manuel Kauers for their patient help with Unix and LaTeX technicalities.

Contents

Introduction **1**

Structure of the Thesis **5**

Main Contributions **7**

Basic Notations **9**

I Zeilberger's Algorithm in the Macsyma and Maxima Computer Algebra Systems **11**

1 Zeilberger's Algorithm in Macsyma **13**
1.1 Introduction 13
1.2 Some Theory 14
1.2.1 Some Fundamental Definitions 14
1.2.2 Some Fundamental Facts 15
1.3 The Algorithms 15
1.3.1 Gosper's Algorithm 15
1.3.2 Zeilberger's Algorithm 17
1.4 The Implementation 19
1.4.1 Gosper's Algorithm 20
1.4.2 Zeilberger's Fast Algorithm 20
1.5 Performance 20
1.5.1 Timings 21
1.5.2 Stability 21
1.5.3 Future Improvements 21
1.6 Manual 22
1.6.1 Loading the files 22
1.6.2 The Commands 22
1.7 Some Examples 24
1.8 The Code 31
1.8.1 Low Level Routines 31

1.8.2 Gosper Form Routines 32
1.8.3 Gosper Equation Routines 33
1.8.4 Gosper's and Zeilberger's Algorithm 33

II Solving Systems of Linear Equations by Interpolation 35

2 Cramer Driven Interpolation 37
2.1 Introduction 37
2.2 Notation 37
2.3 The Homogeneous Problem 38
2.4 Rational Coefficients 39
2.4.1 Definitions 40
2.4.2 Modular Approach 41
2.4.3 Bounding the Solutions 44
2.5 Rational Functions 45
2.5.1 Definitions 45
2.5.2 Interpolation 45
2.5.3 Bounding the Degrees of the Solutions 46
2.5.4 A Combination of the two Approaches 47
2.6 Distributed Computation 47
2.7 The Package 48
2.7.1 Loading the Packages 49
2.7.2 The Commands 49
2.8 Conclusion 51

3 Rational Interpolation 53
3.1 Introduction 53
3.2 Reconstructing Rational Functions 53
3.3 Cauchy Interpolation 54
3.4 Padé Approximation 56
3.5 Homogeneous Linear Systems 57
3.5.1 The First Try 58
3.5.2 A Heuristic Method 58
3.5.3 A Recursive Method 58
3.5.4 A Hybrid Method 58
3.6 The Package 59
3.6.1 The Commands 59
3.6.2 Modular Null Space Computation 60
3.6.3 Completing the Null Space 62
3.6.4 Hybrid Modular Symbolic Null Space 63
3.6.5 Rational Function Interpolation 66

3.7 Conclusion 67

4 Linear Systems in Zeilberger's Algorithm **69**
4.1 Introduction 69
4.2 The Abstract Model 69
4.2.1 Notation 69
4.2.2 The Fundamental Bijection 70
4.2.3 More General Bijections 74
4.2.4 Some Useful Properties 76
4.2.5 Cardinality 77
4.2.6 Maximal Cardinality Property 78
4.3 Shift Quotients in Zeilberger's Algorithm 79
4.3.1 Notation and Definitions 80
4.3.2 Interpretation of the Model 81
4.3.3 Gosper Equation 89
4.3.4 Lower Bounds on Degrees 91
4.3.5 Some Examples 92
4.4 Conclusion 93

III Polynomial Arithmetic Library **95**

5 Combinatorial Interpretation of Division **97**
5.1 Introduction 97
5.2 Definitions 97
5.3 Existence of the Inverse 99
5.4 Relation between C and q 100
5.5 General Recurrence for $C(i)$ 102
5.6 Lifting to the $b(x^\alpha)$ Case 103
5.7 Combinatorial Meaning 104
5.8 Binomial Case 105
5.8.1 Diagrams of Compositions 107
5.8.2 Combinatorial Proof 109
5.9 Powers of Polynomials 114
5.9.1 Initial Values 116
5.10 The Algorithms 116
5.10.1 Modular Inverse 116
5.10.2 Division by Powers of Polynomials 116
5.11 Summary 117

6 The *PolyComb* Package **119**
6.1 Introduction 119
6.2 Structure of the Library 119

- 6.2.1 The Mathematica Component 119
- 6.2.2 The C++ Component 120
- 6.2.3 The Communication 121
- 6.3 Relations among Classes 122
- 6.4 The Algorithms 122
- 6.4.1 Dense Polynomials 122
- 6.4.2 Sparse Polynomials 126
- 6.5 The Mathematica Component 127
- 6.5.1 Loading 128
- 6.5.2 The Parser 128
- 6.5.3 The Mathematica Commands 132
- 6.6 The C++ Library 134
- 6.6.1 Sparse Polynomial Library 135
- 6.6.2 Dense Polynomial Library 138
- 6.7 A Small Program 140
- 6.8 Benchmarks 141
- 6.8.1 Measurements 142
- 6.8.2 Polynomial Multiplication and Expansion 142
- 6.8.3 Polynomial Exponentiation 144
- 6.9 Conclusion 145

Appendix A **147**

Appendix B **149**

Curriculum Vitae **153**

Bibliography **155**

Index **159**

Introduction

The object of the thesis is twofold: improving Zeilberger's algorithm for symbolic definite hypergeometric summation by reconstructing the solutions of the related system of linear equations with rational function coefficients by interpolation, and creating a library for polynomial arithmetic that implements some of the operations related to homogeneous linear diophantine inequalities and equations, namely fast multiplication of sparse polynomials, exponentiation of low degree polynomials, division by powers of low degree polynomials. The first object is covered in the first two parts (chapters 1, 2, 3, 4) and the second by the last part (chapters 5, 6).

Zeilberger's algorithm (see [Zei90], [Zei91]) is the standard computational method for symbolic definite hypergeometric summation. Its computational bottle-neck is a sparse system of linear equations with rational function coefficients. The first two parts of the thesis deal with the implementation of the algorithm, with two different approaches aimed at speeding up the solution of systems of linear equations with rational function coefficients by polynomial and rational interpolation, and with a new abstract model that gives information on the degree of the solutions to the systems of linear equations related to Zeilberger's algorithm.

The third part of the thesis deals with the combinatorial interpretation and with the implementation of some operations between polynomials that appear as subproblems in the algorithms that implement MacMahon's Ω-*calculus* (see [Mac16], [APR01a], [AP99], [APRS], [APR01c], [APR01d], [APR01e], [APR01b]), which is a computational method for solving problems in connection with linear homogeneous diophantine inequalities and equations.

The first part of the thesis (*Zeilberger's Algorithm in the Macsyma and Maxima Computer Algebra Systems*) consists of one chapter:

- Chapter 1: *Zeilberger's Algorithm in Macsyma*

In this chapter we describe Zeilberger's algorithm and our implementation on the Macsyma and Maxima computer algebra systems. Zeilberger's algorithm solves the definite hypergeometric summation problem, i. e. given a 2-variable term $F(n, k)$, we want to rewrite the definite sum $f(n) = \sum_{k=0}^{n} F(n, k)$ in a form free of $\sum$. This is done by finding a special k-free linear recurrence with polynomial coefficients for the summand $F(n, k)$, which, under the condition of natural boundary for $F(n, k)$, can be extended into a k-free homogeneous linear recurrence for the sum $f(n)$ with polynomial coefficients. The desired linear recurrence for the summand has the form: $\sum_{i=0}^{m} a_i(n)F(n+i, k) = G(n, k+1) - G(n, k)$. The search for this recurrence is done by guessing the order m of the left hand side of this recurrence and then trying to find the corresponding $G(n, k)$ of the right hand side by means of a modified *Gosper's algorithm* ([Gos78]). We note that a priori upper bounds for m can be given; however they turn out to be too large and therefore not useful for practical use. Gosper's algorithm for the indefinite summation problem solves the hypergeometric telescoping problem, i.e. given a hypergeometric term $t(k)$, it decides if there is a hypergeometric term $T(k)$ such that $t(k) = \Delta_k T(k) = T(k+1) - T(k)$, and if it exists, it finds it.

The second part (*Solving Systems of Linear Equations by Interpolation*) contains three chapters:

- Chapter 2: *Cramer Driven Interpolation*
- Chapter 3: *Rational Interpolation*
- Chapter 4: *Linear Systems in Zeilberger's Algorithm*

The first two chapters describe two different methods for solving systems of linear equations with rational function coefficients. The last chapter deals with linear systems related to Zeilberger's algorithm and it gives more understanding of some aspects of Zeilberger's algorithm (which is described in the first chapter).

In Chapter 2 (*Cramer Driven Interpolation*) a well-known algorithm based on Cramer's rule and on polynomial interpolation is described. Such algorithm solves overdetermined systems of homogeneous linear equations over $\mathbb{Q}$ and over the field of rational functions over $\mathbb{Q}$ under the hypothesis that the null space is one-dimensional. The algorithm reconstructs the numerators and denominators of the components of the solutions by computing different homomorphic systems over the finite fields $\mathbb{Z}_p$, with p prime.

In Chapter 3 (*Rational Interpolation*) we give a brief description of the standard theory of the rational function reconstruction problem, i. e. the problem of finding a rational function of small degree that is congruent to some polynomial modulo some other polynomial. Rational function reconstruction has two important applications: rational interpolation and rational function approximation. We have used this theory to build an algorithm that computes the null space of a rectangular matrix with entries in $\mathbb{Q}(y_1, \ldots, y_n)$ by reconstructing some low degree components by rational function interpolation and the others symbolically. This is done by plugging the components reconstructed by rational function interpolation into the system in order to simplify it. This approach generalizes the previous work [RZ] on methods for speeding up hypergeometric summation. In such a way the new system can become solvable by Gaussian elimination over $\mathbb{Q}(y_{i_1}, \ldots, y_{i_r})$ with $1 \leq i_j \leq n$ and $r < n$. Alternatively the system can be solved by iterating the procedure on the remaining variables.

In Chapter 4 (*Linear Systems in Zeilberger's Algorithm*) we construct an abstract model to study the degree of the polynomial coefficients of the linear recurrence which is computed by Zeilberger's algorithm. Given a proper hypergeometric term $F(n, k)$ and a linear shift operator $\mathcal{L}$ of order d with polynomial coefficients, we prove that the *Gosper form* $(p(n, k),\ q(n, k),\ r(n, k))$ of the shift quotient in k of $\mathcal{L}F(n, k)$ is such that the degree in n of $q(n, k)$ and $r(n, k)$ can be bounded independently of d. This gives some insight on the degree in n of all the polynomial coefficients $a_i(n)$ of the linear recurrence satisfied by $F(n, k)$ that is computed by Zeilberger's algorithm. We do this in two steps: firstly by proving the existence of certain bijections between subsets of the natural numbers which reflect the properties of the *Gosper form*; secondly by interpreting this abstract model in terms of the polynomials involved in the *Gosper form*.

The third part of the thesis (*Polynomial Arithmetic Library*) contains two chapters:

- Chapter 5: *Combinatorial Interpretation of Division*
- Chapter 6: *The Polynomial Package*

In Chapter 5 (*Combinatorial Interpretation of Division*) we give a combinatorial interpretation of polynomial division and of modular inverse of a polynomial. Such interpretation relates the coefficients of the modular inverse of a polynomial with compositions (i. e. ordered partitions) of non-negative integers. We will describe the recurrence for the coefficients of the polynomial $y(x)$ such that $p(x) \cdot y(x) \equiv 1 \mod x^l$, where l is a non-negative integer and $p(x)$ is an invertible polynomial over a ring R, i. e. $x \nmid p(x)$, for which therefore $y(x)$ exists. We will use a transcendental extension of R as a tool to study this recurrence.

Chapter 6 (*The Polynomial Package*) contains a description of algorithms and the structure of a C++/Mathematica library for fast polynomial arithmetic and a manual of both the Mathematica component and the C++ component of the library. The library has two parts: a C++ implementation of fast polynomial arithmetic for dense and sparse polynomials, and a Mathematica [Wol99] interface which makes it possible to use the library from within the Mathematica environment. The implemented methods and operators between polynomials involve addition, multiplication, exponentiation and division with remainder and various other auxiliary methods.

The C++ component of the library makes full use of the object oriented paradigm and of generic programming through the extensive use of *templates*.

Thus polynomials over rings are defined over any ring of coefficients that provide the necessary *methods* (operations) required by a ring.

Structure of the Thesis

Linear Systems

Part 1

Zeilberger's Algorithm in the Macsyma and Maxima Computer Algebra Systems

Chapter 1

Zeilberger's Algorithm in Macsyma

First implementation on a public domain system

Part 2

Solving Systems of Linear Equations by Interpolation

Chapter 2

Cramer Driven Interpolation

It could be used to parallelize Zeilberger's Algorithm

Chapter 3

Rational Interpolation

It can be used for a probabilistic Zeilberger's Algorithm

Chapter 4

Linear Systems in Zeilberger's Algorithm

New abstract model to undertsand Zeilberger's Algorithm

Polynomial Arithmetic

Part 3

Polynomial Arithmetic Library

Chapter 5

Combinatorial Interpretation of Division

Relation between polynomial division and integer compositions

Chapter 6

The Polynomial Package

Implementation of fast polynomial arithmetic related to some combinatorial problems

Main Contributions

The following items present a condensed list of my scientific contributions which I have achieved in my PhD thesis work and which are new.

- The first implementation of Zeilberger's algorithm for definite hypergeometric summation in the Maxima and Macsyma computer algebra systems. (Chapter 1)

- Two Mathematica libraries for the computation of the null space of a homogeneous system of linear equations by interpolation. The second library uses a hybrid method based on rational interpolation that generalizes [RZ] (Chapter 2 and Chapter 3).

- A Combinatorial model that interprets the Gosper form of a rational function in a new way. The application of this model to the study of the degree of the components of the solutions of the linear systems related to Zeilberger's algorithm (Chapter 4).

- Combinatorial interpretation of the computation of polynomial division and of the modular inverse of a polynomial in terms of integer compositions (Chapter 5).

- A C++/Mathematica library for polynomial arithmetic with special focus on the computation of modular powers of polynomials and division by powers of polynomials (Chapter 6).

Basic Notations

$\mathbb{N}$	set of non-negative integer numbers
$\mathbb{Z}$	ring of integer numbers
$\mathbb{Z}_p$	field of the residue classes modulo p (p is a prime)
$\mathbb{Q}$	field of rational numbers
$\emptyset$	empty set
$R[x]$	ring of polynomials in x over the ring R
$R[x_1, \ldots, x_n]$	ring of polynomials in x_1,...,x_n over the ring R
$R[[x]]$	ring of power series in x over R
$R^{m,n}$	ring of $m \times n$ matrices over the ring R
$F(x)$	field of rational functions over the field F
$\lvert a\rvert$	absolute value of a
$\lvert A\rvert$	determinant of A
$\lvert\lvert a\rvert\rvert_1$	1-norm of a vector
$\lvert\lvert a\rvert\rvert_2$	Euclidean norm of a vector
$a \mid b$	a divides b
$a \nmid b$	a does not divide b
$\lceil a \rceil$	smallest integer greater or equal to a (a is real)
$\lfloor a \rfloor$	greatest integer less or equal to a (a is real)
$[a_1, \ldots, a_n]$	n-tuple of a_1,...,a_n
$\mathcal{O}$	complexity class in the sense of [GKP94]
iff	"if and only if"
:=	assignment
$\square$	end of proof

Part I

Zeilberger's Algorithm in the Macsyma and Maxima Computer Algebra Systems

Chapter 1

Zeilberger's Algorithm in Macsyma

1.1 Introduction

We present the first implementation within the Macsyma and Maxima computer algebra systems (see [Sch], [Mac96]) of Zeilberger's Fast Algorithm for the definite summation problem for the large class of proper hypergeometric terms, i. e. given a 2-variable term $F(n,k)$, we want to rewrite the definite sum $f(n) = \sum_{k=0}^{n} F(n,k)$ in a form free of $\sum$. We do this by finding a special k-free linear recurrence with polynomial coefficients for the summand $F(n,k)$, which, under the condition of natural boundary for $F(n,k)$, can be extended into a k-free homogeneous linear recurrence for the sum $f(n)$ with polynomial coefficients. The desired linear recurrence for the summand has the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$. The search for this recurrence is done by guessing the order m of the left hand side of this recurrence and then trying to find the corresponding $G(n,k)$ of the right hand side by means of a modified Gosper's algorithm (We note that a priori upper bounds for m can be given; however they turn out to be too large and therefore not useful for practical use). A new version of Gosper's algorithm for this purpose has also been implemented and included in this package.

Gosper's algorithm for the indefinite summation problem solves the hypergeometric telescoping problem, i. e. given a hypergeometric term $t(k)$, it decides if there is a hypergeometric term $T(k)$ such that $t(k) = \Delta_k T(k) = T(k+1) - T(k)$, and if it exists, it finds it.

The standard Gosper's algorithm cannot be used in Zeilberger's algorithm because it does not take into account that there are some polynomial parameters. Therefore a modified version of this algorithm, in which the polynomial unknown parameters a_i's are taken into account, has been implemented and it is used by our implementation of Zeilberger's algorithm to compute the right hand side of the desired recurrence.

These implementations do not consider the more general q-case, for which a Mathematica version has already been developed at RISC, University of Linz, Austria (see [PR97]). Moreover we remark that at RISC a Mathematica version of Zeilberger's fast algorithm for the ordinary case $q = 1$, considered in this report, has already been implemented by Paule and Schorn [PS95]. Both packages are available at the world wide web address

http://www.risc.uni-linz.ac.at/research/combinat/risc/

1.2 Some Theory

We now introduce the basic definitions of the theory of definite and indefinite hypergeometric summation and some fundamental results of the theory which are necessary for the correctness of Zeilberger's fast algorithm and Gosper's algorithm.

For a formal description of Gosper's algorithm and Zeilberger's fast algorithm and of the proofs of their correctness we refer to Gosper's original paper [Gos78] and to Zeilberger's papers [Zei90], [Zei91], respectively. For a tutorial description of these algorithms we refer to [GKP94] and [PWZ97].

1.2.1 Some Fundamental Definitions

We are interested in treating sequences $(t(n))_n$ but in our algebraic setting we will treat terms $t(n)$ with free variable n over the integers.

Definition 1.2.1. Given a term $t(n)$ over a field K of characteristic 0, we call $t(n+1)/t(n)$ *shift quotient* of $t(n)$.

Definition 1.2.2. A term $t(n)$ over a field K of characteristic 0 is said to be *hypergeometric* if and only if the shift quotient $t(n+1)/t(n)$ is a rational function in n with coefficients in K.

We can extend this definition to multivariate terms in the natural way:

Definition 1.2.3. A term $t(x_1, \ldots x_m)$ over a field K of characteristic 0 is said to be *hypergeometric* with respect to x_s, where $1 \leq s \leq m$, if and only if the quotient $t(x_1, \ldots, x_s + 1, \ldots x_m)/t(x_1, \ldots, x_m)$ is a rational function in n with coefficients in K.

Definition 1.2.4. Given a 2-variable term $F(k,n)$ over a field K of characteristic 0 is said *proper hypergeometric* if and only if it is in the following form:

$$F(n,k) = p(n,k)\frac{\prod_{i=0}^{I}(a_i k + b_i n + c_i)!}{\prod_{j=0}^{J}(d_j k + e_j n + f_i)!}x^k$$

where $p(k,n)$ is a polynomial in k and n, and a_i, b_i, c_i d_j, e_j, f_i are integers and x is a parameter.

This definition extends in the natural way to m-variable case, where $m \geq 2$.

1.2.2 Some Fundamental Facts

In the following we will always denote by K a field of characteristic 0.

Property 1.2.1. *Given a hypergeometric term $t(n)$ over a field K. If there is a hypergeometric term $T(n)$ such that $t(n) = T(n+1) - T(n)$, then we have $t(n+1)/t(n) = r(n)T(n)$ where $r(n)$ is a rational function.*

Property 1.2.2. *Any proper hypergeometric term $F(n,k)$ is hypergeometric with respect to n and k.*

Theorem 1.2.3. (Existence of a k-free recurrence) *Let $F(n,k)$ be a proper hypergeometric term, then F satisfies a nontrivial recurrence of the form:*

$$\sum_{i=0}^{m} a_i(n)F(n,k) = G(n,k+1) - G(n,k), \tag{1.1}$$

where $G(n,k)/F(n,k)$ is a rational function in n and k.

We will refer to the rational function $G(n,k)/F(n,k)$ of Theorem 1.2.3 as “rational certificate”.

1.3 The Algorithms

1.3.1 Gosper's Algorithm

Gosper's algorithm solves the hypergeometric indefinite summation problem: given a hypergeometric term $t(n)$, it decides whether there exists another hypergeometric term $T(n)$ such that $t(n) = T(n+1) - T(n)$. See Algorithm 1.1 for the pseudo-code.

Algorithm 1.1 Gosper's Algorithm

Input: A hypergeometric term $t(n)$ over a field K of characteristic 0.

Output: If there is a hypergeometric term $T(n)$ such $t(n) = T(n+1) - T(n)$, then return $T(n)$, else return `‘‘no hypergeometric solution’’`

1: Compute $rat_t(n) := t(n+1)/t(n)$;

2: Write $rat_t(n)$ in the following form:

$$rat_t(n) = \frac{p(n+1)}{p(n)} \frac{q(n)}{r(n+1)} \tag{1.2}$$

where $\gcd(q(n), r(n+j)) = 1$ for all positive integers j's, and p, q, and r are polynomials. This process is described in detail in Chapter 4 in Algorithm 4.1.

3: Solve the following linear polynomial recurrence equation:

$$p(n) = q(n)s(n+1) - r(n)s(n), \tag{1.3}$$

where the polynomial $s(n)$ is the unknown.

4: **if** no $s(n)$ found **then**

5: **return** `‘‘no hypergeometric solution’’`

6: **else**

7: **return**

$$T(n) = \frac{1}{(\frac{s(n+1)}{s(n)} \frac{q(n)}{r(n)} - 1)} \cdot t(n).$$

Remark 1.3.1. The special form for the rational function rat_t, called *Gosper condition*, (see Algorithm 1.1) can be computed by forcing the gcd condition by resultant computation and simple rewriting. For more details see Algorithm 4.1. In the proper hypergeometric case, this step can also be done by simple inspection.

Remark 1.3.2. The linear recurrence equation (4.71) in Algorithm 1.1, known as *Gosper equation*, can be computed by plugging into it a polynomial with unknown coefficients, whose degree can be properly bounded. For more details on how to bound the degree of the polynomial we refer to [GKP94], page 227.

Why it works

For a formal proof of the correctness of this algorithm we refer to [PWZ97], [Gos78], [Sch95] and [GKP94], pages 223–241.
The idea of this algorithm is that we can rewrite an indefinite hypergeometric summation problem into a linear recurrence. In fact, by Property 1.2.1 and Property 1.2.2, we have that the solution $T(n)$ of the indefinite summation problem and the input function $t(n)$ must differ by a factor given by rational function $r(t)$:

$$T(n) = r(n)t(n).$$

By plugging this into the desired property: $t(n) = T(n+1) - T(n)$ and by dividing both sides by $t(n)$, we obtain the following recurrence equation over rational functions:

$$1 = r(n+1)rat_t(n) - r(n)$$

where $rat_t(n) := t(n+1)/t(n)$. Gosper could not solve this kind of recurrence equations, which today can be solved by Abramov's method.[Abr95] For an alternative approach and a detailed explanation of the connection between these strategies see, for instance, [Pau95]. Gosper had to find a way to get round this problem: he rewrites rat_t in a special form (See (1.2) in Algorithm 1.1) that leads to a simpler linear recurrence equation in which the unknown is a polynomial, and that can be solved by simple linear algebra.

1.3.2 Zeilberger's Algorithm

Zeilberger's algorithm solves the definite hypergeometric summation problem: given a proper hypergeometric term $F(n,k)$ with natural boundary, (i.e. for any fixed n, $F(n,k)$ has finite support with respect to k), it finds a k-free recurrence for $F(n,k)$ of the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$, which under the condition of natural boundary for $F(n,k)$ yields a homogeneous linear recurrence for the sum $f(n) = \sum_{k=0}^{n} F(n,k)$. See Algorithm 1.2 for the pseudo-code.

Algorithm 1.2 Zeilberger's Algorithm

Input: A hypergeometric term $F(n,k)$ in n and k over a field K of characteristic 0.

Output: A k-free linear recurrence with polynomial coefficients for $F(n,k)$ of the form:

$$\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k) \tag{1.4}$$

Try Gosper's algorithm, on $F(n,k)$;
if it finds a solution **then**
  return this solution;
$m := 1$;
while no solution yet **do**
  Try the modified version of Gosper's algorithm on the term

$$\sum_{i=0}^{m} a_i(n)F(n,k)$$

  in which the a_i's are polynomial unknowns.
  if some polynomials a_i's and a hypergeometric term $G(n,k)$ can be found for which the above recurrence (1.4) is satisfied **then**
    return $[G, [a_0, \ldots, a_m]]$
  else
    $m := m + 1$;

Remark 1.3.3. The modified version of Gosper's algorithm is formally the same as the standard one; the only difference is that in this version the a_i's appear as polynomial parameters and the solving of the recurrence equation $p(n) = q(n)s(n+1) - r(n)s(n)$ is now done by plugging in unknown polynomials whose coefficients are no more in the ground field K, but in the ring of rational functions with coefficients in K.
Therefore the problem of definite summation can be rewritten as the problem of solving systems of linear equations with polynomial coefficients.

Remark 1.3.4. Under the hypothesis of natural boundary (finite support) of $F(n,k)$, for instance let us assume that the support is strictly contained in the interval $[0,p]$, with $p>0$, the special recurrence for the summand $F(n,k)$, namely $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$ leads to a (homogeneous) recurrence for the sum $f(n)$, which is what we are ultimately looking for.
In fact, summing the left hand side and right hand side of the special recurrence over a boundary larger than the support of $F(n,k)$ we get:

$$\sum_{k=0}^{p}\sum_{i=0}^{m} a_i(n)F(n+i,k) = \sum_{k=0}^{p}(G(n,k+1) - G(n,k)) = G(n,p+1) - G(n,0) = 0$$

Hence:

$$\sum_{i=0}^{n} f(n+i) = G(n,p+1) - G(n,0) = 0$$

When the summand does not have finite support the above equation will be non-homogeneous.

Why it works

For a formal proof of the correctness we refer to the literature on this topic ([PWZ97], [Zei90], [Zei91], [Sch95]). Here we present the basic idea behind it.

From Theorem 1.2.3 that states the existence of a k-free recurrence (see Section 1.2.2) we know that for any proper hypergeometric term $F(n,k)$, there exists a recurrence of the form: $\sum_{i=0}^{m} a_i(n)F(n+i,k) = G(n,k+1) - G(n,k)$. Zeilberger's algorithm tries to find this recurrence by iteratively increasing the order of the recurrence and by using a modified version of Gosper's algorithm to find its right hand side; this procedure stops because of the termination of Gosper's algorithm and because of the theorem mentioned above.

1.4 The Implementation

In this section we report on some of the details of our implementation of *Zeilberger's fast algorithm* and of *Gosper's algorithm*.

The implementation of these two algorithms has been done in the internal LISP-like language of the Macsyma and Maxima computer algebra systems.

I implemented both algorithms in Macsyma in a straightforward way and no significant changes have been made in the original algorithms.

1.4.1 Gosper's Algorithm

My implementation of Gosper's algorithm works in the special case of proper hypergeometric terms, but it can be easily extended to the general hypergeometric case by substituting the Gosper form related routine with a more general routine in which the Gosper form is achieved by a resultant computation. (See [GKP94] for more details).

As a consequence of this implementation choice the desired special form at step 1 for the rational function $rat_t(n)$ is constructed by initializing $p(n) := 1$, $q(n) := numerator(rat_t)$, $r(n+1) := denominator(rat_t)$ and by simple inspection and rewriting of the factors of the polynomials q and r (See Algorithm 4.1 for more details).

1.4.2 Zeilberger's Fast Algorithm

This version of Zeilberger's algorithm works in the proper hypergeometric case. It was conjectured that Zeilberger's algorithm could have worked on a larger class of terms, namely the class of holonomic terms (see [Chy98]) but it has been proved that the class of proper hypergeometric terms and holonomic terms coincide (see [APb], [APa]).

The computation of the recurrence for the sum and its solution is left to the user as a post-processing. Moreover the existence of a solution expressible in elementary terms is not always guaranteed, therefore leaving the solution as a recurrence is a reasonable choice.

1.5 Performance

First of all we must point out that this implementation is only the first attempt to incorporate Zeilberger's fast algorithm into the Macsyma and Maxima computer algebra systems and therefore it is very far from being competitive with the best existing optimized implementations (for example see [PS95]).

No large scale test has been carried out to assess the speed of this implementation. The only test that has been run on this implementation is the collection of terms contained in the files `testGosper.macsyma` and `testZeilberger.macsyma`, among which there are some powers of the binomial coefficient. Increasing powers of the binomial coefficients have been used to test the stability of the system and to obtain a rough idea of the performance of the system and to compare it to some other implementations. I used this test suite because of its simplicity and because we know a priori that the order of the recurrence for the summand is $\lceil \frac{p}{2} \rceil$, where p is the power. (see [Cus89]).

1.5.1 Timings

We present some timings obtained by testing the function `parGosper` on increasing powers of the binomial coefficients in which no loop on the order of the recurrence is run. In these tests the Paule-Schorn Mathematica implementation [PS95] and our Macsyma implementation are compared. These implementations have been tested on an SGI Octane with 2 gigabytes of RAM and two 250 Mhz RISC processors (although much less memory was necessary to run the programs on these examples).

Results Note: Timings are in seconds

power	**order**	Paule/SchornMathematica	Macsyma	Ratio
3	2	$0.36s$	$3.31s$	9.19
4	2	$0.92s$	$3.95s$	4.29
5	3	$4.47s$	$37.11s$	8.30
6	3	$16.98s$	$121.64s$	7.16
Average ratio :				7.23

1.5.2 Stability

Our Macsyma implementation tested on an SGI Octane equipped with 2 gigabytes of RAM could no go beyond the sixth power of the binomial coefficient whereas Paule-Schorn Mathematica implementation was able to handle the eleventh power. The sixth power of the binomial coefficient required less than 50 megabytes of memory.

1.5.3 Future Improvements

An analysis on the distribution of the computation time shows that in the "heavy cases" (binomial coefficient to the fifth and sixth powers) the bottle-neck is the computation of the solution of the linear systems of equations that is required to solve the recurrence equation (*Gosper's equation*).

This version uses the built-in linear solver of Macsyma and Maxima, whereas Paule-Schorn Mathematica uses a special linear solver tailored for sparse systems. Future optimized versions should use an ad hoc linear solver that could take advantage of the specific structure of the system.

1.6 Manual

Now we describe how to load and use the package.

1.6.1 Loading the files

The entire package can be downloaded from the RISC combinatorics home page at the following URL http://www.risc.uni-linz.ac.at/research/combinat/risc/
The user will find the following files:

1. algUtil.macsyma
2. shiftQuotient.macsyma
3. poly2quint.macsyma
4. makeGosperForm.macsyma
5. GosperEq.macsyma
6. Gosper.macsyma
7. Zeilberger.macsyma
8. LOADZeilberger.macsyma
9. testZeilberger.macsyma
10. testGosper.macsyma

The entire package can be loaded into memory by simply loading the file `LOADZeilberger.macsyma`; this file will take care of loading the other components except the files containing some examples on which the system has been tested, namely `testZeilberger.macsyma` and `testGosper.macsyma`.

1.6.2 The Commands

Both Gosper and Zeilberger's algorithm have been implemented in two versions: a verbose version, which allows the user to choose different levels of verbosity, and a non-verbose version.

Verbosity

The levels of verbosity are selected by the user just appending a suffix to the command name (`Gosper`, `parGosper`, `Zeilberger`) or by passing a parameter in the generic verbose versions of the command, which is obtained by appending the suffix `VerboseOpt` to the command name. No suffix is interpreted as non-verbose.

These are the levels of verbosity that have been implemented in both algorithms:

Level	Suffix	Scope
summary	`Summary`	summary of the main computations
normal	`Verbose`	verbosity on the main procedure
very	`VeryVerbose`	verbosity on the subroutines
debugging	`Debugging`	strongest verbosity
linsys	`LinSys`	verbosity on the linear system

Example 1.6.1.
`GosperVerbose(f,k)` invokes Gosper's algorithm in verbose mode.

Example 1.6.2.
`ZeilbergerVeryVerbose(f,k,n)` invokes Zeilberger's algorithm in the very verbose mode.

Gosper's Algorithm

- `Gosper(`*f*`,`*k*`)`

  It solves the indefinite summation problem of finding a hypergeometric term $g(k)$ such that $f(k) = \Delta_k g(k) = g(k+1) - g(k)$. If such a hypergeometric term exists it returns it as output, otherwise it will return **NO_HYP_SOL**.

- `GosperVerboseOpt(`*f*`,`*k*`,`*verbosity*`)`

  As `Gosper` but the level of verbosity is passed as a parameter.

- `GosperSum(`*f*`, `*k*`, `*a*`, `*b*`)`

  It computes $\sum_{k=a}^{b} f$ by using `Gosper` to solve the indefinite sum. (It only works if the indefinite sum is Gosper-summable).

- `GosperSumVerboseOpt(`*f*`, `*k*`, `*a*`, `*b*`, `*verbosity*`)`

  As `GosperSum` but the level of verbosity is passed as a parameter.

Zeilberger's Fast Algorithm

- `Zeilberger(`*F, k, n*`)`

  Given a 2-variable proper hypergeometric term $F(n,k)$, it computes by Zeilberger's algorithm a recurrence equation for F of the form:

$$\sum_{i=0}^{d} a_i(n)F(n+i,k) = \Delta_k(Cert(n,k)F(n,k)), \tag{1.5}$$

  where the a_i's are polynomials free of k and $Cert$ ("rational certificate") is a rational function in n and k. The output will be a print-out of the recurrence and the explicit values of the polynomial parameters a_i and the "rational certificate" $Cert(n,k)$.

- `ZeilbergerVerboseOpt(`*F, k, n, verbosity*`)`

  As `Zeilberger` but the level of verbosity is passed as a parameter.

- `parGosper(`*F, k, n, d*`)`

  Given a 2-variable proper hypergeometric term, it computes, when it exists, a recurrence equation of order d for F of the form:

$$\sum_{i=0}^{d} a_i(n)F(n+i,k) = \Delta_k(R(n,k)F(n,k)), \tag{1.6}$$

  where a_i's are polynomials and R is a rational function (the *certificate*), and it yields a term $[R,[a_0,\ldots,a_d]]$; if no such recurrence exists them it yields $[0,[\text{dummy value},\ldots,\text{dummy value}]]$.

- `parGosperVerboseOpt(`*F, k, n, d, verbosity*`)`

  As `parGosper` but the level of verbosity is passed as a parameter.

Settings

The only setting is the environment variable `MAX_ORD` that sets an a priori bound on the order of the recurrence that Zeilberger's algorithm iteratively tries to find by applying `parGosper` with increasing order. The default value of `MAX_ORD` is 3.

1.7 Some Examples

Let us take a look at some examples, that can be found in the files

- `testGosper.macsyma`

- `testZeilberger.macsyma`.

Example 1.7.1. A simple Gosper-summable example
Let us try to telescope $\frac{1}{4k^2-1}$ by Gosper's algorithm:
(prompt) `Gosper(1/4*k^2-1,k);`
output: $-\frac{1}{2(2k-1)}$

Let us now try to sum it over the interval $[1,4]$, i.e. evaluate $\sum_{k=1}^{4} \frac{1}{4k^2-1}$
(prompt) `GosperSum(1/4*k^2-1, k, 1, 4);`
output: $\frac{4}{9}$

Example 1.7.2. A less simple Gosper-summable example
Let us try to telescope $\frac{(-1)^k k}{4k^2-1}$ by Gosper's algorithm:
(prompt) `Gosper(((-1)^k*k)/(4k^2-1),k);`
output: $-\frac{(-1)^k}{4(2k-1)}$

Example 1.7.3. An example involving factorials
(prompt) `Gosper((a!*(-1)^k)/((a-k)!*k!),k);`
output: $\frac{a!k(-1)^k}{a(a-k)!k!}$

Example 1.7.4. A non-Gosper-summable case
(prompt) `Gosper(binomial(n,k),k,n);`
output: **NO_HYP_SOL**

To handle this we must use `parGosper` or `Zeilberger` as shown in the next example.

Example 1.7.5. Binomial coefficient
To evaluate $\sum_{i=0}^{m} \binom{n}{k}$ we can use `Zeilberger` or `parGosper` and use the fact that we expect a first order recurrence:
(prompt) `Zeilberger(binomial(n,k),k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \binom{n}{k}$$

and

$$Cert(n,k) = -\frac{k}{n-k+1}$$

and

$$a[0](n) = -2$$
$$a[1](n) = 1$$

Summing both sides of the recurrence and computing the sum is left as simple post-processing.

Assuming that the recurrence has order 1, we could have used `parGosper`:

(prompt) `parGosper(binomial(n,k),k,n,1);`

output:

$$\left\{-\frac{k}{n-k+1}, \{-2, 1\}\right\}$$

Example 1.7.6. Squared binomial coefficient
We can try to sum the squared binomial coefficient with `parGosper`, but to do this we must guess the order of the recurrence:

(prompt) `parGosper(binomial(n,k)^2,k,n,1);`

output:

$$\left\{-\frac{k^2\ (3\,n-2\,k+3)}{(n-k+1)^2}, \{-2\ (2\,n+1)\,, n+1\}\right\}$$

Example 1.7.7. Binomial Theorem
Let us try a similar example:

(prompt) `Zeilberger(binomial(n,k)*x^k,k,n);`

output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \binom{n}{k} x^k$$

and

$$Cert(n,k) = -\frac{k}{n-k+1}$$

and

$$a[0](n) = -(x+1)$$
$$a[1](n) = 1$$

Example 1.7.8. Vandermonde identity recurrence (see [GKP94], page 169)
(prompt) `Zeilberger(binomial(a,k)*binomial(b,n-k),k,n);`

output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \binom{a}{k}\binom{b}{n-k}$$

and

$$Cert(n,k) = \frac{k(-n+k+b)}{n-k+1}$$

and

$$a[0](n) = -(n-b-a)$$
$$a[1](n) = -(n+1)$$

Example 1.7.9. First Karlsson-Gosper identity (see [Kar86])
(prompt)

```
Zeilberger(binomial(n,k)*(n-1/4)!/(n-k-1/4)!/
           (2*n+k +1/4)!*9^(-k),k,n);
```

output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{(n-\frac{1}{4})!\binom{n}{k}}{9^k(n-k-\frac{1}{4})!(2n+k+\frac{1}{4})!}$$

and

$$Cert(n,k) = \frac{144k(8n+4(k-1)+13)(52n^2+16kn+75n-32k^2+24k+26}{(n-k+1)(4n-4k+3)(8n+4k+5)(8n+4k+9)}$$

and

$$a[0](n) = 2^8$$
$$a[1](n) = -27(3n+2)(12n+13)$$

Example 1.7.10. Second Karlsson-Gosper identity (see [Kar86])
(prompt)

```
Zeilberger(binomial(n,k)*(n-1/4)!/(n-k-1/4)!/
          (2*n+k+5/4)!*9^(-k),k,n);
```

output:

$$a[0]f(n,k) + a[1]f(n+1,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{(n-\frac{1}{4})!\binom{n}{k}}{9^k(n-k-\frac{1}{4})!(2n+k+\frac{5}{4})!}$$

and

$$Cert(n,k) = \frac{144k(8n+4(k-1)+17)(52n^2+16kn+127n-32k^2-4k+72)}{(n-k-1)(4n-4k+3)(8n+4k+9)(8n+4k+13)}$$

and

$$a[0](n) = 2^8$$
$$a[1](n) = -27(3n+4)(12n+17)$$

Example 1.7.11. Trinomial coefficients
(prompt) `Zeilberger(n!/k!/(k+m)!/(-2*k-m+n)!,k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k) = \frac{n!}{k!(n+k)!(n-m-2k)!}$$

and

$$Cert(n,k) = \frac{4k(m+k)(n+1)(n+2)}{(n-m-2k+1)(n-m-2k+2)}$$

and

$$a[0](n) = 3(n+1)(n+2)$$
$$a[1](n) = (n+2)(2n+3)$$
$$a[2](n) = -(n-m+2)(n+m+2)$$

Example 1.7.12. Special case of the Strehl identity (see [Str94a])
(prompt) `Zeilberger(binomial(2*k,k)*binomial(n,k)^2,k,n);`
output:

$$a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) = \Delta_k(Cert(n,k)f(n,k))$$

where

$$Cert(n,k) = -\frac{k^3(n+1)^2(4n-3k+8)}{(n-k+1)^2(n-k+2)^2}$$

and

$$\begin{aligned} a[0](n) &= 9(n+1)^2 \\ a[1](n) &= -(10n^2+30n+23) \\ a[2](n) &= (n+2)^2 \end{aligned}$$

Example 1.7.13. Another second order recurrence
(prompt) `Zeilberger((n!*(n+k)!)/(k!^3*(n-k)!^2),k,n));`
output:

$$\begin{aligned} a[0]f(n,k) + a[1]f(n+1,k) + a[2]f(n+2,k) + a[3]f(n+3,k) &= \\ \Delta_k(Cert(n,k)f(n,k)) \end{aligned} \tag{1.7}$$

where

$$f(n,k) = \frac{n!(n+k)!}{k!^3(n-k)!^2}$$

and

$$Cert(n,k) = -\frac{k^3(n+1)(11n^2-6kn+37n-k^2-7k+30)}{(n-k+1)^2(n-k+2)^2}$$

and

$$\begin{aligned} a[0](n) &= -(n+1)^2 \\ a[1](n) &= -(11n^2+33n+25) \\ a[2](n) &= (n+2)^2 \end{aligned}$$

Example 1.7.14. The binomial coefficient to the third power
We can also use `parGosper` to find a recurrence on such sequence, but we must guess its order:
(prompt) `parGosper(binomial(n,k)^3,k,n,1);`
output: $\{0, \{0, 0\}\}$
This means that `parGosper` could not find a first order recurrence of the desired form but we don't give up. Let us look for a second order recurrence:
(prompt) `parGosper(binomial(n,k)^3,k,n,2);`
output:

$$\begin{aligned}&\big\{(k^3\,(n+1)^2\,\big(14n^3-27\,kn^2+74n^2+18k^2n\\&-93kn+128\,n-4k^3+30k^2-78k+72\big))/\\&\quad((n-k+1)^3\,(n-k+2)^3),\\&\;\big\{8\,(n+1)^2\,,7n^2+21\,n+16,-(n+2)^2\big\}\big\}\end{aligned}\tag{1.8}$$

Example 1.7.15. Binomial coefficient to the fourth power
(prompt) `Zeilberger(binomial(n,k)^4,k,n);`
output:

$$a[0]f(n,k)+a[1]f(n+1,k)+a[2]f(n+2,k)=\Delta_k(Cert(n,k)f(n,k))$$

where

$$f(n,k)=\binom{n}{k}^4$$

and

$$\begin{aligned}&Cert(n,k)=\\&=-\frac{\begin{array}{c}k^4(n+1)(74n^6-260kn^5+725n^5+374k^2n^4-2056kn^4+2885n^4-\\276k^3n^3-6420kn^3+6045n^3+104k^4n^2-1244k^3n^2+5298k^2n^2-\\9892kn^2+7030n^2-16k^5n+298k^4n-1884k^3+5322k^2n-7520kn+\\4300n-20k^5+210k^4-900k^3+1980k^2-2256k+1080\end{array}}{(n-k+1)^4(n-k+2)^4}\end{aligned}\tag{1.9}$$

and

$$a[0](n) = -4(n+1)(4n+3)(4n+5)$$
$$a[1](n) = -2(2n+3)(3n^2+9n+7)$$
$$a[2](n) = (n+2)^3$$

1.8 The Code

The implementations of both Gosper's algorithm and Zeilberger's fast algorithm have been entirely coded in the internal LISP-like language of the computer algebra systems Macsyma vers.419 and Maxima 5.5. The implementation exploits Macsyma computer algebra engine for factorizing, simplifying and normalizing polynomials and rational functions, and the modularity of the Macsyma language, but it does not use Macsyma hypergeometric tools like Gosper's implementation of his own algorithm because Zeilberger's algorithm requires a parametrized version of Gosper's algorithm.

The entire code can be found in the RISC combinatorics home page at the U.R.L.: http://www.risc.uni-linz.ac.at/research/combinat/risc/.

The code is contained in the following files:

`algUtil.macsyma`	algebraic utilities
`shiftQuotient.macsyma`	shift quotient computation
`poly2quint.macsyma`	internal data structures conversions
`makeGosperForm.macsyma`	Gosper form related routines
`GosperEq.macsyma`	Gosper equation related routines
`Gosper.macsyma`	Gosper's algorithm main routines
`Zeilberger.macsyma`	Zeilberger's algorithm main routines
`LOADZeilberger.macsyma`	Zeilberger's routines loader
`testZeilberger.macsyma`	Some examples
`testGosper.macsyma`	Some Gosper's algorithm related examples

1.8.1 Low Level Routines

The low level routines are contained in the files `algUtil.macsyma`, `shiftQuotient.macsyma` and `poly2quint.macsyma`.

The first file contains the lowest level algebraic routines for handling polynomials (extracting components of polynomials, degree, etc).

The second file contains routines necessary for computing the shift quotient of a hypergeometric term and for computing the result of the application of a linear recurrence operator to a hypergeometric term, which is computed by a strategy similar to the one used in Horner's algorithm for polynomial evaluation:

```
niceForm(hyp,var,parName,ord) :=
  block(
  [shQuo,num,den,res],
  res:parName[ord],
  shQuo :  shiftQuo(hyp,n),
  for i :  ord step -1 thru 1 do
   res :  xthru(parName[i-1] +
     shiftFactPoly(shQuo,n,i-1)*res),
  return(res)
  );
```

The third file contains the routines necessary for storing polynomials in special data structures; namely *quintuples* that are well-suited for building the desired *Gosper form* of the shift quotient of a proper hypergeometric term. Such *quintuples* are arrays in which the following information of a polynomial is stored:

- degree
- leading coefficient
- second leading coefficient
- tail (polynomial without the first two monomials)
- multiplicity (number of occurrences) of the polynomial in the shift quotient

Example 1.8.1.
If the polynomial $(7x^3 - 5x^2 + 4x + 8)$ appears in the shift quotient with a multiplicity, say 5, it will be encoded in the following quintuple : $[3, 7, -5, x^2 + 4x, 5]$.

1.8.2 Gosper Form Routines

All the Gosper form (1.2) related routines are in `makeGosperForm.macsyma`. The main routine builds the desired form by checking iteratively the gcd condition required by the Gosper form. Whenever the condition is violated the undesired factors will be moved from the polynomials q and r to the polynomial p. This procedure exploits the ad hoc data structure *quintuple*, which has been coded in `poly2quint.macsyma`. A detailed description of this process is treated in Chapter 4.

1.8.3 Gosper Equation Routines

The computation of the degree of the solution and the solution of the *Gosper equation* (1.3) has been coded in the file `GosperEq.macsyma`. Solving is done by simple linear algebra on the unknown coefficients of the solution.

1.8.4 Gosper's and Zeilberger's Algorithm

Gosper's and Zeilberger's algorithm have been coded in two versions (verbose and non-verbose) in the file `Gosper.macsyma`.

Zeilberger's fast algorithm is a parametrized version of Gosper's algorithm, in which the linear solving of the recurrence equation is done with polynomial parameters.

The coding follows in a straightforward way Gosper's algorithm as described in the previous sections.

Part II

Solving Systems of Linear Equations by Interpolation

Chapter 2

Cramer Driven Interpolation

2.1 Introduction

We describe an algorithm based on Cramer's rule and on polynomial interpolation that solves overdetermined systems of homogeneous linear equations over $\mathbb{Q}$ and over the field of rational functions over $\mathbb{Q}$ under the hypothesis that the null space has dimension 1. We restrict our attention to the case of overdetermined systems with one-dimensional null space because they are easier to study and because the systems involved in hypegeometric summation are very often of the same type. The algorithm described is a generalization of the algorithm in [Lip81], pages 243–251. The algorithm computes the numerators and denominators of the components of the solutions as determined by Cramer's rule. It does so by first computing homomorphic systems over finite fields $\mathbb{Z}_p$, with p prime and then by polynomial interpolation or by the Chinese remainder theorem.

2.2 Notation

In the following K will be a field containing $\mathbb{Q}$ and A an $m \times n$ matrix with entries over K, with $m \geq n$.

Let us denote the entries of A with $a_{i,j}$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, i. e.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \multicolumn{4}{c}{\dotfill} \\ \multicolumn{4}{c}{\dotfill} \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}. \tag{2.1}$$

Definition 2.2.1. We define the *the largest square upper part* of A, denoted by

$\overline{A}$, as the matrix:

$$\overline{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \multicolumn{4}{c}{\dots\dots\dots\dots\dots\dots} \\ \multicolumn{4}{c}{\dots\dots\dots\dots\dots\dots} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}. \tag{2.2}$$

For any given $m \times s$ matrix M and any column vector b with s entries, we denote by $M|b$ the $m \times (s+1)$ matrix obtained by appending the vector b to M.

Definition 2.2.2. We define by $L(A)$ the matrix obtained from A by removing its last column and by $R(A)$ the last column of A.

In the following we will refer to Gaussian elimination applied on a matrix as to any process that produces a (lower or upper) triangular matrix by operations that do not change the solutions of the corresponding system of linear equations. In practice when using homomorphic images the choice of the matrix should be done in a way that makes it very likely to have images that can be used for reconstructing the solutions of the system (see Remark 2.4.2).

Definition 2.2.3. For any matrix M we define the *triangular form* of M, denoted by $\tau(M)$, to be the matrix obtained from M by performing Gaussian elimination on it.

Definition 2.2.4. Given a matrix $M \in K^{m,s}$ and a column vector with m components, we call the i-th *Cramer matrix* of M with respect to b, denoted by $C_i(M, b)$, the matrix obtained from M by substituting the i-th column with b, for $1 \leq i \leq s$.

Example 2.2.1.
Let us consider

$$A = \begin{pmatrix} 2 & 4 & 5 \\ 7 & 0 & 3 \\ 1 & 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 6 \\ 2 \\ 0 \end{pmatrix} \tag{2.3}$$

Then

$$L(A) = \begin{pmatrix} 2 & 4 \\ 7 & 0 \\ 1 & 1 \end{pmatrix}, \quad R(A) = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}, \quad C_1(A, b) = \begin{pmatrix} 6 & 4 & 5 \\ 2 & 0 & 3 \\ 0 & 1 & 2 \end{pmatrix}. \tag{2.4}$$

2.3 The Homogeneous Problem

We consider the problem of solving the homogeneous system of linear equations

$$Ax = 0, \tag{2.5}$$

with $A \in K^{m,n}$, $m \geq n$ and when the dimension of the null space of A is 1.

Let us denote $L(A)$ by A' and let us assume without loss of generality that A' has maximal rank, i. e. $n-1$. Since the null space of A has dimension 1, then A has rank $n-1$ and we can set the value of the last component of the solution to (2.5) to 1 and consider the problem:

$$A'x = b, \tag{2.6}$$

where b is a column vector obtained by taking the opposite of the last column of A, i. e. its component i-th component is $-a_{i,n}$, for all $1 \leq i \leq m$.

A solution to (2.6) gives us the remaining $n-1$ components of the solution to (2.5) whose last component is 1. All the other solutions will be multiples.

We now face the problem of solving an overdetermined system of inhomogeneous linear equations of maximal rank.

2.4 Rational Coefficients

We denote by m a positive integer and by r a non-negative integer.

We consider the problem of finding the integers x such that

$$x \equiv r \mod m, \tag{2.7}$$

with

$$0 \leq r < m. \tag{2.8}$$

We consider an elementary result[1] which tells us how to recover an integer solution from a modular one.

Definition 2.4.1. For any r such that $0 \leq r < m$, we define $\lambda_m(r)$ as follows:

$$\lambda_m(r) = \begin{cases} r, & \text{if } r < m/2, \\ r - m, & \text{otherwise.} \end{cases} \tag{2.9}$$

[1]For a more detailed study of the solutions to integer congruences we refer to [Lip81], pages 243–251.

Theorem 2.4.1. *The congruence*

$$x \equiv r \mod m, \tag{2.10}$$

with

$$0 \leq r < m \tag{2.11}$$

has a unique solution s, such that

$$-m/2 \leq s < m/2, \tag{2.12}$$

given by $s = \lambda_m(r)$.

PROOF. Clearly the definition of $\lambda_m(r)$ (2.9) gives a solution in the desired range. Let us now denote by s_1 and s_2 two solutions of (2.10) in the range given by (2.11).

Therefore we have

$$s_1 \equiv s_2 \mod m. \tag{2.13}$$

If they were not equal, we could assume, without loss of generality, $0 \leq s_1 < s_2 < m$ and therefore we would have

$$0 < s_2 - s_1 < m; \tag{2.14}$$

but then, by (2.13), we would also have that $s_2 - s_1$ is a multiple of m, which contradicts (2.14).

□

Remark 2.4.1. The function λ_m describes the least positive solution to the congruence (2.10). Such a function is necessary when reconstructing possibly negative integer solutions for which only a bound on the absolute value is known.

2.4.1 Definitions

Now let us assume that the field K is the field $\mathbb{Q}$ of the rational numbers.

For a given matrix $M \in \mathbb{Q}^{m,n}$ with $m \geq n$ and a given prime p such that it does not divide any of the denominators of the entries of M, we denote by $\mathrm{mod}_p(M)$ the matrix obtained from M by taking its entries modulo p.

In the following we will always consider primes that do not divide any of the denominators of the entries in the considered matrix.

We consider those primes p such that, for $1 \leq i \leq n$

$$\begin{aligned}
&\det(\overline{L(\tau^{(p)})}) \neq 0,\\
&\det(\overline{L(\tau^{(p)})}) = \mathrm{mod}_p(\det(\overline{L(\tau)})),\\
&\det(\overline{C_i(L(\tau^{(p)}), R(\tau^{(p)}))}) = \mathrm{mod}_p(\det(\overline{C_i(L(\tau), R(\tau))})),
\end{aligned} \tag{2.15}$$

where

$$\begin{aligned}\tau &= \tau(M),\\ \tau^{(p)} &= \tau(\mathrm{mod}_p(M)).\end{aligned} \tag{2.16}$$

We call such primes "lucky" for M.

Remark 2.4.2. The luckiness of a prime does not only depend on M but also on τ. In order to reduce the risk of having "unlucky" choices, it is better to perform *Gaussian elimination* by only allowing addition to a row of a scalar multiple of some other row and interchange of two rows and by keeping some history of the row switches in order to later check whether a prime was probably "lucky".

2.4.2 Modular Approach

We can solve the overdetermined system of linear equations of maximal rank (2.6) by a modular approach, namely by considering modular homomorphisms.

Let us denote $\tau(A'|b)$ by τ and $\tau(\mathrm{mod}_{p_j}(A'|b))$ by $\tau^{(p_j)}$.

We consider a set of primes $\{p_1, \dots, p_l\}$ such that $B < \prod_{i=1}^{l} p_i$, where B is a known bound such that

$$B > 2\max(|d|, |d_1|, \dots, |d_{n-1}|), \tag{2.17}$$

where

$$\begin{aligned}d &= \det(\overline{L(\tau)}),\\ d_i &= \det(\overline{C_i(L(\tau), R(\tau))}).\end{aligned} \tag{2.18}$$

For each of the primes p_j we compute

$$\begin{aligned}d^{(p_j)} &= \det(\overline{L(\tau^{(p_j)})}),\\ d_i^{(p_j)} &= \det(\overline{C_i(L(\tau^{(p_j)}), R(\tau^{(p_j)}))}).\end{aligned} \tag{2.19}$$

An efficient way to compute $d^{(p_j)}$ in (2.19) is to first compute $\tau^{(p_j)}$ and then compute the determinant of its upper part as the product of the diagonal elements. The determinants $d_i^{(p_j)}$ can be computed by solving the triangularized system $\tau(\mathrm{mod}_{p_j}(A'|b))$ by back-substitution and taking, for each computed component of the solution, the product by $d^{(p_j)}$.

In fact if p_j is lucky, then by Cramer's rule we have

$$x_i^{(p_j)} = \frac{d_i^{(p_j)}}{d^{(p_j)}}, \quad 1 \le i \le n-1, \tag{2.20}$$

where $x_i^{p_j}$ is the i-th component of the solution modulo p_j of the system corresponding to $\text{mod}_{p_j}(A'|b)$.

Moreover, if the primes p_j are lucky for $A'|b$, then by (2.15), we must also have:

$$d^{(p_j)} \equiv d \mod p_j, \tag{2.21}$$

and

$$d_i^{(p_j)} \equiv d_i \mod p_j, \tag{2.22}$$

for $1 \leq j \leq l$ and $1 \leq i \leq n-1$.

We now consider the following Chinese remainder problems:

$$z \equiv d^{(p_j)} \mod p_j \ \ 1 \leq j \leq l, \tag{2.23}$$

and

$$z_i \equiv d_i^{(p_j)} \mod p_j \ \ 1 \leq j \leq l, \tag{2.24}$$

for $1 \leq i \leq n-1$.

By the Chinese remainder theorem (see [Win96] and Appendix A) we can solve the systems of congruences (2.23) and (2.24). Therefore we would find a non-negative integer solution d^* to (2.23) with $0 \leq d_i^* < \prod_{j=1}^{l} p_j$ and the non-negative integer solutions d_i^* to (2.24) with $0 \leq d_i^* < \prod_{j=1}^{l} p_j$.

We consider the system of congruences

$$y \equiv d^* \mod \prod_{j=1}^{l} p_j, \tag{2.25}$$

with

$$-(\prod_{j=1}^{l} p_j)/2 \leq y < (\prod_{j=1}^{l} p_j)/2, \tag{2.26}$$

and

$$y_i \equiv d_i^* \mod \prod_{j=1}^{l} p_j, \tag{2.27}$$

with

$$-(\prod_{j=1}^{l} p_j)/2 \leq y_i < (\prod_{j=1}^{l} p_j)/2 \tag{2.28}$$

for $1 \leq i \leq n-1$.

We notice that by (2.17) we have

$$\prod_{j=1}^{l} p_j \geq B > 2 \max(|d|, |d_1|, \cdots, |d_{n-1}|), \tag{2.29}$$

and that d and d_i are solutions respectively of (2.25) and (2.27). Therefore Theorem 2.4.1 guarantees the existence and uniqueness of the solution to (2.25) and (2.27) which must then be respectively d and d_i.

The components of the solution to (2.6) are obtained by Cramer's rule.

Example 2.4.1.
Let us consider the simple example

$$A = \begin{pmatrix} 2 & 3 & -5 \\ 1 & 1 & -4 \\ 4 & -2 & 3 \\ 3 & 4 & -9 \end{pmatrix}, \quad b = \begin{pmatrix} 3 \\ 2 \\ 2 \\ 5 \end{pmatrix}. \tag{2.30}$$

We know with hindsight that 89 is a big enough prime to reconstruct the components of solution of the matrix correctly, i. e. we know that

$$89 > 2 \cdot \max(|d|, |y_1|, |y_2|, |y_3|), \tag{2.31}$$

where d is the determinant of A and y_i is the determinant of $C_i(A, b)$, i. e. the i-th Cramer of A with respect to b.

Then performing Gaussian elimination on $\mathrm{mod}_{89}(A|b)$ over $\mathbb{Z}_{89}$ produces:

$$\mathrm{mod}_{89}(A|b) = \left(\begin{array}{ccc|c} 2 & 3 & 84 & 3 \\ 1 & 1 & 85 & 2 \\ 4 & 87 & 3 & 2 \\ 3 & 4 & 8 & 5 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 2 & 3 & 84 & 3 \\ 0 & 46 & 62 & 46 \\ 0 & 81 & 13 & 85 \\ 0 & 44 & 43 & 45 \end{array}\right) \rightsquigarrow \tag{2.32}$$

$$\left(\begin{array}{ccc|c} 2 & 3 & 84 & 3 \\ 0 & 46 & 62 & 46 \\ 0 & 0 & 47 & 4 \\ 0 & 0 & 34 & 1 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 2 & 3 & 84 & 3 \\ 0 & 46 & 62 & 46 \\ 0 & 0 & 47 & 4 \\ 0 & 0 & 0 & 0 \end{array}\right).$$

By back-substitution in $\mathbb{Z}_{89}$ we get

$$x_1 = 32, \quad x_2 = 12, \quad x_3 = 55. \tag{2.33}$$

The determinant in $\mathbb{Z}_{89}$ is given the product of the diagonal elements:

$$d' = \mathrm{mod}_{89}(\det(A)) = 52. \tag{2.34}$$

Using Cramer's rule (2.20) we get that the numerators in $\mathbb{Z}_{89}$ of the components of the solution are:

$$\hat{y}_1 = x_1 \cdot d' = 32 \cdot 52 \equiv 62 \qquad \mathrm{mod}\ 89, \tag{2.35}$$

$$\hat{y}_2 = x_2 \cdot d' = 12 \cdot 52 \equiv 1 \qquad \mathrm{mod}\ 89, \tag{2.36}$$

$$\hat{y}_3 = x_3 \cdot d' = 55 \cdot 52 \equiv 12 \qquad \mathrm{mod}\ 89. \tag{2.37}$$

$$\tag{2.38}$$

In order to reconstruct the determinants in $\mathbb{Z}$ we take the least absolute value integer (2.9):

$$\begin{aligned} d &= \lambda_{89}(d') = 52 - 89 = -37, \\ y_1 &= \lambda_{89}(y_1') = 62 - 89 = -27, \\ y_2 &= \lambda_{89}(y_2') = 1, \\ y_3 &= \lambda_{89}(y_3') = 12. \end{aligned} \tag{2.39}$$

The solution in $\mathbb{Q}$ will then be:

$$(y_1/d, y_2/d, y_3/d) = (27/37, -1/37, -12/37). \tag{2.40}$$

2.4.3 Bounding the Solutions

Unless a good bound for $|d|$, $|d_1|, |d_2|, \ldots, |d_{n-1}|$ is known because of extra information about the specific problem, a reasonable bound for the absolute value of the determinant is given in most cases by the *Hadamard bound*.

Definition 2.4.2. For a given matrix $A \in \mathbb{Q}^{n,n}$, the *Hadamard bound*, denoted by $H(A)$, for the absolute value of the determinant of A is given by

$$H(A) = \prod_{i=1}^{n} ||A_i||_2, \tag{2.41}$$

where A_i is the i-th row in A.

Example 2.4.2.
For the matrix

$$A = \begin{pmatrix} -1 & 2 \\ 3 & 5 \end{pmatrix}, \tag{2.42}$$

we have

$$\det(A) = -11, \quad H(A) = ||(-1,2)||_2 \cdot ||(3,5)||_2 = (1+4)^{1/2} \cdot (9+25)^{1/2} \simeq 11.66. \tag{2.43}$$

Remark 2.4.3. Since for any matrix A we have $|A| = |A^T|$, we can get a sharper bound by taking the minimum of the Hadamard bounds for both the matrix and its transpose.

For a rectangular matrix $A \in \mathbb{Q}^{m,n}$, with $m > n$ a bound is obtained by taking the maximum of the Hadamard bounds for all possible minors of size $n \times n$.

2.5 Rational Functions

In the following section we present our way of adapting the results presented in the previous section to the rational function case.

2.5.1 Definitions

Let us now consider the case when the field K is the field $\mathbb{Q}(y)$ of univariate rational functions over the rationals.

The notations $\mathrm{eval}_x(M)$ and $A(x)$, with $x \in \mathbb{Q}$ and such that it is not a root of any of the denominators of the entries of M, denote the matrix obtained from M by substituting y by x.

In the following we will always consider rational values x such that they do not divide any of the denominators of the entries of the considered matrix.

For any given matrix A we consider the evaluation homomorphisms with respect to the upper determinant of the triangular form, obtained by substituting the variable with rational values. We consider the rationals x such that, for $1 \leq i \leq n$, the following holds:

$$\begin{aligned} &\det(\overline{L(\tau_x)}) \neq 0, \\ &\det(\overline{L(\tau_x)}) = \mathrm{eval}_x(\det(\overline{L(\tau)})), \\ &\det(\overline{C_i(L(\tau_x), R(\tau_x))}) = \mathrm{eval}_x(\det(\overline{C_i(L(\tau), R(\tau))})), \end{aligned} \tag{2.44}$$

where we have used the following abbreviations

$$\begin{aligned} \tau &:= \tau(M), \\ \tau_x &:= \tau(M(x)). \end{aligned} \tag{2.45}$$

We call "lucky" those rationals x for which (2.44) holds.

2.5.2 Interpolation

Analogously to the case for $K = \mathbb{Q}$ we can solve the overdetermined system of linear equations of maximal rank (2.6) by interpolation.

We consider s distinct rationals $x_1, \ldots, x_s$, with s such that $D < s$, where D is a known bound such that

$$D > \max(\deg(d), \deg(d_1), \ldots, \deg(d_{n-1})), \tag{2.46}$$

where, as in the case $K = \mathbb{Q}$,

$$\begin{aligned} \tau &= \tau(A'|b), \\ d &= \det(\overline{L(\tau)}), \\ d_i &= \det(\overline{C(L(\tau), R(\tau))}). \end{aligned} \tag{2.47}$$

For each evaluation we compute the corresponding upper determinants.

In order to combine the solutions corresponding to different evaluations we use polynomial interpolation (e. g. Lagrange polynomial interpolation, Newton polynomial interpolation, etc.). Choosing lucky evaluations guarantees a correct result. For the definition of Lagrange and Newton polynomial interpolation we refer to Appendix A.

This approach can be recursively generalized to the multivariate case by simply considering the isomorphism $\mathbb{Q}(y_1, y_2, \ldots, y_s) = \mathbb{Q}(y_1, \ldots, y_{s-1})(y_s)$ and iterating the method for each indeterminate.

2.5.3 Bounding the Degrees of the Solutions

In order to have a good bound on the degree of d, $d_1, d_2, \ldots, d_{n-1}$ we can again use *Hadamard bound.*

In fact, for any square matrix $M \in \mathbb{Q}(y)^{n,n}$ we have that

$$|\det(M(x))| \leq \prod_{i=1}^{n} ||M_i(x)||_2 \leq \prod_{i=1}^{n} ||M_i(x)||_1; \quad x \in \mathbb{Q}, \tag{2.48}$$

where M_i is the i-th row in M.

Since this holds for any $x \in \mathbb{Q}$ (and such that it does not divide any of the denominators of the entries of the M) we must also have

$$\begin{aligned} \deg(\det(M(x))) \leq \deg(\prod_{i=1}^{n} ||M_i(x)||_1) = \sum_{i=1}^{n} \deg(||M_i(x)||_1) = \\ = \sum_{i=1}^{n} \deg(\sum_{j=1}^{n} |m_{i,j}|) = \sum_{i=1}^{n} \max_{j=1,\ldots,n} (\deg(m_{i,j})), \end{aligned} \tag{2.49}$$

where $m_{i,j}$ is the element of M in the i-th row and j-th column.

This is generalized to the upper determinant of a matrix by simply taking all possible determinants of submatrices of size $n \times n$.

Then for a rectangular matrix $A \in \mathbb{Q}^{m,n}$ a bound on the degree of the determinant of A is also given by:

$$\sum_{i=1}^{m} \max_{j=1,\ldots,n} (\deg(a_{i,j})).$$

Example 2.5.1.
Let us consider the matrix

$$A = \begin{pmatrix} n-1 & n^2 \\ n^3+2n & n+5 \end{pmatrix}. \tag{2.50}$$

Then a bound on the degree of the determinant of A is given by

$$\max(1,2) + \max(3,1) = 5, \tag{2.51}$$

which, in this case, coincides with the degree of $\det(A) = -n^5 - 2n^3 + n^2 + 4n - 5$.

2.5.4 A Combination of the two Approaches

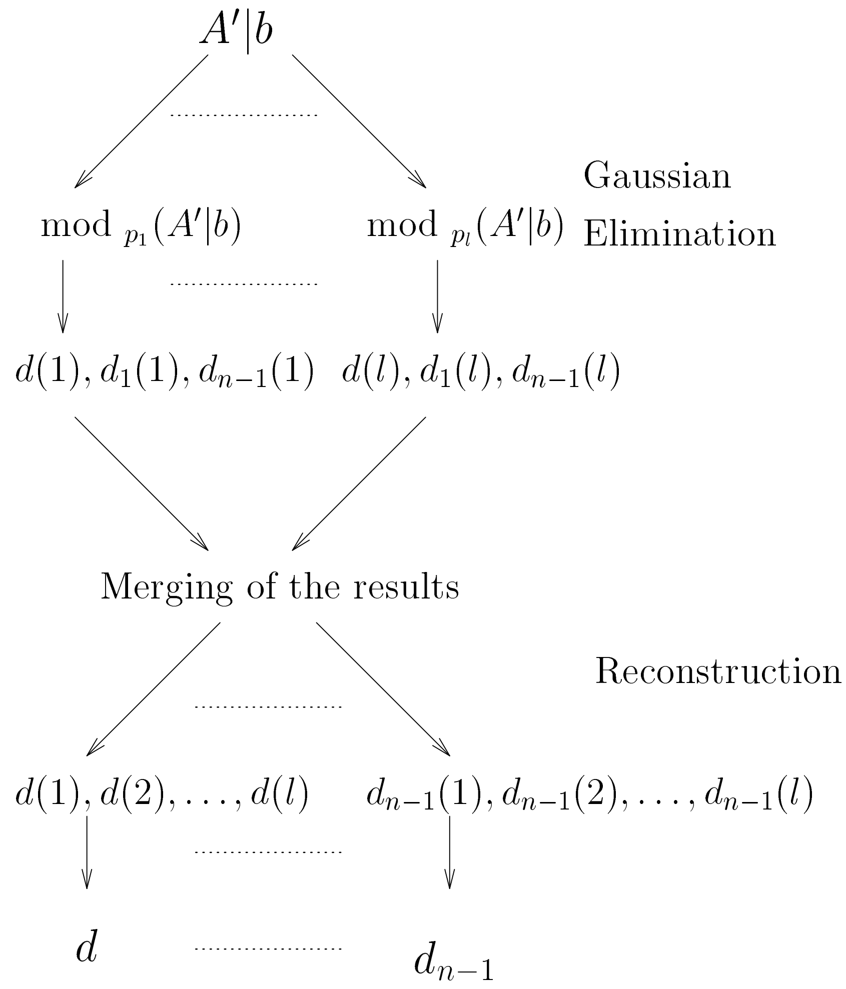
The modular approach and the interpolation approach can work together when working on matrices with entries in $\mathbb{Q}(y_1, \ldots, y_s)$. For each indeterminate and for the ground field $\mathbb{Q}$ we can recursively reconstruct d, d_1, d_2,..., d_{n-1} by interpolation over each indeterminate and by the Chinese remainder theorem over $\mathbb{Q}$.

2.6 Distributed Computation

Both the case $K = \mathbb{Q}$ and the case $K = \mathbb{Q}(y_1, \ldots, y_s)$ perform the following main operations:

1. Solving the homomorphic images of the original problem,
   - modular image in the rational case,
   - evaluation image in the rational function case.
2. Collecting the results,
3. Reconstructing the solutions.
   - Reconstruction by the Chinese remainder theorem in the rational case,
   - Recontruction by interpolation in the rational function case.

Solving the homomorphic images and reconstructing the solutions can be performed in parallel, namely each image can be solved independently and each component of the solutions can be reconstructed independently as in the following scheme:



2.7 The Package

My library implements the algorithms described above for the univariate case and the rational case has been implemented in the Mathematica language [Wol99]. A generalization to the multivariate case would be easy to implement but the poor performance of this approach discouraged me to proceed further into this direction. A better approach is presented in the next chapter. The library is available on the world wide web at the combinatorics home page of RISC (University of Linz, Austria) at the following web address

http://www.risc.uni-linz.ac.at/research/combinat/

2.7.1 Loading the Packages

All the routines are contained in the following Mathematica packages:

`matrixTools.m`	Matrix manipulations
`detBound.m`	Bounds for absolute values of determinants
`detDegreeBound.m`	Bounds for degrees of determinants
`CramerNullSpace.m`	Interpolation Routines

The main package `CramerNullSpace.m` will automatically load the remaining packages. It can be evaluated within the Mathematica environment as any other package by the command:

```
<< CramerNullSpace.m
```

2.7.2 The Commands

The package `CramerNullSpace.m` provides the following commands:

`ChNullSpaceAt[`*`coefs, evalPoint, detBound`*`]`
`ChNullSpaceUntil[`*`coefs, intPoints`*`]`
`DistChNullSpaceUntil[`*`coefs, intPoints`*`]`

Example 2.7.1 and Example 2.7.2 are related to the systems that are solved in Zeilberger's algorithm for definite hypergeometric summation (see Algorithm 1.2 in the first chapter).

Solving Linear Systems at One Point

Command

`ChNullSpaceAt[`*`coefs, evalPoint, detBound`*`]`

Parameters

- *coefs* is the matrix whose null space has to be found
- *evalPoint* is the point at which we intend to evaluate the matrix
- *detBound* is the bound (2.17)

Semantics

The command `ChNullSpaceAt[`*`coefs, evalPoint, detBound`*`]` solves the system of linear equations contained in *coefs* for $n = evalPoint$ by solving for different machine size primes and then by reconstructing the solution via the Chinese remainder theorem with the bound given by *detBound*. It outputs a pair containing the solution and an ordered list containing the indexes of the rows in the upper part of the triangularized matrix.

Example 2.7.1.
We can use `ChNullSpaceAt` to find the null space of

$$A = \begin{pmatrix} n+1 & 3 & 0 & 2n \\ 2n & 0 & 5n & -n^2+1 \\ n^2+n+1 & n^2 & -n^2-2n+3 & n^2+2n \end{pmatrix} \tag{2.52}$$

for $n = 1, 2, 3$, as follows

```
Table[ChNullSpaceAt[A, i, 100], i, 1, 3]
```

$\{\{\{-35, 0, 14, 35\}, \{1, 2, 3\}\},$
$\{\{-35, -165, 59, 150\}, \{1, 2, 3\}\},$
$\{\{423, -1086, -30, 261\}, \{1, 2, 3\}\}\}$

Solving a System by Interpolation

Commands

- `ChNullSpaceUntil[`*`coefs, intPoints`*`]`
- `DistChNullSpaceUntil[`*`coefs, intPoints`*`]`

Parameters

- *coefs* is the matrix whose null space has to be found
- *intPoints* is the number of interpolation points that have to be considered

Semantics

Both commands solve the system of linear equations contained in *coefs* by iteratively invoking `ChNullSpaceAt` for as many choices of n as *intPoints*, and then by interpolating the results via *Lagrange interpolation*.

The command `DistChNullSpaceUntil` does the same as `ChNullSpaceUntil` but uses the package *Distributed Mathematica* (see [PS00]) to solve with respect to as many interpolation points as possible in parallel over a network of computers.

Example 2.7.2.
We can use `ChNullSpaceUntil` or `DistChNullSpaceUntil` to find the null space of the following matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 2n+1 \\ -2n-2 & 0 & 2n+2 & -n^2+1 \\ n^2+2n+1 & n^2+2n+1 & -n^2-2n-1 & -n^2-2n-1 \end{pmatrix} \tag{2.53}$$

as follows

`ChNullSpaceUntil[A, 10] // Simplify`
$\{\{2(1+n)^3(1+2n), -(1+n)^4, 3(1+n)^4, -2(1+n)^3\}\}$

or in parallel with the command `DistChNullSpaceUntil`:

`<< dist.m`
Distributed Mathematica V1.0.0 (c) 2000 Cleopatra Pau (RISC-Linz)
See http://www.risc.uni-linz.ac.at/software/distmath

`InitializeD[pinwheel, octane, andromeda, octane, galaxy, solaris]`
Connecting pinwheel...
Connecting andromeda...
Connecting galaxy...
okay

`AllD["<< CramerNullSpace.m"]`
okay

`AllD["myCoefs =` $\{\{1, 0, 0, 1+2n\}, \{-2-2n, 0, 2+2n, 1-n^2\}, \{1+2n+n^2, 1+2n+n^2, -1-2n-n^2, -1-2n-n^2\}\}$`"]`
okay

`DistChNullSpaceUntil[myCoefs, 10] // Simplify`
$\{\{2(1+n)^3(1+2n), -(1+n)^4, 3(1+n)^4, -2(1+n)^3\}\}$

Remark 2.7.1. The command `InitializeD` initializes the computers available on the network. The command `AllD` "broadcast" a command to all the computers in the network.

2.8 Conclusion

The two variations of the approach based on Cramer's rule and on polynomial interpolation and on the Chinese remainder theorem perform very badly when compared with an approach based on rational interpolation, which is presented in the next chapter. Therefore I have not further developed nor extended the implementation of this method.

The problems encountered are:

- The method works under restrictive hypotheses (rank, matrix form, choice of primes)
- The method requires the detection of unlucky cases
- Too many homomorphic images are computed (see explanation in the next paragraph)

The excessive number of necessary homomorphic images is not caused by a bad bound on the degree of the polynomials but it is intrinsic in the approach. The determinant of the upper matrix and those of the Cramer matrices are the denominator and numerators of the components of the solution but they often have common factors that cannot be predicted in advance.

This experience made clear that an alternative had to be considered. A better approach is treated in the next chapter in which rational function interpolation and a hybrid method based on guessing and symbolic computation is used.

Chapter 3

Rational Interpolation

3.1 Introduction

We give a brief description of the theory behind the rational function reconstruction problem as described in [GvzG99], pages 106–115. Rational function reconstruction is the problem of finding a rational function that is congruent to some polynomial modulo some other polynomial. Rational function reconstruction has two important applications: rational interpolation and rational function approximation. We use rational function interpolation to build an algorithm that computes the null space of a rectangular matrix with entries in $\mathbb{Q}(y_1, \ldots, y_n)$ by reconstructing some low degree components of the solutions by rational function interpolation and the others symbolically. This is done by plugging the components reconstructed by rational function interpolation into the system in order to simplify it. In such a way the new system can become solvable by Gaussian elimination over $\mathbb{Q}(y_{i_1}, \ldots, y_{i_r})$ with $1 \leq i_j \leq n$ and $r < n$. Alternatively the system can be solved by iterating the procedure on the remaining variables. These algorithms have been implemented in the Mathematica computer algebra system.

3.2 Reconstructing Rational Functions

In the following let F be a field, $m \in F[x]$ be a polynomial of positive degree n (which will serve as modulus), and $f \in F[x]$ be a polynomial of degree less than n.

Definition 3.2.1. Given $a, b \in F[x]$, we define the j-th *row of the extended Euclidean algorithm for a and b* to be the ordered triplet (r, s, t), with $r, s, t \in F[x]$, where r is the remainder, and s, t are the corresponding cofactors computed at the j-th step of the Algorithm 3.1 (generalized extended Euclidean algorithm) for a and b.

Algorithm 3.1 Generalized Extended Euclidean Algorithm

Input: (f, g) with $f, g \in F[x]$.

Output: $l \in \mathbb{N}, r_j, s_j, t_j \in R$ for $0 \leq j \leq l$ as computed below.

1: $r_0 := \operatorname{Norm}(f)$; $s_0 := \operatorname{Lcoeff}_x(f)^{-1}$; $t_0 := 0$;
$r_1 := \operatorname{Norm}(g)$; $s_1 := 0$; $t_1 := \operatorname{Lcoeff}_x(g)^{-1}$;
2: $i := 1$;
3: **while** $r_i \neq 0$ **do**
4: $q_i := r_{i-1} \operatorname{quot} r_i$; `// Quotient of` r_{i-1} `divided by` r_i
$r_{i+1} := r_{i-1} \mod r_i$;
$s_{i+1} := (s_{i-1} - q_i s_i)/\operatorname{Lcoeff}_x(r_{i+1})$; `//` Lcoeff_x `= leading coefficient`
$t_{i+1} := (t_{i-1} - q_i t_i)/\operatorname{Lcoeff}_x(r_{i+1})$; `//` $\operatorname{Norm}(f) = f \operatorname{Lcoeff}_x(f)^{-1}$
$i := i + 1$;
$r_{i+1} := \operatorname{Norm}(r_{i+1})$
5: **return** i, r_j, s_j, t_j, for $0 \leq j \leq i$.

Given $k \in \{0, 1, \ldots, n\}$ we want to find a rational function $r/t \in F(x)$, with $r, t \in F[x]$ such that

$$\begin{aligned} &\gcd(t, m) = 1; \\ &rt^{-1} \equiv f \mod m, \quad \deg r < k, \quad \deg t \leq n - k, \end{aligned} \tag{3.1}$$

where t^{-1} is the inverse of t modulo m.

The following theorem describes the conditions under which (3.1) has solutions.

Theorem 3.2.1. *Let $r_j, s_j, t_j \in F[x]$ be the components of the j-th row in the extended Euclidean algorithm for m and f, where j is the minimal integer such that $\deg r_j < k$.*

1. *If $\gcd(r_j, t_j) = 1$ then r_j and t_j solve (3.1).*
2. *If r/t is a canonical form solution to (3.1) then $\gcd(r_j, t_j) = 1$ and $r = \tau^{-1} r_j$ and $t = \tau^{-1} t_j$, where τ is the leading coefficient of t_j.*

Corollary 3.2.2. *There is an algorithm which decides whether (3.1) has a solution, and if so, computes the solution in $\mathcal{O}(n^2)$ operations in F.*

For the proofs and more details we refer to [GvzG99], pages 106–115.

3.3 Cauchy Interpolation

In the following let $u_0, u_1, \ldots, u_{n-1}$ be distinct elements of F, and for $0 \leq i < n$, let $v_i = g(u_i)$ be a collection of samples of an unknown function $g : F \to F$.

Cauchy interpolation is the problem of rational interpolation where, given a collection of samples v_i, for $0 \le i < n$ of an unknown function g at distinct points $u_0, u_1, \ldots, u_{n-1}$ and given $k \in \{0, 1, \ldots, n\}$, we want to find a rational function $r/t \in F(x)$, with $r, t \in F[x]$, such that

$$\begin{aligned} &t(u_i) \neq 0; \\ &r(u_i)/t(u_i) = v_i; \quad 0 \le i < n, \quad \deg r < k, \ \deg t \le n - k. \end{aligned} \tag{3.2}$$

We will refer to the solutions of (3.2) as "rational interpolant" .

Lemma 3.3.1. *Let (r_i, s_i, t_i) be the i-th row in the extended Euclidean algorithm for $(a(x), b(x)) \in F[x]$. Then*

$$\gcd(r_i, t_i) = \gcd(a, t_i). \tag{3.3}$$

For the proof we refer to [GvzG99], pages 106–115.

The following corollary of Theorem 3.2.1 describes the solutions to the problem (3.2).

Corollary 3.3.2. *Let $f \in F[x]$ of degree less than n, with $f(u_i) = v_i$ for $0 \le i < n$, let $k \in \{0, 1, \ldots, n\}$ and $r_j, s_j, t_j \in F[x]$ be the components of the j-th row in the extended Euclidean algorithm for $((x - u_0) \cdot (x - u_1) \cdots (x - u_{n-1}), f)$, where j is minimal such that $\deg r_j < k$.*

1. *If $\gcd(r_j, t_j) = 1$ then r_j and t_j solve (3.2).*
2. *If $r/t \in F(x)$ is a canonical form solution to (3.2), then $\gcd(r_j, t_j) = 1$ and $r = \tau^{-1} r_j$ and $t = \tau^{-1} t_j$, where τ is the leading coefficient of t_j.*

Proof. Problem (3.2) is a special instance of problem (3.1) where

$$m = (x - u_0) \cdot (x - u_1) \cdot \cdots \cdot (x - u_{n-1}), \tag{3.4}$$

and where f is the interpolating polynomial of degree less than n.
□

Therefore in order to find a solution to (3.2) we apply Algorithm 3.1 (generalized extended Euclidean algorithm) to (m, f), with m as in (3.4). If the $\gcd(r_j, t_j) \neq 1$ then, by Lemma 3.3.1, also $\gcd(m, t_j) \neq 1$ and therefore $t(u_i) = 0$ for some $i \in \{0, 1, \ldots, n-1\}$ and (3.2) has no solution.

Example 3.3.1.
Let us take $F = \mathbb{Z}_5$ and let us find the rational functions $\rho = r/t \in \mathbb{Z}_5$ such that

$$\begin{aligned} \rho(0) &= 1, \\ \rho(1) &= 2, \\ \rho(2) &= 4. \end{aligned}$$

First of all we compute the interpolating polynomial f of degree less than 3:

$$f = 3x^2 + 3x + 1.$$

Then the extended Euclidean algorithm for (m, f) with $m = x(x-1)(x-2) = x^3 + 2x^2 + 2x$ and f yields the following sequence of remainders and cofactors

j	r_j	s_j	t_j
0	$x^3 + 2x^2 + 2x$	1	0
1	$x^2 + x + 2$	0	2
2	$x + 2$	4	$2x + 2$
3	1	$4x + 1$	$2x^2 + 1$
4	0	$x^2 + x + 2$	$3x^3 + x^2 + x$

From the sequence we can read off the following solutions corresponding to the rows i where the gcd of r_i and t_i is one

$$\begin{aligned} r_1/t_1 &= \frac{x^2 + x + 2}{2}, \\ r_2/t_2 &= \frac{x + 2}{2x + 2}, \\ r_3/t_3 &= \frac{1}{2x^2 + 1}. \end{aligned} \tag{3.5}$$

Corollary 3.3.3. *There is an algorithm the decides whether the problem (3.2) has a solution, and if so, it computes it in $\mathcal{O}(n^2)$ arithmetic operations in F.*

3.4 Padé Approximation

Padé approximation (see [GvzG99], pages 112–115) is the problem of approximating a power series $\sum_{i \geq 0} f_i x^i \in F[[x]]$ with all $f_i \in F$ to a rational function $r/t \in F[x]$ with $r, t \in F[x]$ and $x \nmid t$.

Formally, given $k \in \{0, 1, \ldots, n\}$ and $f \in F[x]$ of degree less than n we want to find a rational function $r/t \in F(x)$ with $r, t \in F[x]$ such that

$$x \nmid t; \;\; r/t \equiv f \mod x^n, \;\; \deg r < k, \;\; \deg t \leq n - k. \tag{3.6}$$

Corollary 3.4.1. *Let $f \in F[x]$ of degree less than n, $k \in \{0, 1, \ldots, n\}$, and $r_j, s_j, t_j \in F[x]$ be the components of the j-th row of the extended Euclidean algorithm for (x^n, f), where j is minimal such that $\deg r_j < k$.*

1. *If $\gcd(r_j, t_j) = 1$ then r_j and t_j are solutions to (3.6).*

2. *If r/t is a canonical form solution to (3.6), then $\gcd(r_j, t_j) = 1$ and $r = \tau^{-1} r_j$, $t = \tau^{-1} t_j$, where τ is the leading coefficient of t_j.*

Proof. Problem (3.6) is an instance of (3.1) for $m = x^n$.
□

We will refer to the solutions of (3.6) as "Padé approximants".

Thus the Padé approximants are obtained by computing r_j and t_j through Algorithm 3.1 (generalized extended Euclidean algorithm) and by checking whether their gcd is one, and if so, no such approximant exists, otherwise r_j/t_j is a solution to (3.6).

Corollary 3.4.2. *There is an algorithm that decides whether (3.6) has a solution, and if so, it computes it in $\mathcal{O}(n^2)$ arithmetic operations in F.*

3.5 Homogeneous Linear Systems

We can apply rational function interpolation to reconstruct the components of the solutions of a homogeneous system of linear equations with polynomial or rational function coefficients.

Given $A \in \mathbb{Q}(y_1, \ldots, y_p)^{m,n}$, we want to find $x = (x_1, \ldots, x_n)$, with $x_i \in \mathbb{Q}(y_1, \ldots, y_p)$, $1 \le i \le n$ such that

$$Ax = 0. \tag{3.7}$$

Definition 3.5.1. We define $\nu(A)$ to be the ordered sequence of n-tuples containing the null space of A obtained by triangularization and back-substitution.

Given $A \in \mathbb{Q}(y_1, \ldots, y_p)^{m,n}$ and a p-tuple $v = (v_1, v_2, \ldots, v_p)$ of rational numbers, the notation $\mathrm{eval}_v(A)$ denotes the matrix obtained from A by substituting in all entries of A v_i to y_i, for $1 \le i \le p$, whenever no denominator of any of the entries of A becomes zero. Besides, given an ordered sequence ν of n-tuples of rational functions in $\mathbb{Q}(y_1, \ldots, y_p)$, the notation $\mathrm{eval}_v(\nu)$ denotes the ordered sequence obtained from ν by substituting in all n-tuples v_i to y_i, for $1 \le i \le p$, whenever no denominator of any of the components of the n-tuples becomes zero.

Definition 3.5.2. We consider the vectors $v \in \mathbb{Q}(y_1, \ldots, y_p)$ for which $\mathrm{eval}_v(A)$ and $\mathrm{eval}_v(\nu(A))$ are defined and such that

$$\nu(\mathrm{eval}_v(A)) = \mathrm{eval}_v(\nu(A)). \tag{3.8}$$

We call the vectors v *lucky evaluations* for A.

In the next sections we describe various strategies and tools for experimentations for the computation of the null space by rational function interpolation that I have implemented in the Mathematica package `RatNullSpace` (for the details on the package see Section 3.6).

3.5.1 The First Try

We can reconstruct $\nu(A)$ by taking the solutions $\nu(\mathrm{eval}_v(A))$ of sufficiently many lucky evaluations and by rational function interpolation.

If we always choose lucky evaluations, this method is an algorithm that correctly reconstructs the solutions. What makes this brute force approach useless in many cases is the fact that the bound on the number of necessary evaluations is often too high.

3.5.2 A Heuristic Method

A slightly different method would be using rational function interpolation in an iterative way. We remove those possible solutions that change once a new evaluation is considered and keep the others. Once we have a unique candidate, we take that as the solution.

I could not find examples for which this heuristic failed to reconstruct the correct solution. Moreover this method does not require any bound and it is much faster than the previous method. I implemented this method in the package `RatNullSpace.m` which is described in this chapter (see `ModularNullSpace`).

3.5.3 A Recursive Method

We describe a recursive method that improves the previous heuristic method. Such improved heuristic reconstructs first those components that require few evaluations to stabilize, then plugs their values into the system in order to simplify it and recursively reconstructs the remaining components in the simplified system. I implemented this method in the package `RatNullSpace.m` which is described in this chapter (see `CompleteNullSpace`).

3.5.4 A Hybrid Method

An ever faster method is to use the heuristic method only for those components that require few evaluations to stabilize. Once these components have been computed their values can be plugged in into the original system in order to simplify it. Then the simplified system can be solved by a symbolic algorithm. This method generalizes the work in [RZ]. I implemented this method in the package `RatNullSpace.m`, which is described in this chapter (see `RatNullSpace`).

This method can be sped up by putting into the system extra information on the degree of the components of the solutions.

Black Lists

If we know in advance which components require fewer evaluations we can avoid trying to reconstruct the others. This is implemented by putting these components into a *black list*. These components will be computed symbolically.

Delayed Interpolation

If we have a lower bound for the number of evaluations necessary to reconstruct the components we can avoid interpolation until the necessary number of evaluations has been reached. Having this information can be particularly useful when using non-incremental interpolation algorithms.

3.6 The Package

A package that implements null space computation through rational function reconstruction has been implemented in the Mathematica computer algebra system [Wol99]. The package is available on the world wide web at the combinatorics home page of RISC (University of Linz, Austria) at the following web address

http://www.risc.uni-linz.ac.at/research/combinat/

The package can be loaded as any other Mathematica package by the command:

```
<< RatNullSpace.m
```

The package provides functions for computing the null space by the *heuristic method*, by the *recursive method* and by the *hybrid method* as described in the previous section. The package provides also some lower level routines that implement rational function reconstruction and rational function interpolation.

3.6.1 The Commands

Null space computation is implemented in the functions

- `RatNullSpace`
- `ModularNullSpace`
- `MultiModularNullSpace`
- `HybridNullSpace`
- `MultiHybridNullSpace`
- `CompleteNullSpace`

All the commands except `CompleteNullSpace` take as input a matrix and output the null space. Both input and output follow the Mathematica convention used in the internal `NullSpace` command.

A component is given the conventional value `"?"` meaning *non-computed* when either it cannot be computed because of too few interpolating points or the user desires not to compute it.

The command `InterpolatingRatFunction` provides univariate rational function interpolation.

Example 3.6.1, 3.6.2, 3.6.3 and 3.6.4 are related to the systems that are solved in Zeilberger's algorithm for definite hypergeometric summation (see Algorithm 1.2 in the first chapter).

3.6.2 Modular Null Space Computation

Commands

- `ModularNullSpace[`*coefs*`]`
- `MultiModularNullSpace[`*coefs*`]`

Parameter

coefs contains the matrix whose null space has to be computed.

Options

Name	Default Value
`StartingInterpolation`	3
`Offset`	3
`DegreeBound`	6
`Var`	0 (not defined)
`Verbose`	Off
`Algorithm`	NullSpace
`BlackList`	{ }

Semantics

The commands `ModularNullSpace` and `MultiModularNullSpace` compute all the components of the vectors in the null space except those whose positions are in `BlackList`.

`ModularNullSpace` uses rational function interpolation on the variable passed as a single element through the option `Var` or, when no option is given (`Var` is 0), on the first variable in the lexicographic order contained in *coefs*.

The command `MultiModularSpace` uses rational function interpolation on each variable passed as a list through the option `Var` or, when no option is given (`Var` is 0), on all variables.

Meaning of the options

- `StartingInterpolation` is the number of minimal interpolation points at which the interpolation algorithm is invoked.

- `Offset` is the initial integer interpolating point.
- `DegreeBound` is the maximum number of interpolating point to be considered after which, in case of unsuccessful result, the value `"?"` is assigned to the component.
- `Var` is the variable or variables (passed as a list) with respect to which the interpolation should be performed. When it has value 0 it means "the first variable found in *coefs*" if used in `ModularNullSpace`, or "all the variables found in *coefs*" when used in `MultiModularNullSpace`.
- `Verbose` toggles the verbosity and can be set either to `On` or `Off`.
- `Algorithm` is the algorithm which has to be used to solve each homomorphic image.
- `BlackList` is a the set (passed as a list of positive integers) containing the positions of those components that are not to be computed. Those components are given the conventional value `"?"`.

Example 3.6.1. A Univariate Example
Let us compute the null space of the following matrix with univariate polynomial entries:

$$A = \begin{pmatrix} 1 & n^3+2 & n-2 & 4n+1 \\ -2n-2 & 0 & 2n+2 & -n^2+1 \\ n^2+2n+1 & n^2+2n+1 & -n^2-2n-1 & -n^2-2n-1 \end{pmatrix} \tag{3.9}$$

This can be done `ModularNullSpace`. Let us try to reconstruct the components by using at most 5 interpolating points:

```
ModularNullSpace[A, DegreeBound -> 5]
```

Output =

$$\{\{?, \frac{1+n}{2}, ?, 1\}\} \tag{3.10}$$

Two components are left without a value. Trying with an extra point yields the complete solution:

```
ModularNullSpace[A, DegreeBound -> 6]
```

Output =

$$\{\{-\frac{6+7n+n^2+n^3+n^4}{2(-1+n)}, \frac{1+n}{2}, -\frac{5+9n+n^3+n^4}{2(-1+n)}, 1\}\} \tag{3.11}$$

Example 3.6.2. A Multivariate Example
Let us now consider the following matrix with bivariate polynomial entries:

$$A = \begin{pmatrix} 1 & m+n & 0 & m+1 \\ -2 & 0 & 2n+2 & 0 \\ m & 0 & 0 & n \end{pmatrix} \tag{3.12}$$

We can find the null space of A by either the command `ModularNullSpace` or by the command `MultiModularNullSpace`.

```
ModularNullSpace[A, DegreeBound -> 6, Var->m]
```

or

```
MultiModularNullSpace[A, DegreeBound -> 6]
```

Output =

$$\{\{-\frac{n}{m}, \frac{-m-m^2+n}{m^2+mn}, -\frac{n}{m(1+n)}, 1\}\} \tag{3.13}$$

3.6.3 Completing the Null Space

Command
`CompleteNullSpace[`*coefs*`, `*parSols*`]`

Parameters

- *coefs* is the matrix whose null space has to be found
- *parSols* is a partially computed null space, i. e. it can contain components whose value is `"?"`. It can be the output of `ModularNullSpace` or of the command `MultiModularNullSpace`.

Options

Name	Default Value
`Verbose`	Off
`Algorithm`	NullSpace

Semantics
`CompleteNullSpace` takes *parSols* and uses this information to simplify the system by plugging in the values of the components in *parSols* that do not have value `"?"`. The new system will be devoid of these unknowns. Then it invokes the null space algorithm passed through the option `Algorithm` on the simplified system and returns a list of lists containing those components that had value `"?"` in *coefs* and the last component.

Meaning of the options

- `Verbose` toggles the verbosity level and can be either `On` or `Off`.
- `Algorithm` is the null space algorithm which is used to find the components whose value in *parSol* is `"?"`, i. e. those components that have not been computed.

Example 3.6.3.
Let us consider the matrix:

$$A = \begin{pmatrix} n-1 & 0 & 0 & 2n+1 \\ -2n & 0 & 2n & -n^2+1 \\ n+1 & n^2+3n+1 & -n^2-1 & -n^2-1 \end{pmatrix} \tag{3.14}$$

We can guess some components of the solutions by:

```
ms = ModularNullSpace[A, DegreeBound -> 4]
```

Output =

$$\{\{\frac{-1-2n}{-1+n}, ?, ?, 1\}\} \tag{3.15}$$

Then we can use this result to find the remaining components by the command `CompleteNullSpace`:

```
CompleteNullSpace[A,ms]
```

Output =

$$\{\{\frac{-1+2n-2n^2-2n^3+n^4}{2n(1+3n+n^2)}, \frac{1-3n-5n^2+n^3}{2(-1+n)n}, 1\}\} \tag{3.16}$$

3.6.4 Hybrid Modular Symbolic Null Space

Commands

- `HybridNullSpace[`*coefs*`]`
- `MultiHybridNullSpace[`*coefs*`]`
- `RatNullSpace[`*coefs*`]`

Parameter
coefs contains the matrix whose null space has to be computed.

Options

Name	Default Value
`Verbose`	Off
`ModularAlgorithm`	NullSpace
`SymbolicAlgorithm`	NullSpace
`StartingInterpolation`	3
`Offset`	3
`DegreeBound`	6
`Var`	0 (not defined)
`BlackList`	{ }

Semantics

`HybridNullSpace` invokes `ModularNullSpace` and then `CompleteNullSpace` to find the null space.

`MultiHybridNullSpace` and `RatNullSpace` are two different names for the same command. They invoke `MultiModularNullSpace` and `CompleteNullSpace` to find the null space.

Meaning of the options

- `StartingInterpolation` is passed to the modular null space algorithm.
- `Offset` is passed to the modular null space algorithm.
- `DegreeBound` is passed to the modular null space algorithm.
- `Var` is passed to the modular null space algorithm.
- `BlackList` is passed to the modular null space algorithm.
- `Verbose` toggles the verbosity level, it can be either `On` or `Off` and it is passed to `ModularNullSpace` and to `CompleteNullSpace`.
- `ModularAlgorithm` is passed through the option `Algorithm` to the modular null space algorithm.
- `SymbolicAlgorithm` is passed through the option `Algorithm` to the command `CompleteNullSpace`.

Example 3.6.4.

Let us consider the matrix A with polynomial entries:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 2 \\ -6n-9 & 0 & 0 & 0 & 2 & -3n-3 \\ 15n^2+4n+3 & 0 & 0 & 2 & -3n-4 & 3n-3 \\ -n^3-n^2-n-1 & -3n-1 & 2 & 3n+1 & n & n \\ 0 & 7n^2+n+6 & n+1 & 6 & -n^3+1 & 0 \end{pmatrix} \tag{3.17}$$

We can find its null space by any of the null space algorithms that use the hybrid method:

```
RatNullSpace[A]
```

Output=

$$\begin{aligned}&\{\{-2, \frac{530+639n-801n^2-143n^3+71n^4}{4(13+6n+17n^2)},\\ &\frac{486+2614+2727n^2+429n^3+1115n^4-803n^5}{4(13+6n+17n^2)}, \frac{1}{4}(-42-71n+33n^2),\\ &-\frac{3}{2}(5+3n), 1\}\}\end{aligned} \tag{3.18}$$

Example 3.6.5. Superiority over Mathematica
We can use `HybridNullSpace` to find the null space of matrices faster than by the Mathematica built-in null space algorithm. For instance we can consider the system that has to be solved by Zeilberger's algorithm applied to the binomial coefficient to the fifth power, i. e. the problem of finding a special linear recurrence (see Theorem 1.2.3) with polynomial coefficients in n and free of k (see recurrence (1.1)) for

$$\sum_{k=0}^{n} \binom{n}{k}^5. \tag{3.19}$$

One possible way to obtain the matrix is to run `parGosperLinSys` in the Maxima implementation of Zeilberger's algorithm, which performs Zeilberger's algorithm with verbose linear system solving. For more details on how Zeilberger's algorithm works we refer to the first chapter of this thesis and to [Car99], [PS95], [Zei90], [Zei91].

Once we have the matrix, say, in `b5coefs` we can find its null space in $37.69s$ as follows:

```
HybridNullSpace[b5coefs,
                BlackList -> {2, 3, 4, 5, 6, 7, 8, 9, 10}]
```

Output =

$$\begin{aligned}&\{\{2, (-514048-2341112n-4509834n^2-4764843n^3-\\ &2982616n^4-1106342n^5-225214n^6-19415n^7)/\\ &(16(292+253n+55n^2)), \ldots\}\end{aligned} \tag{3.20}$$

The Mathematica command `NullSpace` can solve the problem in 163.44 seconds. The tests were run on an SGI Octane (with two 200 Megahertz processors) with 2 Gigabytes RAM. We used Mathematica 4.0.

The command was used as follows

```
NullSpace[b5coefs] // Together
```

The same command not followed by the command `Together`, which takes care of adding up rational functions, produces a crash of Mathematica 4.0.

3.6.5 Rational Function Interpolation

Rational function interpolation is implemented in the package in the command `InterpolatingRatFunction`.

Command

`InterpolatingRatFunction[`*`interpList, numBound, var`*`]`.

Parameters

- *interpList* is a list of interpolation points passed as a list of couples (p, v), where v is the value assumed at p.
- *numBound* is a bound on the degree of the numerator.
- *var* is the indeterminate of the searched rational functions.

Semantics

`InterpolatingRatFunction` implements rational function interpolation by first invoking a polynomial interpolation algorithm and then by invoking Algorithm 3.1 (generalized extended Euclidean algorithm) on the polynomial interpolant and on the polynomial $\prod_i (var - p_i)$, where p_i are the interpolation points.

Example 3.6.6.
Let us interpolate the rational function f:

$$f(x) = \frac{x-1}{x^2+3}, \tag{3.21}$$

by only knowing f at the values $1, 2, 3, 4$:

$$f(1) = \frac{1}{7},\ f(2) = \frac{2}{6},\ f(3) = \frac{3}{19},\ f(4) = \frac{1}{7}. \tag{3.22}$$

This can be done by `InterpolatingRatFunction` in the following way:

```
InterpolatingRatFunction[{{1,0},{2,1/7},{3,1/6},{4,3/19}},3,x]
```

Output =

$$\{\frac{-556+766x-233x^2+23x^3}{1596}, \frac{-9+10x-x^2}{3+23x}, \frac{-1+x}{3+x^2}\} \tag{3.23}$$

The result contains f but also other rational functions that interpolate the points in (3.22) but have numerator and denominator with different degrees.

If we knew the degree of the numerator of f we could choose the correct answer. Otherwise we can try to guess it with very high probability by taking into account extra points and taking those solutions that are both in the old and new set of interpolants.

Let us take for instance

$$f(5) = \frac{1}{7}. \tag{3.24}$$

Then we get

```
InterpolatingRatFunction[
{{1,0},{2,1/7},{3,1/6},{4,3/19},{5,1/7}},3,x]
```

Output =

$$\{\frac{-68 + 82x - 15x^2 + x^3}{28(3 + 4x)}, \frac{-1 + x}{3 + x^2}\} \tag{3.25}$$

We see that $f = (-1 + x)/(3 + x^2)$ is the only rational function contained in both solution sets ((3.23),(3.25)).

3.7 Conclusion

As seen in Example 3.6.5 rational function interpolation for guessing low degree components of solutions of homogeneous systems of linear equations performs much better than the method based on Cramer's rule and polynomial interpolation (described in the previous chapter).

The method based on Cramer's rule and polynomial interpolation reconstructs the numerators and denominators of the components of the solutions as given by Cramer's rule, whereas the method based on rational function interpolation reconstructs the simplified numerators and denominators.

The bottle-neck of this approach, as it is implemented now, is the interpolation step in which the generalized extended Euclidean algorithm (Algorithm 3.1) is invoked.

A future version of the package could use a more efficient interpolation algorithm.

Chapter 4

Linear Systems in Zeilberger's Algorithm

4.1 Introduction

Given a proper hypergeometric term $F(n,k)$ (see Definition 4.3.1) and a linear shift operator $\mathcal{L}$ of order d with polynomial coefficients, we prove that the *Gosper form* $(p(n,k), q(n,k), r(n,k))$ (see Definition 4.3.6 and Algorithm 4.1) of the shift quotient in k of $\mathcal{L}F(n,k)$ is such that the degree in n of $q(n,k)$ and $r(n,k)$ can be bounded independently of d. This gives some insight on the degree in n of the polynomial coefficients of the linear recurrence satisfied by $\sum_{k=0}^{n} F(n,k)$ that is computed by Zeilberger's algorithm. We do this in two steps: firstly by proving the existence of certain bijections between subsets of the natural numbers which reflect the properties of the *Gosper form*; secondly by interpreting this abstract model in terms of the polynomials involved in the *Gosper form*.

4.2 The Abstract Model

Given any positive integers n, s with $s \neq 0$, $n \geq 2s$, and a subset H of $\{1,\ldots,n\}$, we construct a bijection ψ between a subset of $\{s+1,\ldots,n\} \setminus (H+s)$ and a subset $\{s+1,\ldots,n\} \setminus H$ with the property that for any i in its domain, $\psi(i) - i$ is a non-negative multiple of s. Besides we show that ψ is the bijection between a subset of $\{s+1,\ldots,n\} \setminus (H+s)$ and a subset $\{s+1,\ldots,n\} \setminus H$ with the largest possible domain and such that $\psi(i) - i$ is a non-negative multiple of s.

4.2.1 Notation

Throughout this section n, s are fixed positive integers with $s \neq 0$, $n \geq 2s$, and H is a given subset of $\{1,\ldots,n\}$. We use the following notation:

$$\begin{aligned}
A(H) &= H \cap \{1, \ldots, s\};\\
B(H) &= H \cap \{s+1, \ldots, n-s\};\\
C(H) &= H \cap \{1, \ldots, n-s\};\\
D(H) &= H \cap \{n-s+1, \ldots, n\}.
\end{aligned} \tag{4.1}$$

Pictorially:

$$\overbrace{\underbrace{1, 2, \ldots, s}_{A(H)=H\cap},\ \underbrace{s+1, s+2, \ldots, n-s}_{B(H)=H\cap}}^{C(H)=A(H)\cup B(H)},\ \underbrace{n-s+1, n-s+2, \ldots, n}_{D(H)=H\cap}$$

Given a function $f : X \to Y$, we denote by $\mathrm{dom}(f)$ the *domain* of f, i. e. X and by $\mathrm{im}(f)$ the *image*, i. e. the set $\{f(x)|x \in X\}$.

Given a subset $S \subseteq \mathbb{N}$ and $q \in \mathbb{N}$ we denote by $S + q$ the set $\{p + q|p \in S\}$.

4.2.2 The Fundamental Bijection

In the next two theorems we treat the case $H \subseteq \{s + 1, \ldots, n - s\}$.

Definition 4.2.1. Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{s + 1, \ldots, n - s\}$. We define the function ϕ_H

$$\phi_H : \{s + 1, \ldots, n\} \setminus (H + s) \to \{s + 1, \ldots, n\} \setminus H, \tag{4.2}$$

as follows:

If H is the empty set, we take the identity function

$$\phi_\emptyset(i) = i, \tag{4.3}$$

for all $i \in \mathrm{dom}\, \phi_H$.

If H is not the empty set, we take for all $i \in \mathrm{dom}\, \phi_H$

$$\phi_H(i) = \begin{cases} \phi_{H\setminus\{a\}}(i), & \text{if } i \in (\{s+1, \ldots, n\} \setminus (H+s)) \setminus \phi^{-1}_{H\setminus\{a\}}(a), \\ \phi_{H\setminus\{a\}}(a+s), & \text{if } i = \phi^{-1}_{H\setminus\{a\}}(a), \end{cases} \tag{4.4}$$

where a denotes the minimum element in H with respect to the natural ordering.

Example 4.2.1 describes the function ϕ_H for $n = 9$, $s = 3$ and $H = \{5, 6\}$.

Theorem 4.2.1. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{s+1, \ldots, n-s\}$. The function ϕ_H (see Definition 4.2.1) is well-defined, is a bijection and is such that $\phi_H(i) - i$ is a non-negative multiple of s, i. e. there exists a non-negative integer m_i such that*

$$\phi_H(i) - i = m_i \cdot s. \tag{4.5}$$

Proof. Let us prove this theorem by induction on the cardinality of H.

When H is the empty set, we have the identity function

$$\phi_\emptyset(i) = i,$$

which is a bijection and it satisfies the condition (4.5) because:

$$\phi_\emptyset(i) - i = 0 = 0 \cdot s.$$

Let us assume that the theorem holds for all subsets of $\{s+1, \ldots n-s\}$ of a certain given cardinality j. We prove it for an arbitrary subset H of $\{s+1, \ldots n-s\}$ of cardinality $j+1$.

We consider the function ϕ_H

$$\phi_H : \{s+1, \ldots, n\} \setminus (H+s) \rightarrow \{s+1, \ldots, n\} \setminus H,$$

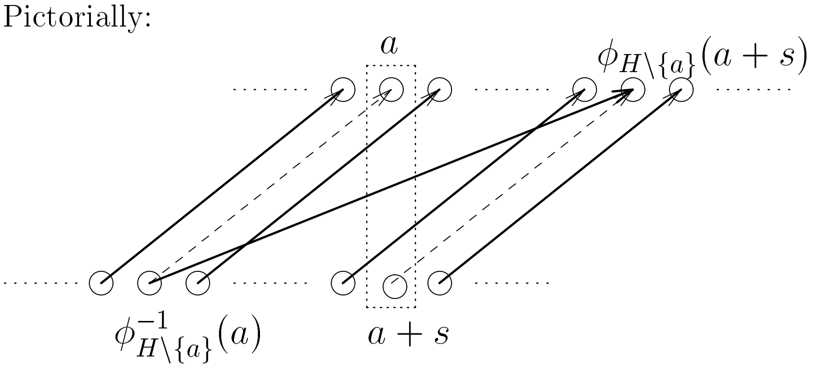
which, by Definition 4.2.1, is

$$\phi_H(i) = \begin{cases} \phi_{H\setminus\{a\}}(i), & \text{if } i \in (\{s+1, \ldots, n\} \setminus (H+s)) \setminus \phi^{-1}_{H\setminus\{a\}}(a), \\ \phi_{H\setminus\{a\}}(a+s), & \text{if } i = \phi^{-1}_{H\setminus\{a\}}(a), \end{cases} \tag{4.6}$$

where a is the minimum element in H with respect to the natural ordering.

We notice that a must be in the image of $\phi_{H\setminus\{a\}}$ and therefore $\phi^{-1}_{H\setminus\{a\}}(a)$ is an element of the domain of $\phi_{H\setminus\{a\}}$.

Pictorially:



The condition (4.5) holds by the induction hypothesis for $i \neq \phi^{-1}_{H\setminus\{a\}}(a)$.

When $i = \phi^{-1}_{H\setminus\{a\}}(a)$ we have that by definition of inverse and by the induction hypothesis

$$\underbrace{\phi_{H\setminus\{a\}}(\phi^{-1}_{H\setminus\{a\}}(a))}_{=a} - \phi^{-1}_{H\setminus\{a\}}(a) = l \cdot s, \tag{4.7}$$

where l is a non-negative integer.

Hence, by (4.7) and by the fact that $\phi_{H\setminus\{a\}}(a+s) - (a+s) = m \cdot s$, with m a non-negative integer,

$$\begin{aligned} \phi_H(\phi^{-1}_{H\setminus\{a\}}(a)) - \phi^{-1}_{H\setminus\{a\}}(a) &= \phi_{H\setminus\{a\}}(a+s) - (a - l \cdot s) = \\ &= a + s + m \cdot s - a + l \cdot s = (m + l + 1) \cdot s. \end{aligned} \tag{4.8}$$

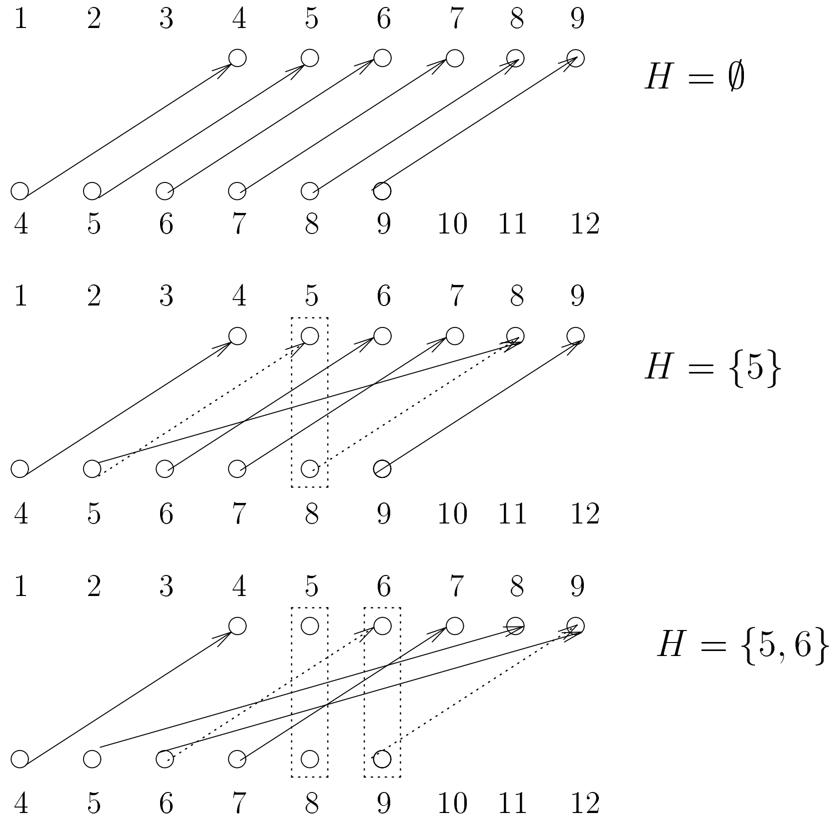
The function ϕ_H is surjective, because for $i \notin \{\phi^{-1}_{H\setminus\{a\}}(a), a+s\}$, ϕ_H spans the image of $\phi_{H\setminus\{a\}}$ expect for the values a and $\phi_{H\setminus\{a\}}(a+s)$ and for $i = \phi_{H\setminus\{a\}}(a)$ it covers the value $\phi_{H\setminus\{a\}}(a+s)$.

Injectivity follows from the inductive hypothesis and from the fact that $a+s \notin \mathrm{dom}(\phi_H)$ and therefore for any $i \in \mathrm{dom}(\phi_H)$, $i \neq \phi^{-1}_H(a)$,

$$\phi_H(i) = \phi_{H\setminus\{a\}}(i) \neq \phi_{H\setminus\{a\}}(a+s) = \phi_H(\phi^{-1}_{H\setminus\{a\}}(a)). \tag{4.9}$$

□

Example 4.2.1.
$n = 9$, $s = 3$, $H = \{5, 6\}$:



Theorem 4.2.2. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{s+1, \ldots, n-s\}$. The function ϕ_H has the property that for any $i \in \mathrm{dom}(\phi_H)$, $\phi_H(i) - i$ is the minimum non-negative multiple of s, i. e. we have*

$$\frac{\phi_H(i) - i}{s} = \min_{l \in \mathbb{N}}(l \mid i + l \cdot s \in \{s+1, \ldots, n\} \setminus H). \tag{4.10}$$

Proof. We proceed by induction on $|H|$.

When H is the empty-set, we have $\phi_H(i) = i$ for any $i \in \mathrm{dom}(\phi_H)$ and therefore the minimality condition (4.10) holds.

Given an arbitrary subset H of $\{s+1, \ldots, n-s\}$, let us assume that the property holds for a subsets of $\{s+1, \ldots, n-s\}$ of lower cardinality than H, and let us prove it for H.

Let us denote by a the minimum element in H with respect to the natural ordering. The property holds for any $i \neq \phi^{-1}_{H\setminus\{a\}}(a)$ because in this case $\phi_H(i) = \phi_{H\setminus\{a\}}(i)$.

We have that $\phi_H(\phi^{-1}_{H\setminus\{a\}}(a)) = \phi_{H\setminus\{a\}}(a+s)$.

Moreover, we have

$$a = \phi^{-1}_{H\setminus\{a\}}(a) + l_1 \cdot s, \tag{4.11}$$

with

$$l_1 = \min_{l \in \mathbb{N}}(l \mid \phi^{-1}_{H\setminus\{a\}}(a) + l \cdot s \in \mathrm{im}(\phi_{H\setminus\{a\}})); \tag{4.12}$$

and

$$\phi_{H\setminus\{a\}}(a+s) = a + s + l_2 \cdot s, \tag{4.13}$$

with

$$l_2 = \min_{l \in \mathbb{N}}(l \mid a + s + l \cdot s \in \mathrm{im}(\phi_{H\setminus\{a\}})); \tag{4.14}$$

and therefore

$$l_2 + 1 = \min_{l \in \mathbb{N}}(l \mid a + l \cdot s \in \mathrm{im}(\phi_{H\setminus\{a\}}) \setminus \{a\}). \tag{4.15}$$

Hence

$$\underbrace{\phi_H(\phi^{-1}_{H\setminus\{a\}}(a))}_{=\phi_{H\setminus\{a\}}(a+s)} = \phi^{-1}_{H\setminus\{a\}}(a) + (l_1 + l_2 + 1) \cdot s \tag{4.16}$$

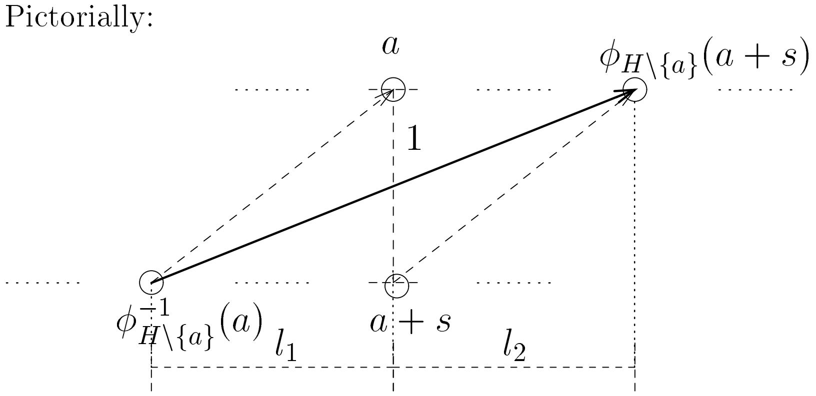
where

$$l_1 + l_2 + 1 = \min_{l \in \mathbb{N}}(l \mid \phi^{-1}_{H\setminus\{a\}}(a) + l \cdot s \in \mathrm{im}(\phi_{H\setminus\{a\}}) \setminus \{a\}); \tag{4.17}$$

because of the minimality of l_1 and l_2.

□

Pictorially:



4.2.3 More General Bijections

The following lemma and theorem generalize the previous theorems respectively to the case $H \subseteq \{1, \ldots, n-s\}$ and to the case $H \subseteq \{1, \ldots, n\}$.

Definition 4.2.2. Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n-s\}$, we define the function μ_H to be the following restriction of ϕ_H:

$$\mu_H = \phi_{B(H)}\big|_{\mathrm{dom}\,(\phi_{B(H)})\setminus(A(H)+s)} . \tag{4.18}$$

Lemma 4.2.3. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n-s\}$. The function μ_H is such that for any $i \in \{s+1, \ldots, n\} \setminus (H+s)$, $\mu_H(i) - i$ is a non-negative multiple of s, i. e. there exists a non-negative integer m_i such that*

$$\mu_H(i) - i = m_i \cdot s. \tag{4.19}$$

Moreover

$$\mu_H : \{s+1, \ldots, n\} \setminus (H+s) \to R, \tag{4.20}$$

where

$$R \subseteq \{s+1, \ldots, n\} \setminus H, \tag{4.21}$$

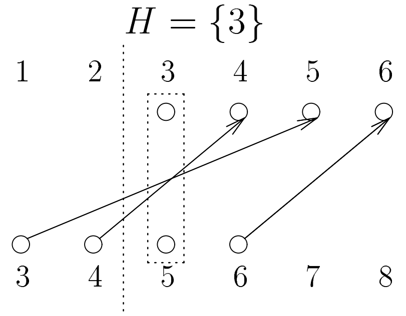
Proof. The function μ_H is a bijection that satisfies the condition (4.19) because it is a restriction of $\phi_{B(H)}$.

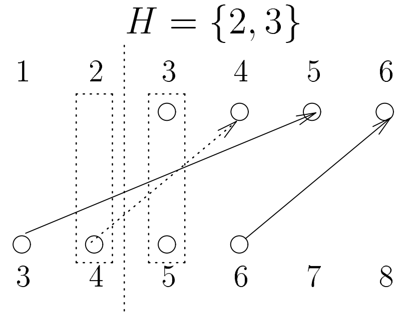
It is defined over $\{s+1, \ldots, n\} \setminus (H+s)$ because $\phi_{B(H)}$ is defined over $\{s+1, \ldots, n\} \setminus (B(H)+s)$.

Its image is a subset of $\{s+1, \ldots, n\} \setminus H$ because the image of $\phi_{B(H)}$ is $\{s+1, \ldots, n\} \setminus B(H)$ and μ_H is a restriction of ϕ_H.

□

Example 4.2.2.
$n = 6$, $s = 2$





In the second case we have: $A(H) = \{2\}$ and $B(H) = \{3\}$.

We introduce the notation

$$\bar{D}(H) = D(H) \cap \mathrm{im}\,(\mu_{C(H)}). \tag{4.22}$$

Definition 4.2.3. Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n\}$, we define the function ψ_H to be the following restriction of μ_H

$$\psi_H = \mu_{C(H)}\big|_{\mathrm{dom}\,(\mu_{C(H)}) \setminus \mu^{-1}_{C(H)}(\bar{D}(H))}. \tag{4.23}$$

Theorem 4.2.4. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n\}$. We have that*

$$\psi_H : P \to R \tag{4.24}$$

where

$$\begin{aligned} P &\subseteq \{s+1, \ldots, n\} \setminus (H + s), \\ R &\subseteq \{s+1, \ldots, n\} \setminus H, \end{aligned} \tag{4.25}$$

and

$$|P|, |R| = n - s - |C(H)| - |\bar{D}(H)|, \tag{4.26}$$

Moreover ψ_H is such that for any $i \in P$, $\psi_h(i) - i$ is a non-negative multiple of s, i. e. there exists a non-negative integer m_i such that

$$\psi_H(i) - i = m_i \cdot s. \tag{4.27}$$

Proof. The function ψ_H is a bijection that satisfies the condition (4.27) because it is a restriction of $\mu_{C(H)}$.

We notice that ψ_H is then defined over

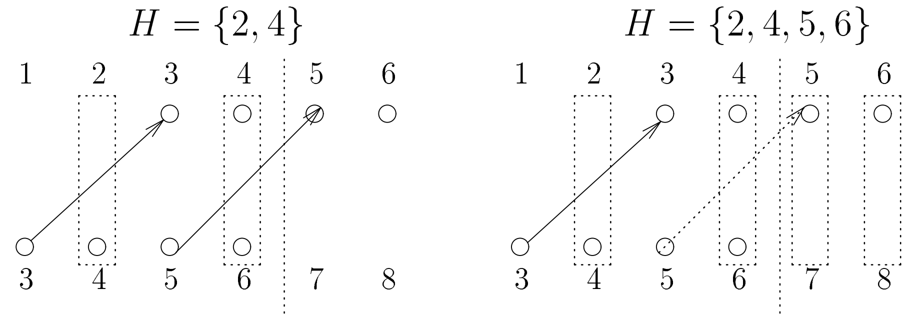
$$(\{s+1, \ldots, n\} \setminus (C(H) + s)) \setminus \mu^{-1}_{C(H)}(\bar{D}(H))$$

which has cardinality

$$n - s - |C(H)| - |\bar{D}(H)|.$$

□

Example 4.2.3.
$n = 6$, $s = 2$:



In the second case we have: $C(H) = \{2, 4\}$, $D(H) = \{5, 6\}$ and $\bar{D}(H) = \{5\}$.

4.2.4 Some Useful Properties

Property 4.2.5. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n\}$. The bijective function ψ_H is such that for any $i \in \operatorname{dom}(\psi_H)$, $\psi_H(i) - i$ is the minimum non-negative multiple of s, i. e. we have*

$$\frac{\psi_H(i) - i}{s} = \min_{l \in \mathbb{N}}(l \mid i + l \cdot s \in \operatorname{im}(\psi_H)). \tag{4.28}$$

Proof. The proof follows from the the fact that ψ_H is a restriction of ϕ_H.
□

Property 4.2.6. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n\}$. The inverse function of ψ_H is such that, for any $j \in \operatorname{im}(\psi_H)$, there exists a non-negative integer m_j such that*

$$j - \psi_H^{-1}(j) = m_j \cdot s. \tag{4.29}$$

Proof. The proof follows from (4.19). In fact for any $j \in \operatorname{im}(\psi_H)$ we have

$$\underbrace{\psi_H(\psi_H^{-1}(j))}_{=j} - \psi_H^{-1}(j) = l_j \cdot s, \tag{4.30}$$

where l_j is a non-negative integer.
□

Let us denote by $\delta(H)$ the set $D(H) \setminus \bar{D}(H)$.

4.2.5 Cardinality

The next two theorems give us the cardinality of the sets:

$$\begin{aligned}&(\{s+1,\dots,n+s\}\setminus(H+s))\setminus\operatorname{dom}\psi_H,\\&(\{1,\dots,n\}\setminus H)\setminus\operatorname{im}\psi_H.\end{aligned}\tag{4.31}$$

The meaning of these apparently obscure sets will become clear in the next section where they are interpreted in terms of factors of polynomials involved in Zeilberger's algorithm that satisfy a certain condition (*Gosper condition*).

Theorem 4.2.7. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, $H \subseteq \{1,\dots,n\}$. Denoted by P_H and R_H the domain and image of the ψ_H. We have*

$$\begin{aligned}&|(\{s+1,\dots,n+s\}\setminus(H+s))\setminus P_H| = s-|\delta(H)|;\\&|(\{1,\dots,n\}\setminus H)\setminus R_H| = s-|\delta(H)|.\end{aligned}\tag{4.32}$$

Proof. By Theorem 4.2.4, we have

$$|P_H| = |R_H| = n-s-|C(H)|-|\bar{D}(H)|.\tag{4.33}$$

Moreover, since

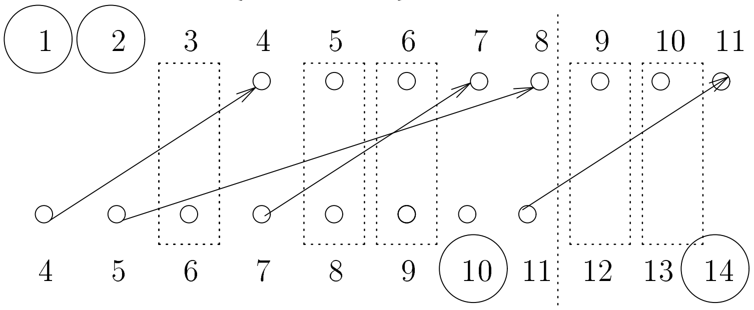
$$\begin{aligned}&H\subseteq\{1,\dots,n\},\\&P_H\subseteq\{s+1,\dots,n+s\}\setminus(H+s),\\&R_H\subseteq\{1,\dots,n\}\setminus H,\end{aligned}\tag{4.34}$$

we have

$$\begin{aligned}&|(\{s+1,\dots,n+s\}\setminus(H+s))\setminus P_H| = |(\{1,\dots,n\}\setminus H)\setminus R_H| =\\&= |(\{1,\dots,n\}|-|H|-|R_H| = n-|H|-(n-s-|C(H)|-|\bar{D}(H)|) =\\&= n-|C(H)|-|D(H)|-n+s+|C(H)|+|\bar{D}(H)| = s-|\delta(H)|;\end{aligned}\tag{4.35}$$

□

Example 4.2.4.
$n = 11$, $s = 3$, $H = \{3,5,6,9,10\}$:



In this case we have $C(H) = \{3, 5, 6\}$, $D(H) = \{9, 10\}$, $\bar{D}(H) = \{10\}$, $\delta(H) = \{9\}$.

We see that the cardinality of the sets $(\{s+1, \ldots, n+s\} \setminus (H+s)) \setminus \mathrm{dom}(\psi_H)$ and $(\{1, \ldots, n\} \setminus H) \setminus \mathrm{im}(\psi_H)$ is $s - |\delta(H)| = 3 - 1 = 2$.

Theorem 4.2.8. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, $H \subseteq \{1, \ldots, n\}$. The set $\delta(H) = D(H) \setminus \bar{D}(H)$ is such that:*

$$0 \leq |\delta(H)| \leq s, \tag{4.36}$$

and

$$|\delta(H)| = s \ \textit{ iff } \ \{1, \ldots, n\} = H. \tag{4.37}$$

Proof. The fact that $0 \leq |\delta(H)| \leq s$ follows immediately from the definition of $\delta(H)$ and from the fact that $\bar{D}(H)$ is a subset of $D(H)$.

We see that if $\{1, \ldots, n\} = H$ then $\mathrm{im}(\mu_{C(H)}) = \emptyset$ and therefore $\bar{D}(H) = \emptyset$ which implies $\delta(H) = D(H)$ and $|\delta(H)| = s$ since $D(H) = \{n - s + 1, \ldots, n\}$.

On the other hand, $|\delta(H)| = s$ implies $\bar{D}(H) = \emptyset$ and $D(H) = \{n - s + 1, \ldots, n\}$. If we assumed that $\{1, \ldots, n-s\} \setminus H$ were not the empty-set we would have that

$$\max(i \,|\, i \in \{1, \ldots, n - s\} \setminus H) + s \in \mathrm{dom}(\mu_{C(H)}) \tag{4.38}$$

and its image would be in

$$\bar{D}(H) = \mathrm{im}(\mu_{C(H)}) \cap D(H);$$

which contradicts the fact that $\bar{D}(H) = \emptyset$.

□

4.2.6 Maximal Cardinality Property

We now prove that ψ_H is the bijection between a subset of $\{s+1, \ldots, n\} \setminus (H+s)$ and a subset of $\{s+1, \ldots, n\} \setminus H$ such that condition (4.27) holds, whose domain and image have the largest possible cardinality.

Theorem 4.2.9. *Given $n, s \in \mathbb{N}$, $s \neq 0$, such that $n \geq 2s$, and $H \subseteq \{1, \ldots, n\}$. There exists no bijection $\chi_H : M \to N$ where*

$$\begin{array}{l} M \subseteq \{s + 1, \ldots, n\} \setminus (H + s), \\ N \subseteq \{s + 1, \ldots, n\} \setminus H, \end{array} \tag{4.39}$$

such that for any $i \in P$, there exists a non-negative integer m_i such that

$$\chi_H(i) - i = m_i \cdot s. \tag{4.40}$$

and $\mathrm{dom}(\psi_H) \subsetneq M$.

PROOF. We notice that

$$\mathrm{dom}(\psi_H) = (\{s+1, \ldots, n\} \setminus (H+s)) \setminus \mu^{-1}_{C(H)}(\bar{D}(H)). \tag{4.41}$$

We show that there cannot be a bijection $\chi_H : M \to N$ such that the condition (4.40) holds and such that $M \subseteq \{s+1, \ldots, n\} \setminus (H+s)$, $N \subseteq \{s+1, \ldots, n\} \setminus H$ and that has any of the elements in $\mu^{-1}_{C(H)}(\bar{D}(H))$ in its domain.

If this were not the case, by Property 4.2.5 and by the fact that for any $i \in \bar{D}(H)$, $\chi_H(\mu^{-1}_{C(H)}(i)) \neq i$, we would also have

$$\chi_H(\mu^{-1}_{C(H)}(i)) > \mu_{C(H)}(\mu^{-1}_{C(H)}(i)) = i. \tag{4.42}$$

By (4.40) we have

$$\chi_H(\mu^{-1}_{C(H)}(i)) - \mu^{-1}_{C(H)}(i) = m \cdot s, \tag{4.43}$$

where m is a non-negative integer.

By Property 4.2.6 on the inverse function

$$i - \mu^{-1}_{C(H)}(i) = l \cdot s, \tag{4.44}$$

where l is a non-negative integer.

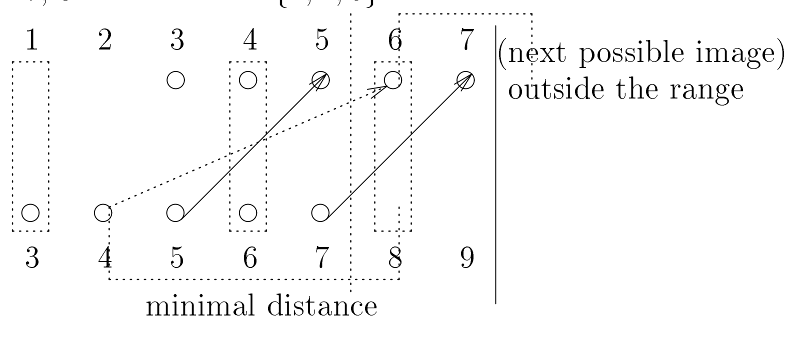
Therefore from (4.42) we have

$$\chi_H(\mu^{-1}_{C(H)}(i)) \geq i + s, \tag{4.45}$$

which is impossible because $i \geq n - s + 1$ since $i \in \bar{D}(H)$.

□

Example 4.2.5.
$n = 7$, $s = 2$ and $H = \{1, 4, 6\}$:



Here we have $C(H) = \{1, 4\}$, $D(H) = \bar{D}(H) = \{6\}$.

4.3 Shift Quotients in Zeilberger's Algorithm

In this section we interpret the previous results in terms of linear factors of polynomials involved in Zeilberger's algorithm.

4.3.1 Notation and Definitions

As in Chapter 1 (Definition 1.2.4) we define:

Definition 4.3.1. We will denote by $F(n,k)$ a proper hypergeometric term, i. e.

$$F(n,k) = P(n,k)x^k \frac{\prod_{i=0}^{A}(a_i n + b_i k + c_i)!}{\prod_{j=0}^{U}(u_j n + v_j k + w_j)!} \tag{4.46}$$

where a_i, b_i, c_i, u_j, v_j and w_j are integers, x is a parameter, and where $P(n,k) \in \mathbb{Q}[n,k]$.

Given a hypergeometric term $F(n,k)$ in n and k, we denote by N and K the *shift operators* whose action on $F(n,k)$ is

$$N s_{n,k} = s_{n+1,k}; \quad K s_{n,k} = s_{n,k+1}. \tag{4.47}$$

We denote by $\mathcal{L}$ the linear recurrence operator with polynomial coefficients in n (free of k) $\sum_{i=0}^{d} z_i(n)N^i$, with $z_i(n) \in \mathbb{Q}(n)$.

Definition 4.3.2. For any hypergeometric term $F(n,k)$ we define the *shift quotient* of $F(n,k)$ in n (respectively in k) as follows:

$$\mathcal{Q}_n(F(n,k)) = \frac{F(n+1,k)}{F(n,k)}, \tag{4.48}$$

and

$$\mathcal{Q}_k(F(n,k)) = \frac{F(n,k+1)}{F(n,k)}, \tag{4.49}$$

respectively.

We denote by $\hat{F}(n,k)$

$$F(n,k)/(P(n,k)x^k) = \frac{\prod_{i=0}^{A}(a_i n + b_i k + c_i)!}{\prod_{j=0}^{U}(u_j n + v_j k + w_j)!}. \tag{4.50}$$

We will denote by num and den the operators that extract respectively numerators and denominators out of a rational function $f(n,k) \in \mathbb{Q}(n,k)$, with $n, k \in \mathbb{Z}$, in normal form, i. e. given

$$f(n,k) = \frac{p_1(n,k)}{p_2(n,k)}, \tag{4.51}$$

with $p_1(n,k), p_2(n,k) \in \mathbb{Q}[n,k]$ such that $\gcd(p_1(n,k), p_2(n,k)) = 1$, we have

$$\operatorname{num} f(n,k) = p_1(n,k); \quad \operatorname{den} f(n,k) = p_2(n,k). \tag{4.52}$$

For the sole purpose of studying the degree it suffices to consider numerators and denominators defined up to multiplicative constant coefficients.

Definition 4.3.3. We define a proper hypergeometric term $F(n,k)$ to be *n-regular* (k-regular) in n if and only if the numerator and the denominator of its *shift quotient* in n (respectively in k) have the same degree in n, i. e.

$$\deg_n(\text{num }\mathcal{Q}_n(F(n,k))) = \deg_n(\text{den }\mathcal{Q}_n(F(n,k))), \tag{4.53}$$

and

$$\deg_n(\text{num }\mathcal{Q}_k(F(n,k))) = \deg_n(\text{den }\mathcal{Q}_k(F(n,k))). \tag{4.54}$$

respectively.

We define *non-n-regular* (non-k-regular) in n any term which does not satisfy (4.53) ((4.54) respectively).

Example 4.3.1. The Binomial Coefficient
The binomial coefficient $\binom{n}{k}$ is n-regular in n but it is non-k-regular in n. In fact we have:

$$\mathcal{Q}_n(\binom{n}{k}) = \frac{n+1}{n-k+1} \tag{4.55}$$

and

$$\mathcal{Q}_k(\binom{n}{k}) = \frac{n-k}{k+1}. \tag{4.56}$$

Example 4.3.2. Rational Functions
Any rational function $F(n,k) = p_1(n,k)/p_2(n,k)$, with $p_1, p_2 \in \mathbb{Q}[n,k]$ is both n-regular in n and k-regular in n. In fact we have:

$$\mathcal{Q}_n(\frac{p_1(n,k)}{p_2(n,k)}) = \frac{p_1(n+1,k)}{p_2(n+1,k)} \cdot \frac{p_2(n,k)}{p_1(n,k)} \tag{4.57}$$

and

$$\mathcal{Q}_k(\frac{p_1(n,k)}{p_2(n,k)}) = \frac{p_1(n,k+1)}{p_2(n,k+1)} \cdot \frac{p_2(n,k)}{p_1(n,k)}. \tag{4.58}$$

Example 4.3.3. The Factorial Function
The factorial term $F(n,k) = n!$ is trivially k-regular in n but non-n-regular in n. In fact

$$\mathcal{Q}_n(n!) = \frac{(n+1)!}{n!} = n+1. \tag{4.59}$$

4.3.2 Interpretation of the Model

With the next theorem and corollary we prove that the degree in n of some polynomials involved in Zeilberger's algorithm can be bounded in a way which is independent of the order d of the recurrence. We do this by interpreting the abstract model described in the previous section in terms of factors of polynomials breaking a certain condition.

In the sequel we will use the following notation:

$$\begin{array}{ll} \mathcal{A}^+ = \{i \mid 0 \le i \le A, a_i > 0\}; & \mathcal{A}^- = \{i \mid 0 \le i \le A, a_i < 0\}; \\ \mathcal{U}^+ = \{j \mid 0 \le j \le U, u_j > 0\}; & \mathcal{U}^- = \{j \mid 0 \le j \le U, u_j < 0\}; \end{array} \tag{4.60}$$

Definition 4.3.4. Given $f \in \mathbb{Z}[n,k]$ and $i \in \mathbb{N}$ we define the *i-rising factorial* of $f(n,k)$, denoted by $f^{\overline{i}}$, to be

$$f^{\overline{i}} = \prod_{j=0}^{i-1}(f+j), \quad \text{for } i > 0, \qquad f^{\overline{0}} = 1. \tag{4.61}$$

Definition 4.3.5. Given $f \in \mathbb{Z}[n,k]$ and $i \in \mathbb{N}$ we define the *i-falling factorial* of $f(n,k)$, denoted by $f^{\underline{i}}$, to be

$$f^{\underline{i}} = \prod_{j=0}^{i-1}(f-j), \quad \text{for } i > 0, \qquad f^{\underline{0}} = 1. \tag{4.62}$$

Lemma 4.3.1. $\operatorname{den} \mathcal{Q}_n(\hat{F}(n,k))$ *must be a divisor of*

$$\prod_{j \in \mathcal{U}^+} (u_j n + v_j k + w_j + 1)^{\overline{u_j}} \cdot \prod_{i \in \mathcal{A}^-} (a_i n + b_i k + c_i)^{\underline{a_i}}. \tag{4.63}$$

Proof. By definition of $\hat{F}(n,k)$ we have

$$\begin{aligned} \mathcal{Q}_n(\hat{F}(n,k)) = & \frac{\prod_{i\in\mathcal{A}^+}(a_i n + b_i k + c_i + a_i)!}{\prod_{j\in\mathcal{U}^+}(u_j n + v_j k + w_j + u_j)!} \cdot \frac{\prod_{j\in\mathcal{U}^+}(u_j n + v_j k + w_j)!}{\prod_{i\in\mathcal{A}^+}(a_i n + b_i k + c_i)!} \cdot \\ & \cdot \frac{\prod_{i\in\mathcal{A}^-}(a_i n + b_i k + c_i + a_i)!}{\prod_{j\in\mathcal{U}^-}(u_j n + v_j k + w_j + u_j)!} \cdot \frac{\prod_{j\in\mathcal{U}^-}(u_j n + v_j k + w_j)!}{\prod_{i\in\mathcal{A}^-}(a_i n + b_i k + c_i)!} = \\ = & \frac{\prod_{i\in\mathcal{A}^+}(a_i n + b_i k + c_i + 1)^{\overline{a_i}}}{\prod_{j\in\mathcal{U}^+}(u_j n + v_j k + w_j + 1)^{\overline{u_j}}} \cdot \frac{\prod_{j\in\mathcal{U}^-}(u_j n + v_j k + w_j)^{\underline{u_j}}}{\prod_{i\in\mathcal{A}^-}(a_i n + b_i k + c_i)^{\underline{a_i}}}. \end{aligned} \tag{4.64}$$

Taking the denominator completes the proof.
□

In the following we will refer to the linear factors in n and k of $\operatorname{den} \mathcal{Q}_n(\hat{F}(n,k))$, i. e. those linear factors in n in k from

$$\prod_{j \in \mathcal{U}^+} (u_j n + v_j k + w_j + 1)^{\overline{u_j}} \cdot \prod_{i \in \mathcal{A}^-} (a_i n + b_i k + c_i)^{\underline{a_i}}$$

that are not canceled in $\mathcal{Q}(\hat{F}(n,k))$, simply as *linear factors* of $\operatorname{den} \mathcal{Q}_n(\hat{F}(n,k))$.

We will refer to the linear factors of $N^h \operatorname{den} \mathcal{Q}_n(\hat{F}(n,k))$, for $h = 0, 1, \ldots, d-1$, as linear factors of $\prod_{h=0}^{d-1}(N^h \operatorname{den} \mathcal{Q}_n(\hat{F}(n,k)))$.

Algorithm 4.1 Gosper Form

Input: $a(n,k)/b(n,k) \in \mathbb{Q}(n,k)$, with $a(n,k), b(n,k) \in \mathbb{Q}[n,k]$

Output: $p(n,k), q(n,k), r(n,k) \in \mathbb{Q}[n,k]$ such that

$$\frac{a(n,k)}{b(n,k)} = \frac{p(n,k+1)}{p(n,k)} \cdot \frac{q(n,k)}{r(n,k+1)}. \tag{4.65}$$

and

$$\gcd(q(n,k), r(n,k+j)) = 1, \tag{4.66}$$

for all $j \geq 1$, $j \in \mathbb{N}$.

$i := 0$;
$p_0(n,k) := 1$;
$q_0(n,k) := a(n,k)$;
$r_0(n,k) := b(n,k-1)$.
while there exists $j \in \mathbb{N}$ such that

$$\gcd(q_i(n,k), r_i(n,k+j)) = f(n,k) \neq 1, \tag{4.67}$$

do
  $q_{i+1}(n,k) := q_i(n,k)/f(n,k)$;
  $r_{i+1}(n,k) := r_i(n,k)/f(n,k-j+1)$;
  $p_{i+1}(n,k) := p_i(n,k) \cdot f(n,k-1)^{\underline{j-1}}$;
  $i := i+1$.
return $(p_i(n,k), q_i(n,k), r_i(n,k))$

Definition 4.3.6. Given a rational function $a(n,k)/b(n,k) \in \mathbb{Q}[n,k]$, we define the *Gosper form* of $a(n,k)/b(n,k)$ to be the triplet $(p(n,k), q(n,k), r(n,k))$ of polynomials in $\mathbb{Q}[n,k]$ such that

$$\frac{a(n,k)}{b(n,k)} = \frac{p(n,k+1)}{p(n,k)} \cdot \frac{q(n,k)}{r(n,k+1)}. \tag{4.68}$$

and such that

$$\gcd(q(n,k), r(n,k+j)) = 1, \quad \forall j > 0, j \in \mathbb{N}. \tag{4.69}$$

We refer to the condition (4.69) as *Gosper condition* .

We refer to the additional condition on p, q and r

$$\begin{aligned} &\gcd(p(n,k+1), r(n,k+1)) = 1, \\ &\gcd(p(n,k), q(n,k)) = 1. \end{aligned} \tag{4.70}$$

as *Petkovšek condition* [Pet92].

We refer to the problem of finding $y(n,k) \in \mathbb{Q}[n,k]$ such that

$$p(n,k) = q(n,k)y(n,k+1) - r(n,k)y(n,k), \tag{4.71}$$

as *Gosper equation* and we call $y(n,k)$ *Gosper polynomial* .

More details on the Gosper form can be found in the first chapter. We also refer to [Gos78], [Pau95], [PS95], [Zei90], [Zei91], [GvzG99], pages 622–639 and [GKP94], pages 223–241.

Remark 4.3.1. Imposing to the polynomials in Gosper form the additional Petkovšek condition (4.70) produces a unique Gosper form up to invertible elements (*Gosper-Petkovšek form*) .

Theorem 4.3.2. *Algorithm 4.1 generates the Gosper form of a rational function.*

For the proof we refer to [PWZ97], pages 79–84.

We will prove the next theorem by interpreting the condition (4.27) on the function ψ_H as a violation of the *Gosper condition.*

Theorem 4.3.3. *If the order d of $\mathcal{L}$ is such that*

$$d \geq M(F(n,k)), \tag{4.72}$$

where

$$M(F(n,k)) = \max(\max_{j\in\mathcal{U}^+}(\lceil 2|v_j|/u_j\rceil), \max_{i\in\mathcal{A}^-}(\lceil 2|b_i|/|a_i|\rceil)), \tag{4.73}$$

then, the number of linear factors f in

$$t(n,k) := \prod_{h=0}^{d-1}(N^h \operatorname{den} \mathcal{Q}_n(\hat{F}(n,k))) \tag{4.74}$$

such that

$$\begin{aligned} &\nexists g \text{ linear factor in } Kt(n,k) \text{ such that} \\ &f = K^m g \text{ for some } m \in \mathbb{N}, \end{aligned} \tag{4.75}$$

is bounded by

$$\sum_{j\in\mathcal{U}^+} |v_j| + \sum_{i\in\mathcal{A}^-} |b_i|, \tag{4.76}$$

Proof. By Lemma 4.3.1, we can write

$$\begin{aligned} t(n,k) = &\prod_{j\in\mathcal{U}^+}\prod_{s=1}^{A_{j,d}}(u_j n + v_j k + w_j + f_{j,s})\cdot \\ &\cdot \prod_{i\in\mathcal{A}^-}\prod_{t=1}^{B_{i,d}}(a_i n + b_i k + c_i + 1 - g_{i,t}); \end{aligned} \tag{4.77}$$

where $A_{j,d}$, $B_{i,d}$ are non-negative integers and $f_{j,s}$, $g_{j,t}$ are non-negative distinct integers such that

$$1 \le f_{j,s} \le d \cdot u_j; \quad 1 \le g_{i,t} \le d \cdot |a_i|.$$

Choosing $d \ge M(F(n,k))$ (4.73) guarantees

$$d \cdot u_j \ge 2 \cdot |v_j|, \ \forall j \in \mathcal{U}^+, \tag{4.78}$$

and

$$d \cdot |a_i| \ge 2 \cdot |b_i|, \ \forall i \in \mathcal{A}^-. \tag{4.79}$$

Hence, if we exclude the trivial cases $v_j = 0$ and $b_i = 0$, we are under the hypotheses of Theorem 4.2.7 and Theorem 4.2.8 which guarantee that for each factor $(a_i n + b_i k + c_i)\underline{^{d\cdot a_i}}$, $i \in \mathcal{A}^-$ and $(u_j n + v_j k + w_j)\overline{^{d\cdot u_j}}$, $j \in \mathcal{U}^+$, respectively at most $|b_i|$ and $|v_j|$ linear factors satisfy the condition in (4.75). Therefore a bound for the number of linear factors in $Kt(n,k)$ that do not violate the condition (4.75), is $\sum_{j\in\mathcal{U}^+}|v_j| + \sum_{i\in\mathcal{A}^-}|b_i|$.

□

Example 4.3.4.
If we take

$$F(n,k) = \frac{(2n+k-1)!}{(4n+2k)!},$$

and $d = 2$ we have:

$$\begin{aligned}\mathcal{Q}_n(\hat{F}(n,k)) = \mathcal{Q}_n(F(n,k)) &= \frac{(2n+k+1)!}{(4n+2k+4)!}\frac{(4n+2k)!}{(2n+k-1)!} = \\ &= \frac{2n+k}{(4n+2k+1)2(4n+2k+3)(4n+2k+4)}.\end{aligned} \tag{4.80}$$

Therefore

$$\begin{aligned}t(n,k) = 4(4n+2k+1)(4n+2k+3)(4n+2k+4)\cdot \\ (4n+2k+5)(4n+2k+7)(4n+2k+8),\end{aligned} \tag{4.81}$$

where

$$t(n,k) := \prod_{h=0}^{d-1}(N^h \operatorname{den} \mathcal{Q}_n(\hat{F}(n,k))), \tag{4.82}$$

and

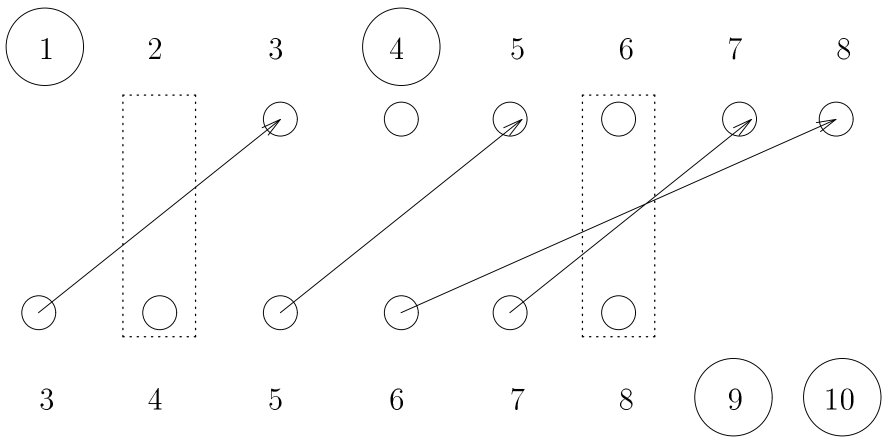
$$\begin{aligned}&\mathcal{Q}_k(1/t(n,k)) = \\ &= \frac{(X+1)(X+3)(X+4)(X+5)(X+7)(X+8)}{(X+3)(X+4)(X+6)(X+7)(X+9)(X+10)}.\end{aligned} \tag{4.83}$$

where $X = 4n + 2k$.

The violations of the Gosper condition (including cancellations) between factors in the numerator and denominator of

$$\mathcal{Q}_k(1/t(n,k)) = \frac{t(n,k)}{Kt(n,k)}, \tag{4.84}$$

are shown in the following diagram describing ψ_H, defined in Definition 4.2.3, with $H = \{2, 6\}$, $n = 8$ and $s = 2$:



The set $H = \{2, 6\}$ describes the missing factors in $t(n, k)$ (i.e. $4n + 2k + 2$, $4n + 2k + 6$). The circled integers are the elements of the sets

$$(\{1, \dots, n\} \setminus H) \setminus \mathrm{im}\,(\psi_H), \tag{4.85}$$

and

$$(\{s + 1, \dots, n + s\}) \setminus (H + s)) \setminus \mathrm{dom}\,(\psi_H). \tag{4.86}$$

namely those integers in the upper set (i.e. $\{1, \dots, 8\}$) that are nor in a "hole" (2 or 6) nor in the the image of ψ_H (i.e. $\{3, 5, 7, 8\}$), and those integers in the lower set (i.e. $\{3, \dots, 10\}$) that are nor in a "shifted hole" (4 or 8) nor in the domain of ψ_H (i.e. $\{3, 5, 6, 7\}$), respectively. The first ones correspond to the factors of the polynomial $q(n, k)$ in the Gosper form of $\mathcal{Q}_k(1/t(n, k))$ and the latter to the factors in the polynomial $r(n, k)$ in the Gosper form of $\mathcal{Q}_k(1/t(n, k))$.

Theorem 4.2.7 and Theorem 4.2.8 bound the cardinality of these sets and therefore also the degree in n of $q(n, k)$ and $r(n, k)$ by $s = 2$.

In the following we will use the following notation

$$\hat{Q}_n = \mathcal{Q}_n(\hat{F}(n, k)), \quad \hat{Q}_k = \mathcal{Q}_k(\hat{F}(n, k)) \tag{4.87}$$

Lemma 4.3.4. *For any $d \in \mathbb{N}$ we have*

$$\begin{aligned}(\sum_{i=0}^{d} z_i(n) N^i) F(n, k) =& (\sum_{i=0}^{d} z_i(n)(N^i P(n, k))\prod_{j=0}^{i-1} N^j \,\mathrm{num}\, \hat{Q}_n \prod_{j=i}^{d-1} N^j \,\mathrm{den}\, \hat{Q}_n)\cdot \\ &\cdot \frac{1}{\prod_{i=0}^{d-1}(N^i \,\mathrm{den}\, \hat{Q}_n)} \cdot x^k \hat{F}(n, k).\end{aligned} \tag{4.88}$$

Proof. We notice that

$$Q_n = \frac{NF(n, k)}{F(n, k)} = \frac{NP(n, k)}{P(n, k)} \cdot \frac{N\hat{F}(n, k)}{\hat{F}(n, k)} = \frac{NP(n, k)}{P(n, k)} \cdot \hat{Q}_n. \tag{4.89}$$

Therefore, using (4.89) we get

$$
\begin{aligned}
&(\sum_{i=0}^{d} z_i(n)N^i)F(n,k) = (\sum_{i=0}^{d} z_i(n)\prod_{j=0}^{i-1} N^j Q_n)F(n,k) = \\
&= (\sum_{i=0}^{d} z_i(n)\prod_{j=0}^{i-1} N^j \frac{NP(n,k)}{P(n,k)} \cdot N^j \hat{Q}_n)F(n,k) = \\
&= (\sum_{i=0}^{d} z_i(n)\frac{N^i P(n,k)}{P(n,k)}\prod_{j=0}^{i-1} N^j \frac{\operatorname{num}\hat{Q}_n}{\operatorname{den}\hat{Q}_n})F(n,k) = \\
&= (\sum_{i=0}^{d} z_i(n)N^i P(n,k)\prod_{j=0}^{i-1} N^j \frac{\operatorname{num}\hat{Q}_n}{\operatorname{den}\hat{Q}_n}) \cdot x^k \hat{F}(n,k) = \\
&=(\sum_{i=0}^{d} z_i(n)(N^i P(n,k))\prod_{j=0}^{i-1} N^j \operatorname{num}\hat{Q}_n \prod_{j=i}^{d-1} N^j \operatorname{den}\hat{Q}_n) \cdot \\
&\cdot \frac{1}{\prod_{i=0}^{d-1}(N^i \operatorname{den}\hat{Q}_n)} \cdot x^k \hat{F}(n,k).
\end{aligned}
\tag{4.90}
$$

□

Corollary 4.3.5. *If the order d of $\mathcal{L}$ is such that*

$$d \geq \max(\max_{j\in\mathcal{U}^+}(\lceil 2|v_j|/u_j\rceil), \max_{i\in\mathcal{A}^-}(\lceil 2|b_i|/|a_i|\rceil)), \tag{4.91}$$

then the Gosper form $(p(n,k), q(n,k), r(n,k))$ of $\mathcal{Q}_k(\mathcal{L}F(n,k))$, will be such that the degrees of $q(n,k)$ and $r(n,k)$ in n are bounded by

$$\deg_n(\operatorname{num}(\hat{Q}_k)) + (\sum_{j\in\mathcal{U}^+} |v_j| + \sum_{i\in\mathcal{A}^-} |b_i|), \tag{4.92}$$

and

$$\deg_n(\operatorname{den}(\hat{Q}_k)) + (\sum_{j\in\mathcal{U}^+} |v_j| + \sum_{i\in\mathcal{A}^-} |b_i|), \tag{4.93}$$

respectively.

Proof. From Lemma 4.88 and from the distributivity of the shift quotient we have that

$$
\begin{aligned}
\mathcal{Q}_k(\mathcal{L}F(n,k)) = \mathcal{Q}_k(&\sum_{i=0}^{d} z_i(n)(N^i P(n,k))\prod_{j=0}^{i-1} N^j \operatorname{num}\hat{Q}_n \prod_{j=i}^{d-1} N^j \operatorname{den}\hat{Q}_n) \cdot \\
&\cdot \mathcal{Q}_k(1/(\prod_{h=0}^{d-1} N^h \operatorname{den}\hat{Q}_n)) \cdot x \cdot \hat{Q}_k;
\end{aligned}
\tag{4.94}
$$

We notice the following facts:

- $\mathcal{Q}_k\,(1/(\prod_{h=0}^{d-1} N^h \operatorname{den}\hat{Q}_n))$, by Theorem 4.3.3, contributes to the degree in n of $q(n,k)$ and $r(n,k)$ by a degree bounded by $\sum_{j\in\mathcal{U}^+}|v_j| + \sum_{i\in\mathcal{A}^-}|b_i|$;

- $x\cdot\hat{Q}_k$ gives a contribution to $q(n,k)$ and $r(n,k)$ with degree in n respectively $\deg_n \operatorname{num}(\hat{Q}_k)$ and $\deg_n \operatorname{den}(\hat{Q}_k)$;

- $\mathcal{Q}_k(\sum_{i=0}^{d} z_i(n)(N^i P(n,k)) \prod_{j=0}^{i-1} N^j \operatorname{num}\hat{Q}_n\cdot \prod_{j=i}^{d-1} N^j \operatorname{den}\hat{Q}_n)$ violates the Gosper condition (see (4.69)) because it is the shift quotient in k of a polynomial, and therefore does not contribute to the degree in n of either $q(n,k)$ or $r(n,k)$.

□

Remark 4.3.2. While Corollary 4.3.5 holds for any Gosper form, condition (4.70) (Petkovšek condition) can produce a Gosper form with lower degree polynomials.

4.3.3 Gosper Equation

In the next two theorems we find conditions under which it is possible, at least heuristically, to find a relation between the highest degree in n of $z_i(n)$ for $0 \le i \le d-1$ and of the coefficients $y_i(n)$ with respect to k of the Gosper polynomial $y(n,k)$.

In the following we assume that $\mathcal{L}$ is such that $\mathcal{L}F(n,k)$ is *Gosper-summable.*

We denote the *Gosper form* of $\mathcal{L}F(n,k)$ by $(p(n,k), q(n,k), r(n,k))$.

We also introduce the following definitions:

$$\begin{aligned}
&\zeta := \max_{0\le i\le d}(\deg_n(z_i(n))); \quad \gamma := \deg_n(y(n,k));\\
&\alpha := \deg_n(q(n,k)); \quad \beta := \deg_n(r(n,k)); \quad \delta := \deg_n(\operatorname{num}\hat{Q}_n);
\end{aligned} \tag{4.95}$$

where $y(n,k)$ is the *Gosper polynomial.* We denote the coefficient of k^i in $y(n,k)$ by $y_i(n)$.

We denote by $(\bar{p}(n,k), \bar{q}(n,k), \bar{r}(n,k))$ the *Gosper form* of

$$\mathcal{Q}_k\,(1/(\prod_{h=0}^{d-1} N^h \operatorname{den}\hat{Q}_n))\cdot x\cdot \hat{Q}_k.$$

For a given polynomial $a(n,k)\in\mathbb{Q}[n,k]$ we denote by $\operatorname{Lcoeff}_n(a(n,k))$ the *leading coefficient* of $a(n,k)$ with respect to n, i. e. the coefficient of highest degree with respect to n.

Remark 4.3.3. Since $\mathcal{L}F(n,k)$ is assumed to be *Gosper-summable* we have $\zeta \geq 0$. Since $y(n,k)$ cannot be identically zero we must also have $\gamma \geq 0$.

Lemma 4.3.6. *Given $F(n,k)$ proper hypergeometric, n-regular in n, then the degree in n of $p(n,k)$ is bounded by:*

$$\deg_n(\bar{p}(n,k)) + \zeta + \deg_n(P(n,k)) + d \cdot \delta. \tag{4.96}$$

PROOF. $\mathcal{Q}_k(\sum_{i=0}^{d} z_i(n)(N^i P(n,k)) \prod_{j=0}^{i-1} N^j \operatorname{num} \hat{Q}_n \cdot \prod_{j=i}^{d-1} N^j \operatorname{den} \hat{Q}_n)$ violates the Gosper condition because it is the shift quotient of a polynomial.

Then the *Gosper form* $(p(n,k), q(n,k), r(n,k))$ of $\mathcal{L}F(n,k)$ has the following form:

$$p(n,k) = p'(n,k) \cdot \bar{p}(n,k); \quad q(n,k) = \bar{q}(n,k); \quad Kr(n,k) = K\bar{r}(n,k). \tag{4.97}$$

where

$$p'(n,k) = \sum_{i=0}^{d-1} z_i(n)(N^i P(n,k)) \prod_{j=0}^{i-1} N^j \operatorname{den} \hat{Q}_n \cdot \prod_{j=i}^{d-1} N^j \operatorname{num} \hat{Q}_n, \tag{4.98}$$

whose degree is bounded by $\zeta + \deg_n(P(n,k)) + d \cdot \delta$.
□

Lemma 4.3.7. *Let $F(n,k)$ be proper hypergeometric and non-k-regular in n, and let us denote by $(p(n,k), q(n,k), r(n,k))$ the Gosper form for $\mathcal{L}F(n,k)$. Then $q(n,k)y(n,k+1) - r(n,k)y(n,k)$ has degree in n:*

$$\gamma + \max(\alpha, \beta). \tag{4.99}$$

PROOF. We notice that the hypothesis of non-k-regularity in n of $F(n,k)$ together with the fact that $1/(\prod_{h=0}^{d-1}(N^h \operatorname{den} \hat{Q}_n))$ is k-regular in n implies that $\hat{F}(n,k) \cdot 1/(\prod_{h=0}^{d-1}(N^h \operatorname{den} \hat{Q}_n))$ is non-k-regular in n, from which follows $\alpha \neq \beta$. Therefore there is no cancellation between the leading coefficients of the polynomials $q(n,k)y(n,k+1)$ and $r(n,k)y(n,k)$. □

Theorem 4.3.8. *Given $F(n,k)$ proper hypergeometric, n-regular in n. If*

$$\sum_{i \mid \deg_n(z_i)=M} \operatorname{Lcoeff}_n(z_i) \operatorname{Lcoeff}_n(P(n,k)) \operatorname{Lcoeff}_n(\operatorname{den} \hat{Q}_n)^i \cdot \operatorname{Lcoeff}_n(\operatorname{num} \hat{Q}_n)^{d-i} \neq 0 \tag{4.100}$$

and either $F(n,k)$ is non-k-regular in n,

or

$F(n,k)$ is k-regular in n and

$$\sum_{j\,|\,\deg_n(y_j)=M'} \operatorname{Lcoeff}_n(q)\operatorname{Lcoeff}_n(y_j(n))(k+1)^j - \operatorname{Lcoeff}_n(r)\operatorname{Lcoeff}_n(y_j(n))k^j \neq 0, \tag{4.101}$$

where M and M' are the highest degrees in n of the $z_i(n)$ for $0 \le i \le d-1$ and $y(n,k)$'s, respectively, then we have

$$\zeta + \deg_n(\bar{p}(n,k)) + \deg_n(P(n,k)) + d\cdot\delta = \gamma + \max(\alpha,\beta). \tag{4.102}$$

Proof. We see that the first condition (4.100) guarantees that the left hand side of the *Gosper equation* (4.71), i. e. $p(n,k)$ has degree exactly $\zeta + \deg_n(\bar{p}(n,k)) + \deg_n(P(n,k)) + d\cdot\delta$.

The second condition guarantees that the degree in n of the right hand side, i. e. of $q(n,k)y(n,k+1) - r(n,k)y(n,k)$ is $\gamma + \max(\alpha,\beta)$.

□

Theorem 4.3.9. *Given $F(n,k)$ proper hypergeometric, n-regular in n, non-k-regular in n, then we have*

$$\zeta + \deg_n(\bar{p}(n,k)) + \deg_n(P(n,k)) + d\cdot\delta \ge \gamma + \max(\alpha,\beta). \tag{4.103}$$

Proof. By Lemma 4.3.6, the fact that $F(n,k)$ is non-k-regular in n guarantees that the right hand side of the *Gosper equation* (4.71), i. e.$q(n,k+1)y(n,k) - r(n,k)y(n,k)$ has degree in n exactly $\gamma + \max(\alpha,\beta)$.

By Lemma 4.3.7, the degree in n of the left hand side, i. e. $p(n,k)$ is at most

$$\zeta + \deg_n(\bar{p}(n,k)) + \deg_n(P(n,k)) + d\cdot\delta.$$

□

4.3.4 Lower Bounds on Degrees

We now use the previous result to get lower bounds on the degree in n of the coefficients $z_i(n)$, for $0 \le i \le d-1$, of the linear recurrence operator $\mathcal{L}$, and on the Gosper polynomial $y(n,k)$.

Definition 4.3.7. Let us define:

$$D(F(n,k),d) = \max(\alpha,\beta) - (\deg_n(\bar{p}(n,k)) + \deg_n(P(n,k)) + d\cdot\delta). \tag{4.104}$$

Remark 4.3.4. By Corollary 4.3.5, we have that α and β can be bounded independently of d, whereas $\deg_n(\bar{p}(n,k)) + \deg_n(P(n,k)) + d\cdot\delta$ grows with d when $\delta \ge 1$. Therefore, if $\delta \ge 1$, for d big enough we expect $D(F(n,k),d)$ to be negative.

Theorem 4.3.10. *Given $F(n,k)$ proper hypergeometric, n-regular in n, non-k-regular in n, then the following lower bound on the degree in n of $z_i(n)$, for $0 \le i \le d-1$, and of the coefficients $y_i(n)$ of $y(n,k)$ with respect to k must hold:*

$$\zeta \ge D(F(n,k), d). \tag{4.105}$$

Proof. It follows from Theorem 4.3.9, and from the fact that

$$\zeta \ge \gamma + D(F(n,k), d) \ge D(F(n,k), d),$$

because the Gosper polynomial cannot be identically zero.

□

Theorem 4.3.11. *Given $F(n,k)$ proper hypergeometric, n-regular in n, then if condition (4.100) holds and either $F(n,k)$ is non-k-regular in n or condition (4.101) holds, then we have*

$$\zeta \ge D(F(n,k), d); \quad \gamma \ge -D(F(n,k), d). \tag{4.106}$$

Proof. It follows from Theorem 4.3.8, and from the fact that

$$\begin{aligned} \zeta &= \gamma + D(F(n,k), d) \ge D(F(n,k), d), \\ \gamma &= \zeta - D(F(n,k), d) \ge -D(F(n,k), d), \end{aligned} \tag{4.107}$$

because the Gosper polynomial cannot be identically zero and not all the $z_i(n)$ can be zero.

□

4.3.5 Some Examples

We run Zeilberger's algorithm implemented as described in Chapter 1 on increasing powers of binomials $\binom{n}{k}^p$, for $p = 1, 2, \ldots, 7$ since the order d of the recurrence found by the algorithm is related to the power p by the simple formula $d = \lfloor p/2 \rfloor$.

power	order	α	β	δ	γ	ζ	$\gamma - \zeta$
1	1	1	0	1	0	0	0
2	1	2	0	2	1	1	0
3	2	3	0	3	5	2	3
4	2	4	0	4	7	3	4
5	3	5	0	5	16	6	10
6	3	6	0	6	20	9	11
7	4	7	0	7	37	16	21

We observe how $\gamma - \zeta$ grows in this case with the order of the recurrence.

In all the presented examples we have

$$\begin{aligned} \deg_n(\bar{p}(n,k)) &= 0; \\ \deg_n(P(n,k)) &= 0. \end{aligned} \tag{4.108}$$

We notice that $\binom{n}{k}^p$ with $p \in \mathbb{N}$ is n-regular in n and non-k-regular in n:

$$\begin{aligned} \mathcal{Q}_n(\binom{n}{k}^p) &= \frac{(n+1)^p}{(n-k+1)^p}; \\ \mathcal{Q}_k(\binom{n}{k}^p) &= \frac{(n-k)^p}{(k+1)^p}. \end{aligned} \tag{4.109}$$

Some other examples:

term	order	α	β	δ	γ	ζ	$\gamma - \zeta$
$\frac{n!}{k!(m+k)!(n-m-2k)!}$	2	2	0	1	1	1	0
$\frac{(n-k)!}{k!(n-2k)!}$	2	2	1	1	0	0	0
$\binom{2k}{k}\binom{n}{k}^2$	2	2	1	2	3	2	1

4.4 Conclusion

The main result achieved in this chapter is a combinatorial model that describes the Gosper condition and therefore the Gosper form. By this model we know that the degree of certain polynomials ($q(n,k)$ and $r(n,k)$) does not grow with the order of the recurrence (see Corollary 4.3.5). From this, under some hypotheses on the term (see (4.91)), we can estimate the difference between the degree (in n) of the coefficients of the linear recurrence (i. e. z_i) and the coefficients (i. e. y_i) of the Gosper polynomial.

We can apply this result to get lower bounds on the degree of solutions of related systems of linear equations. This is useful when computing the components of the solutions by polynomial interpolation. How to use this result together with rational function interpolation still has to be investigated and some preliminary experiments are encouraging. We could use interpolation only on low degree components and standard symbolic Gaussian elimination for the remaining components.

The main limitation of this result on the degree of the components of the solutions is given by the hypotheses on the term ($F(n,k)$). Such hypotheses are often fulfilled when Zeilberger's algorithm is implemented as described in Chapter 1, but can become a serious restriction when Zeilberger's algorithm is implemented in a different way (for details on how to "tune" Zeilberger's algorithm we refer to [Rie01]). For instance, when summing

$$\sum_{k=0}^{n} \binom{n}{k} \tag{4.110}$$

we could rewrite the problem as follows

$$\sum_{k=0}^{n} \binom{n}{k} = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} = n! \sum_{k=0}^{n} \frac{1}{k!(n-k)!}, \tag{4.111}$$

where $1/(k!(n-k)!)$ is simpler to sum because it generates a simpler system of linear equations but it is no longer n-regular in n as $\binom{n}{k}$, and therefore does not satisfy the conditions of theorems 4.3.9, 4.3.8, 4.3.10, 4.3.11.

Further study will be necessary to implement these results and extend them to more general cases.

Part III

Polynomial Arithmetic Library

Chapter 5

Combinatorial Interpretation of Division

5.1 Introduction

We describe the recurrence for the coefficients of the polynomial $y(x)$ such that $p(x) \cdot y(x) \equiv 1 \mod x^l$, where l is a non-negative integer and $p(x)$ is an invertible polynomial over a ring R, i. e. $x \nmid p(x)$, for which therefore $y(x)$ exists. We will use a transcendental extension of R as a tool to study the recurrence. Besides, we give a combinatorial interpretation of the solution $y(x)$ that relates the coefficients of $y(x)$ with non-negative integer compositions.

5.2 Definitions

Let us denote by K a field containing $\mathbb{Q}$, and let R either be equal to K or to the ring $K[v_1, v_2, \ldots, v_r]$ of multivariate polynomials over K.

Let a_0, a_1, ... a_n be transcendental elements over the ring R and let $\bar{R} = R(a_0, a_1, \ldots, a_n)$ be the transcendental ring extension over R with respect to a_0, a_1, ... a_n.

Given these transcendentals a_i, let $a(x) \in \bar{R}[x]$ be

$$a(x) = \sum_{i=0}^{n} a_i x^i.$$

Hence $\deg(a) = n$.

Let us consider a polynomial $b(x) \in R[x]$ with $\deg(b) = m$ and $b_m \in K \setminus \{0\}$:

$$b(x) = \sum_{i=0}^{m} b_i x^i$$

and the polynomials $q(x)$, $r(x) \in \bar{R}[x]$, such that:

$$a(x) = b(x) \cdot q(x) + r(x) \tag{5.1}$$

and either

$$r(x) = 0 \tag{5.2}$$

or

$$\deg(r) < \deg(b). \tag{5.3}$$

The transcendentals $a_0,\ldots,a_n$ are a tool to study a recurrence related to $b(x)$.

Remark 5.2.1. The existence and uniqueness of $q(x)$, $r(x)$ are guaranteed by the fact that b_m, being an element of $K \setminus \{0\}$, is invertible.

Notation

- $\delta_{i,j}$ denotes the Kronecker delta function, which is 1 if $i = j$ and 0 otherwise.
- $[x^l]$ is the operator which extracts the coefficient of x^l from a polynomial expanded with respect to x.

Definition 5.2.1. Given $p(x) \in \bar{R}[x]$ and $d \in \mathbb{Z}$ such that $d \geq \deg(p)$, we define the *d-reversed polynomial* of the polynomial $p(x)$, denoted by $\mathrm{rev}_d(p(x))$, to be

$$\mathrm{rev}_d(p(x)) = x^d p(1/x). \tag{5.4}$$

Convention: Unless differently specified, the j-th coefficients of a, b, q, r and of other polynomials will be denoted with the corresponding indexed letter, i. e. $q_j = [x^j]q(x)$. Coefficients with indeces outside the range will be assumed to be zero.

Definition 5.2.2. Given the equality (5.1), we define

$$C_b(i) := [a_n]q_{n-m-i},$$

i. e. $C_b(i)$ is the coefficient of a_n in $[x^{n-m-i}]q(x)$.

When b is clear from the context we use $C(i)$ instead of $C_b(i)$.

5.3 Existence of the Inverse

Property 5.3.1. *For any polynomial* $p(x) \in R[x]$ *(or respectively* $\bar{R}[x]$*),* $\deg(p) = m$*, with* $p_0 \neq 0$*, and for any non-negative integer* l*, there exists always a polynomial* $y(x) \in R[x]$ *(respectively* $\bar{R}[x]$*) such that*

$$p(x) \cdot y(x) \equiv 1 \mod x^l. \tag{5.5}$$

Moreover,

$$y_0 = p_0^{-1}. \tag{5.6}$$

PROOF. The condition (5.5) is equivalent to

$$\sum_{k=0}^{l-1} \left(\sum_{i=0}^{k} p_i y_{k-i} \right) x^k = 1. \tag{5.7}$$

Comparing the coefficients with respect to x yields a system of linear equations in triangular form:

$$\begin{array}{lllcl} p_0 y_0 & & & = & 1 \\ p_1 y_0 + & p_0 y_1 & & = & 0 \\ p_2 y_0 + & p_1 y_1 + & p_0 y_2 & = & 0 \\ \dots\dots & \dots\dots & \dots\dots & & \dots \end{array} \tag{5.8}$$

in which the coefficient of the highest variable in the ordering $y_0 < y_1 < \cdots < y_{l-1}$ is always p_0. Therefore such a system can be solved by back-substitution, that yields

$$y_0 = p_0^{-1}, \tag{5.9}$$

which proves the second statement (5.6) , and for all $1 \leq k \leq l-1$ we have

$$y_k = -\frac{1}{p_0} \sum_{i=1}^{k} p_i y_{k-i}; \quad k \in \mathbb{N} \setminus \{0\}. \tag{5.10}$$

□

Remark 5.3.1. The recurrence (5.10) gives an algorithm for computing the inverse. Since p_i is zero for $i \leq 0$ and $i \geq m$, such algorithm performs $\mathcal{O}(l \cdot m)$ multiplications in the ground field.

5.4 Relation between C and q

For the sake of simplicity we will consider $b_m = 1$ in the following. The more general case when b_m is an arbitrary element of $K \setminus \{0\}$ can be reduced to the case $b_m = 1$ by dividing the coefficients of $b(x)$ by the leading coefficient.

We will need the following lemmas to show how $C(i)$ gives the necessary information to invert the m-reversed of $b(x) = \sum_{i=0}^{m} b_i x^i \in R[x]$:

Lemma 5.4.1. *For any non-negative integer i, the coefficients of the quotient of $a(x)$ divided by $b(x)$ can be expressed by the following formula:*

$$q_{n-m-i} = a_{n-i} - \sum_{k=\max(0,i-m)}^{i-1} b_{m-i+k} q_{n-m-k}. \tag{5.11}$$

Moreover, q_{n-m-i} is linear in a_n, a_{n-1}, ..., a_{n-i}.

Proof. The proof of the above formula follows immediately from the correctness of the school division algorithm. In fact this formula describes in a recursive fashion a step of the school division algorithm:

$$\begin{array}{llll}
\boxed{a_n}\, x^{n-m} & + \boxed{(a_{n-1} - a_n b_{m-1})}\, x^{n-m-1} & + \boxed{(a_{n-2} - a_n b_{m-2} - (a_{n-1} - a_n b_{m-1}) b_{m-1})}\, x^{n-m-2} & \\
+ \dots & + q_1 x + q_0 = & & \\
(\boxed{a_n}\, x^n & + a_{n-1} x^{n-1} & + a_{n-2} x^{n-2} + \dots + a_0) \div \; (x^m + b_{m-1} x^{m-1} + \dots + b_0) & \\
a_n x^n & + a_n b_{m-1} x^{n-1} & + a_n b_{m-2} x^{n-2} + \dots & \\
\hline
 & \boxed{(a_{n-1} - a_n b_{m-1})}\, x^{n-1} & + (a_{n-2} - a_n b_{m-2}) x^{n-2} + \dots & \\
 & (a_{n-1} - a_n b_{m-1}) x^{n-1} & + (a_{n-1} - a_n b_{m-1}) b_{m-1} x^{n-2} + \dots & \\
\hline
 & & \boxed{(a_{n-2} - a_n b_{m-2} - (a_{n-1} - a_n b_{m-1}) b_{m-1})}\, x^{n-2} + \dots & \\
 & & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots &
\end{array}$$

Since $b(x) \in R[x]$, the linearity of q_{n-m-i} is immediately proved by induction on i.
□

Lemma 5.4.2. *For any $h \in \{1, 2, \dots, n\}$ and for any non-negative integer i, we have*

$$[a_{h-1}] q_{n-m-i-1} = [a_h] q_{n-m-i}. \tag{5.12}$$

Proof. The statement is true when $i = 0$ because by Lemma 5.4.1 (5.11) we have $q_{n-m} = a_n$ and $q_{n-m-1} = a_{n-1} - a_n b_{m-1}$ and therefore it follows that

$$[a_{h-1}]q_{n-m-1} = [a_{h-1}](a_{n-1} - a_n b_{m-1}) = \delta_{h,n} = [a_h]a_n = [a_h]q_{n-m}.$$

Assuming that the statement holds for values less than a certain positive i, we prove it for i.

By Lemma 5.4.1 (5.11), we have:

$$\begin{aligned}
&[a_{h-1}]q_{n-m-(i+1)} = \\
=&[a_{h-1}](a_{n-(i+1)} - \sum_{k=\max(0,i+1-m)}^{i} b_{m-(i+1)+k} q_{n-m-k}) = \\
=&\delta_{h,n-i} - \sum_{k=\max(0,i+1-m)}^{i} b_{m-i+k-1}[a_{h-1}]q_{n-m-k} = \\
=&\delta_{h,n-i} - \underbrace{b_{m-i-1}[a_{h-1}]q_{n-m}}_{=0} - \sum_{k=\max(1,i+1-m)}^{i} b_{m-i+k-1}[a_{h-1}]q_{n-m-k} = \\
=&\delta_{h,n-i} - \sum_{k=\max(0,i-m)}^{i-1} b_{m-i+k}[a_{h-1}]q_{n-m-(k+1)} = \\
=&\delta_{h,n-i} - \sum_{k=\max(0,i-m)}^{i-1} b_{m-i+k}[a_h]q_{n-m-k} = \\
=&[a_h](a_{n-i} - \sum_{k=\max(0,i-m)}^{i-1} b_{m-i+k} q_{n-m-k}) = [a_h]q_{n-m-i}.
\end{aligned} \tag{5.13}$$

□

Theorem 5.4.3. *The coefficients of the quotient $q(x)$ are given by the following formula:*

$$q_l = \sum_{j=0}^{n-m-l} a_{n-j} C(n-m-l-j). \tag{5.14}$$

Moreover, the polynomial $\sum_{i=0}^{n-m} C(i)x^i$ is the modular inverse of $\mathrm{rev}_m(b)$ *modulo x^{n-m+1}, i. e.*

$$\sum_{i=0}^{n-m} C(i)x^i \equiv \mathrm{rev}_m(b)^{-1} \mod x^{n-m+1}. \tag{5.15}$$

Proof. By definition of $C(i)$ and by Lemma 5.4.2 (5.12) we have that

$$C(n-m-l-j) = [a_n]q_{l+j} = [a_{n-j}]q_l. \tag{5.16}$$

By Lemma 5.4.1 (5.11), we know that q_l is linear in a_n, a_{n-1}, ..., a_{m+l}. Therefore we have:

$$q_l = \sum_{j=0}^{n-m-l} a_{n-j} \cdot [a_{n-j}]q_l = \sum_{j=0}^{n-m-l} a_{n-j} C(n-m-l-j). \tag{5.17}$$

In order to prove the second part of the lemma, we notice that, by the previous statement (5.17), we must have

$$\begin{aligned} \mathrm{rev}_n(a) \cdot \sum_{i=0}^{n-m} C(i)x^i \equiv \sum_{l=0}^{n-m} \left(\sum_{j=0}^{l} a_{n-j} C(l-j) \right) x^l \equiv \\ \equiv \sum_{l=0}^{n-m} q_{n-m-l} x^l \equiv \mathrm{rev}_{n-m}(q) \mod x^{n-m+1}. \end{aligned} \tag{5.18}$$

By the existence of $\mathrm{rev}_m(a)^{-1}$, proved in Property 5.3.1, we also have

$$\sum_{i=0}^{n-m} C(i)x^i \equiv \mathrm{rev}_n(a)^{-1} \cdot \mathrm{rev}_{n-m}(q) \mod x^{n-m+1}. \tag{5.19}$$

Applying rev_n to a yields, because of (5.1),

$$\mathrm{rev}_n(a) = \mathrm{rev}_m(b) \cdot \mathrm{rev}_{n-m}(q) + x^{n-m+1} \mathrm{rev}_{m-1}(r), \tag{5.20}$$

and therefore

$$\mathrm{rev}_n(a) \equiv \mathrm{rev}_m(b) \cdot \mathrm{rev}_{n-m}(q) \mod x^{n-m+1}. \tag{5.21}$$

Hence by the existence of $\mathrm{rev}_m(b)^{-1}$, guaranteed by Property 5.3.1, we also have

$$\mathrm{rev}_m(b)^{-1} \equiv \mathrm{rev}_n(a)^{-1} \cdot \mathrm{rev}_{n-m}(q) \mod x^{n-m+1}. \tag{5.22}$$

which, with the previous congruence (5.19), completes the proof.

□

5.5 General Recurrence for $C(i)$

Theorem 5.5.1. *The sequence $C(i)$ satisfies the following m-th order linear recurrence:*

$$C(i) = \begin{cases} 0, & \text{if } i < 0, \\ 1, & \text{if } i = 0, \\ -\sum_{k=\max(0,i-m)}^{i-1} b_{m-i+k} C(k), & \text{if } i > 0. \end{cases} \tag{5.23}$$

PROOF. For any negative i we clearly have:

$$C(i) = [a_n]q_{n-m-i} = [a_n]0 = 0.$$

From Lemma 5.4.1 (5.11) we have the following recurrence equation for q_{n-m-i}

$$q_{n-m-i} = a_{n-i} - \sum_{k=\max(0,i-m)}^{i-1} b_{m-i+k}q_{n-m-k}. \tag{5.24}$$

Applying $[a_n]$ to both sides of this recurrence completes the proof.
□

Remark 5.5.1. Theorem 5.5.1 can be also proved by using the fact that the $C(i)$ are the coefficients of the inverse of $\text{rev}_m(b)$ as seen in Theorem 5.4.3 (5.15) and therefore satisfy the recurrence (5.10) shown in the proof of the Property 5.3.1, which is, up to a change in the indices the same as (5.23).

Remark 5.5.2. This m-th order recurrence describes an algorithm for computing the coefficients $C(i)$ of the modular inverse of $\text{rev}_m(b(x))$, which computes each coefficient by m multiplications in R.

5.6 Lifting to the $b(x^\alpha)$ Case

Here we show how $C_{b(x)}$ can be used to construct $C_{b(x^\alpha)}$, for any positive integer α.

Remark 5.6.1. $C_{b(x^\alpha)}(i)$ is, by Theorem 5.4.3 (5.15), the coefficient of degree i in the inverse of $\text{rev}_{m\cdot\alpha}(b(x^\alpha))$.

Theorem 5.6.1. *For any monic polynomial $b(x)$ and any positive integer α we have*

$$C_{b(x^\alpha)}(i) = \begin{cases} C_{b(x)}(i/\alpha), & \text{if } i \equiv 0 \mod \alpha, \\ 0, & \text{otherwise.} \end{cases} \tag{5.25}$$

PROOF. Let us denote $C_{b(x)}$ and $C_{b(x^\alpha)}$ by C and C^α, respectively.
By Theorem 5.4.3 (5.15) we have that

$$\sum_{i=0}^{n-m}(C(i)x^i) \cdot \text{rev}_m(b(x)) \equiv 1 \mod x^{n-m+1}. \tag{5.26}$$

Therefore, by definition of rev_m (5.4), substituting x with x^α, we have

$$\sum_{i=0}^{n-m}(C(i)x^{\alpha i}) \cdot \mathrm{rev}_{\alpha m}(b(x^{\alpha})) \equiv 1 \mod x^{\alpha(n-m+1)}, \tag{5.27}$$

and, since for any positive α, $\alpha(n-m+1) \geq n - \alpha m + 1$, we also have

$$\sum_{i=0}^{n-m}(C(i)x^{\alpha i}) \cdot \mathrm{rev}_{\alpha m}(b(x^{\alpha})) \equiv 1 \mod x^{n-\alpha m+1}. \tag{5.28}$$

Moreover, by Theorem 5.4.3 (5.15) applied to C^{α} we have that

$$\sum_{i=0}^{n-\alpha m}(C^{\alpha}(i)x^{i}) \cdot \mathrm{rev}_{\alpha m}(b(x^{\alpha})) \equiv 1 \mod x^{n-\alpha m+1}. \tag{5.29}$$

Hence, from (5.28) and (5.29) we get

$$\sum_{i=0}^{n-\alpha m}(C^{\alpha}(i)x^{i}) = \sum_{i=0}^{n-m}(C(i)x^{\alpha i}). \tag{5.30}$$

Comparing the coefficients with respect to x completes the proof.
□

5.7 Combinatorial Meaning

Definition 5.7.1. A *composition* of a non-negative integer i is a sequence $[p_1, p_2, \ldots, p_n]$ of positive integers such that $\sum_{j=1}^{n} p_j = i$. By convention the empty sequence [] is the only composition of 0.

For any non-negative integer i, we denote by $\mathrm{Comp}(i)$ the set of all the compositions of i.

Definition 5.7.2. For any non-negative integers i and j, we define the *concatenation* of $\mathrm{Comp}(i)$ with j, denoted by $\mathrm{Comp}(i)|j$, as the set of sequences obtained from $\mathrm{Comp}(i)$ by appending j to the end of all the sequences (compositions of i) in $\mathrm{Comp}(i)$.

Example 5.7.1.
For $i = 3$ and $j = 5$ we have:

$$\begin{aligned} \mathrm{Comp}(3) &= \{[1,1,1],[2,1],[1,2],[3]\} \\ \mathrm{Comp}(3)|5 &= \{[1,1,1,5],[2,1,5],[1,2,5],[3,5]\} \end{aligned} \tag{5.31}$$

Corollary 5.7.1. *For any non-negative integers m and i we have:*

$$C_{b(x)}(i) = \sum_{t \in \mathrm{Comp}(i)} \prod_{c \in^* t} (-b_{m-c}), \tag{5.32}$$

where $\in^*$ means being a component of t.

Proof. This can be proven by induction on i and by using the recursive formula (5.23) for $C(i)$:

From (5.23) we have $C(0) = 1 = \sum_{t \in \{[]\}} \prod_{c \in^* t} (-b_{m-c})$.

The induction step is proven by the fact that the set $\mathrm{Comp}(i)$ is given by

$$\mathrm{Comp}(i) = \cup_{0 \le k < i} \mathrm{Comp}(k)|(i-k), \tag{5.33}$$

where $\mathrm{Comp}(k)|(i-k)$ corresponds to the summand $b_{m-i+k}C(k)$ in the formula (5.23).

□

Example 5.7.2.
Let us see how the compositions of 4 can be written in terms of compositions of 0, 1, 2 and 3:

$$\begin{aligned} &\mathrm{Comp}(4) = \mathrm{Comp}(0)|4 \cup \mathrm{Comp}(1)|3 \cup \mathrm{Comp}(2)|2 \cup \mathrm{Comp}(3)|1 = \\ &= \{[]\}|4 \cup \{[1]\}|3 \cup \{[1,1],[2]\}|2 \cup \{[1,1,1],[1,2],[2,1],[3]\}|1 = \\ &= \{[4],[1,3],[1,1,2],[2,2],[1,1,1,1],[1,2,1],[2,1,1],[3,1]\}. \end{aligned} \tag{5.34}$$

5.8 Binomial Case

The previous formula has an interesting special case when $b(x) = (x-d)^m$, i. e. $b_i = \binom{m}{i}(-d)^{m-i}$:

Corollary 5.8.1. *For any non-negative integers m, i and $d \in K \setminus \{0\}$ we have:*

$$C_{(x-d)^m}(i) = \sum_{t \in \mathrm{Comp}(i)} \prod_{c \in^* t} ((-1)^{c+1} d^c \binom{m}{c}) = \binom{m+i-1}{i} \cdot d^i, \tag{5.35}$$

Proof. By Corollary 5.7.1 we have

$$C_{(x-d)^m}(i) = \sum_{t \in \mathrm{Comp}(i)} \prod_{c \in^* t} ((-1)^{c+1} d^c \binom{m}{c}). \tag{5.36}$$

We prove that $C_{(x-d)^m}(i) = \binom{m+i-1}{i} \cdot d^i$ by induction on i.

By Theorem 5.5.1 (5.23) we have

$$C_{(x-d)^m}(0) = 1 = \binom{m+0-1}{0} \cdot d^0.$$

We now assume that for a certain positive i, $C_{(x-d)^m}(k) = \binom{m+k-1}{k} \cdot d^k$ for any $0 \le k < i$, and we prove it for i.

By Theorem 5.23 we have

$$\begin{aligned} C_{(x-d)^m}(i) &= -\sum_{k=0}^{i-1}((-d)^{i-k}\binom{m}{i-k}C_{(x-d)^m}(k)) = \\ &= -\sum_{k=0}^{i-1}((-d)^{i-k}\binom{m}{i-k}\binom{m+k-1}{k} \cdot d^k) = \\ &= \sum_{k=0}^{i-1}((-1)^{i-k+1}d^i\binom{m}{i-k}\binom{m+k-1}{k}). \end{aligned} \tag{5.37}$$

At first glance to solve a definite sum like in (5.37) one would use Zeilberger's algorithm [Zei90], [Zei91]. However it turns out that this problem is also solvable by Gosper's algorithm [Gos78] for indefinite hypergeometric summation. Both algorithms have been implemented in various packages for different computer algebra systems [Car99], [PS95], and are described in Section 1.3.1 and Section 1.3.2 as well as in [PWZ97], [Win96] and in [GvzG99], pages 622–639 and [GKP94], pages 223–241. By using one of these packages, one obtains

$$\begin{aligned} &(-1)^{i-k+1}d^i\binom{m}{i-k}\binom{m+k-1}{k} = \\ &= \Delta_k\left(\frac{(-1)^{i-k} \cdot d^i k(m+k-i)}{i \cdot m} \cdot \binom{m}{i-k}\binom{m+k-1}{k}\right), \end{aligned} \tag{5.38}$$

where Δ_k denotes the *forward difference* operator, i. e. $\Delta_k(f(k)) = f(k+1) - f(k)$.

Therefore from (5.37) we get

$$\begin{aligned} &C_{(x-d)^m}(i) = \\ &= \sum_{k=0}^{i-1}\Delta_k\left(\frac{(-1)^{i-k} \cdot d^i k(m+k-i)}{i \cdot m} \cdot \binom{m}{i-k}\binom{m+k-1}{k}\right) = \\ &= \binom{m+i-1}{i} \cdot d^i, \end{aligned} \tag{5.39}$$

which, with (5.36), completes the proof.

□

Remark 5.8.1. The formula (5.25) allows us to generalize (5.35) to the case $b(x) = (x^\alpha - d)^m$, for any positive integer α. The corresponding formula becomes

$$C_{(x^\alpha - d)^m}(i) = \begin{cases} \binom{m+i/\alpha-1}{i/\alpha} d^{i/\alpha}, & \text{if } i \equiv 0 \mod \alpha, \\ 0, & \text{otherwise.} \end{cases} \tag{5.40}$$

Remark 5.8.2. The binomial coefficient in the right hand side of (5.35):

$$\binom{m+i-1}{i} = \binom{m+i-1}{m-1},$$

also counts all the compositions of $m+i$ into exactly m parts. For a simple bijective proof of this see [And94], pages 54–57.

5.8.1 Diagrams of Compositions

In this section we define a relation between two integer compositions and we prove a property on such relation that will be later useful to prove Corollary 5.8.1 in a combinatorial fashion.

Definition 5.8.1. For any non-negative integer i we define the relation $\longrightarrow$, called *diagram relation*, between two compositions $C_1, C_2 \in \mathrm{Comp}(i)$ as follows: $C_1 \longrightarrow C_2$ if and only if C_2 is obtained from C_1 by merging two consecutive parts of C_1 into one.

We denote by $\stackrel{*}{\longrightarrow}$ the transitive and reflexive closure of the diagram relation. Then $C_1 \stackrel{*}{\longrightarrow} C_2$ means that C_2 can be obtained from C_1 by merging consecutive parts of C_1 into one.

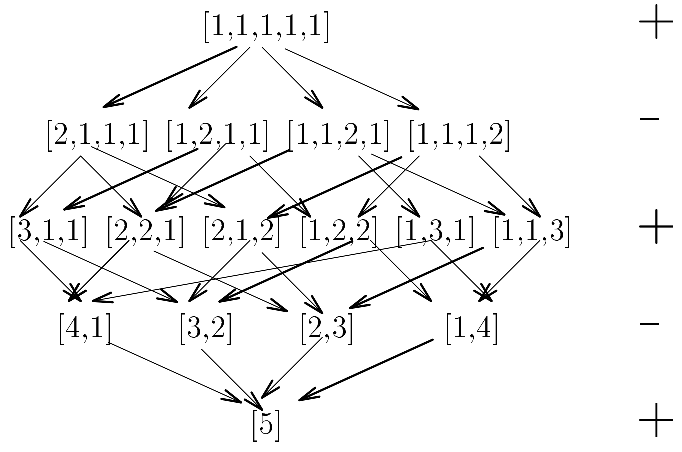
For each composition C we define the *sign function* σ as follows

$$\sigma(C) = \begin{cases} 1, & \text{if in C has an even number of even parts;} \\ -1, & \text{otherwise.} \end{cases} \tag{5.41}$$

Example 5.8.1.
For $i = 4$ we have:

$$\begin{tikzcd}
 & {[1,1,1,1]} \arrow[dl] \arrow[d] \arrow[dr] & & + \\
{[2,1,1]} \arrow[d] \arrow[dr] & {[1,2,1]} \arrow[dl] \arrow[dr] & {[1,1,2]} \arrow[dl] \arrow[d] & - \\
{[3,1]} \arrow[dr] & {[2,2]} \arrow[d] & {[1,3]} \arrow[dl] & + \\
 & {[4]} & & -
\end{tikzcd}$$

For $i = 5$ we have:



Definition 5.8.2. For any non-negative i we denote by 1_i the composition of i with i parts:

$$1_i = [\underbrace{1, 1, \ldots, 1}_{i}]. \tag{5.42}$$

Lemma 5.8.2. *For any positive integer i and any composition $C \in \mathrm{Comp}(i)$, the number of compositions $D \in Comp(i)$ with exactly h parts, such that $D \xrightarrow{*} C$ is given by*

$$\binom{i-k}{h-k}, \tag{5.43}$$

where k is the number of parts of C.

Proof. We prove this by using the bijection between compositions of i into exactly h parts and choices without repetitions of $h-1$ elements in $\{1, 2, \ldots, i-1\}$. We can visualize each chosen element j as a line between j and $j+1$ in the range $[1, 2, \ldots, i]$.

The condition $D \xrightarrow{*} C$ means, by definition of $\xrightarrow{*}$, that the composition C is obtained from D by merging some adjacent parts. In terms of choices in the set $\{1, 2, \ldots, i-1\}$ this means that the choices corresponding to the composition C must also correspond to the composition D.

Therefore the number of compositions D with h parts, such that $D \xrightarrow{*} C$ is given by the number of possible choices of $h - 1 - (k - 1)$ elements in a set with $i - 1 - (k - 1)$ elements, since C has k parts to which $k - 1$ chosen elements correspond. Thus we have that the final count is:

$$\binom{i-1-(k-1)}{h-1-(k-1)} = \binom{i-k}{h-k}. \tag{5.44}$$

□

Theorem 5.8.3. *For any positive integer i and any composition C of i we have*

$$\sum_{D|D\xrightarrow{*}C} \sigma(D) = \delta_{C,1_i}. \tag{5.45}$$

Proof. We notice that for any integer l such that $1 \leq l \leq i+1$, the number of even parts in a composition $D \in \mathrm{Comp}(i)$ with $i-l+1$ parts, will be even if l is even and odd otherwise. Therefore for the compositions $D \in \mathrm{Comp}(i)$ with $i-l+1$ we have $\sigma(D) = (-1)^{l+1}$.

By Lemma 5.8.2, we have that the number of compositions $D \in \mathrm{Comp}(i)$ with $i-l+1$ parts, such that $D \xrightarrow{*} C$, is given by

$$\binom{i-k}{i-l+1-k}, \tag{5.46}$$

where k is the number of parts of C.

Summing with the corresponding sign over all possible number of parts of D, i. e. $1 \leq l \leq i-k+1$ we get

$$\sum_{D|D\xrightarrow{*}C} \sigma(D) = \sum_{l=1}^{i-k+1} (-1)^{l+1} \binom{i-k}{i-l+1-k}. \tag{5.47}$$

The sum in (5.47), being a definite hypergeometric sum, can be proved by Zeilberger's algorithm, but it happens to be also solvable by Gosper's algorithm for indefinite hypergeometric summation.

By performing one of these algorithms on (5.47) we get

$$\sum_{l=1}^{i-k+1} (-1)^{l+1} \binom{i-k}{i-l+1-k} = \delta_{C,1_i}, \tag{5.48}$$

which proves (5.45).

□

Remark 5.8.3. The sign function σ is a special case of the *Möbius function* (see [Str94b], pages 116–117) for partially ordered sets, where the partial ordering here is given by the relation $\xrightarrow{*}$.

5.8.2 Combinatorial Proof

Corollary 5.8.1 can also be proved in a purely combinatorial way, namely by finding a bijection between two sets whose cardinality is given by the left hand side and by the right hand side of (5.35), respectively.

For any non-negative integer k, we call a set (respectively multiset) with k elements, a *k-set* (respectively *k-multiset*). We call a sequence with k elements (respectively without repetitions) a *k-sequence* (respectively a *k-sequence without repetitions*) .

Definition 5.8.3. For any non-negative integer k and any set S we define a *combination* (respectively *combination without repetitions*) from S of size k, to be any k-subset (respectively k-multiset) of S.

Definition 5.8.4. For any non-negative integer k and any set S we define a *k-permutation* (respectively *k-permutation without repetition*) from S, to be any k-sequence of elements from S (respectively k-sequence of elements from S without repetitions).

PROOF. First of all we observe that the binomial coefficient in the right hand side of (5.35):

$$\binom{m+i-1}{i}$$

counts all the combinations with repetitions of size i from an m-set. For a simple combinatorial proof of this we refer to [Str94b], pages 13–17.

We show that the left hand side of the formula (5.35) describes an inclusion-exclusion[1] procedure which counts the combinations with repetitions of size i from an m-set by first counting all the i-permutations with repetitions and then by subtracting and adding i-permutations with repetitions where increasingly larger blocks of consecutive elements are inversely ordered with respect to a certain order.

Let us first count all the i-permutations with repetitions from an m-set. Since for each position we have m choices, this overcounting yields:

$$\underbrace{\binom{m}{1}\cdots\binom{m}{1}}_{i} = \binom{m}{1}^{i}, \tag{5.49}$$

which in fact corresponds to the composition of i:

$$[\underbrace{1, 1, \cdots, 1}_{i}].$$

In order to count the i-combinations with repetitions from an m-set, superfluous i-permutations with repetitions must be removed.

[1]For a formal and very general definition of inclusion-exclusion see [Sta97], pages 64–67.

Let us consider the m elements from which we have built all i-permutations with repetitions totally ordered with respect to $\preceq$.

We disregard those i-permutations in which in any position (i. e. from 1 to $i-1$) two consecutive elements are not ordered[2] with respect to the relation $\preceq$.

This is done for all choices of 2 elements from m and for each position from 1 to $i-1$; therefore we subtract from the count:

$$\binom{m}{2} \cdot \underbrace{\binom{m}{1} \cdots \binom{m}{1}}_{i-2} = \binom{m}{2}\binom{m}{1}^{i-2}, \tag{5.50}$$

which corresponds to the compositions of i:

$$\begin{array}{llllll}
[2, & 1, & 1, & \cdots, & 1, & 1], \\
[1, & 2, & 1, & \cdots, & 1, & 1], \\
\multicolumn{6}{c}{\dots\dots\dots\dots\dots\dots} \\
[1, & 1, & 1, & \cdots, & 1, & 2].
\end{array}$$

By removing the superfluous i-permutations where two consecutive elements are not ordered with respect to $\preceq$, we have also overcounted superfluous i-permutations with repetitions where two consecutive elements are not ordered with respect to $\preceq$, in fact cases with 3 consecutive elements in the opposite with respect to $\preceq$ and cases with two couples of consecutive elements in the wrong order are overcounted, for instance cases with 3 consecutive elements inversely ordered are removed both when the first two consecutive elements and when the second two consecutive elements are considered.

The overcounted i-permutations with repetitions correspond to the next class of compositions of i in which two parts are merged into one:

$$\begin{array}{llllll}
[3, & 1, & 1, & \cdots, & 1, & 1], \\
[1, & 3, & 1, & \cdots, & 1, & 1], \\
\multicolumn{6}{c}{\dots\dots\dots\dots\dots\dots} \\
[1, & 1, & 1, & \cdots, & 1, & 3];
\end{array}$$

and

[2] This is how the sorting algorithm "bubble sort" works.

$$\begin{array}{llllllll}
[2, & 2, & 1, & 1 & \cdots, & 1, & 1], \\
[2, & 1, & 2, & 1, & \cdots, & 1, & 1], \\
\multicolumn{7}{l}{\dots\dots\dots\dots\dots\dots\dots\dots} \\
[2, & 1, & 1, & 1, & \cdots, & 1, & 2]; \\
\\
[1, & 2, & 2, & 1, & \cdots, & 1, & 1], \\
[1, & 2, & 1, & 2, & \cdots, & 1, & 1], \\
\multicolumn{7}{l}{\dots\dots\dots\dots\dots\dots\dots\dots} \\
[1, & 2, & 1, & 1, & \cdots, & 1, & 2]; \\
\multicolumn{7}{l}{\dots\dots\dots\dots\dots\dots\dots\dots} \\
\multicolumn{7}{l}{\dots\dots\dots\dots\dots\dots\dots\dots}
\end{array}$$

In this way we have again overcounted those i-permutations with repetitions where larger groups of consecutive elements are inversely ordered with respect to $\preceq$, which correspond to compositions of i with larger parts obtained by merging two parts into one.

In the inclusion-exclusion procedure the sign of the correction will be positive if the corresponding compositions of i have an even number of even parts and will be negative otherwise, in fact merging two odd parts generates a new even part and merging two parts where at least one is even decreases the number of even parts by 1.

This process can be described as a diagram of i. Theorem 5.8.3 guarantees that each class of i-permutations with blocks of consecutive elements inversely ordered is counted correctly, i. e. the sum of the times in which it is included and excluded is zero.

$\square$

Some related combinatorial proofs on compositions can be found in [And94], pages 54–57 and [Str94b], pages 13–17.

Example 5.8.2.
Let us consider all combinations with repetitions of size 3 from the set $\{a, b, c, d\}$. Then we are in the case $m = 4$, and $i = 3$.

Counting all 3-permutations with repetitions from a 4-set, yields:

$$\binom{4}{1} \cdot \binom{4}{1} \cdot \binom{4}{1} = \binom{4}{1}^3 = 64,$$

which corresponds to the composition $[1, 1, 1]$.

Let us describe 3-permutations with repetitions from a 4-set as strings[3] of size 3 from the set $\{a, b, c, d\}$ where the considered ordering $\preceq$ is the alphabetical ordering.

[3]Strings are represented as sequences of characters between round parentheses.

We disregard those 3-permutations with repetitions where two consecutive elements are not in alphabetical order. Therefore we are counting those 3-permutations with repetitions which match the following patterns[4]:

$$\begin{array}{ll} (b \quad a \quad *), & (* \quad b \quad a), \\ (c \quad a \quad *), & (* \quad c \quad a), \\ (d \quad a \quad *), & (* \quad d \quad a), \\ (c \quad b \quad *), & (* \quad c \quad b), \\ (d \quad b \quad *), & (* \quad d \quad b), \\ (d \quad c \quad *); & (* \quad d \quad c). \end{array} \tag{5.51}$$

The number of all these cases is therefore

$$2 \cdot \binom{4}{2} \cdot \binom{4}{1} = 48,$$

where the patterns on the left and right column of (5.51) correspond respectively to the compositions $[2, 1]$ and $[1, 2]$.

By doing so we are overcounting 3-permutations with repetitions where 3 consecutive elements are not alphabetically ordered:

$$\begin{array}{l} (c \quad b \quad a), \\ (d \quad b \quad a), \\ (d \quad c \quad a), \\ (d \quad c \quad b). \end{array} \tag{5.52}$$

The number of these cases is:

$$\binom{4}{3} = 4,$$

which corresponds to the composition $[3]$.

In fact the 3-permutations with repetitions in (5.52) match patterns both on the left and on the right column of (5.51):

$$\begin{aligned} (c\, b\, a) &\in (*\, b\, a) \cap (c\, b\, *); \\ (d\, b\, a) &\in (*\, b\, a) \cap (d\, b\, *); \\ (d\, c\, a) &\in (*\, c\, a) \cap (d\, c\, *); \\ (d\, c\, b) &\in (*\, c\, b) \cap (d\, c\, *). \end{aligned} \tag{5.53}$$

Hence the final count is:

$$\binom{4}{1} - 2\binom{4}{2} \cdot \binom{4}{1} + \binom{4}{3} = 20 = \binom{4+3-1}{3}. \tag{5.54}$$

[4]In this context a pattern is a set of strings represented with the same notation as of a string but with at least one occurrence of the wild-card character "*", which denotes any possible element of the set {a,b,c,d}.

5.9 Powers of Polynomials

We now consider a well-known result[5] that generalizes Corollary 5.8.1 to the case where $b(x)$ is any rational power of a polynomial in $R[x]$ with constant coefficient in $K \setminus \{0\}$. This result is a generalization of (5.35) but is obtained in a totally different way, namely by seeing powers of polynomials as formal power series (*generating functions*) whose coefficients can be described by a recurrence. We will refer to this method as *exponentiation by generating functions*. When this method is used for dividing polynomials we will refer to it as *dividing by generating functions*.

We introduce the following notation:

$$\begin{aligned} &d(x) \in R[x], \;\; \deg(d) = s, \;\; d_0 \in K \setminus \{0\}, \\ &\beta_{d^p}(x) = d(x)^{-p} = \sum_{i=0}^{\infty} G_{d^p}(i)x^i, \; G_{d^p}(i) \in R, \; \forall i \in \mathbb{N}; \end{aligned} \tag{5.55}$$

where $p \in \mathbb{Q}$.

For brevity of notation we use $\beta(x)$ for $\beta_{d^p}(x)$ and $G(i)$ for $G_{d^p}(i)$ whenever this does not cause ambiguity.

Theorem 5.9.1. *When $d(0) = 1$ and p is a non-negative integer, we have*

$$G_{d^p}(i) = C_{\mathrm{rev}_{p\cdot s}(d^p)}(i), \tag{5.56}$$

for any non-negative integer i.

PROOF. By definition, $G_{d^p}(i)$ is the coefficient of degree i in $d(x)^{-p}$, which is the inverse of $d(x)^p$. By Theorem 5.4.3 (5.15) $C_{\mathrm{rev}_{p\cdot s}(d^p)}(i)$ is the coefficient of degree i in the inverse of the $\mathrm{rev}_{p\cdot s}(\mathrm{rev}_{p\cdot s}(d^p)) = d(x)^p$.
□

Theorem 5.9.2. *For any non-negative integer i and any $p \in \mathbb{Q}$ $G_{d^p}(i)$ is solution of the following linear order recurrence of order s (degree of $d(x)$) with linear coefficients:*

$$\sum_{i=0}^{s}(j + i\cdot(p-1))d_iG(t-j) = 0 \;\; (\forall j \in \mathbb{N} \setminus \{0\}). \tag{5.57}$$

PROOF. First of all we notice that using the definition of $\beta(x)$ and differentiating we obtain

$$\beta'(x) = -p \cdot d(x)^{-p-1} \cdot d'(x).$$

[5] For more details we refer to [GCL92], pages 114–116.

This, together with the definition of $\beta(x)$, yields a first order linear differential equation:

$$d(x) \cdot \beta'(x) + p \cdot d'(x) \cdot \beta(x) = 0. \tag{5.58}$$

Plugging the explicit forms of $d(x)$ and $\beta(x)$ into this equation, yields

$$\sum_{i=0}^{s} d_i x^i \cdot \sum_{i=0}^{\infty} G(i) i x^{i-1} + p \cdot \sum_{i=0}^{s} d_i i x^{i-1} \cdot \sum_{i=0}^{\infty} G(i) x^i = 0.$$

Computing the Cauchy product and shifting summation indices gives

$$\begin{aligned} &\sum_{l=0}^{\infty} \left(\sum_{i=0}^{l} (i+1) G(i+1) d_{l-i} \right) x^l + \\ &+ p \cdot \sum_{l=0}^{\infty} \left(\sum_{i=0}^{l} (G(i)(l-i+1) d_{l-i+1} \right) x^l = \\ = &\sum_{l=0}^{\infty} \left(\sum_{i=1}^{l+1} i G(i) d_{l-i+1} \right) x^l + \\ &+ p \cdot \sum_{l=0}^{\infty} \left(\sum_{i=0}^{l} (G(i)(l-i+1) d_{l-i+1} \right) x^l = \\ = &\sum_{l=0}^{\infty} \left(\sum_{i=0}^{l+1} (i + (l-i+1)p) d_{l-i+1} G(i) \right) x^l = 0. \end{aligned} \tag{5.59}$$

Comparing the coefficients with respect to x gives

$$\sum_{i=0}^{l+1} (i + (l-i+1)p) d_{l-i+1} G(i) = 0; \ \forall l \in \mathbb{N}.$$

We notice that $(d_{l-i+1})_{i \in \mathbb{N}}$ has the finite support $[l+1-s, l+1]$, and therefore

$$\sum_{i=l+1-s}^{l+1} (i + (l-i+1)p) d_{l-i+1} G(i) = 0; \ \forall l \in \mathbb{N}.$$

Substituting $l+1$ by j and an index transformation give

$$\sum_{i=0}^{s} (j + i \cdot (p-1)) d_i G(j-i) = 0 \ \ \forall j \in \mathbb{N} \setminus \{0\}. \tag{5.60}$$

□

5.9.1 Initial Values

From Property 5.3.1 (5.6) we have that the initial value $G_{d^p}(0)$ must be the inverse of the constant coefficient of $d(x)^p$, namely d_0^{-p}. Moreover, the recurrence (5.57) encodes the remaining $s-1$ initial values, which are obtained as the first $s-1$ solutions of (5.57) for $t \in [1, s-1]$.

5.10 The Algorithms

The recurrence (5.57) in Theorem 5.9.2 is an efficient algorithm for computing the modular inverse of powers of polynomials. Besides, fast inversion of powers of polynomials by this method can be used to speed-up polynomial division by powers of polynomials.

5.10.1 Modular Inverse

The recurrence (5.57) describes an algorithm that performs s multiplications over R at each step. In fact from (5.57) we see that for $t \in \mathbb{N} \setminus \{0\}$, we can write $G(t)$ as s summands:

$$G(t) = -\frac{1}{d_0} \cdot \sum_{i=1}^{s} \frac{(t + i \cdot (p-1))}{t} \cdot d_i \cdot G(t-i). \tag{5.61}$$

Therefore the complexity for the computation of the inverse modulo x^l by this algorithm is $\mathcal{O}(s \cdot l)$.

Exponentiation of dense univariate polynomials has been implemented in C++. For more details and for a comparison with repeated squaring we refer to the next chapter.

Remark 5.10.1. The complexity of the previous algorithm is independent of the power p.

Remark 5.10.2. For $p = 1$ the recurrence (5.61) becomes the recurrence (5.10) used in the proof of Property 5.3.1.

5.10.2 Division by Powers of Polynomials

Division with remainder of the polynomial $\alpha(x) \in R[x]$ of degree n by $d(x)^p$, where $d(x) \in R[x]$ is a polynomial of degree s, can be done by first computing the inverse of $\text{rev}_{p \cdot s}(d(x)^p)$ modulo x^{n-ps} and then by multiplying the inverse of $\text{rev}_{p \cdot s}(d(x)^p)$ by $\text{rev}_{p \cdot s}(\alpha(x))$.

Let us here denote by $q(x)$ the quotient and by $r(x)$ the remainder in the division of $\alpha(x)$ by $d(x)^p$.

As seen in the proof of Theorem 5.4.3 in (5.22), we have

$$\begin{aligned}\operatorname{rev}_{p\cdot s}(d)^{-p} &= \operatorname{rev}_{p\cdot s}(d^p)^{-1} \equiv \\ &\equiv \operatorname{rev}_n(\alpha)^{-1} \cdot \operatorname{rev}_{n-p\cdot s}(q) \mod x^{n-p\cdot s+1},\end{aligned} \tag{5.62}$$

from which it follows

$$\operatorname{rev}_{n-p\cdot s}(q) \equiv \operatorname{rev}_{p\cdot s}(d)^{-p} \cdot \operatorname{rev}_n(\alpha) \mod x^{n-p\cdot s+1}. \tag{5.63}$$

Since $\deg(q) = n - ps$, the quotient can be computed by (5.63) and by taking the $(p \cdot s)$-reversed.

Computing $\operatorname{rev}_{p\cdot s}(d)^{-p}$ modulo $x^{n-p\cdot s+1}$ by (5.61) takes $n - ps$ steps, each of which requires s multiplications in the ground field R. Therefore the overall complexity of the modular inversion is $\mathcal{O}(s(n - ps)) = \mathcal{O}(sn - ps^2)$, which, as long as s is small, is linear in the degree of the quotient, i. e. $n - ps$.

The complexity of the computation of the quotient will then depend on the complexity of the multiplication of $\operatorname{rev}_{p\cdot s}(d)^{-p}$ by $\operatorname{rev}_n(\alpha)$, which dominates the complexity of the modular inversion.

The remainder is then computed by the formula

$$r(x) = \alpha(x) - d(x)^p \cdot q(x), \tag{5.64}$$

which has the complexity of the multiplication of $d(x)^p$ by $q(x)$.

Remark 5.10.3. When computing the remainder with (5.64), $d(x)^p$ can be computed by the formula (5.61).

Division by powers of polynomial has been implemented in a C++ package for polynomial arithmetic. For more details and for the implementation in C++ we refer to the next chapter.

5.11 Summary

In this chapter we have used the school division algorithm for polynomials to find a description of the modular inverse of a polynomial (Theorem 5.4.3). With this result we can describe the coefficients of the modular inverse of a polynomial over a field containing $\mathbb{Q}$ by a linear recurrence (Theorem 5.5.1). Such a linear recurrence has a combinatorial meaning in terms of integer compositions (Corollary 5.7.1).

We have treated the special case of the modular inverse of a binomial or power of a binomial of a special form (Corollary 5.8.1), which leads to prove the combinatorial identity (5.35), for which we can provide a combinatorial proof using the principle of inclusion-exclusion.

We have related these results to previous well-known facts which generalize the recurrence corresponding to the special binomial case (Theorem 5.9.1).

In the end we describe how to use the recurrences as algorithms to compute powers of polynomials and how to divide polynomials by powers of polynomials.

Chapter 6

The *PolyComb* Package

6.1 Introduction

We describe the algorithms and the structure of the C++/Mathematica library *PolyComb* for fast polynomial arithmetic. This chapter also contains a manual of the Mathematica and C++ commands provided by *PolyComb*. The library has two parts: a C++ implementation of fast polynomial arithmetic for dense and sparse polynomials, and a Mathematica interface which makes it possible to use the library from within the Mathematica environment. The implemented methods and operators between polynomials include addition, multiplication, exponentiation and division with remainder and various other auxiliary methods.

The C++ component of the library makes full use of the object oriented paradigm and of generic programming through the extensive use of *templates*. Thus polynomials over rings are defined over any ring of coefficients that provide the necessary *methods* (operations) required by a ring.

6.2 Structure of the Library

The library has a Mathematica [Wol99] component that provides an easy-to-use interface, and a C++ component that implements fast polynomial arithmetic.

6.2.1 The Mathematica Component

The part of the library written in Mathematica [Wol99] is composed by an interface and by a parser which is used by the interface to translate the polynomial expression into a format which can be easily processed by the C++ library.

The Interface

Different versions of the interface are available depending on the type of polynomials involved in the expressions, e. g. whether the polynomials are sparse or dense, whether their power products are sparse or dense, whether their coefficients are arbitrary length integers or floating point reals, etc. The interface takes as input a polynomial expression in the Mathematica syntax.

The interface invokes the parser, which, using also the information on the type of polynomial expression, rewrites the expression in a special compressed postfix format (see next section for more details).

The interface uses the C++ library to perform the computation and waits for the result which is read in from the standard output device.

The Parser

The parser performs two tasks

1. It parses a polynomial expression, it matches the largest possible set of subexpressions in it that can be considered a polynomial and writes them as one polynomial. In this way it minimizes the size of the output and the overhead time for communication (see Example 6.2.1).

2. It transforms the polynomial expression into a special postfix format that can be read by the C++ component of the library.

Example 6.2.1.
The polynomial expression

$$3x^2 + (4x^3 + 2x)(x + 1) + x^3 - 2$$

is parsed as

$$(\underbrace{x^3 + 3x^2 - 2}_{\text{polynomial}}) + (\underbrace{4x^3 + 2x}_{\text{polynomial}})(\underbrace{x + 1}_{\text{polynomial}})$$

The postfix form will be saved as a parenthesisless sequence of polynomials and operators. At the end of the process the postfix form is passed by the interface to the library.

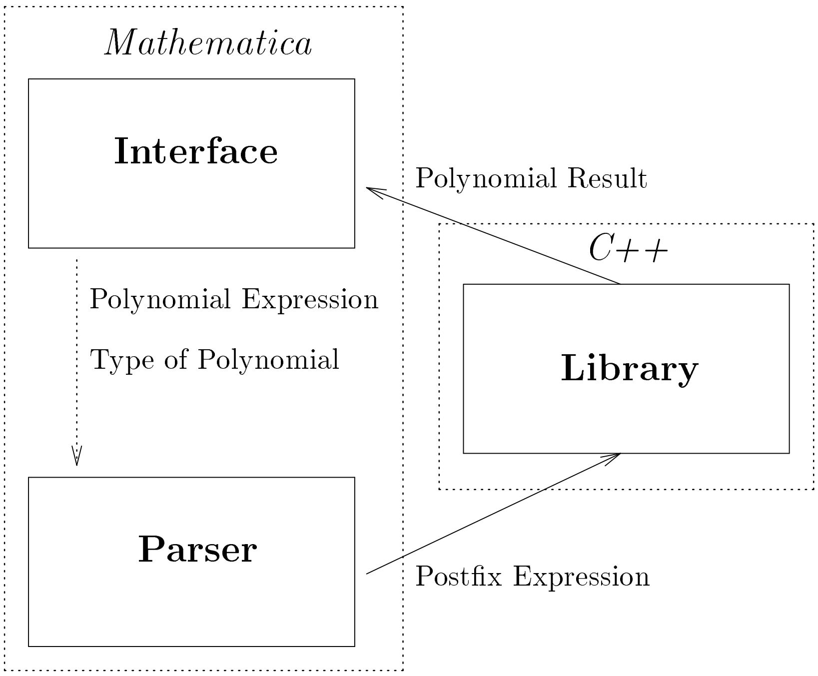
6.2.2 The C++ Component

The C++ component is the core of the library which does all the computation and outputs the result into a file or screen in a format that is both human readable and that can be read and processed by Mathematica.

The library processes the input in postfix form by a stack. It iteratively reads an element in postfix form and if it is a polynomial, it puts it into the stack, if it is an n-ary operand, it takes n polynomials from the top of the stack, it performs the corresponding operation on them and puts the result into the stack. The final result will be the polynomial left in the stack.

6.2.3 The Communication

The following picture briefly describes how the components interact with each other.



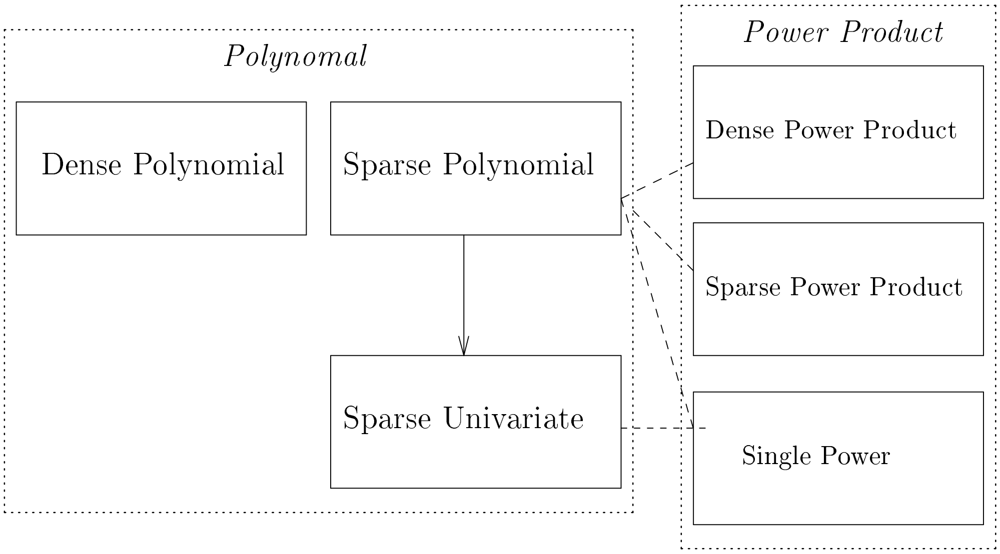
The dotted arrow within the interface box means that the interface simply calls the parser within its code and uses its result.

The communication between the interface and the library is implemented by saving the postfix expression into a file which is then read by the C++ library. The output is written in the standard output device. The interface reads the output of the C++ library through the pipe mechanism.

This implementation is good when the library is called only once because at each new call the library has to be restarted. When this is not the case a better mechanism should be used. For instance the communication could be done through MathLink ([Wol99]) or through a different mechanism in which the library is always running in a stand-by mode as a *server* and whenever an expression is sent to it by a *client* (e. g. the Mathematica interface), it “wakes up” and processes the input.

6.3 Relations among Classes

The following diagram describes the relations among the classes of polynomials and power classes present in the library.



The class of *sparse polynomials* implements sparse polynomials where the class of power products to be used is a *template parameter*. The library provides already three different classes of power products: sparse, dense and single power.

The class of *univariate sparse polynomials* is derived form the class of *sparse polynomials* with single power as power product. It implements univariate sparse polynomials and provides a few additional operations like Karatsuba multiplication.

The class of *dense polynomials* implements dense univariate polynomials with Karatsuba multiplication, school and Newton division, and exponentiation by generating functions.

6.4 The Algorithms

Various algorithms for fast polynomial arithmetic that speed up product, exponentiation and division have been implement for sparse and dense polynomials.

6.4.1 Dense Polynomials

The implemented fast arithmetic for dense polynomials includes: Karatsuba multiplication, division by Newton method, exponentiation by generating functions, division by powers of polynomials by generating functions.

Karatsuba Multiplication

Karatsuba algorithm is based on a divide-and-conquer strategy. For more details we refer to [Win96], page 39 and [GvzG99], pages 210–215. Multiplying two polynomials $a(x)$ and $b(x)$ can be done by splitting each polynomial into its higher and lower part. For the sake of simplicity let us consider that both $a(x)$ and $b(x)$ have degree n and that n is even.

Then we can write:

$$\begin{aligned} a(x) &= a_0(x) + a_1(x)x^{n/2}, \\ b(x) &= b_0(x) + b_1(x)x^{n/2}, \end{aligned} \tag{6.1}$$

where a_0, a_1, b_0 and b_1 have degree less than or equal to $n/2$.

Multiplying $a(x)$ by $b(x)$ yields:

$$\begin{aligned} &a(x) \cdot b(x) = (a_0(x) + a_1(x)x^{n/2}) \cdot (b_0(x) + b_1(x)x^{n/2}) = \\ &= a_1(x)b_1(x)x^n + a_0(x)b_0(x) + (a_1(x)b_0(x) + a_0(x)b_1(x))x^{n/2}. \end{aligned} \tag{6.2}$$

We observe that

$$\begin{aligned} &(a_0(x) + a_1(x)) \cdot (b_0(x) + b_1(x)) = \\ &a_0(x)b_0(x) + a_0(x)b_1(x) + a_1(x)b_0(x) + a_1(x)b_1(x). \end{aligned} \tag{6.3}$$

Therefore

$$\begin{aligned} &a_0(x)b_1(x) + a_1(x)b_0(x) = \\ &(a_0(x) + a_1(x)) \cdot (b_0(x) + b_1(x)) - a_0(x)b_0(x) - a_1(x)b_1(x). \end{aligned} \tag{6.4}$$

So (6.4) allows us to simplify (6.2) as follows:

$$\begin{aligned} &a(x) \cdot b(x) = \\ &a_1(x)b_1(x)x^n + a_0(x)b_0(x) + \\ &(a_0(x) + a_1(x)) \cdot (b_0(x) + b_1(x)) - a_0(x)b_0(x) - a_1(x)b_1(x). \end{aligned} \tag{6.5}$$

By applying (6.5) instead of (6.2) we perform only 3 multiplications over polynomials of degree $n/2$ instead of 4 multiplications.

Computing $a(x) \cdot b(x)$ by applying recursively (6.5) has complexity $\mathcal{O}(n^{\log_2 3})$, whereas the school algorithm has complexity $\mathcal{O}(n^2)$

Division by Newton Method

Division by Newton method (see [GvzG99], pages 243–249) is based on the idea of dividing $a(x)$ by $b(x)$ by first computing the inverse of the reversed of $b(x)$ (see Definition 5.2.1) modulo an appropriate power of x through a symbolic version of Newton iteration method, and then computing the quotient by a fast multiplication algorithm (e. g. Karatsuba multiplication).

When dividing $a(x)$ by $b(x)$ we want to find $q(x)$, $r(x)$ such that

$$a(x) = b(x)q(x) + r(x), \tag{6.6}$$

where $\deg r(x) < \deg b(x)$ or $r(x) = 0$.

Let us also assume:

$$\deg a(x) = n,\ \deg b(x) = m.$$

Taking the n-reversed of $a(x)$ (see Definition 5.2.1) yields

$$\begin{aligned} \operatorname{rev}_n a(x) &= \operatorname{rev}_m b(x) \operatorname{rev}_{n-m} q(x) + \operatorname{rev}_n r(x) = \\ &= \operatorname{rev}_m b(x) \operatorname{rev}_{n-m} q(x) + x^{n-m+1} \operatorname{rev}_{m-1} r(x). \end{aligned} \tag{6.7}$$

Thus we have the following congruence

$$\operatorname{rev}_n a(x) \equiv \operatorname{rev}_m b(x) \operatorname{rev}_{n-m} q(x) \mod x^{n-m+1}, \tag{6.8}$$

and also

$$\operatorname{rev}_{n-m} q(x) = \operatorname{rev}_n a(x)(\operatorname{rev}_m b(x))^{-1} \mod x^{n-m+1}. \tag{6.9}$$

Therefore once $(\operatorname{rev}_m b(x))^{-1}$ is computed, the quotient can be computed by the formula (6.9).

For the sake of simplicity let us assume that $b(0) = 1$.

Computing $b(x)^{-1}$ can be done by a symbolic version of Newton iteration for root finding. In this case the function to which we apply Newton iteration is:

$$\Phi(g) = 1/g - f. \tag{6.10}$$

The general Newton iteration formula is:

$$g_{i+1} = g_i - \frac{\Phi(g_i)}{\Phi'(g_i)}. \tag{6.11}$$

which for (6.10) becomes:

$$g_{i+1} = g_i - (1/g_i - f)(-g_i^2) = 2g_i - fg_i^2. \tag{6.12}$$

The following theorem gives a good initial point for the iteration and tells us that (6.12) quickly converges to the desired solution.

Theorem 6.4.1. *Given a commutative ring with unity R, $f, g_0, g_1, \dots \in R[x]$, with $f(0) = 1$, $g_0 = 1$ and $g_{i+1} = 2g_i - fg_i^2$ for all i. Then*

$$fg_i \equiv 1 \mod x^{2^i} \text{ for all } i \geq 0. \tag{6.13}$$

PROOF. Let us proceed by induction on i.

For $i = 0$ we have

$$fg_0 \equiv f(0)g_0 \equiv 1 \mod x^{2^0}. \tag{6.14}$$

Let us assume that (6.13) holds for a given positive integer j and prove it for $j + 1$.

We see that

$$\begin{aligned} 1 - fg_{i+1} &\equiv 1 - f(2g_i - fg_i^2) \equiv 1 - 2fg_i + f^2g_i^2 \equiv \\ &\equiv (1 - fg_i)^2 \equiv 0 \mod x^{2^{i+1}}. \end{aligned} \tag{6.15}$$

□

Therefore g_i converges to the correct solution in a logarithmic number of steps. The overall complexity of the Newton method for division is $\mathcal{O}(M(n))$, where $M(n)$ is the complexity of multiplying two polynomials of degree n. For more details on the complexity analysis we refer to [GvzG99], pages 243–263.

Exponentiation by Generating Functions

Exponentiating a polynomial by generating functions is done by considering the power of polynomial as a formal power series (generating function). Such power series satisfies a certain differential equation (5.58) from which a recurrence (5.57) satisfied by its coefficients can be obtained. For the details we refer to the previous chapter and to [GCL92], pages 114-116.

Fast Division by Powers of Polynomials

When diving by possibly high powers of polynomials of low degree polynomials, we can follow the same strategy as for division by Newton method, where the inversion is done by exponentiation through generating functions. For more details we refer to the previous chapter and to [GCL92], pages 114-116.

Repeated Squaring for Fast Exponentiation

Exponentiation by repeated squaring (see [GvzG99], pages 69–70) of a polynomial $p(x)$ by a power n, with $n \in \mathbb{N}$ is based on the simple recursive formula:

$$p(x)^n = \begin{cases} p(x), & \text{if } n = 1, \\ p(x)^{n/2} \cdot p(x)^{n/2}, & \text{if } n \text{ is even, } n \neq 1, \\ p(x)^{(n-1)/2} \cdot p(x)^{(n-1)/2} \cdot p(x), & \text{if } n \text{ is odd, } n \neq 1. \end{cases} \tag{6.16}$$

This procedure performs $\lfloor \log_2 n \rfloor$ squarings and $H(n) - 1 \leq \lfloor \log_2 n \rfloor$ multiplications, where $H(n)$ is the *Hamming weight* of the binary representation of n, i. e. the number of 1s in it. For more details we refer to [GvzG99], pages 69–70.

6.4.2 Sparse Polynomials

Fast arithmetic for polynomials in sparse representation includes: Karatsuba multiplication (it pays off if the involved sparsely represented polynomials are not very sparse), multiplication by geobuckects, repeated squaring for fast exponentiation.

Karatsuba Multiplication

Karatsuba multiplication can also be used for polynomials in sparse representation but its advantage over the school algorithm degrades quickly when many coefficients are zero. For more details we refer to Section 6.4.1 and also to [Win96], page 39 and [GvzG99], pages 210–215.

Repeated Squaring for Fast Exponentiation

Repeated squaring for exponentiation of a polynomial is also implemented for sparse polynomials. For more details we refer to Section 6.4.1 and also to [GvzG99], pages 69–70.

Multiplication by Geobuckets

Multiplication of sparse possibly multivariate polynomials can be drastically sped up by using the geobucket data-structure to save intermediate sums. The original paper where geobuckets were introduced is [Yan98].

The geobucket data-structure speeds up the addition of many polynomials. When multiplying two polynomials we have to sum up many polynomials to get the result. Adding up many polynomials by simple insertion into an ordered list representing the partial sum ends up almost always adding small polynomials to a big partial sum. This has a bad worst case complexity corresponding to the case when the entire list containing the partial sum has to be visited to append the terms of the new polynomial at the end.

The geobucket data-structure avoids this problem by saving the partial sum into a vector of lists of polynomials B where the i-th cell can only contain 4^i monomials. Each cell is implemented as an ordered list of pairs containing a power product and the corresponding coefficient. The addition of a polynomial f to the geobucket B is described in Algorithm 6.1. A new polynomial f with l monomials is added into the cell at position $\lfloor \log_4(l) \rfloor$, i.e. the lowest cell that could contain the polynomial. If the maximum number of monomials in the cell is exceeded, we have an *overflow*, the cell is zeroed and its content is added to the next cell until a cell is reached where the maximum number of allowed monomials is not exceeded.

We remark that the geobucket does not represent a polynomial in a unique way, and that monomials in different cells are not automatically canceled.

Algorithm 6.1 Addition of a Single Polynomial to a Geobucket

Input: A polynomial f.

Output: The polynomial f is added into B, i. e. $B := B + f$.

```
i := max(1, ⌈log_4(#f)⌉); // # means number of power products
if i ≤ m then
    f := f + B[i];
    while i ≤ m and #f > 4^i do
        f := f + B[i + 1];
        B[i] := 0;
        i := i + 1;
m := max(m, i);
B[i] := f.
```

By adding a polynomial to a geobucket, we only add polynomials of similar size and the maximum number of overflows for an addition is bounded by a logarithm of the size of the polynomial.

In order to have the final result the geobucket needs to be *canonicalized*, i. e. the content of all the cells must be added up.

Adding n polynomials $f_1, f_2, \ldots, f_n$ by a geobucket such that the total number of monomials of $f_1, f_2, \ldots, f_n$ is N has complexity $\mathcal{O}(Nlog(N))$, whereas the school algorithm has complexity $\mathcal{O}(N^2)$.

Example 6.4.1.
The geobucket $B = [B[1], B[2], B[3]]$ with

$$\begin{aligned} B[1] &= \boxed{-5x^7y} + \boxed{-13x^5} \\ B[2] &= \boxed{y^{24}} + \boxed{3x^4y^6} + \boxed{5x^7y} + \boxed{17x^5} + \boxed{-2x^2} \\ B[3] &= \boxed{-4x^5} + \boxed{x^2} \end{aligned} \tag{6.17}$$

has "internal cancellations" and represents $y^2 4 + 3x^4y^6 - x^2 + 4$. In fact the "canonicalization" gives

$$\begin{aligned} &B[1] + B[2] + B[3] = \\ &= (-5x^7y - 13x^5) + (y^{24} + 3x^4y^{+}5x^7y + 17x^5 - 2x^2) + (-4x^5 + x^2) = \\ &= y^2 4 + 3x^4y^6 - x^2. \end{aligned} \tag{6.18}$$

6.5 The Mathematica Component

The Mathematica interface consists of the following components:

- The Mathematica package `pPostify.m`,

- The Mathematica notebook `expand.nb`,
- The executable program `expand.out`,
- The executable program `power.out` (only for dense polynomials).

These files are available on the world wide web at the combinatorics home page of RISC (University of Linz, Austria) at the following web address

http://www.risc.uni-linz.ac.at/research/combinat/

6.5.1 Loading

In order to use the Mathematica interface `expand.out` and `pPostify.m` must be in the same directory as `expand.nb`.

To load the package it is enough to load and evaluate the Mathematica notebook `expand.nb`, which will also take care of loading the package `pPostify.m`.

6.5.2 The Parser

The package `pPostify.m` contains the parser which provides the functions necessary to convert polynomial expressions into a special postfix format that can be interpreted by the C++ library.

The package exists in various versions that only differ by the kind of polynomials that are treated (dense univariate, sparse univariate, sparse multivariate) and the type of power products (sparse, dense).

The following types of polynomials are so far supported

- dense univariate
- sparse univariate
- sparse multivariate with sparse power products
- sparse multivariate with dense power products

Support for additional types of polynomials (e. g. dense multivariate, recursively represented, etc.) can be easily added without having to rewrite the parser. It suffices to rewrite the function `WritePoly` of the parser (contained in `pPostify.m`) that translates a polynomial in the Mathematica syntax into its internal representation (for more details see the section on the polynomial representation).

The Postfix Format

The command `pPostify` outputs a text file into a stream ([Wol99]) containing a sequence of lines. Each line begins with a character (meaning either a polynomial or an operator) followed by some arguments, as in the following scheme:

Character	*Meaning*	*Arguments*
`X`	polynomial identifier	polynomial representation
`P`	sum	sum arity
`T`	product	product arity
`E`	exponentiation	exponent

Power Product Representation

The Mathematica interface provides the following types of representations of power products:

- univariate (single power)
- dense
- sparse

Powers are simply represented by their exponent.

Sparse power products are represented as a sequence starting with the number of powers followed by couples $(varId, power)$ where $varId$ is an integer that identifies an indeterminate and $power$ is the corresponding power.

Dense power products are represented as a sequence of powers corresponding to the powers of the indeterminates in the polynomial expression which is taken into consideration.

Example 6.5.1.
The power products $x^5y^7z^2$ and x^3z^6 are represented sparsely by the sequences

$$3\ \underbrace{1\ 5}_{x^5}\ \underbrace{2\ 7}_{y^7}\ \underbrace{3\ 2}_{z^2}, \qquad 2\ \underbrace{1\ 3}_{x^3}\ \underbrace{3\ 6}_{z^6},$$

respectively, and densely by the sequences

$$5\ 7\ 2, \qquad 3\ 0\ 6,$$

respectively.

Polynomial Representation

The following two kinds of polynomial representation are available:

- dense
- sparse

Dense polynomials are represented as a sequence starting with an integer indicating the number of power products followed by a sequence of coefficients.

Sparse polynomials are represented as a sequence beginning with an integer indicating the number of power products and then by a sequence of couples $(pp, coef)$, where pp is the representation of a power product and $coef$ is the corresponding coefficient.

Example 6.5.2. A univariate polynomial
The polynomial $3x^5 + 7x^2 + 2x + 1$ is sparsely represented by the sequence

$$4\ \underbrace{0\ 1}_{1}\ \underbrace{1\ 2}_{2x}\ \underbrace{2\ 7}_{7x^2}\ \underbrace{5\ 3}_{3x^5}$$

and it is densely represented by the sequence

$$6\ 1\ 2\ 7\ 0\ 0\ 3$$

Example 6.5.3. A multivariate sparse polynomial
The polynomial $3xz + 5y^2 + 1$ is represented as a sparse multivariate polynomial with sparsely represented power products by the sequence

$$3\ \underbrace{0\ 1}_{1}\ \underbrace{1\ 2\ 2\ 5}_{5y^2}\ \underbrace{2\ 1\ 1\ 3\ 1\ 3}_{3xz}$$

and it is represented as a sparse multivariate polynomial with densely represented power products by the sequence

$$3\underbrace{0\ 0\ 0\ 1}_{1}\ \underbrace{0\ 2\ 0\ 5}_{5y^2}\ \underbrace{1\ 0\ 1\ 3}_{3xz}$$

The Postfix Translator Command

The command `pPostify` is contained in the Mathematica package `pPostify.m`.
Command
`pPostify[`*`strm, expr`*`]`

Parameters

- *strm* is the *stream* where we want `pPostify` to write its output. It can also be `"stdout"` which is the *standard output*, i. e. the Mathematica window.
- *expr* is a Mathematica polynomial expression containing the operators +, *, ^ .

Semantics

`pPostify` transforms a polynomial into the postfix format that can be interpreted by the C++ library. Its output depends on the kind of polynomials that are treated.

The type of polynomial is given by loading the corresponding routines. This is so far implemented[1] by having more versions of the package `pPostify.m`.

Example 6.5.4. A univariate polynomial
The expression $(3x^5 + 2x^3 + 4x + 7)(x^3 + 2) + 5x + 2$ is converted by `pPostify` in the following way:

```
pPostify["stdout", (3x^5 + 2x^3 + 4x + 7)(x^3 + 2) + 5x + 2]
```

(sparse version)
Output =

```
X  2  0  2   1 5
X  2  0  2   3 1
X  4  0  7   1 4   3 2   5 3
T 2
P 2
```

(dense version)
Output =

```
X 2 2 5
X 4 2 0 0 1
X 6 7 4 0 2 0 3
T 2
P 2
```

Example 6.5.5. A multivariate sparse polynomial
The polynomial expression $(3xz + 5y^2 + 1)(2y + x) + 7x$ is converted into postfix form by `pPostify` as follows

[1] This could also be done with a unified package and by passing the type of polynomials and the type of power products through an option or parameter to the command `pPostify`.

```
pPostify["stdout", (3 x z + 5 y^2 + 1)(2y + x) + 7x]
```

(sparsely represented power products)
Output =

```
X  1  1  1 1 7
X  2  1  1 1 1  1  2 1 2
X  3  0  1  1  2 2 5  2  1 1 3 1  3
T 2
P 2
```

(densely represented power products)
Output =

```
X  1  1 0 0  7
X  2  1 0 0  1  0 1 0  2
X  3  0 0 1  0 2 0  5  1 0 1  3
T 2
P 2
```

6.5.3 The Mathematica Commands

The Mathematica commands use the library to perform expansion of polynomial expressions and exponentiation of polynomials.

The Polynomial Expansion Command

The command `PolynomialExpand` is contained in `expand.nb`. It also requires the files `pPostify.m` and `expand.out`.

Command
`PolynomialExpand[`*expr*`]`

Parameter
expr is a Mathematica polynomial expression possibly containing the operators +, ∗ , ˆ .

Options

Name	Default Value
`Algorithm`	`expand.out`
`Verbose`	Off

Semantics
`PolynomialExpand[`*expr*`]` fully expands the polynomial expression *expr* by using `pPostify` and `expand.out`. It first invokes `pPostify` to transform the expression into the postfix format. Then it uses the C++ library by the executable

`expand.out` to expand the expression. The output from `expand.out` is read by `PolynomialExpand` from a temporary file.

Meaning of the options

- `Algorithm` is the name of the executable file that uses the C++ library to perform the polynomial expansion. Different versions of the executable, using for instance different coefficient rings, can exist.
- `Verbose` toggles the verbosity level.

Example 6.5.6.
The expression $((x+1)^{10}+(y-1)^3)+x^5z^3(x^3-1)$ can be expanded by the command `PolynomialExpand` as follows

```
PolynomialExpand[((x + 1)^10 + (y - 1)^3) + x^5 z^3 (x^3 - 1)]
```

Output =

$$\begin{aligned} &10x^1+45x^2+120x^3+210x^4+252x^5-x^5z^3+210x^6\\ &+120x^7+45x^8+x^8z^3+10x^9+x^{10}+3y^1-3y^2+y^3 \end{aligned} \tag{6.19}$$

The Exponentiation Command

The polynomial command `ModPower` is contained in the file `power.nb`. It also requires the files `pPostify.m` and `power.out`.

Command

`ModPower[`*`poly, power, thr`*`]`

Parameter

- *poly* is a polynomial (in the Mathematica syntax).
- *power* is the power to which we want to exponentiate *poly*.
- *thr* is the exponent of the modulo x^{thr} with respect to which we compute the power (x is the indeterminate).

Options

Name	Default Value
`Algorithm`	`power.out`
`Verbose`	Off

Semantics

`ModPower[`***`poly,power,thr`***`]` computes

$$poly^{power} \mod x^{thr}, \tag{6.20}$$

by calling `power.out` which reads the polynomial through a file. `ModPower` uses the command `WritePoly` (contained in `pPostify.m`) to translate *poly* into its internal representation. The so far implemented version assumes that *poly* has constant coefficient equal to one. The output is then read by `ModPower` through the pipe mechanism.

Meaning of the options

- `Algorithm` is the name of the executable file that uses the C++ library to perform the polynomial exponentiation. Different versions of the executable, using for instance different coefficient rings, can exist.
- `Verbose` toggles the verbosity level.

Example 6.5.7.
We can compute the modular inverse of $(x+1)$ modulo x^{10} by `ModPower` as follows

```
ModPower[(x + 1), -1, 10]
```

Output =

$$1 - x + x^2 - x^3 + x^4 - x^5 + x^6 - x^7 + x^8 - x^9 \tag{6.21}$$

Example 6.5.8.
We can compute $(2x^3 - 3x + 1)^{12}$ modulo x^8 by `ModPower` as follows

```
ModPower[(2x^3 - 3x + 1), 12, 8]
```

Output =

$$1 - 36x + 594x^2 - 5916x^3 + 39303x^4 - 180576x^5 + 566940x^6 - 1098504x^7 \tag{6.22}$$

6.6 The C++ Library

The C++ library for polynomial arithmetic consists of the following parts:

- dense polynomial library
- sparse polynomial library

The C++ libraries are available on the world wide web at the combinatorics home page of RISC (University of Linz, Austria) at the following web address

http://www.risc.uni-linz.ac.at/research/combinat/

6.6.1 Sparse Polynomial Library

The sparse polynomial library is contained in the following files (followed by the extensions `.h` and `.cxx`):

File name	Description
`log4`	logarithm base four
`classLong`	class for long integers
`uniPowers`	class of powers
`powerProducts`	class of sparse power products
`densePowerProducts`	class of dense power products
`polynomialRing`	sparse polynomial ring class
`uniPolynomials`	univariate sparse polynomial ring class

The additional file `expand.cxx` is required to generate `expand.out`, which is required by the Mathematica interface (`expand.nb`).

The Power Products Classes

The implemented power products types are contained in the following template classes

Type of power product	File	Template Parameters
`uniPowers`	single powers	*expType*
`powerProducts`	sparse power products	*varType, expType*
`densePowerProducts`	dense power products	*nVars, expType*

whose parameters are

Parameter	Type of parameter	Description
expType	*class*	Type of the exponents of the powers
varType	*class*	Type describing the indeterminates
nVars	*int*	Number of variables

A power product implementation in order to be used as a parameter of `polynomialRing` must provide a constructor with no arguments, the overloaded operators for product, product "in place", comparison, input, output and the method `one()` to compute the unit power product.

The Coefficients

The coefficient ring is not implemented in the library but has to be provided as a parameter to the polynomial template classes.

A coefficient ring implementation must at least provide a constructor with no arguments, all the overloaded operators for sum, sum "in place", difference, prefix and postfix increment, prefix and postfix decrement, product, product in "place", comparison, assignment input, output and the method `zero()` to compute the zero element.

The template class `classLong` (in `classLong.h` and `classLong.cxx`) adds the method `zero()` to the C++ predefined types (`int`, `float`, `double`, etc.) and to the GMP classes ([GNU]) `mpz_class` and `mpr_class`.

The library has so far been tested with the following built-in coefficient rings:

- `int`
- `long long`
- `float`
- `double`
- `long double`

and with the following additional implementations of coefficient rings:

- `mpz_class` (GMP[2] implementation of arbitrary length integers)
- `mpr_class` (gmp implementation of rationals)
- `NTL::ZZ` (NTL[3] implementation of arbitrary length integers)
- user defined template class for $\mathbb{Z}_p$, with p prime

The `polynomialRing` Class

Sparse polynomials are implemented in the template class `polynomialRing` which take the following parameters

Parameter	Description
ppType	Power Product Class
coeRing	Coefficient Ring Class

The class `polynomialRing` implements

- binary operators between polynomials for sum and product,
- unary operators $+=$, $*=$ for "in place" sum and product,
- the exponentiation operator ^ ,
- the method

[2] For an on line documentation we refer to [GNU].
[3] For an on line documentation we refer to [Sho].

```
void
  gbMult(polynomialRing<ppType,coeRing> & poly)
```

which multiplies the polynomial "in place" by *poly* using a geobucket,

- the method

```
void
  expandFromFile(istream & sor)
```

which expands "in place" a polynomial expression from the stream *sor*,

- the operators <<, >> for input and output from and to *streams*.

A text file with name `varNames` has to be present in the same directory as `expand.out`. Such file must contain an integer representing the maximum number of indeterminates followed by the names of the indeterminates used for the output format. When the library is used through the Mathematica interface the appropriate file `varNames` is generated automatically.

The `uniPolynomial` Class

Sparse univariate polynomials are implemented in `uniPolynomial`, which is a derived class of `polynomialRing`, namely in the C++ syntax:

```
template<class degType, class coeRing>
class uniPolynomial :
   public polynomialRing<uniPower<degType>,coeRing>;
```

The class `uniPolynomial` takes the following parameters

Parameter	Description
degType	Type representing the exponents
coeRing	Class of the coefficient ring

It inherits all the methods of `polynomialRing` and implements

- the friend function

```
uniPolynomial<degType,coeRing>
  Karatsuba(const uniPolynomial<degType,coeRing> & lhs,
            const uniPolynomial<degType,coeRing> & rhs)
```

which multiplies two polynomials by Karatsuba's algorithm,

- the method

```
uniPolynomial<degType,coeRing>
  shift(const degType d)
```

which multiplies a polynomial in x by the power x^d,

- the method

```
uniPolynomial<degType,coeRing>
  modulo(const degType m)
```

which computes a polynomial in x modulo x^m,

- the method

```
uniPolynomial<degType,coeRing>
  reversed(const degType d)
```

which computes the d-reversed polynomial.

A text file with name `varName` has to be present in the same directory as `expand.out`. Such file must contain the name of the indeterminate used for the output format. When the library is used through the Mathematica interface the appropriate file `varName` is generated automatically

6.6.2 Dense Polynomial Library

The dense polynomial library is implemented in `uniPolynomial` that is a template class contained in the following files (followed by the extensions `.h` and `.cxx`):

File name	Description
`log2`	logarithm in base two
`classLong`	class for long integers
`uniPolynomials`	class of univariate dense polynomials

The executables `expand.out` and `power.out` that are used with the Mathematica interface are produced by the files

File name	Description
`expand.cxx`	expand input from Mathematica
`power.cxx`	exponentiates input from Mathematica

The dense polynomial class implements

- binary operators between polynomials for sum and product;

- unary operators $+=$, $*=$ for "in place" sum and product;
- the exponentiation operator ˆ ;
- the friend function

  ```
  uniPolynomial<coeRing>
    Karatsuba(const uniPolynomial<coeRing> & lhs,
              const uniPolynomial<coeRing> & rhs)
  ```

  which performs multiplication by Karatsuba algorithm;
- the method

  ```
  pair<uniPolynomial<coeRing>,uniPolynomial<coeRing> >
    divide(const uniPolynomial<coeRing> & lhs,
           const uniPolynomial<coeRing> & rhs)
  ```

  which performs division by the school algorithm;
- the method

  ```
  pair<uniPolynomial<coeRing>,uniPolynomial<coeRing> >
    Newton(const uniPolynomial<coeRing> & lhs,
           const uniPolynomial<coeRing> & rhs)
  ```

  which divides *lhs* by *rhs* using Newton division;
- the method

  ```
  pair<uniPolynomial<coeRing>,uniPolynomial<coeRing> >
    GFdivide(const uniPolynomial<coeRing> & lhs,
             const uniPolynomial<coeRing> & rhs,
             const long p)
  ```

  which divides *lhs* by rhs^p by generating function driven division;
- the method

  ```
  void
    modPower(const uniPolynomial<coeRing> & base,
             const long p,
             const long m)
  ```

  which computes the p-th power of *base* modulo x^m.

- the method

  ```
  void
    expandFromFile( istream & sor)
  ```

  expands "in place" a polynomial expression from the stream *sor*;

- the operators <<, >> for input and output from and to *streams*.

6.7 A Small Program

The following program reads two sparsely represented polynomials with sparsely represented power products from the files `polyA.pol` and `polyB.pol`, prints on the screen the two polynomials, their sum, their product (computed by the school algorithm), their squares, then it computes the product by geobucket multiplication, it stores it into the first polynomial and prints the result.

The coefficient ring `coeR` is provided by the template class `mpz_class` of the GMP library [GNU] for arbitrary length integers and rationals (by including `gmp.h` and `gmpxx.h`). In order to make `mpz_class` a suitable parameter for `polynomialRing` it is wrapped in the template class `classLong` which gives the extra methods required by the template parameter of `polynomialRing`.

```
#include <gmp.h>
#include <gmpxx.h>
#include <iostream>
#include <fstream>
#include "classLong.h"
#include "polynomialRing.h"

typedef classLong<mpz_class> coeR;
typedef long degType;
typedef short varType;

void main()
{
polynomialRing<powerProduct<varType,classLong<degType> >,coeR> a;
polynomialRing<powerProduct<varType,classLong<degType> >,coeR> b;

ifstream r1Stream("polyA.pol");
r1Stream >> a;
r1Stream.close();
ifstream r2Stream("polyB.pol");
r2Stream >> b;
```

```
r2Stream.close();

cout << "a : " << a << endl;
cout << "b : " << b << endl;
cout << "a + b = "  << a+b << endl;
cout << "a * b = " << a * b << endl;
cout << "a^2 = " << (a^2) << endl;
cout << "b^2 = " << (b^2) << endl;
a.gbMult(b);
cout << "(using Geobuckets) a := a * b" << endl;
cout << "a     = " << a << endl;
};
```

When `polyA.pol` contains

```
2  1  1 1 1  1  2 1 1
```

which represents $x^1 + y^1$, and `polyB.pol` contains

```
2  1  2 2 1  2  1 1 3 1  1
```

which represents $y^2 + x^1 z^1$, the program will output

```
a : (1)x^1 +(1)y^1
b : (1)y^2 +(1)x^1 z^1
a + b = (1)x^1 +(1)y^1 +(1)y^2 +(1)x^1 z^1
a * b = (1)x^1 y^2 +(1)y^3 +(1)x^2 z^1 +(1)x^1 y^1 z^1
a^2 = (1)x^2 +(2)x^1 y^1 +(1)y^2
b^2 = (1)y^4 +(2)x^1 y^2 z^1 +(1)x^2 z^2
(using Geobuckets) a := a * b
a     = (1)x^1 y^2 +(1)y^3 +(1)x^2 z^1 +(1)x^1 y^1 z^1
```

6.8 Benchmarks

Here we present some benchmarks that compare the performance of the Mathematica built-functions for polynomial expansion and polynomial exponentiation with *PolyComb* (the C++ library run through the Mathematica interfaces) for polynomial expansion (to be found in the file `expand.nb`) and the for polynomial exponentiation (to be found in the file `power.nb`).

6.8.1 Measurements

We present some timings of the Mathematica built-in functions versus our library used through its Mathematica interface for polynomial expansion.

These timings measure the overall time spent by the library when used through the Mathematica interface. The overhead produced by the parsing of the expressions and by the data interchange is also included in the timings.

The coefficient arithmetic is provided by the class `mpz_class` of the GMP library [GNU] for arbitrary length arithmetic.

Modality of Measurement

All the tests have been run on a Pentium IV 1,5 Megahertz equipped with 512 Megabytes of memory and under the Linux operating system. For the Mathematica code of the tests we refer to Appendix B. The timings are measured only in seconds and are not to be considered very accurate since slight variations exist in different executions of the tests.

6.8.2 Polynomial Multiplication and Expansion

Dense Polynomials

Here we present some timings of the Mathematica built-in functions versus our library used through the Mathematica interface for polynomial expansion using Karatsuba multiplication and using school multiplication.

degree	Mathematica	Karatsuba Mult.	School Mult.
1000	10	1	1
2000	43	3	4
3000	97	5	7
4000	167	6	12

For more details see Appendix B.

Sparse Univariate Polynomials

Here we present some timings of the Mathematica built-in functions compared with the timings of our library when used through its Mathematica interface for polynomial expansion using geobucket multiplication and using school multiplication.

The test measures in seconds the time spent for computing the product of two univariate polynomials with exactly $t = m \cdot 100$ monomials, for $m = 2, 3, 4, 5, 6, 7, 8, 9$.

Each coefficient is in the range between 0 and 9 and has degree in the range between 1 and $m \cdot 10^7$. We refer to the Appendix B for more details on how the tests were executed.

power products	Mathematica	Geobucktes	School Algorithm
200	2	2	1
300	7	3	4
400	17	5	7
500	33	8	12
600	56	12	19
700	88	17	28
800	133	22	39
900	192	29	54
1000	264	35	72

For more details see Appendix B.

Sparse Multivariate Polynomials

Here we present some timings of the Mathematica built-in functions versus the polynomial library used through the Mathematica interface for polynomial expansion in a version using both geobucket multiplication and in a version using school multiplication.

The test measures in seconds the time spent for expanding an expression of the form $p_1 \cdot p_2 + (-p_1 + 1) \cdot p_2$ where p_1 and p_2 are randomly generated polynomial in $m = 4, 8$ indeterminates with exactly $t = 300, 400, 500, 600, 700$ power products.

Each coefficient is in the range between 0 and 9 and each power product is a product of the m indeterminates whose exponent has a 90% probability of being 0 and a 10% probability of being a value in the range between 1 and 99. We refer to the Appendix B for more details on how the tests were executed.

Case $m = 4$

power products	Mathematica	Geobucktes	School Algorithm
300	23.5	4.0	6.0
400	54.0	7.5	14.0
500	102.5	11.5	26.5
600	177.0	19.0	46.0
700	275.0	27.0	72.5
800	438.0	36.0	113.0
900	587.0	51.0	166.0
1000	636.0	62.0	232.0

Case $m = 8$

power products	Mathematica	Geobuckets	School Algorithm
300	27.0	4.5	7.5
400	62.5	9.0	16.0
500	119.5	13.5	31.0
600	200.5	21.0	57.0
700	327.5	27.5	91.5
800	348.0	38.0	140.0
900	506.0	47.0	204.5
1000	707.0	61.0	335.0

For more details see Appendix B.

6.8.3 Polynomial Exponentiation

Here we present some timings of the Mathematica built-in functions versus the Mathematica interface for polynomial exponentiation using the generating function method and repeated squaring.

The tests measure in seconds the time spent for exponentiating to the power $p = 50, 100, 150, 200, 250$ randomly generated dense polynomials with degree $d = 10, 20, 40$ and coefficients in the range $\{0, 99\}$ by

- Built-in Mathematica `Expand` command
- Mathematica interface using exponentiation by generating functions
- Mathematica interface using repeated squaring

Case $d = 10$

p (power)	Mathematica	Generating Functions	Repeated Squaring
50	1	<1	<1
100	6	<1	1
150	15	1	3
200	27	1	6
250	44	2	13

Case $d = 20$

p (power)	Mathematica	Generating Functions	Repeated Squaring
50	5	<1	1
100	23	1	3
150	54	1	11
200	106	2	28
250	212	4	58

Case $d = 40$

p (power)	Mathematica	Generating Functions	Repeated Squaring
50	22	<1	2
100	92	1	13
150	214	3	46
200	382	5	109
250	781	8	238

For more details we refer to Appendix B.

6.9 Conclusion

The library so far provides only basic arithmetic operations for polynomials in distributed representation. The next step could be to extend the library into two directions:

- providing direct support for polynomials in recursive representation
- implementing more operations
- improving efficiency of the overloaded operators

The library allows having polynomials as coefficients of polynomials but no operations tailored for such representation, like conversions between different representations, have been implemented, yet.

The library could also be extended with more operations like multivariate greatest common divisor.

Overloaded operators such as `+` and `*` are so far implemented in a straightforward way and do not delay evaluation to improve efficiency. For instance, the command:

$$a = b + c \tag{6.23}$$

would compute the sum of b and c and write the result in a temporary location and then it would copy it into a.

Delaying the addition until an assignment operator is being evaluated would solve the problem. This can be done by having the operators output an unevaluated result in a special data structure. The assignment would then perform the operation (or operations).

Appendix A

Chinese Remainder Problem

The Chinese remainder algorithm solves a system of congruences of the form

$$\begin{aligned} x &\equiv r_1 \mod m_1 \\ &\cdots \\ x &\equiv r_t \mod m_t \end{aligned} \tag{6.24}$$

where the r_i and m_i, for $1 \leq i \leq t$, are elements of a ring and the m_i are pairwise relatively prime.

We refer to the r_i as *remainders* and to the m_i as *moduli*.

Lagrange Interpolation

Lagrange Interpolation is based on the *Lagrange interpolating polynomial*, i. e. the polynomial $P(x) \in F[x]$, with F field, of degree $n-1$ which passes through the points

$$\begin{aligned} y_1 &= P(x_1), \\ &\cdots \\ y_n &= P(x_n), \end{aligned} \tag{6.25}$$

which is given by

$$P(x) = \sum_{j=1}^{n} P_j(x), \tag{6.26}$$

with

$$P_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} y_j. \tag{6.27}$$

Newton Interpolation

Newton Interpolation uses the Chinese remainder algorithm to interpolate a polynomial where the considered ring is the ring of polynomials over a field. We consider the Chinese remainder problem where the remainders $r_i = f(p_i)$ are values of the unknown function f at p_i and the moduli are linear functions of the form $x - p_i$, where x is the indeterminate:

$$\begin{aligned} x &\equiv r_1 \ \ (x - p_1) \\ &\dots \\ x &\equiv r_t \ \ (x - p_t) \end{aligned} \tag{6.28}$$

Remark: Newton interpolation is superior to Lagrange interpolation in that it can be used in an incremental fashion.

Appendix B

Here we give some details on how the timings of the library for polynomial arithmetic were obtained.

Multiplication of Dense Univariate Polynomials

For testing multiplication between two dense univariate polynomials we have used the following Mathematica code:

```
rC := Random[Integer,{0,10^10-1}]
rP[deg_] := Sum[rC x^i,{i,0,deg}]

p1 = rP[...];
p2 = rP[...];

initT = AbsoluteTime[];
r2 = PolynomialExpand[p1 p2];
Print["[",
      Max[N[AbsoluteTime[]-initT-1,1],0],",",
          N[AbsoluteTime[]-initT+1,1],"]"];

initT = AbsoluteTime[];
r1 = Expand[p1 p2];
Print["[",
      Max[N[AbsoluteTime[]-initT-1,1],0],",",
          N[AbsoluteTime[]-initT+1,1],"]"];
```

Each coefficient is in the integer range $[0, 10^{10} - 1]$.

Multiplication of Sparse Univariate Polynomials

For testing multiplication between two sparse univariate polynomials we have used:

```
rc:=Random[Integer,{1,10-1}];
dr[i_]:=Random[Integer,{1,i}];
rP[range_,size_] := Module[{res},
    res = 0;
    While[Length[res] != size,
      res = res + rc y^dr[range]
      ];
    Return[res]
```

```
    ];

m = ...;
p1 = rP[m * 10^7,m * 100];
p2 = rP[m * 10^7,m * 100];

initT = AbsoluteTime[];
r1 = Expand[p1 p2];
Print[AbsoluteTime[]-initT]

initT = AbsoluteTime[];
r2 = PolynomialExpand[p1 p2];
Print[AbsoluteTime[]-initT]
```

Each coefficient is in the range between 0 and 9 and has degree in the range between 1 and $m \cdot 10^7$.

Expansion of Sparse Multivariate Polynomials

We have tested multiplication and addition with the following Mathematica code:

```
sparsity=9;
rc:=Random[Integer,{1,10-1}];
dr[i_]:=If[Random[Integer,{0,9}]<sparsity,
           0,
           Random[Integer,{1,10^5-1}]];
var[i_]:=ToExpression[StringJoin["x",ToString[i]]];
rT[nVars_]:=Product[var[i]^dr[i],{i,1,nVars}];
rP[nVars_,size_] := Module[{res},
    res = 0;
    While[Length[res] != size,
      temp =rc rT[nVars];
      If[ !IntegerQ[temp],
        res = res + temp;
        ];
      ];
    Return[res]
    ];

m = ...;
p1 = rP[m,...];
p2 = rP[m,...];
p3 = (-p1+1);p4 = p2;

initT = AbsoluteTime[];
r1 = Expand[p1 p2+p3 p4];
Print["[",Max[N[AbsoluteTime[]-initT-1,1],0],",",
              N[AbsoluteTime[]-initT+1,1],"]"];

initT = AbsoluteTime[];
r2 = PolynomialExpand[p1 p2 + p3 p4];
Print["[",Max[N[AbsoluteTime[]-initT-1,1],0],",",
              N[AbsoluteTime[]-initT+1,1],"]"];
```

Each coefficient is in the integer range between 0 and 9 and each power product is a product of the m indeterminates whose exponent has 90% probability of being 0 and 10% probability of being a value in the integer range between 1 and 99.

This test was run twice and the average timings were considered.

Note: We notice that incrementing the number of variables *nVars* does not always make the problem of multiplying two polynomials generated by `rP[nVars, size]` harder.

Exponentiation of Dense Univariate Polynomials

For testing exponentiation we have used:

```
rC := Random[Integer,{0,10^2-1}]
rP[deg_] := Sum[rC x^i,{i,0,deg}]

deg = ...;

pAUX = rP[pDeg];
p = pAUX - Coefficient[pAUX,x,0]+1;
modulo = exponent*Exponent[p,x]+1;

initT = AbsoluteTime[];
res = ModPower[p,exponent,modulo];
AbsoluteTime[]-initT

initT = AbsoluteTime[];
res2 = Expand[p^exponent];
AbsoluteTime[]-initT
```

In this test each coefficient is in the integer range $[0, 99]$.

Curriculum Vitae

Personal Data

First Name:	Fabrizio
Family Name:	Caruso
Birthday:	19/09/1972
Address:	Via Del Popolo, 45/a, 95040, Licodia Eubea (CT), Italy

Studies

1986–1991	**High School** ("Maturità scientifica"), Liceo Scientifico, Caltagirone, Italy
1991–1996	**Master Degree** in Computer Science ("Laurea in Informatica"), University of Catania, Italy
1997–	**PhD Studies** in Symbolic Computation RISC, Johannes Kepler University, Linz, Austria

Grants

- SFB grant F1305 of the Austrian FWF
- Grant from the University of Catania for specialization in a foreign institution ("Borsa di studio per corsi di perfezionamento presso istituzioni estere")
- Grant no. 203.15.10 from the CNR (Italian national research council)

Bibliography

[Abr95] S.A. Abramov. Rational solutions of linear differential and difference equations with polynomial coefficients. In *Proc. ISSAC '95*. ACM Press, 1995.

[And94] G.E. Andrews. *The Theory of Partitions*. Cambridge University Press, 1994.

[APa] S.A. Abramov and M. Petkovšek. Minimal decomposition of indefinite hypergeometric sums. (corrected) preprint.

[APb] S.A. Abramov and M. Petkovšek. On the structure of multivariate hypergeometric terms. preprint.

[AP99] G.E. Andrews and P. Paule. MacMahon's Partition Analysis IV: Hypergeometric multisums. *Sém. Lothar. Combin.*, (B42i):1–24, 1999.

[APR01a] G.E. Andrews, P. Paule, and A. Riese. MacMahon's Partition Analysis III:The Omega Package. *European J. Combin.*, (22):887–904, 2001.

[APR01b] G.E. Andrews, P. Paule, and A. Riese. MacMahon's Partition Analysis IX: k-gon partitions. *Bull. Austral. Math. Soc.*, (64):321–329, 2001.

[APR01c] G.E. Andrews, P. Paule, and A. Riese. MacMahon's Partition Analysis VI: A new reduction algorithm. *Ann. Comb.*, (5):251–270, 2001.

[APR01d] G.E. Andrews, P. Paule, and A. Riese. MacMahon's Partition Analysis VII: Constrained compositions. In B.C. Berndt and K. Ono, editors, *q-Series with Applications to Combinatorics, Number Theory, and Physics*, volume 291 of *Contemp. Math.*, pages 11–27. Amer. Math. Soc., 2001.

[APR01e] G.E. Andrews, P. Paule, and A. Riese. MacMahon's Partition Analysis VIII: Plane partition diamonds. *Adv. in Appl. Math.*, (27):231–242, 2001.

[APRS] G.E. Andrews, P. Paule, A. Riese, and V. Strehl. MacMahon's Partition Analysis V: Bijections, recursions, and magic squares. In *Algebraic Combinatorics and Applications*.

[Car99] F. Caruso. A Macsyma Implementation of Zeilberger's Fast Algorithm. SFB-Report 99-16, RISC, J.K. University, Linz, Austria, 1999.

[Chy98] F. Chyzak. *Fonctions holonomes en calcul formel.* PhD thesis, INRIA (France), May 27 1998. Th'ese universitaire no. TU 0531.

[Cus89] T.W. Cusick. Recurrences for sums of powers of binomial coefficients. *J. Combin. Theory Ser. A*, 52(1):77–83, 1989.

[GCL92] O.K. Geddes, R.S. Czapor, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer, 1992.

[GKP94] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics - A Foundation for Computer Science.* Addition Wesley, second edition, 1994.

[GNU] GNU Project. *GMP on line documentation.* Available at http://www.swox.com/gmp/.

[Gos78] R.W. Gosper. Decision procedure for indefinite hypergeometric summation. In *Proceedings of the National Academy of Sciences of USA*, number 75, pages 40–42, 1978.

[GvzG99] J. Gerhard and J. von zur Gathen. *Modern Computer Algebra.* Cambridge University Press, 1999.

[Kar86] P.W. Karlsson. On two hypergeometric summation formulas conjectured by Gosper. *Simon Stevin*, 60(4):329–337, 1986.

[Lip81] J.D. Lipson. *Algebra and Algebraic Computing.* Addison-Wesley, Reading, Massachusetts, 1981.

[Mac16] P.A. MacMahon. *Combinatory Analysis*, volume 2. Cambridge University Press, 1915-1916. Reprinted: Chelsea, New York, 1960.

[Mac96] Macsyma, Inc. *Mathematics and System Reference Manual*, sixteenth edition, 1996.

[Pau95] P. Paule. Greatest factorial factorization and symbolic summation. *J. of Symbolic Computation*, (20):235–268, 1995.

[Pet92] M. Petkovšek. Hypergeometric solutions of linear recurrences with polynomial coefficients. *J. of Symbolic Computation*, (14):243–264, 1992.

[PR97] P. Paule and A. Riese. A Mathematica q-Analogue of Zeilberger's Algorithm Based on an Algebraically Motivated Approach to q-Hypergeometric Telescoping. *Special Functions, q-Series and Related Topics, Fields Institute Communications*, (14):179–210, 1997.

[PS95] P. Paule and M. Schorn. A Mathematica Version of Zeilberger's Algorithm for Proving Binomial Coefficient Identities. *J. of Symbolic Computation*, (20):673–698, 1995.

[PS00] C. Pau and W. Schreiner. Distributed Mathematica - User and Reference Manual. RISC Report 00-25, RISC, J.K. University, Linz, Austria, 2000.

[PWZ97] M. Petkovšek, H. Wilf, and D. Zeilberger. *A=B*. A K Peters, MA, 1997.

[Rie01] A. Riese. Fine-Tuning Zeilberger's Algorithm: The Methods of Automatic Filtering and Creative Substituting. In F.G. Garvan and M.E.H. Ismail, editors, *Symbolic Computation, Number Theory, Special Functions, Physics and Combinatoric*, volume 4 of *Developments in Mathematics*, pages 243–254. Kluwer, 2001.

[RZ] A. Riese and B. Zimmermann. Randomization Speeds up Hypergeometric Summation. Preprint.

[Sch] W.F. Schelter. *Maxima on line documentation.* Available at
http://www.ma.utexas.edu/maxima/index.html
http://maxima.sourceforge.net/referencemanual/maxima_toc.html.

[Sch95] M. Schorn. Contributions to Symbolic Summation, December 1995.

[Sho] V. Shoup. *NTL on line documentation.* Available at
http://www.shoup.net/ntl/.

[Sta97] R.P. Stanley. *Enumerative Combinatorics*, volume I. Cambridge University Press, 1997.

[Str94a] V. Strehl. Binomial identities – combinatorial and algorithmic aspects. *DMATH: Discrete Mathematics*, 136, 1994.

[Str94b] V. Strehl. Binomial identities - combinatorial and algorithmical aspects. *Discrete Mathematics*, (136):309–346, 1994.

[Win96] F. Winkler. *Polynomial Algorithms in Computer Algebra.* SpringerWienNewYork, 1996.

[Wol99] S. Wolfram. *The Mathematica Book*, fourth edition, 1999.

[Yan98] T. Yan. The Geobucket Data Structure for Polynomials. *J. Symbolic Computation*, 25(3):285–293, March 1998.

[Zei90] D. Zeilberger. A fast algorithm for proving terminating hypergeometric identities. *Discrete Mathematics*, (80):207–211, 1990.

[Zei91] D. Zeilberger. The method of creating telescoping. *J. of Symbolic Computation*, (11):195–204, 1991.

Index

1_i, 108
$A(H)$, 69
$B(H)$, 69
$C(H)$, 69
$C_b(i)$, 98
$C_i(M, b)$, 38
$Comp$, 104
$D(F(n, k), d)$, 91
$D(H)$, 69
$H(A)$, 44
Δ_k, 106
$\mathcal{Q}_k$, 80
$\mathcal{Q}_n$, 80
$\bar{D}(H)$, 75
$\beta_{d^p}(x)$, 114
$\delta(H)$, 76
$\delta_{i,j}$, 98
den, 80
dom, 70
eval_v, 57
eval_x, 45
$\hat{F}(n, k)$, 80
$\hat{Q}_k$, 87
$\hat{Q}_n$, 87
im, 70
λ_m, 39
$\mathcal{L}$, 80
mod_p, 40
μ_H, 74
$\nu(A)$, 57
num, 80
ϕ_H, 70, 74
ψ_H, 75
rev_d, 98
σ, 107
τ, 38
`‘‘?’’`, 60

Abramov’s method, 17
`Algorithm`, 60–63
algorithm
    addition to geobucket, 126
    division
        by generating functions, 122
        by powers of polynomials, 125
        Newton, 122, 123
        school, 100
    division by powers of polynomials
        by generating functions, 122
    exponentiation
        by generating functions, 122, 125
        by repeated squaring, 125
    extended Euclidean, 53–55, 57, 66
        row of, 53
    generalized extended Euclidean, 53–55, 57, 66
    Gosper, 15
    Gosper Form, 83
    multiplication
        by geobuckets, 126
        Karatsuba, 122, 123
    polynomial evaluation
        Horner, 32
    Zeilberger, 17
`AllD`, 51

back-substitution, 41, 43, 57, 99
bijection
    fundamental, 70
binomial theorem, 26
black list, 58

BlackList, 60, 61, 64
bubble sort, 111

C++ component, 120
canonicalization, 127
Cauchy interpolation, 54
Chinese remainder
  problem, 42, 147
  theorem, 37, 42, 47, 49, 51
ChNullSpaceAt, 49, 50
ChNullSpaceUntil, 50
client, 121
combination, 110
  without repetitions, 110
combinatorial proof, 109
CompleteNullSpace, 62, 64
composition, 104
  concatenation, 104
concatenation of a composition, 104
Cramer matrix, 38
Cramer's rule, 41
CramerNullSpace.m, 49

d-reversed polynomial, 98, 138
DegreeBound, 60, 61, 64
diagram relation, 107
  transitive and reflexive closure, 107
diophantine
  equations, 1
  inequalities, 1
DistChNullSpaceUntil, 50, 51
Distributed Mathematica, 50
divide, 139
divide-and-conquer strategy, 123
domain, 70

expandFromFile, 137, 140

falling factorial, 82
finite support, 17, 19, 115
forward difference operator, 106
function
  Kronecker delta, 98
  sign, 107

fundamental bijection, 70

Gaussian elimination, 38, 41, 43, 53, 93
gbMult, 137
gcd condition, 17, 32
generalized extended Euclidean algorithm, 53–55, 57, 66
generating function, 114
geobucket, 126
GFdivide, 139
GMP, 136, 140, 142
Gosper
  condition, 17, 84, 86, 89, 90
  equation, 17, 33, 84, 89, 91
  form, 20, 32, 69, 83, 84, 87–90
  polynomial, 84, 89, 91, 92
Gosper, 23
Gosper's algorithm, 15
Gosper-Petkovšek
  form, 84, 89
Gosper-summable, 25, 89, 90
GosperSum, 23
GosperSumVerboseOpt, 23
GosperVerbose, 23
GosperVerboseOpt, 23

Hadamard bound, 44, 46
Hamming weight, 125
heuristic method, 58
homomorphism
  evaluation, 45
  modular, 41
Horner's algorithm, 32
hybrid method, 58
HybridNullSpace, 63

identity
  first Karlsson-Gosper, 27
  second Karlsson-Gosper, 28
  Strehl, 29
  Vandermonde, 27
image, 70
in place

expand
of a dense polynomial, 140
of a sparse polynomial, 137
product
of coefficients, 136
of dense polynomials, 139
of power products, 135
of sparse polynomials, 136
sum
of coefficients, 135
of dense polynomials, 139
of sparse polynomials, 136
inclusion-exclusion, 110, 112
`InitializeD`, 51
integer congruence, 39, 40, 42, 147
interpolant
polynomial, 66
rational, 55
`InterpolatingRatFunction`, 66
interpolation
Cauchy, 55
delayed, 59
Lagrange, 46
Newton, 46
polynomial, 46
rational, 55

k-free recurrence, 15, 17
k-multiset, 110
k-permutation, 110
without repetition, 110
k-regular, 80
k-sequence, 110
without repetitions, 110
k-set, 110
`Karatsuba`, 137, 139
Karatsuba multiplication, 122, 123
Karlsson-Gosper
first identity, 27
second identity, 28
Kronecker delta function, 98

Lagrange
interpolating polynomial, 147
polynomial interpolation, 46, 147
largest square upper part, 37
lifting, 103
linear recurrence operator, 80
LISP, 20, 31
`LOADZeilberger.macsyma`, 22
lucky
evaluation, 57
primes, 41
rationals, 45

MacMahon, 1
Macsyma, 20, 31
Mathematica, 57, 59, 119
component, 127
interface, 120
parser, 120
MathLink, 121
matrix
$L(A)$, 38
$R(A)$, 38
mod_p, 40
Cramer, 38
triangular form of a, 38
`MAX_ORD`, 24
Maxima, 20, 31
maximal cardinality property, 78
method
heuristic, 58
hybrid, 58
recursive, 58
method (obj. or. progr.), 119
`ModPower`, 133
`modPower`, 139
modular inverse, 101, 116, 123, 134
`ModularAlgorithm`, 64
`ModularNullSpace`, 60
`modulo`, 138
`mpz_class`, 136
`MultiHybridNullSpace`, 63
`MultiModularNullSpace`, 60
Möbius function, 109

n-regular, 80
natural boundary, 17, 19
network, 50, 51
Newton
 division, 122
 iteration, 123, 124
  formula, 124
 polynomial interpolation, 46, 148
`Newton`, 139
non-Gosper-summable, 25
non-k-regular, 81
non-n-regular, 81
NTL, 136
`NTL::ZZ`, 136

`Offset`, 60, 64
Ω-calculus, 1
`one()`, 135
operator
 forward difference, 106
 linear recurrence, 80
 shift, 3, 80
overflow, 126

Padé
 approximant, 57
 approximation, 56
`parGosper`, 24
`parGosperVerboseOpt`, 24
parser, 120, 128
Paule-Schorn implementation, 21
Petkovšek condition, 84
PolyComb, 119
`PolynomialExpand`, 132
`polynomialRing`, 136
postfix form, 120, 121, 128–131
power product, 129, 135
`pPostify`, 130

quintuple, 32

rational certificate, 15
rational function approximation, 2, 53
`RatNullSpace`, 63
recursive method, 58
regular, 80
repeated squaring
 for polynomial exponentiation, 125
resultant, 17, 20
`reversed`, 138
reversed polynomial, 98, 138
rising factorial, 82

school
 division, 100, 139
 multiplication, 142, 143
server, 121
shift
 operator, 3, 80
 quotient, 14, 80
`shift`, 138
`StartingInterpolation`, 60, 64
stream, 129
Strehl identity, 29
summation
 hypergeometric
  definite, 17
  indefinite, 15
  q-case, 14
`SymbolicAlgorithm`, 64

telescoping, 2, 13
template, 119
term
 holonomic, 20
 hypergeometric, 14
  proper, 15, 80
  regular proper, 80
transcendental extension, 97
triangular form, 38
triangularization, 57
trinomial coefficients, 28

`uniPolynomial`, 137, 138

Vandermonde identity, 27
`Var`, 60, 61, 64

`Verbose`, 60–64

`WritePoly`, 128, 134

`Zeilberger`, 24
Zeilberger's algorithm, 17
`ZeilbergerVerboseOpt`, 24
`ZeilbergerVeryVerbose`, 23
`zero()`, 136