Dipartimento di Informatica e
Scienze dell'Informazione

The Design of the CoCoA 3 System

by

Antonio Capani

Theses Series **DISI-TH-2000-02**

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

The Design of the CoCoA 3 System

by

Antonio Capani

February, 2000

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35
I-16146 Genova, Italy
http://www.disi.unige.it/

Ph.D. Thesis in Computer Science

Submitted by Antonio Capani
DISI, Univ. di Genova
capani@disi.unige.it

Date of submission: February 2000

Title: The Design of the CoCoA 3 System

Supervisor: Lorenzo Robbiano
DIMA, Univ. di Genova, Italy robbiano@dima.unige.it

Ext. Reviewers:
Alessandro Logar
University of Trieste
logar@univ.trieste.it

Martin Kreuzer
University of Regensburg
martin.kreuzer@mathematik.uni-regensburg.de

Abstract

This thesis presents the design of CoCoA, a special purpose computer algebra system dedicated to the Commutative Algebra. After an introduction to the field of the Computer Algebra the system is presented from the architectural point of view. Two chapters expose the Gröbner Framework, an advanced tool for doing computations based on an Generalization of the Buchberger Algorithm, and an application of this tool to the computation of Minimal Free Resolutions. Then it is introduced EEE, an new advanced polynomial representation which exhibits both great compactness of space and good performance. A final chapter present a general approach for accessing computing systems in an interactive Web-based environment.

To Caterina, Paolo, and Samuele

Acknowledgements

For their support, collaboration and assistance with my thesis I would like to thank my advisor, Lorenzo Robbiano, my Phd Program Coordinator, Eugenio Moggi, and the reviewers Massimo Ferri, Alessandro Logar, Martin Kreuzer, and Rocco De Nicola.

I am thankful to TXT Ingegneria Informatica, in which I am currently employed, which gave to me all the necessary time to complete my PhD work.

Thanks to Paul Hearmon, for help with the english.

Finally a very special thanks to Caterina whose love fueled me during all these years.

Table of Contents

Introduction

This thesis presents the design of the architecture of CoCoA, a special purpose computer algebra system dedicated to the Commutative Algebra.

Contribution of the Thesis

The major contribution of the thesis consists of the presentation of the approach I have used in order to design and develop the CoCoA system. The approach consists of a combination of methodologies and techniques, used to satisfy the needs of the typical user of the system: a matematician with few or no background on programming.

In particular I have focused my contribution on the design of the architecture of the system, which has been privileged with respect to the mathematical research. The major contribution on the mathematical side has been the invention of EEE, an efficient representation for multivariate polynomials.

The strong points of the system architecture are:

- CoCoAL, the high level language for the user interaction;
- CTL, a low level language used to abstract the system behaviour from the *syntax sugar* of CoCoAL and for efficency reasons;
- the CDevices, which consist of an abstraction of the phisical devices, and are used to ensure a high portability to the system;
- the Gröbner Framework, an advanced technique designed to solve different Gröbner-related problems using a single, parametric Buchberger Algorithm;
- the Session Management Framework, a general approach for writing distributed web-based applications which interact with Computation Systems.

Summary

The thesis consists of six chapters organized as follows.

In Chapter 1 we present an introduction to the field of the Computer Algebra. After few historical notes we introduce the Symbolic Computation and its relation to the Computer Algebra. The chapter contains also an introduction to the Gröbner Bases.

In Chapter 2 we describe the CoCoA system both from the user point of view and from the architectural point of view. In particular it is presented CoCoAL, the high level language of the system, and the decomposition of the architecture in its main modules; the description of the architecture presents also how we solved the problem of the portability by means of the combination of *CTL*, a low level intermediate language and the *CDevices*, an abstraction of the phisical devices.

In Chapter 3 we present the Gröbner Framework, an advanced tool for doing computations based on an Generalization of the Buchberger Algorithm. In brief the framework consists of a parametrization of the Buchberger Algorithm with some parameter functions and the instantiation of the algorithm by using different implementation of these parameters.

In Chapter 4 we explain how we have used the Gröbner Framework for solving the problem of computing Minimal Finite Free Resolutions of homogeneous submodules of graded free modules over polynomial rings.

In Chapter 5 it is introduced EEE, an new advanced representation for polynomials which exhibits both great compactness of space and good performance. This new polynomial representation was inspired by the mathematical structure intrinsic in homogeneous problems, and consists of a parametrized family of representations allowing varying degrees of trade-off between space and time. Its compactness also makes EEE interesting as a protocol for communicating polynomials.

Finally the Chapter 6 contains the presentation of the Session Management Framework, which is a general approach for accessing computing systems in an interactive Web-based environment. SMF is based on the concept of Session which is a list of Queries. Each query is characterized by an Input and an Output. The user writes requests to query inputs and then *submits* the query to an agent, which put the answer in the output field of the query. The framework gives the necessary abstraction from the locality of the sessions and as such permits to write tools for interacting with a computing system from remote clients (e.g. Web browsers).

Chapter 1

Introduction to the Computer Algebra

Recently there has been the publication of the book "Commutative Algebra with a view toward algebraic geometry" [Eis95]. by David Eisenbud. Because of the importance of the author this book has already become a standard reference, in particular for the commutative algebraist and for the algebraic geometer. The book contains a chapter about the Gröbner bases and the related computational aspects of algebra. An eminent algebraic geometer, Marc Green, in the review of this book writes textually:

> *There is one change which has overtaken commutative algebra that is in my view revolutionary in character – the advent of symbolic computation. This is as yet an unfinished revolution. At present, many researchers routinely use Macaulay, Maple, Mathematica, and* CoCoA *to perform computer experiments, and as more people become adept at doing this, the list of theorems that have grown out of such experiments will enlarge. The next phase of this development, in which the questions that are considered interesting are influenced by computation and where these questions make contact with the real world, is just beginning to unfold. I suspect that ultimately there will be a sizable applied wing to commutative algebra, which now exists in embryonic form.*

1.1 Historical Notes

The birth and growth of algebra and of algorithms are stricly related. The origins of both disciplines go back to Muhammed ibn-Mūsa al-Khwarizmi al-Quturbulli, who was an important person at the court of the Calif Al-Mamun of the Abassid dynasty in Baghdad (813-833 d.C.).

He merged the axiomatic mathematics of the Greeks with the algorithmic mathematics of the Hindus, making popular decimal representation, symbolic calculation, etc. His book "al-Jabr wal-Muqabala" originated the terms *algebra* (as a distortion of al-Jabr) and *algorithm* (distortion of al-Khwarizmi).

The two topics developed at different speeds in different communities. The discipline of algorithms remained frozen for years, whilst the discipline of algebra grew at as surprising speed.

In the last century there were the theoretical-functional and the topological approach by Riemann, the more geometrical approach by Brill and Noether, and the purely algebraic approach of Kronecker, Dedekind e Weber. The discipline was further enriched with the work of important algebrist and algebraic geometers (like Newton, Eulero, Jacobi, Noether, Castelnuovo, Macaulay, Hilbert, and a lot of others).

But soon algebra lost its constructive foundations and rose as a calculation tool. For example the influence of the school of Bourbaki mined the reputation of the beautiful and constructive *theory of the elimination*, developed in more than half a century by Sylvester, Kronecker, Mertens, König, Hurwitz e Macaulay.

The revival of the *constructive algebra* is a rather recent phenomenon and gained particular benefit from the work of Tarski, Seidenberg, Ritt, Collins, Hironaka, Buchberger, Bishop, Richman, and others.

Here is what Hensel writes in the preface to the lectures on the Kronecker's number theory.

> *[Kronecker] believed that one could, and that one must, in these parts of mathematics, frame each definition in such a way that one can test in a finite number of steps whether it applies to any given quantity. In the same way, a proof of the existence of a quantity can only be regarded as fully rigorous when it contains a method by which the quantity whose existence is to be proved can actually be found.*

Even though for many years there was a large interest in automation of the computation process, the lack of modern computers discouraged the study of algoriths.

Around the 1670 Gottfied Leibnitz invented the well known *Leibnitz wheel* capable of performing the four operations. Moreover Leibnitz proposed the *characteristica generalis*, a symbolic language for the translation of the methods and the assertions of mathematics in *algorithms* and *formulas*.

In the XIX century Charles Babbage devised, but never built, a powerful computation machine, the *analytical engine*, capable of manipulating problems in algebra and mathematical analysis. Babbage, referring to his machine, said: "it could do everything but compose country dances" [HH80].

During this century two events gave oxigen to this sector: the first was the study of the foundations of mathematics, as stated in the *Hilbert program*; the result where the incompleteness theorems of Gödel, the computational models of Church, Turing, Markov, and Post, and the, existence of a "universal" machine and the computability problem. The second event was the birth of modern computers after the end of World War Two.

Initially the problems managed by computers were purely numerical, but soon it became clear that machines were also able to perform operations with symbolic data.

It is interesting to note that since 1844 this aspect of computers was already clear in the mind of the countess Lady Ada Augusta Lovelace. Here is what she wrote about the ability of Babbage's machine [HH80].

> *Many persons who are not conversant with mathematical studies imagine that because the business of [Babbage's analytical engine] is to give its results in numerical notation, the nature of its process must consequently be aritmetical rather than algebraic and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and, in fact, it might bring out its results in algebraic notation...*

The creation of general purpose programming languages (Post, Chomsky, Backus, Church), and in particular the Lisp language, made it possible for Nolan of M.I.T. and Kahrimanian of Temple University, to develop in 1953 the very first symbolic computing systems.

1.2 Symbolic Computing and Computer Algebra

1.2.1 Symbolic Computing

Nowadays the Symbolic Computing is a field which grows rapidly within the frontier between mathematics and computer science. On one hand formalistic mathematics appeared to be limited if a method which uses existence theorems is preferred to a more constructive approach; on the other hand the numerical techniques, very useful in linear problems, collided with relevant obstacles due to the bigger complexity of non linear problems.

On the computer science side the development of new methodologies and computing methods, together with the growth of computing power and felling of costs, permitted in the last years the diffusion of computing techniques that were considered interesting only from the theoretical point of view. Moreover algebraic-geometrical methods revealed to be a powerful aid in various technological applications (signal processing, code theory, cryptography, geometric modelling, CAD, etc.) and showed multiple benefits in the fields of biology, chemistry, physics, and robotics.

From the mathematical point of view it is possible today to access computation examples which, for the level of complexity, are very far from the possibilities of manual computation; and as a consequence it is possible to experiment in depth with various mathematical structures.

The researcher is now able to formulate with a great precision conjectures which in the past could not be expressed, because it was impossible even to enunciate them.

Mathematicians are now interested in creating more sophisticated symbolic computation tools in order to further explore the field, the target in mind being to expose the properties for which there is a reasonable "experimental evidence".

Franz Winkler wrote in his book "Polinomial Algorithms in computer algebra" [Win96]:

> *By the aid of computer algebra the capacity for mathematical problem solving has been decisively improved.*

Another interesting aspect is the possible interaction between algebraic computation and numerical computation, as again Winkler said [Win96]:

> *Both forms of scientific computation have their merits and they should be combined in a computational environment... Often it is more economical in terms of computation time to simplify an expression algebraically before evaluating it numerically. In this way the expression becomes simpler and less prone to numerical errors.*

1.2.2 Constructive Algebra, Computer Algebra e Intuitionism

Edwards summarized the Kronecker's point of view about the *Constructive Algebra* as follows [Edw89]:

> *Kronecker believed that a mathematical concept was not well defined until you had shown how, in each specific instance, to decide [algorithmically] whether the definition was fulfilled or not.*

One motivation for the study of constructive algebra is the following: it enriches and deepens the classical algebra. If we try to make constructive a theorem, we have to study in detail the structure of the examined object; the result of this study may furnish useful information for the implementation of efficient algorithms.

An eminent algebraic geometer who preferred the algorithmic point of view is Abhyankar. He strongly opposed the non-constructive approach, stressing the importance of constructive theories, for example elimination theory. His point of view is obvious from one of his humorous poetries, which begins... [Mis93]:

> *Polynomials and power series.*
> *May they forever rule the world.*
>
> *Eliminate, eliminate, eliminate.*
> *Eliminate the eliminators of elimination theory.*

The computer theoretician extends Abhyankar's point of view to the extreme limit: he regards the existence of a construction as a first step for the classication of computational complexity, which is inherent to the algebraic problem. It was within general ambit of Simbolic Computing, and based upon the constructive approach, a new discipline began: *Computer Algebra.*

Here is what Bhubaneswar Mishra says in his book "Algorithmic Algebra" [Mis93]:

> *The field of computational algebra ... is a relative newcomer, but holds the promise of adding a new dimension to the subject of algorithms. After a millennium, it appears that the subject of algorithms and algebra may finally converge and coexist in a fruitful symbiosis.*

Thomas Becker e Volker Weispfenning, in their book "Gröbner bases: A Computational Approach to Commutative Algebra" [BW93] wrote:

> *The development of new concepts and results in the area has now established computer algebra as an independent discipline that extends deeply into both mathematics and computer science.*

Computer Algebra is not to be confused with the *intuitionism*, even if the relationships between these terms are really strong. Here is a passage from a conference held by Carlo Traverso during the occasion of the XIII U.M.I. (Union of the Italian Mathematicians) Congress [Tra88]:

If we want to reduce the intuitionism to a slogan, we may say the following: a formula of the type $\forall x \exists y . p(x, y)$ means that exists an algorithm which has an input x, an output y and a proof of $p(x, y)$. The proof, in order to be valid, must demonstrate that the algorithm terminates, and this ($\forall x \exists n \in \mathbf{N}$ such that in n steps the algorithm terminates) has to be intuitionist, that is it must contain an explicity procedure to determine the number of the steps starting from the data; in other words it is an estimation of the complexity of the algorithm.

The emphasis of the Computer Algebra is on the algorithms even more than for the intuitionism, but the notion of algorithm is different: not only are considered the algorithms with an complexity estimation, but also the algorithms for which it is possible to demonstrate the termination in a non constructive manner; for sure any convergence estimation is welcome, but such estimation may be given by means of a proof, or by a probabilistic estimation, or simply by a statistics. Any proof method is good as long as it is given in a formal way.

1.2.3 Gröbner Bases

During the recent history of these modern disciplines there was a fundamental event, the birth of the Gröbner bases. There are a lot of people who claim for the paternity of this notion, and in fact there were a lot of precursors; as a matter of fact a central work was the one made by Bruno Buchberger, who, in his PhD thesis in 1965 [B65] introduced the notion of Gröbner basis and described an algorithm for its computation, the famous "'Buchberger Algorithm".

We can say that the theory of the Gröbner bases consists on one hand on the generalization to the multivariate case of the Euclidean algorithm for the computation of the greatest common divisor between two univariate polynomials; on the other hand it consists of the generalization to the polynomial case of the Gaussian reduction for the solution of linear equations systems.

Thomas Becker and Volker Weispfenning in their book "Gröbner bases: A Computational Approach to Commutative Algebra" wrote [BW93]:

There is one particular contribution made by computational algebra that is in most dire need of being introduced in the mathematical mainstream, namely, the theory of Gröbner bases.

David Cox, John Little, and Donald O'Shea on the foreward of their book "Ideals, Varieties and Algorithms" wrote, [CLD92]:

In the 1960's, Buchberger and Hironaka discovered new algoriths for manipulating systems of polynomial equations. Fueled by the development of computers fast enough to run these algorithms, the last two decades have seen a minor revolution in commutative algebra. The ability to compute efficiently with polynomial equations has made it possible to investigate complicated examples that would be impossible to do by hand, and has changed the practice of much research in algebraic geometry. This has also enhanced the importance of the subject for computer scientists and engineers, who have begun to use these techniques in a whole range of problems.

Some importants papers among like [Rob86] and [MM86] consolidated, in the middle of the '80, the mathematical foundations of the Gröbner bases and of some related notions.

Here is what Lorenzo Robbiano wrote in the review [Rob96] of the books "Gröbner Bases: A computational approach to commutative algebra", by Becker and Weispfenning [BW93], and "An introduction to Gröbner Bases" by Adams and Loustaunau [AL94]:

For many years the importance of Buchberger's work was not fully appreciated. Only in the eighties did researchers in mathematics and computer science start a deep investigation of the wide variety of application that could be developed from the new theory. Many generalizations and applications to different problems were investigated. It immediately became clear that the theory of Gröbner Bases was about to become widely used and immensely populare in many areas of science. What I have not yet said, but what is rather starling, is the utter simplicity of the fundamental ideas. Simplicity *and* power*: two ingredients that make a perfect mix for the success of a theory.*

1.3 The Problem of the Representation

Kronecker once said:

God created integers: the rest is human's stuff.

Anyway the man has problems even with the representation of integers! The majority of programming languages treat integers as a finite set, such as $\{-2^{31}, \ldots, 2^{31}-1\}$. It's easy to say: "My data is small, hence the answer will be small, so I do not need big integers and this set will suffice. This assertion is completely wrong. Suppose you want to compute the GCD (greatest common divisor), of the following two polynomials ([Knu69, Bro71]):

$$A(x) = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$$
$$B(x) = 3x^6 + 5x^4 - 4x^2 - 9x + 21$$

By using the well known Euclidean algorithm it is easy to see that the last computed integer has 35 digits (117 bits!), even if the input has small coefficients and the output consists of the result 1 (the two polynomials are coprime).

In the book "Computer Algebra: Systems and Algorithms for Algebraic Computation" by Davenport, Siret, Tournier [DST88] you find:

> *[The problem] of the "intermediate expression swell" is one of the biggest problems in Computer Algebra.*

1.3.1 Representation of Polynomials

In the field of Commutative Algebra the problem of having a good representation of multivariate polynomials is of fundamental importance.

The two main points to consider are: the representation of the coefficients and the representation of the support of polynomials, independently of the format of the coefficients.

The support is the set of the power products of the terms whose coefficients is not null.

Representation of Coefficients

Computing with variable sized coefficients implies the use of different strategies: the coefficients in Z_p have a fixed size as the floating points and hence the computational cost of the operations on the coefficients is constant.

An interesting point is to ask if it is better to keep the polynomials monic (i.e. with the leading coefficient 1) or not; in the case of Z_p the overhead is not considerable and hence may be useful to keep the polynomials monic; on the contrary on Q the cost of dividing by the leading coefficient is very high and hence it is important to adopt an alternative representation, like the ones proposed in [GPT88].

Representation of the Support

Lets start by considering the univariate polynomials. A first distinction is between the *dense* and the *sparse* representation. A representation is said to be sparse if the terms with non null coefficients are not explicitly represented; on the contrary a representation is said to be dense if at least all the monomials between the minimum and the maximum degree are represented.

The dense representation is more immediate from the coding point of view (an array of coefficients is sufficient); on the contrary the sparse representation is more difficult (we need something like a linked list); on the other hand the sparse representation turns out to be closer to the mathematical notation, more space efficient, and finally more time efficient for the basic operations.

As we want a unique representation it is appropriate to choose a term ordering. The term ordering has to be compatible w.r.t. the monoid structure of the set of the terms: $\forall x, y, z \quad (x > y \Rightarrow xz > yz)$.

The univariate polynomials have only two possible ordering, increasing or decreasing w.r.t. the degrees.

The multivariate polynomials allow again sparse or dense representation. But the term ordering turns out to be a more difficult stuff, becase we have a lot of possible term orderings.

For a complete classification of the term orderings see [Rob85].

Whatever ordering is chosen it turns out that the single indeterminates are sorted, so that it is possible to speak of the first indeterminate, the second one, and so on.

The lexicographic ordering has an interesting property: if all the terms with x^n are grouped together (supposing to have x as the first indeterminate) we may consider this polynomial as being an univariate polynomial in the x whose coefficients are in turn polynomials in the remaining indeterminates.

By applying this reasoning to the other indeterminates we obtain a *recursive* representation (i.e. $x^2 + x(2y + 1) + (y^2 + y + 1)$); in contrast with the distributive representation (i.e. $x^2 + 2xy + x + y^2 + y + 1$).

In practice it is always possible to use term orderings of the type given by an $n \times n$ matrix of integers, where n is the number of the indeterminates.

By multiplying the ordering matrix by the vector of the natural numbers representing the exponents of a given term we obtain another integer vector called the *vector of the scalar products*.

So the comparison between terms is reduced to the lexicographic comparison of the corresponding vectors of the scalar products.

In Chapter 5 it is introduced EEE, a flexible and efficient hierarchical representation for multivariate polynomials.

1.4 Computer Algebra Systems

The first program developed for doing algebraic computations has been a program for the differential computation realized for UNIVAC I by Kahrimanian in the year 1953.

An important step forward in the develop of the algebraic manipulation systems corresponds to the creation of the LISP programming language by McCarty at MIT in the 1960. One of the first systems, ALPAK (ALgebraic PACKages, 1963), was a collection of routines written in assembler for the manipulation of polynomials and rational functions (arithmetics, differentiation, etc.).

The very first system for algebraic manipulation, ALTRAN (ALgebra TRANslator), go back to the sixties and is based on the Fortran programming language. Among other things it implemented new algoritms for the multiplication of polynomials and the computation of the GCD.

The first interactive system appeared in 1968. It was MATHLAB, a system developed by Carl Engleman at MIT.

In the '80 there is the birth of some general purpose systems, like Axiom (previously known as SCRATCHPAD), Derive, Macsyma, Maple, Mathematica, and Reduce; their use by the scientific community has grown rapidly; soon appeared also the first special purpose systems, specialized in the field of the Commutative Algebra and the Algebraic Geometry: Macaulay [BS92] and CoCoA[GN, CN95].

1.4.1 Why a Special Purpose System on Computer Algebra

Among the main basic books on computer algebra there is the one by Adams and Loustaunau [AL94] in which it is textually written:

> *There are many Computer Algebra Systems which have a Gröbner basis package, for example AXIOM, MAPLE, and MATHEMATICA. There are other packages which are entirely devoted to computing in polynomial rings and which have an extensive list of commands to perform some of the computations presented in this book. In particular, we mention* CoCoA *and Macaulay. Most of the computations in the examples and exercises in this book (except in Chapter 4) were performed using* CoCoA. *Many of these computations could not have been done with the other systems listed above.*

On the Eisenbud book [Eis95] it is textually written:

Several current computer algebra systems allow the computation of Gröbner bases. Unfortunately, as of this writing the general–purpose systems such as Macsyma, Maple, Mathematica, and Axiom do not have the flexibility in their algorithms of simply do not run fast enough to make experimentation of the sort suggested below very attractive.

From these two citations it turns to be clear that the develop of a computer algebra system specialized to the polynomial rings and to the Gröbner bases, and hence a special purpose system, appears to be of fundamental importance mainly, but not only, for the research in the field of the commutative algebra.

1.4.2 Constructive Commutative Algebra

A constructive approach to the Commutative Algebra takes its origins by Kronecker, which is one of the founders of the intuitionism. In particular is due to Kronecker [Kro82] the first algorithm for the factorization of polynomials with integer coefficients.

Such algorithm is inefficient but was the unique one until the '60, when the first computers appeared and fueled this new research field, and the first algorithms due to Berlekamp-Hensel e Cantor-Zassenhaus [Knu82], where first introduced.

The first systematic treatment of the *Constructive Commutative Algebre* is due to a student of Kronecker, Grete Hermann [Her26]; other contributes are from Van der Waerden and, more recently, from Seidenberg [Sei74] and Lazard [Laz77]; this is befor the big boost due to the Buchberger's algorithm which originated a new big research field which involves till now a lot of different schools of computer scientist and mathematicians.

The objects considered in Commutative Constructive Algebra are rings of polynomials and other rings connected to them. Withing these rings are studied the elements (the polynomials) and some more complex structures such as ideals and modules.

In fact the central subject of the Commutative Algebra and the Algebraic Geometry is the study of the solutions of systems of polynomial equations. Such solutions does not depend on the specific equations, but on the ideal generated by the corresponding polynomials; from the set point of vew they depend on the radical of the mentioned ideal.

The fundamental idea is to extract the information about the "geometry" of the space of the solutions starting from the"algebra" of the corresponding ideal. Traditionally there was the construction of a "dictionary" between the geometrical problems on one hand and the algebraic problems on the other hand. For example to project a coordinated subspace is translated to the problem of intersecting the corresponding ideal with the subring of the indeterminates (elimination theory); to obtain the union of the sets of the solutions of two polynomial systems is translated to the intersection of the corresponding ideals; and so on. As a consequence, tipical problems of the Commutative Algebra are:

- find when possible the explicit solutions of a polymial system
- eliminate some indeterminates: given $I \subseteq k[X, Y]$ find $I \cap k[X]$;
- given I, J ideals of a ring A determine $I \cap J$, $I : J$, etc.;
- find the radical of an ideal, or in general its primary decomposition;
- find a free resolution of a module.

A key problem in the Commutative Algebra is about the *canonical representation of the elements*, [DST88]. Let's consider two classical cases, respectively on one indeterminate and the linear case.

One indeterminate: in the univariate case the ideal are *principal* that is they are generated by a unique polynomial; given more generators of an ideal, find their GCD, which is the generator; an element of $k[x]/(f)$ is canonically represented as the rest of its division by f.

Linear Systems: reduce a system of linear equations to the Gauss-Jordan form; the canonycal representation of an equation is obtained by reducing the equation by means of the equations in the Gauss-Jordan form.

The problem of the unique representation of the elements is solved in both cases using a rewriting operation.

Rewriting

Let's consider the ring $A = k[X]/I$, and a polynomial $f \in I$ such that $f = T - r$ where T is a term and r is another polynomial which does not contain the term T. The polynomial f may be interpreted as a *rewriting rule* in the ring A, as follows: within a polynomial $g \in k[x]$ we can substitute with r one or more occurences of T (as a factor of a monomial), obtaining a new representative of the same element in A.

In order to interpret in a unique way each monic polynomial f as a rewriting rule it is necessary to identify the leading mononomial T in order to write f as the leading monomial minus the remainder.

Orderings

While in the univariate case it is clear what is the leading term (the one with the maximum degree), in the multivariate case this choice is not so automatic. Iven in the linear case it is clear that we have to choose an *ordering* on the set of the indeterminates; in the general case we need a total ordering on the set of the terms (which is isomorphic, as monoid, to N^r, where r is the number of the indeterminates , and N is the set of the natural numbers). Such ordering must make the set of the terms a ordered monoid, with a zero (the term 1) as mimimum. Such ordering is called *term–ordering* (see [Rob85] for a classification).

Examples of term–orderings are:

- the lexicographic ordering (Lex) in which first are sorted the indeterminates and then the terms are first compared w.r.t. the first indeterminate, and in case of parity is considered the second indeterminates, and so on;
- the inverse graded lexicographic ordering (DegRevLex) in which first are compared the whole degrees of the terms, then the degree of all the indeterminates but the last, then the degree of all the indeterminates but the last two, and so on.
- as already stated a generic term–ordering is achieved by comparing lexicographically the vectors of the scalar products obtained by multiplying the vector of the exponents by the ordering matrix.

Gröbner Bases

Once a term ordering is defined, for each polynomial is defined the leading term $\mathrm{Lt}f$ and hence a rewriting rule. Given $G = \{g_1, \ldots, g_s\}$ we define $f \overset{G}{\Rightarrow} f'$ iff f' is obtained by repeatedly applying the rules defined by $g_1, \ldots, g_s$.

Given an ideal I we define $\mathrm{Lt}(I)$ the ideal generated by $\{Lt(f)\}_f \in I$. We say that $G = \{g_1, \ldots, g_s\}$ is a Gröbner basis of I if $\{\mathrm{Lt}g_1, \ldots, \mathrm{Lt}g_s\}$ generate $\mathrm{Lt}(I)$. As an alternate definition we say that G is a Gröbner basis if $f \in I \Leftrightarrow f \overset{G}{\Rightarrow} 0$.

We say that f is *reduced* (w.r.t. G) if no other rule in G may be applied; a Gröbner basis is said to be reduced if any of its element is reduced with respect to the other ones.

The fundamental theorem on the Gröbner bases, due to Buchberger [B65], is the following:

Teorema 1.1 *Given a finite set of polynomials G', and given I the ideal generated by G', and given a term–ordering*

1) there exists a Gröbner basis,

2) there exists an algorithm to compute it,

3) the reduced Gröbner basis is unique.

It is substantially an application of the Knuth-Bendix [KB70] algorithm to make confluent a noetherian rewriting system.

In the univariate case the Gröbner basis of a system of generators $\{g_1, \ldots, g_s\}$ corresponds to the generator $GCD(g_1, \ldots, g_s)$. In the linear case, instead, the Gröbner basis obtained with the Lex ordering is in Gauss-Jordan form.

Starting from the Gröbner basis it is possible to build new important algorithms. A large amount of them is based on the elimination theory, as with some appropriate term orderings (for example Lex) it is possible to eliminate one or more indeterminates. It is almost immediate to show the following theorem:

Teorema 1.2 *Let I be an ideal of $k[X, Y]$, and let's consider a term ordering such that each monomial in Y is smaller than each monomial in the X. Let G be a Gröbner basis of I. Then $G \cap k[Y]$ is a Gröbner basis of $I \cap k[Y]$, w.r.t. the induced term ordering.*

Starting from the elimination algorithm it is possible to build other algorithms: for example we may intersect two ideals as follows: let t be a new indeterminate: then $I \cap J$ is obtained by eliminating t from the ideal $tI + (1 - t)J$.

We may compute also the colon ideal $I : J$; if $J = (g_1, \ldots, g_r)$, then we have that $I : J = (I : (g_1)) \cap \ldots \cap (I : (g_r))$; furthermore $I : (a) = a^{-1}(I \cap (a))$.

We may represent also subrings: if $f_1, \ldots, f_r \in k[X]/I$ and $I = (g_1, \ldots, g_s)$, we define $J' = (g_1, \ldots, g_s, f_1 - t_1, \ldots, f_r - t_r) \subseteq k[X, T]$, with T new indeterminates. Let $J = J' \cap k[T]$, hence we have that $k[f_1, \ldots, f_r] \equiv k[T]/J$.

1.4.3 Basic features of a Computer Algebra System

The main features of a special purpose system specialized in the field of the Computer Algebra, are the followings:

- special data structures to represent non numerical data;
- a language for manipulating them;

- a library of functions to perform the basic algebraic operations;
- ability of working with mathematical expressions;f
- possibility to operate interactively;
- programmability, to extend the system capabilities;
- arbitrary precision for integers and rational numbers.

For sure we also need to have all of these feature implemented in a very efficient way, considered the very high complexity of the basic algorithms [MM82].

A Computer Algebra system must provide an arbitrary precision arithmetics on both the integers and the rational numbers.

Moreover it has to be able to execute operations on polynomials (sum, product, power, derivative, greatest common divisor, least common multiple), on rational functions (sum, product, power, and derivative), on ideals (sum, product, power), and on modules (sum).

Those operations let the user able to prepare the data for doing more advanced computations, which the system has to be able to perform as efficiently as possible. Here are some of the advanced operations that the researchers tipically ask for: computing Gröbner bases, syzygies, Poincaré series of ideals and modules, free resolutions of ideals and modules.

A lot of study has been done for the optimization of the algorithms related to the Gröbner bases [GMNRT, CDNR97, B97, T96, CDR96].

Another important characteristic has to be the simplicity of using the system. It is important to let the matematician abile to benefit of the system even if he/she is not a computer expert.

Finally a desirable feature is the possibility to extend the system with some form of programmability (see Sect. 2.1.2).

Chapter 2

The CoCoA 3 System

My work in the field of the Computer Algebra was mainly in the design and the develop of the CoCoA system [CNR96]. More precisely I am the designer of the *architecture* of the system.

Introduction

CoCoA 3 is a special-purpose system for doing *Computations in Commutative Algebra*. It is the product of a research team in Computer Algebra at the University of Genova (Italy). It belongs to the class of Computer Algebra Systems which are specialized in doing computations on polynomials and which includes for instance Macaulay [BS92], Macaulay 2 [BS92] and Singular [GPS95].

The architecture of CoCoA is designed to offer flexibility through *efficiency*, *portability*, *programmability*, and *multiple choices of coefficient arithmetics*.

Currently CoCoA runs on the following platforms: Sun workstations, DEC-Alpha stations, Macintosh, PC (under DOS, Linux, and Windows 95). The system can be obtained by anonymous ftp from the URL:

`ftp://cocoa.dima.unige.it/cocoa`

CoCoA is the name of a research project, which started in the mid-eighties. It was coordinated by Robbiano and its first official version was due to Alessandro Giovini and Gianfranco Niesi [GN]. The original idea was to produce a piece of software which could be both a useful tool for the working mathematician and a laboratory for research in different fields, ranging from Commutative Algebra to Software Engineering. After the premature death of Giovini (January 1993) a new project started. Robbiano continues to act as the project manager, Antonio Capani and Gianfranco Niesi are the authors and many other researchers like John Abbott, Anna Bigatti, Massimo Caboara and Gabriel De Dominicis contribute with pieces of code.

Currently CoCoA is used by researchers in several countries. Most of them are Commutative Algebraists and Algebraic Geometers, but also people working in different areas like Analysis and Statistics have already benefitted from our system.

CoCoA is also used as the main system for teaching advanced courses in several Universities. Besides Italy, the most intensive use is by Tomas Recio at the University of Santander (Spain), Anthony Geramita at the Queen's University (Canada) and Martin Kreuzer at the University of Regensburg (Germany).

CoCoA is able to perform both simple and sophisticated operations on multivariate polynomial rings and on various data connected with them (ideals, modules, matrices, rational functions). Arbitrarily large integers are also available. Every computation is performed within a ring (or "ring-environment"). The user can define and use many rings, but at any one time there exists a chosen *current ring*.

The system is capable of performing *basic* operations such as: sums, products, powers, derivatives, gcd, and lcm of polynomials; sums, products, powers, and derivatives of rational functions; sums, products, and powers of ideals; sums of modules; sums, products, powers, determinants, and adjoints of matrices; operations between heterogeneous values, like the product between an ideal and a polynomial, and so on.

Besides these, more advanced operations are available. For example: Gröbner bases of ideals and modules; syzygies of ideals and modules; minimal free resolutions of ideals and modules; intersection and division of ideals and modules; inclusion and equality test for ideals and modules; elimination of indeterminates; homogenization of ideals; Poincaré series and Hilbert functions.

The core of the system is a set of algorithms for computing Gröbner bases, syzygies and Poincaré series of ideals and modules over a polynomial ring whose coefficient ring is a field. These algorithms have been optimized in several directions [B97, CDNR97, CDR96, GMNRT, T96] and are used as a foundation for many operations.

One of the main features of CoCoA 3 is a Pascal-like programming language, named CoCoAL, that allows the user to customize the system and extend the embedded library.

CoCoA has a kernel written in the C language (about 55,000 lines) for portability and efficiency, and a library written in CoCoAL (about 2,500 lines), the high–level language of the system.

It has different interfaces for different machines; each interface interacts with the system using a Low Level Protocol (LLP) which is platform independent.

The interface-independent part of CoCoA is divided into two modules: ENGINE and MATH. The first one is the main motor: it includes the interpreter of the CoCoAL language which allows it to exchange LLP-inputs and LLP-outputs with the interface. Requests of computations are forwarded by ENGINE to MATH, which contains all the mathematical algorithms to manipulate coefficients, polynomials, ideals, modules and so on. MATH accepts LLP-requests from ENGINE, converts data from LLP-format into internal data and then performs the mathematical task in an efficient environment.

2.1 Some examples

The CoCoA commands can be roughly divided into two classes: *simple commands* and *structured commands*. Examples of simple commands are: evaluation of an expression, assignment, and printing. Examples of structured commands are: if-then-else, for-loop, while-loop, foreach-loop, and function definition.

The expressions follow a simple syntax which is very close to the usual mathematical one.

Every object in CoCoA is defined over a *base ring* which is a polynomial ring with some annotations (about ordering or weights or other things).

Each computation in CoCoA is performed in a ring-environment which consists of a base ring and a local memory where objects like polynomials or ideals can be stored.

The user can define and use several ring-environments. At any one time a particular *current ring-environment* is active within the system. The system has a default ring-environment named R and whose base ring is $Q[t, x, y, z]$.

The simplest way to use CoCoA after starting it, is to set a ring (or use the default ring), type a command, and ask the system to execute it.

Example 2.1.1 *This example shows how to define and use a ring different from the default ring.*

```
A ::= Z/(7)[xy];
Use A;
(4x^2 - 1/2y)^4;
```

The first command uses the operator "::=" to define A to be the polynomial ring in the indeterminates x and y with coefficients in the field $Z/(5)$ of the integers modulo 5. The second command chooses A as the current ring. Then the expression $(2x^2 - 1/2y)^4$ is evaluated in A and the result is displayed.

```
 - 3x^8 - 2x^6y + 3x^4y^2 - 2x^2y^3 - 3y^4
-------------------------------
```

Example 2.1.2 *In the next session we use the structured command* `For` *to compute* $(x+y)^7$ *in the ring* Z/nZ *for* $n \in \{3,5,7,9\}$. *Note that any text from a* `--` *to the end of the line is a comment.*

```
For N := 3 To 9 Step 2 Do
  S ::= Z/(N)[xy]; --> define the ring S
  PrintLn Ring(S); --> display the ring S
  S :: (x+y)^7; --> evaluate the power in S and print it
  PrintLn; --> start a new line
  PrintLn
End;
```

Special Orderings

Here we discuss two problems whose solutions require using a particular term-ordering. The first problem is how to find the minimal polynomial of an element of an algebraic extension of Q. The second one is how to find a cartesian representation of a space curve given parametrically.

Example 2.1.3 *The minimal polynomial of* $\frac{4\alpha-1}{\alpha^3}$ *over* Q*, where* α *is a root of* x^7-x-1 *can be found by computing the reduced Gröbner basis of the ideal* $(x^7-x-1, x^3y-4x+1)$ *of the ring* $Q[x,y]$ *with respect to the lexicographic term-ordering with* $x > y$. *In* CoCoA *this is achieved by using the following commands:*

```
Use R ::= Q[xy],Lex;
Set Indentation;
ReducedGBasis(Ideal(x^7-x-1, x^3y-4x+1));
```

The output is:

```
[
  x - 10022553737/89893683351809y^6 + 49925279149/89893683351809y^5
+ 17282591991/89893683351809y^4 - 2197476813566/89893683351809y^3
- 3212847751937/89893683351809y^2 + 1627614492145/89893683351809y
- 52555866039552/89893683351809,
  y^7 - 5y^6 + 147y^4 + 640y^3 - 31y^2 + 2176y - 20479 ]
-------------------------------
```

Example 2.1.4 *Given the space curve* $(t^{31}+t^6, t^8, t^{10})$*, its cartesian equations can be found by eliminating the indeterminate* t *in the ideal* $(t^{31}+t^6-x, t^8-y, t^{10}-z)$. *In* CoCoA *this is achieved by using the following commands:*

```
Use R ::= Q[xyzt];
Set Indentation;
Elim(t, Ideal(t^31+t^6-x, t^8-y, t^10-z ) );
```

The system automatically changes the ordering to an elimination term-ordering for t, performs the computation, and, finally, restores the original ordering and gives the result:

```
Ideal( y^5 - z^4,
        - y^4z^5 + y^4 - 2xy^2z + x^2z^2,
        - z^8 - 2xy^3 + x^2yz + z^3,
       2xy^4z^4 + yz^7 + 3x^2y^2 - 2x^3z - yz^2,
        - y^2z^6 - 1/2xz^7 + 1/2x^3y + y^2z - 3/2xz^2,
        - 1/3x^2y^4z^3 - y^3z^5 - 2/3xyz^6 + 1/3x^4 + y^3 - 4/3xyz )
-------------------------------
```

Minimal Free Resolutions

Example 2.1.5 *In this example we compute the minimal free resolution of the ideal I generated by the 2 by 2 minors of a catalecticant matrix A, using the interactive environment of* CoCoA *(see [CN98]). We define the ideal I and start the computation of its minimal free resolution using the Hilbert-driven algorithm described in [CDNR97].*

```
Use R ::= Z/(32003)[z[0..3,0..3,0..3]];

A := Mat[
  [z[3,0,0], z[2,1,0], z[2,0,1]],
  [z[2,1,0], z[1,2,0], z[1,1,1]],
  [z[2,0,1], z[1,1,1], z[1,0,2]],
  [z[1,2,0], z[0,3,0], z[0,2,1]],
  [z[1,1,1], z[0,2,1], z[0,1,2]],
  [z[1,0,2], z[0,1,2], z[0,0,3]]
];

I := Ideal(Minors(2,A));
GB.Start_Res(I);
```

Now we ask the system to perform 1000 steps of computation and then we explore the partial result.

```
GB.Steps(I,1000);
GB.Get_Res(I);
```

```
0 --> R^176(-5) --> R^189(-4) --> R^105(-3) --> R^27(-2)
-------------------------------

GB.Res_Report(I);
--------------------------------------------------------------
Minimal Pairs,             :   650
  Groebner Pairs           :    14
  Minimal (Type S)         :   636
      H-Killed (Type S0)   :     9
--------------------------------------------------------------
```

Finally we complete the computation and see the result.

```
GB.Complete(I);
GB.Get_Res(I);
0 --> R(-9) --> R^27(-7) --> R^105(-6) --> R^189(-5) --> R^189(-4)
--> R^105(-3) --> R^27(-2)
-------------------------------

GB.Res_Report(I);
--------------------------------------------------------------
Minimal Pairs,             :   730
  Groebner Pairs           :    25
  Minimal (Type S)         :   705
    Minimal (Type Smin)    :   616
    Minimal (Type S0)      :    89
      H-Killed (Type S0)   :    78
      Hard (Type S0)       :    11
--------------------------------------------------------------
```

2.1.1 The Architecture of CoCoA

CoCoAis made up of two main modules: the interpreter of the CoCoAL language and the user interface. The user interacts with the system by using CoCoAL, a special purpose Pascal-like language. The interface transforms the user's requests from CoCoAL to an internal protocol called CTL corresponding somewhat to the parse tree of the request (the command).

The use of this internal protocol gives a better independence of the lexical and notational details of the language, guaranteeing a good interfaceability with the external word.

The communication between the interpreter and the interface is implemented via *C* library, which is responsible of initializing the interpreter and submitting the user's command to it; furthemore this interface gives to the interpreter the tools for accessing some Input–Output functions; these functions implement the notion of *device* which is characterized by the phisical support (e.g.: file, string) and the protocol (CTerm, CoCoAL, plain) used to read from and write to the device.

The use of the devices make it possible for the interpreter to have a sufficient access to the external world, even mantaining its independency of the interface and the underlying architercure. For example this techniques facilitated the realization of a prototypal interface of the system with the World Wide Web [CD96].

The interpreter receives the commands in CTerm language and execute them within a *ringenv*, an environment built upon a *base ring*. The result of the evaluation of an CoCoAL expression is a value which in general depends on one or more rings, and which may be stored in a memory of the system, in order to recall it for further computations.

Each computation is performed within the *current ringenv*; in this way the system is abile to completely code each value, possibly involving polynomials, taking into account the coefficient ring, the names of the indeterminates, and the current ordering.

The memory consists of a set of pairs (identifier, value). Each value consists in turn of a tuple $\langle V, T, R, C\rangle$, where V is the internal (efficient) representation of the value, T is the type (int,string,poly,etc.), R is the reference ring which completes the type, and C is the *reference counter*.

The system uses the T field in of the value in order to deduce the type of the expressions which involve that value.

For efficiency each value holds a reference counter, which indicates the number of the reference to that value; in that way it is possible to refer to the same value without the need of duplicating it.

The interpreter forwards the mathematical requests to a kernel of mathematical functions for the manipulation of the coefficients, the polynomials, the ideals, and so on.

The type of each expression is deduced by the operations **and the operands** of the expression itself. A partial ordering is defined on the types in order to implement the notion of subtype. Furthermore the system perform an automatic type conversion in order to enrich the overloading of the operators. For example it is implemented the operation of sum of polynomials; when the system have to sum a integer with a polynomial, it finds out that there not exists a dedicated operation such as `Sum(INT,POLY)`, then discovers the `INT` is a subtype of `POLY`, convers the integer to a polynomial and finally perform the sum of the two polynomials.

The non critical part of the algorithms has been implemented at an high level using the CoCoAL language and is organized in *packages* (Sect. 2.2), a construct of the language which gives a good level of modularity and solves the problem of the name clash.

2.1.2 The CoCoAL programming language

The CoCoAL language has been developed keeping strongly in mind the need of simplicity. The CoCoAL programmer needs to work knowing only a bit of programming notions, and typing in the system using a notation which is similar to the standard mathematical notation he/she is used to work with.

For example the user prefers to write xy instead of $x * y$.

For this reason some aspects like the declaration of the variables and the possibility of creating new types are not part of the CoCoAL.

The language provides a collection of predefined types:

- basic types: integers, rational numbers, integers modulo p, multivariate polynomials, rational multivariate functions, strings, and types.
- structured types: lists, matrices, and open records (also known as associative arrays).

Note that the polynomials and the rational functions are considered as basic types. Then the user accesses directly the values which are implemented in an efficient way.

The lists and the matrices are heterogeneous, that is the entries may have any type, as in the following example:

```
[3,x+y,[2,"hello"],Ideal(x,y)]
```

An open record consists of a set of pairs (name,value); the values may have any type. The record is said to be open because the names are not fixed a priory: you may add pairs dynamically.

```
X := Record[N=3, S="Ciao"];
Num := X.N;
X.P := x+y;
```

By combining lists, matrices and records it is easy to implement a lot of structured values.

An useful feature of the language is the possibility to generate values of type `TYPE`. So, for example, the user may ask to the system the type of a parameter in order to choose the correct implementation.

The values of type `TYPE` are partially sorted. If $t < t'$ then a value v of type t may be always converted to a value of type t'. Furthermore v is said to *admit* both the types t and t'.

So the user may evaluate the admissibility of a type for a given value, and write expressions like the following one.

```
If (Type(X) <= POLY) Then Print "OK" End;
```

As the CoCoAL language does not require declaration of variables and types, the code results to be concise, but readable. Its simplicity implies a big rapidness of develop.

For example the following function computes the *jacobian matrix* of a set S of polynomials w.r.t. the indeterminates of the current ring. The notation used to declare the function is near to the usual mathematical notation.

```
Jacobian(S) := Mat[ [ Der(P,X) | X In Indets() ] | P In S ];
```

Those considerations indicates the CoCoAL language as possible language for specification and rapid prototyping of new (mathematical) algorithms.

For example CoCoAL has been used during the develop of two different algorithms for the computation of the minimal free resolutions [CDNR97, CT].

For efficiency reasons the algorithm developed in [CDNR97] has been later ported to C language; but this porting did not involved logical redefinition, but just a syntax translation.

2.2 Functions and Packages

The user may extend the system by defining new functions or procedures. Furthermore it is possible to collect functions in a package.

The general syntax to define a function is:

```
Define F(X1,...,Xn) C End
```

where F is an identifier, C is a sequence of commands, the Xi's are formal parameters and E is an expression.

This command adds the user-defined function F to the library. The function F can be called in the following way:

```
F(E1,...,En)
```

where the Ei's are expressions. The result of the evaluation of each expression Ei is assigned to the respective formal parameter Xi, and the command sequence C is executed. If, during the execution of C, a statement 'Return E' is executed, then the result of the evaluation of E is the return-value of the function F. If no 'Return' command is executed, or 'Return' is executed without argument, then the return-value is 'Null'.

```
Define Square(X)
  Return X^2;
End;

Square(5);
```

When the body of the function is just the command Return followed by an expressions, then the user may use an abbreviate syntax as follows.

```
Square(X) := X^2;
```

User-defined functions may be saved in separated files and read into a CoCoA session using the 'Source' command. If one sources several such files or, especially, if a file is to be made available for general use, a possible problem arises from conflicting function names. If two functions with the same name are read into a CoCoA session, only the one last read survives. To avoid this, functions may be collected in 'packages'.

A CoCoA package is essentially a list of functions (made using the 'Define' command), labeled with a long prefix. A function from a package is referred to by the package prefix plus the function name. The user may type the full prefix, but the usual method is to create a short alias for the prefix.

The following is an example of a package. It could be typed into a window as-is during a CoCoA session, but we will assume that it is stored in a file in the CoCoA directory under the name "one.pkg."

```
Package $contrib/dave/toypackage

Define IsOne(N)
  If N = 1 Then Return TRUE Else Return FALSE End;
End;

Define Test(N)
  If $.IsOne(N) Then
```

```
    Print 'The number 1.'
  Else
    Print 'Not the number 1.'
  End;
End;

End; -- of toypackage
```

Below is output from a CoCoA session in which this package was used:

```
<<'one.pkg'  -- read in the package
Test(1);  -- error here because the function 'Test' is not defined

-------------------------------
ERROR: Unknown operator Test
CONTEXT: Test(1)
-------------------------------
$contrib/dave/toypackage.Test(1); -- this is the name of the function
                                  -- we are looking for
The number 1.
-------------------------------
Alias Toy := $contrib/dave/toypackage;  -- use an alias to save typing
Toy.Test(3);
Not the number 1.
-------------------------------
Toy.IsOne(3);
FALSE
-------------------------------
```

Once the package is read, the user can choose a "substitute prefix" using the 'Alias' command and in that way avoid conflicts between functions in various packages and save on typing.

2.3 Breakdown of CoCoA Architecture

2.3.1 General Structure of a CoCoA Application

The CoCoA system alone is not an application. It cannot be executed directly because it has not a main. The CoCoA kernel (see Sect. 2.3.2) acts as a *computer algebra server* and must

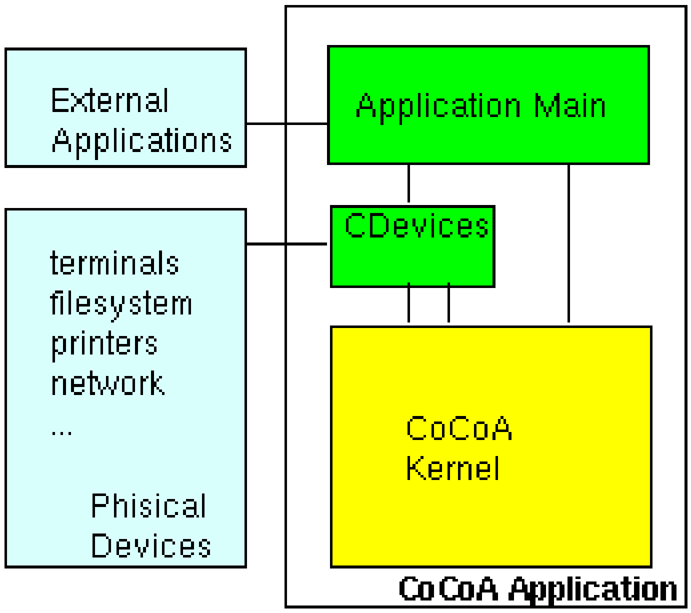

Figure 2.1: General Structure of a CoCoA Application

be linked to another module, a *computer algebra client*, which contains the application main.

A CoCoA application (see Fig. 2.1) is a monolitic program which consists of an entry point for the execution (the *Application Main*), the CoCoA kernel, and an I/O module which consists of a list of "abstract devices" (the CDevices, see below).

Currently we have developed two graphical CoCoA applications, for the Macintosh and the Windows platforms, and a general purpose text based CoCoA application which is useful for Unix platforms and also for DOS platforms (which do not have access to Windows). Moreover we have designed an advanced Web interface which is described in Chap 6.

The Application Main holds the control of the program execution and submits requests to the CoCoA kernel. It may also interact with other applications.

In order to make CoCoA access the external world (i.e. some phisical devices such as terminals, filesystem, printers, and network) the CoCoA application must implement some CoCoA devices (CDevices, see Sect. 2.3.7).

The CoCoA application passes requests (CTL-commands) to the CoCoA kernel by means of a dedicated protocol called CTL (CoCoA terms language, see Sect. 2.3.6). The CoCoA kernel and the application main may also read CTL-commands from input CDevices and write CTL-values (the results) to output CDevices.

2.3.2 The CoCoA Kernel

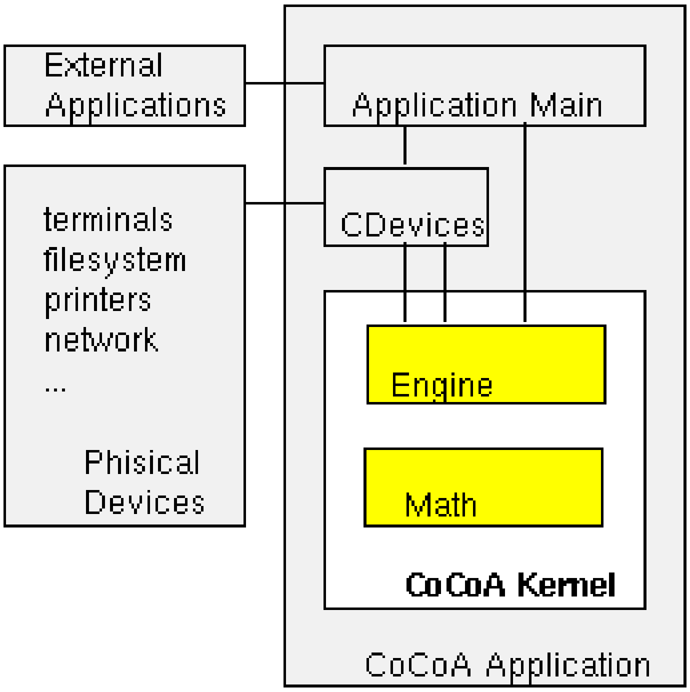

Figure 2.2: The CoCoA Kernel

In a system like CoCoA the mathematics is the *core* target, and hence it contains a lot of advanced mathematical algorithms. On the same time we have decided to give a great importance to the usability and extensibility of the system. For this purpose CoCoA is not just a mathematical library, but is a combination of a powerful mathematical library and a flexible architecture for the access to that library.

For this reason the architecture of the system poses a clear distinction between the module responsible for the interactions management, the *Engine*, and the mathamatical library, the *Math* module (see Fig. 2.2).

The Engine interprets requests coming from the Application Main or from input CDevices, and possibly sends results to output CDevice. The mathematical requests are forwarded by the Engine to the Mathematical Module.

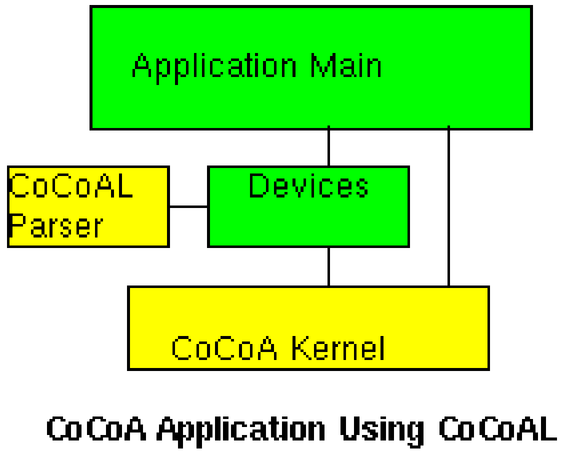

Figure 2.3: The CoCoAL Parser

2.3.3 The CoCoAL Parser

The main interaction with user is made via the CoCoAL language, which is used both for input and for output (see Fig. 2.3).

The CoCoAL parser is responsible for translating CoCoAL data into CTL-commands and CTL-expressions. An input CDevice may implement the CoCoAL protocol by means of CoCoAL parser.

The CTL-values are translated in CoCoAL by means of a "Pretty printing" module. An output CDevice may implement the CoCoAL protocol via the Pretty Printing module.

Note that the CoCoAL language, even if it is currently the only implemented protocol for accessing the system from a high level, is just one of the possible protocols which may be implemented in the CDevices.

2.3.4 The CoCoA Engine Module

The Engine Module (see Fig. 2.4) is responsible for the interaction with the user and for the connection to the Mathematical Library.

The CTL Interpreter receives requests from the Application Main in the form of function calls (`execute(command)`). Moreover the CTL Interpreter may read (and execute) commands from an input CDevice(`source(device)`).

When, during the execution of a command, the CTL Interpreter finds an expression it submits it to the *Expression Evaluator* which in turn makes use of User Defined CTL-

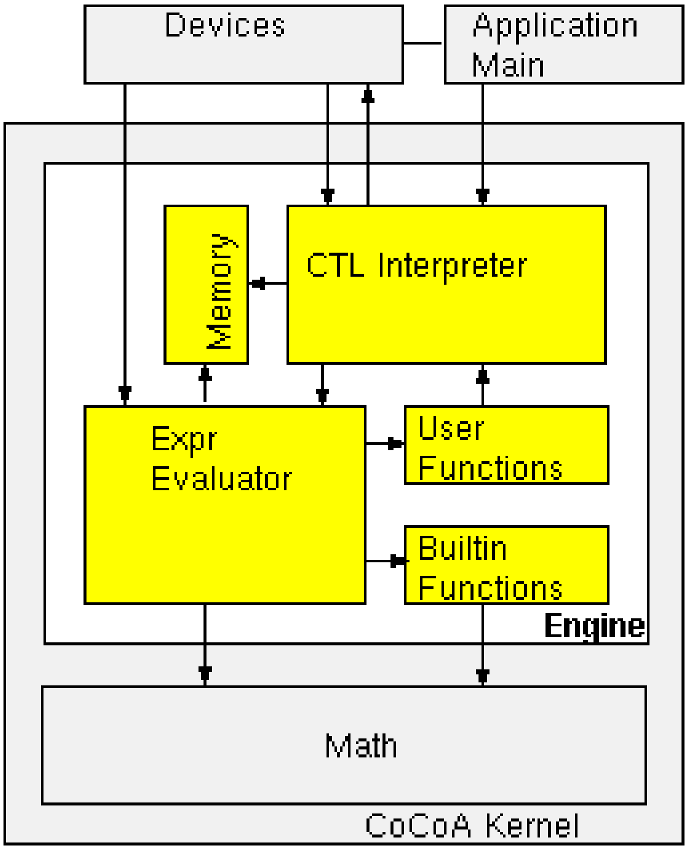

Figure 2.4: The CoCoA Engine Module

Functions or Built-in Functions.

Some of the Built-in functions access the Mathematical Library for creating the mathematical data and for performing all the mathematical computations.

2.3.5 The CoCoA Mathematical Module

The *Math* module (see Fig. 2.4) contains the implemantation of basic data structures, like coefficients, polynomials, and modules.

Advanced algorithms for the computation of Gröbner bases, syzygies, resolutions, and so on, are organized in a dedicated module which is called the "Groebner Framework" (see Chap. 3 and Chap. 4).

Some high-level functions like Intersection of Modules, or the Colon operation are organized

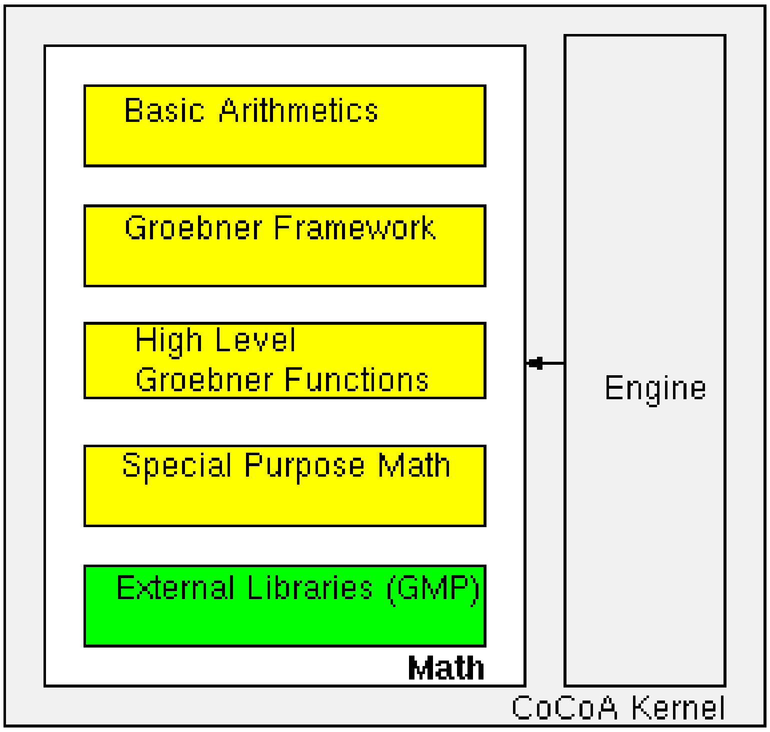

Figure 2.5: The CoCoA Mathematical Module

in a separate module.

Finally the Math module contains powerful special purpose libraries for computation of the Poincaré series, computation of the solutions of Linear Algebra problems, management of Toric Ideals, management of Ideals of Points, and a very efficient Multivariate Factorizer (see Fig. 2.4).

2.3.6 The CTL language

The CTL language is a dedicated language used for the interaction between the CoCoA application main and the CoCoA kernel, and for some high level interaction between the input/output CDevices, the CoCoA kernel and the CoCoA application main.

An element of the CTL language, which is referred as CTerm (CoCoA term), consists of a tree which represent an operation: the root node of the tree holds the operator and its subtrees, which are in turn CTerms, are the parameters of the operation. The leaves of the tree may be constant operations or strings.

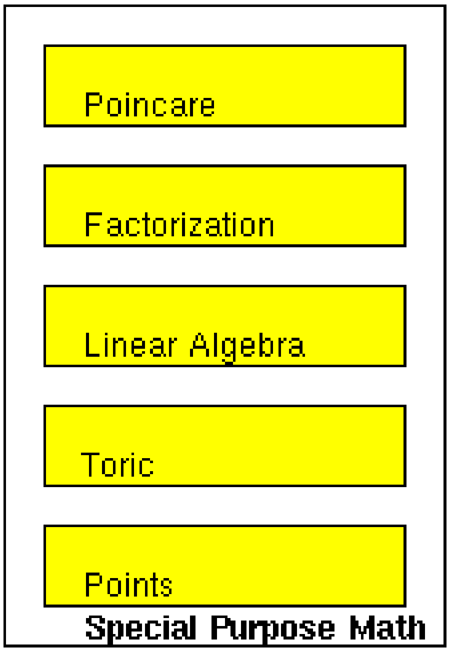

Figure 2.6: Special Purpose Mathematical Modules

The elements of the CTL language are divided into three categories: CTL-commands, CTL-expressions, and CTL-values. The CTerms are built by assembling CTL-operators and strings. More precisely CTerms are of the one of the following forms:

- "a string"
- 0-arity operations: $o()$ (where o is a CTL-operator)
- n-arity operations: $o(t_1, \ldots, t_n)$ (where o is a CTL-operator and t_i are CTerms)

The CTL language is useful for the following reasons:

- it abstracts from a particular syntax (e.g. the CoCoAL syntax) and is better mapped to the semantics (roughtly speaking we have a CTL-operator for each CoCoA operation);
- it makes a useful distinction between expressions and values; for example an item like `3x+y` may be seen as an expression like `sum(mult("3","x"),"y")` or as a value (a polynomial) of the form `poly(mon("3","x"),mon(1,"y"))`. It is obvious that a distinction between the two aspects is important for the efficiency of the system. In the first case the following operations are to be performed: evaluate the expression "3" and obtain an integer, evaluate the expression "x" and obtain a polynomial,

multiply the integer by the polynomial and obtain another polynomial P, evaluate "y" and obtain another polynomial Q, and finally obtain the result of the evaluation of the whole expression by performing the sum of the two polynomials P and Q. In the second case we *directly construct* the polynomial using the given monomials.

- we do not have to interpret the command; we directly *execute* the command in the CoCoA kernel; this is possible because a CTerm is directly a "parse tree" and hence no syntax checking has to be performed; moreover the tree structure gives a better performance.

2.3.7 The CoCoA devices

The CDevices are responsible for the interaction of the CoCoA application with the external world.

The CDevices implement low level operations like `getChar()`, `getString()`, `putChar()`, and `putString()` and more high level operations like `getCTerm() and putCTerm()`, for reading and writing data using the CTL language (see below).

A CDevice is an abstraction of a phisical device and may be an input device, which reads data from a phisical source (e.g. a file, a window, the network, a string in the memory), or an output device, which writes data to a phisical destination (e.g. a file, a window, the network, a printer, a string buffer).

Each device must also implement a *protocol* in order to translate input data to CTerms via the `getCTerm()` function and to translate CTerms to output data.

Chapter 3

The CoCoA 3 Gröbner Framework

In this chapter it is reported how we have implemented in the CoCoA 3 system algorithms for computing Gröbner bases, syzygies, and minimal free resolution of submodules of finitely generated free modules over polynomial rings. In particular we describe the environment we have built where these algorithms are obtained as special instances of a single parametrized algorithm. The outcome is a framework which offers not only efficiency in computing, but also efficency in designing new algorithms; moreover, it allows the user to interactively execute the above computations, and easily customize the way they are carried out.

3.1 Introduction

Computing Gröbner bases, syzygies, and finite free resolutions of ideals and of submodules of finitely generated free modules over polynomial rings is a fundamental task in Computational Commutative Algebra. Nowadays there are several specialized computer algebra systems, developed by researchers in the field, which efficiently perform such computations, for instance Asir [NT92], CoCoA, Macaulay [BS92], Macaulay 2 [GS96], Singular [GPS95].

The cornerstone in this area has been the (now) classical algorithm due to Buchberger [B65] which produces a Gröbner basis of an ideal in a polynomial ring R over a field, starting from any of its possible finite sets of generators. This algorithm can be suitably modified [S80, S77] in such a way that it also computes the syzygy module of a list of polynomials. These procedures can be extended to submodules of the free R-module R^n and lists of vectors of polynomials (see for instance [MM86]).

A significant situation arises from the fundamental problems of Projective Algebraic Geometry and occurs when R is a graded ring and one works with homogeneous polynomials. In this context appropriate techniques and strategies can produce remarkable improvements of the algorithms quoted above.

Since its first release [GN] the system CoCoA has offered an efficient implementation of algorithms for computing Gröbner bases, syzygies, and free resolutions, embodying theoretical results of several authors [B85, CT, GM89, GMNRT, T96] to speed up such computations.

The system was originally designed to work with general polynomials but more recently we have enriched it with special algorithms for the homogeneous case. In particular, in a joint work with G. De Dominicis and L. Robbiano [CDNR97], we developed new algorithms for computing: minimal systems of generators, Gröbner bases, syzygy modules, and minimal free resolutions of a homogeneous submodule of a finitely generated graded free module. Variants of these algorithms use the knowledge of the Hilbert function of the submodule to speed up the computation. We discuss later in this chapter our implementation of these algorithms.

It is worth mentioning other recent contributions to this subject: R. La Scala [L96] has proposed an algorithm for computing a minimal free resolution which has been implemented in Macaulay 2. Another approach, due to M. Caboara (1996), uses an embedding of the module in a suitable polynomial ring; moreover they show how to perform various module operations and present an algorithm to compute a minimal free resolution of a homogeneous module.

The implementation of the algorithms was guided by the following key observation: these algorithms share a common pattern and hence they can be obtained as specialization of an abstract Buchberger-like algorithm which depends on some function parameters, i.e. each of these algorithms can be obtained from the generic one by assigning to each parameter an appropriate function called *key function*. Different algorithms can share some key functions; on the other hand the same computation can be done using different settings of key functions depending on the strategy to be used and on some other options (e.g. if we want some statistics or not). We have supplied several key function sets for the most common algorithms and for some of their variations and we have designed the system in such a way that users may assign their own key function set and easily customize the behaviour of a Buchberger-like algorithm.

Another important consideration is the following: often a complete computation takes a very long time; in most cases the user wishes to execute a computation in an interactive way: that is to start with a part of the computation, explore the partial result, and, if necessary, perform further pieces of computation until a certain result is obtained.

For these reasons we have completely redesigned and reimplemented all the "Gröbner stuff" as a family of algorithms obtained from a unique generic algorithm which, after an initialization stage, can perform a single "step" of computation. We started our work by specifying all the considered algorithms in CoCoAL, the CoCoA 3 programming language and, then, we implemented them in the CoCoA kernel, which is written in the language C.

The key observation here is that CoCoA is a system for doing research, hence we concentrated not only on efficiency in computing but also on efficiency in designing new algorithms. The key functions trace a well defined and well behaved path along which the user can do good prototyping by choosing among the predefined sets of key functions. Moreover it is easier for us to extend the C library by implementing new key functions, if necessary. In both cases the overhead is greatly reduced, and hence also the possibility of introducing errors.

In this chapter we present some aspects of our implementation by using the CoCoAL language. In Section 3.2 we revise the classical Buchberger algorithm for a module. Then we present in Section 3.3 the abstract version of the algorithm, and we discuss how various concrete algorithms can be obtained by specifying suitable sets of key functions. In Section 3.4 we present the algorithms of the homogeneous environment. Finally we conclude the chapter with some remarks (see Sect. 3.5).

3.2 A Review of the Buchberger Algorithm

In this section we recall the classical Buchberger algorithm for a module, and present a short description of it in CoCoAL.

Let $R := k[X_1, \dots, X_n]$ be a polynomial ring over a field k and $n > 0$. Throughout this chapter, for the sake of brevity, we use the word *module* to denote a submodule of the free module R^n. An element of a module is called a *vector*.

Let $\{e_1, \dots, e_n\}$ be the canonical basis of R^n. The terms of R^n are the elements of the form $t\, e_i$ where t is a term of the ring R. We assume that a term-ordering σ is given on the set of the terms of R^n. If $V \in R^n$, then we denote by $\mathrm{LT}(V)$ and $\mathrm{LC}(V)$ the leading term and the leading coefficient of V respectively. Basic facts about polynomial rings, modules, term-orderings, and Gröbner bases can be found in (Adams and Loustaunau 1994, Möller and Mora 1986).

The Buchberger algorithm takes as input a list of vectors, L, and returns a list of vectors, *GBasis*, which is a Gröbner basis of the module generated by L.

We recall that two vectors V and W in R^n give rise to a critical pair if $\mathrm{LT}(V) = s\, e_i$ and $\mathrm{LT}(W) = t\, e_i$ with the same i; the S-polynomial of a critical pair is the vector defined by

$$\mathrm{LC}(W)\,\frac{\mathrm{lcm}(s,t)}{s}\,V - \mathrm{LC}(V)\,\frac{\mathrm{lcm}(s,t)}{t}\,W \ .$$

The first phase of the algorithm initializes the list *IPs* of the *incoming pairs* to the list of the critical pairs of elements in L and initializes *GBasis* to L. Then the algorithm executes a loop: each incoming pair P is processed as follows. First the S-polynomial of the pair is computed; then it is normalized with respect to the list *GBasis*; if the resulting normal form N is different from 0 then the set of pairs is updated with the new critical pairs obtained pairing N with all the elements of the current Gröbner basis; furthermore the element N is added to the Gröbner basis. The algorithm terminates when the list *IPs* is empty [B65].

The performance of the algorithm depends heavily on the selection strategy (i.e. how to choose the pair to be processed), on the simplification strategy (i.e. how to choose the simplifier during the normalization of a vector when many choices are possible), and on the application of criteria to detect a priori useless pairs (i.e. pairs whose S-polynomial reduces to 0).

The following CoCoAL function implements the above algorithm. It uses some auxiliary functions whose behaviour is assumed to be the one described in Remark 1.

```
Define Buchberger_Classic(L)
  IPs := MakeAllPairs(L);
  GBasis := L;
  While IPs <> 0 Do
    P := GetFirst(IPs);
    V := SPoly(P);
    N := NF(V,GBasis);
    If N <> 0 Then
      IPs := UpdatePairs(GBasis,IPs,N);
      Append(GBasis,N);
    End;
  End;
  Return GBasis;
End;
```

Remark 1 The auxiliary functions used by *Buchberger_Classic(L)* behave as follows:

- the function *MakeAllPairs(L)* returns the list of the critical pairs of elements in L.

- the function *UpdatePairs(GBasis,IPs,N)* updates the list of pairs *IPs* by adding the new pairs which involve the vector N and elements of *GBasis*;

- the function *NF(V,GBasis)* returns the normal form of V with respect to the rules *GBasis*

- the function *SPoly(P)* returns the S-polynomial corresponding to the pair P.

- the function *GetFirst(L)* returns the first element of the list L and discards it from L.

- the function *Append(L,X)* appends to the list L the element X.

Remark 2 The functions *MakeAllPairs(L)* and *UpdatePairs(GBasis,L,N)* may also take care of applying the criteria for detecting useless pairs [B85, GM89].

A selection strategy can be implemented by keeping the list *IPs* sorted with respect to some fixed ordering [GMNRT].

The use of the function *Append(L,X)* implies the simplification strategy that chooses the "oldest" simplifier.

3.3 The Implementation of the Buchberger Algorithm in the system CoCoA

As we said earlier, we wish to have an implementation of the Buchberger algorithm which allows us to do the computations "step by step" and allows us to implement other Buchberger-like algorithms as simple variations of the basic algorithm; computations made using such algorithms will be called *Gröbner computations*.

First of all we need to have convenient data structures. In particular we defined a structure, called "module", which contains all the relevant information about the mathematical module itself (for example one of its systems of generators and, when they are known, its Gröbner basis, its Poincaré series, etc.), but also it is the container of a Gröbner computation in progress. This means that all the information (e.g. critical pairs or reductors) is stored in the module. Hence we think of a module M as a record (in the computer science sense) and we denote respectively by *M.Gens*, *M.GBasis*, *M.IPs* the generators, the Gröbner basis, and the list of critical pairs of M. We organized the data structures with two more lists *M.Rules* and *M.IVs*. The first is the set of the rules (reductors) used to perform the normalization and is in general a superset of *M.GBasis* (we prefer to keep the two lists separated); the latter is the list of the *incoming vectors* which is initialized with the generators of the module; this list is particularly useful in the homogeneous context (see Sect. 3.4).

With this setting, the algorithm *Buchberger_Classic* of Section 3.2 is an instantiation of the following parametric algorithm *Buchberger_Abstract*, when an appropriate set of key functions is assigned. This presentation of the algorithm highlights that there is an initialization phase followed by a loop in which the computation is performed one step at a time. The boolean function <u>*OneStep*</u> returns TRUE if and only if some computation has been performed, otherwise it returns FALSE and the loop terminates.

```
Define Buchberger_Abstract(M)
   Initialize(M);
   Repeat
      B := OneStep(M)
   Until Not B;
End;
```

The algorithm puts the result of the computation in the module M passed as input. For example the Gröbner basis is stored in *M.GBasis*.

We say that this algorithm is abstract in the sense that it depends on the "parameter functions" *Initialize* and *OneStep* whose actual values are assigned at run time depending on the kind of Gröbner computation requested. Once the user has assigned to the module the setting of key functions, the computation will use these functions without having to examine any option. Thus to execute *Buchberger_Classic*, the functions *Initialize* and *OneStep* are "instantiated" (i.e. substituted) with the following two functions.

```
Define Initialize_GB(M)
   M.IVs := M.Gens;
   M.IPs := [];
   M.Rules := M.Gens;
   M.GBasis := M.Gens;
End;

Define OneStep_GB(M)
   If M.IVs <> [] Then
      V := GetFirst(M.IVs);
      UpdateBasisAndPairs(M,V);
      Return TRUE
   End;
   If M.IPs = [] Then Return FALSE End;
   P := GetFirst(M.IPs);
   V := SPoly(P);
   ProcessVector(M,V);
   Return TRUE;
End;

Define ProcessVector(M,V)
   W := NF(V,M.Rules);
   If W <> 0 Then
      IPs := UpdateBasisAndPairs(M,W);
```

Else
CaseZero(M,W);
End
End;

The mechanism of "instantiation" works as follows. The user calls a function named *Start(M,KFL)* passing to it a module and a list of key functions; more precisely each key function is identified by a positive integer, and so *KFL* is a list of positive integers. This list is stored in the module and then every time we call a parameter function *KeyFun* it means that we are calling the key function assigned to *KeyFun*.

After the key functions of the module have been assigned with the function *Start*, the user can execute the complete computation by calling the function *Complete*. Since all the information about the computation in progress (i.e. *M.IVs*, *M.IPs*, *M.Rules* and *M.GBasis*) is kept in the module it is also possible to execute the computation "interactively" by repeatedly calling the function *OneStep*. So the user can examine the partial results.

Let's examine the key function *OneStep_GB*. First all the incoming vectors (which were initialized to the generators of the module) are removed one by one and the Gröbner basis, the pairs and the rules are updated with them (by using the key function *UpdateBasisAndPairs*). Each successive step takes the first incoming pair, compute its S-polynomial, and then processes it by calling the auxiliary function *ProcessVector*.

We are using the parameter functions *NF*, *SPoly*, *UpdateBasisAndPairs* and *CaseZero* because we want to be ready to adapt the algorithm to compute the syzygy module of *M.Gens*.

If we want to retrieve the algorithm *Buchberger_Classic* then the key functions assigned to the parameter functions *NF* and *SPoly* have to behave as explained in Remarks 1 and 2. The function *UpdateBasisAndPairs*(M,V) updates the list of the incoming pairs (as *UpdatePairs*) and adds the new element V to *M.GBasis* and to *M.Rules*.

Remark 3 The key function assigned to *UpdateBasisAndPairs* in its turn uses some parameter functions; among these there are *Add_To_Rules* used to add new vectors to the list *M.Rules* and *Pair_Prec* used to keep the list of the incoming pairs sorted with respect to a fixed ordering. By using different key functions for the parameter function *Add_To_Rules* we can apply different simplification strategies; the function *Pair_Prec* is used to keep the list of the incoming pairs sorted; different key functions for *Pair_Prec* allow us to implement different orderings and hence different selection strategies.

Let's now introduce another set of key functions which specializes the abstract algorithm to one which computes the syzygy module of the module generated by a given list of vectors.

If V is a vector belonging to the module generated by $L := \{V_1, \ldots, V_r\}$, then there exists a tuple $Rel := (P_1, \ldots, P_r)$ of polynomials such that $V = \sum_i P_i V_i$. We say that *Rel* is a "relation" of V in terms of L.

It is well known [S80, S77] that one can use the Buchberger algorithm also for the computation of the syzygies: it suffices to keep track of at least one relation (in terms of the given generators) for each rule, S-polynomial and vector in reduction.

So the key functions (say *SPoly_Syz* and *NF_Syz*) corresponding to the parameter functions *SPoly* and *NF* must keep the above relations updated. More precisely each arithmetical operation involving a vector V must be performed also on the corresponding relation.

When a vector reduces to 0, then we have a relation for 0 in terms of the generators L, and hence we have found a syzygy of the elements of L. Collecting all these syzygies, we get a system of generators for the syzygy module of L. So the key function for *CaseZero* will have to add the found syzygy to the generators of the module *M.Syz*.

Define *CaseZero_Syz(M,N)*
  If *N.Res <> 0* **Then** *Append(M.Syz.Gens,N.Rel)* **End***;*
End*;*

Note that only the key functions *SPoly_Syz, NF_Syz, CaseZero_Syz* have to be implemented to achieve the computation of the syzygy module.

3.4 The Homogeneous Algorithms

In this section we assume that the ring $R := k[X_1, \ldots, X_n]$ is graded over $\mathbf{N}$ by $\deg(X_i) > 0$ for $i = 1, \ldots, n$. Let M be an $\mathbf{N}$-graded submodule of a finitely generated graded free R-module given by a homogeneous system of generators. We consider the following problems for M: a) to find a minimal system of generators; b) to compute a Gröbner basis of M; c) to find the syzygy module of a minimal system of generators or of the given generators; d) to compute the minimal free resolution.

Algorithmic answers to these problems are been presented in Chapter 4. Here we show how we implemented these algorithms in CoCoA 3 (in the framework of the abstract Buchberger algorithm) by means of suitable key functions which take into account the homogeneity of the module.

The general idea is to sort the list of incoming vectors *M.IVs* by increasing degree and construct the Gröbner basis in the same order. Processing an incoming vector V means to normalize it with respect to *M.GBasis* and then to do some action depending on its nullity.

Processing a critical pair is essentially the same as in the non-homogeneous case, but we can use the extra knowledge that the degree does not change during reduction.

In this case a degree-compatible strategy is used; this means that the elements of both the incoming queues are processed by increasing degree, and, inside each degree, first all the incoming pairs are processed and then the incoming vectors. If a degree-compatible strategy is used to compute a Gröbner basis of M, then, after passing a certain degree d, a d-truncated Gröbner basis of M is already computed.

The pair of key functions *Initialize_Hom* and *OneStep_HGB* is used when we want to compute a Gröbner basis of the homogeneous module M using a degree-compatible strategy.

Define *Initialize_Hom(M)*
  M.IVs := M.Gens;
  M.IPs := [];
  M.Rules := [];
  M.GBasis := [];
End;

Define *OneStep_HGB(M)*
  If *M.IPs = []* **And** *M.IVs = []* **Then Return** *FALSE* **End**;
  V := GetFirstToProcess(M);
  ProcessVector(M,V);
  Return *TRUE;*
End;

The function *GetFirstToProcess* selects an element of the queues *M.IPs* and *M.IVs* using the degree-compatible strategy: if it is a pair, then the function returns the corresponding S-polynomial.

Capani et al. [CDNR97] showed how to obtain a *minimal* set of critical pair which have to be processed by the Buchberger algorithm in the homogeneous case. This leads to an improved version of the criteria for detecting useless pairs described in [GM89]. Since we want to use different criteria in the homogeneous and non-homogeneous case, the part of *UpdateBasisAndPairs* related to the criteria has been parametrized.

If we want to compute just a minimal system of generators of M, then, by Proposition 4 of [CDNR97], it suffices to assign the key function *Initialize_Hom* to *Initialize* and the following key function to *OneStep*:

Define *OneStep_MinGens(M)*

If *M.IVs* = *[]* **Then Return** *FALSE* **End**;
V := GetFirstToProcess(M);
ProcessVector(M,V);
Return *TRUE;*
End;

We conclude this section by sketching the implementation in our environment of the basic algorithm for computing minimal free resolutions proposed by [CDNR97]. This algorithm takes as input a homogeneous submodule M of a finitely generated graded free module, and gives as output the resolution as a chain of graded modules $M_0, M_1, \ldots, M_r$ such that $M_0 = M$, and for each $i \geq 1$, M_i is the i^{th}-order syzygy module of M i.e. the syzygy module of a minimal system of generators of M_{i-1}. Since each module, as a CoCoA data type, can store inside itself the syzygy module of one of its minimal systems of generators, after the computation the whole resolution can be retrieved in M.

We look at a resolution as a growing bidimensional table having a column for each module M_i and a row for each degree. The rightmost column corresponds to M_0.

So the cell (i, d) of the table (denoted by $cell(i, d)$) represents the degree d part of the module M_i and it contains the incoming pairs and the incoming vectors of degree d of M_i. The algorithm follows a "reading pattern strategy", i.e. the cells are processed left to right and up to down.

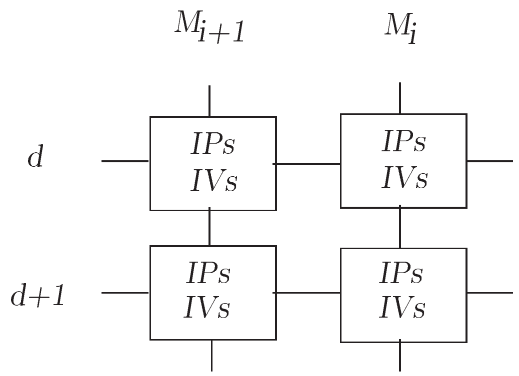

Figure 3.1: The computation matrix

The algorithm is informally described as follows [CDNR97].

Suppose we have completed all the top part of cell$(i{+}1, d)$ *i.e. we have processed* *all the critical pairs related to it. Then we enter* cell(i,d) *where we find critical pairs in the top part and vectors in the bottom part. Some of the critical pairs produce new elements in the Gröbner Basis of* M_i*. The others produce syzygies, i.e. incoming vectors in the* cell$(i+1,d)$*. These vectors may reduce to* 0 *or produce new minimal generators of* M_{i+1}*. In the last case they possibly produce new critical pairs in the cells* cell$(i+1,\delta)$*, with* $\delta > d$*. Then* cell$(i+1,d)$ *is completely done and we enter* cell$(i-1,d)$*.*

Looking at the processing of an element (pair or vector) of *cell*(i,d) as a single step of computation, we can obtain the above algorithm from *Buchberger_Abstract* by assigning to the parameter function <u>*OneStep*</u> the following key function *OneStep_Res* which operates on the rows of the table by increasing degree, calling the auxiliary function *Do_OneStep_Row*.

Define *OneStep_Res(M)*
  D = Min_Deg_Incomings_Res(M);
  If *D = NULL* **Then Return** *FALSE* **End***;*
  Return *Do_OneStep_Row(M,D);*
End*;*

Define *Do_OneStep_Row(M, D)*
  If *M.Syz* <> *NULL* **Then**
    B := Do_OneStep_Row(M.Syz,D);
    If *B* **Then Return** *TRUE* **End***;*
  End*;*
  If *M.IPs = []* **And** *M.IVs = []* **Then Return** *FALSE* **End***;*
  DF := Min_Deg_Incomings_Res(M);
  If *DF = D* **Then**
    V := GetFirstToProcess(M);
    ProcessVector(V);
    Return *TRUE*
  Else
    Return *FALSE*
  End
End*;*

The function *Do_OneStep_Row* reduces an element of the minimum degree D of the highest order syzygy module (the leftmost cell of the first row to be processed) by recursively calling *Do_OneStep_Row* on the syzygy before trying to reduce an element of the "current" module. The function *Min_Deg_Incomings_Res* returns the minimum degree of the incoming elements.

3.5 Final Remarks

In the previous sections we have outlined our framework for the Gröbner computations. The actual implementation uses more parameter functions for still greater flexibility. For example we have implemented a refined notion of "step" by using a parameter function *Reduce* (instead of *NF*) which allows us to consider the application of a single rule as a single step of computation. Moreover we have implemented a set of key functions that inspect some special variables stored in the module called "Truncations" and stop the execution when the truncation is reached. For example it is possible to specify a "Degree Truncation" in the variable *M.DegTrunc*: the execution stops when all the incoming pairs or vectors have degree greater that M.DegTrunc. It is also possible to specify the maximum length of the resolution in the variable *M.ResTrunc* and the Castelnuovo regularity in the variable *M.RegTrunc*.

The minimal interface of CoCoA with the Gröbner framework is given by the functions *Start*, *OneStep* and *Complete*. The use of these functions together with the large number of key functions available allow the experienced user to customize the Gröbner environment.

However the system also offers to the user several predefined settings of key functions for the most common usage. For example the user may compute a Gröbner basis of a module M by typing the CoCoA command `GBasis(M)`; this command selects the appropriate setting of key functions taking into account all the information already present in M: for example it check if it is homogeneous, if there are truncations, if its Poincaré Series is present, and so on.

Furthermore the system offers a CoCoAL package containing several high level functions which allow the user:

- to execute a given number of steps of computation;
- to compute for a given time;
- to obtain summary reports about the current status of the computation (how many IVs, IPs, GBasis elements and so on)

Chapter 4

Computing Minimal Finite Free Resolutions

In this chapter we address the basic problem of computing Minimal Finite Free Resolutions of homogeneous submodules of graded free modules over polynomial rings. We develop a strategy, which keeps the resolution `minimal at every step`. Among the relevant benefits is a marked saving of time, as the first reported experiments in CoCoA show. The algorithm has been optimized using a variety of techniques, such as minimalizing the number of critical pairs and employing an "ad hoc" Hilbert-Driven strategy. The algorithm can also take advantage of various a priori pieces of information, such as the knowledge of the Castelnuovo regularity.

4.1 Introduction

In this chapter we address the basic problem of computing Minimal Finite Free Resolutions (MFFR) of homogeneous submodules of graded free modules over polynomial rings. The theory of Finite Free Resolutions has a long history; it goes back to the fundamental paper by Hilbert ([H]) and is a cornerstone in Algebraic Geometry and Commutative Algebra, since many geometric invariants can be calculated from such resolutions. Despite their outstanding importance, it was only in the 1980's that the first attempt to actually compute them was tried. For instance [RV83] considered some computational problems involved, but only for very special ideals. Then in [MM86] the authors discussed a new computational approach to the problem, which worked in general. The first implementations of the MFFR-algorithms, done in Macaulay 3.0 [BS92], and CoCoA 1.5 [GN] were unsatisfactory for various reasons. Recently we have seen new ideas introduced to deal with this problem, which lead to more efficient implementations (see [GGMNPPSS]). Other researchers are working on the subject, in particular we are aware of some work in progress by La Scala [L96], whose algorithm has been recently implemented in Macaulay 2, and by Caboara-Traverso [CT].

Our idea can be briefly described by saying that our MFFR algorithm features a strategy, which has a great deal of flexibility and keeps the minimality at every step. More specifically, our algorithm is a combination of the following ingredients:

- the use of a minimal set of critical pairs, as explained in Section 4.3;
- the use of left-induced term-orders (see Section 4.4), as suggested many years ago by Schreyer ([S80]);
- the use of Hilbert Functions to discard *Type* S_0 critical pairs (see Section 4.5);
- a "top first" strategy, which first computes the top part of every computational cell, i.e. the part which manages the "critical pairs" and then the bottom part, i.e. the part which manages the "incoming vectors";
- the possibility of placing a priori bounds in the "resolution matrix", when some extra information, such as the Castelnuovo regularity, is known.

The default uses a "reading pattern strategy", i.e. a strategy which computes in increasing degree and from left to right. However the flexibility is such that completely different strategies can be used so that the algorithm may benefit from other sources of knowledge.

The chapter is structured in the following way. After having discussed some more or less well known facts in Section 4.2, we address the problem of minimalizing the set of the critical pairs (see Section 4.3). Then we propose an algorithm to compute a minimal set of syzygies of a graded module (see Section 4.4). The algorithm embodies a subroutine which uses Hilbert-Poincaré series as a tool for discarding useless critical pairs (see Section 4.5), and one should be aware of the fact that useless critical pairs are not the pairs which reduce to zero in Buchberger's Algorithm, but pairs which, by reducing to zero, produce a syzygy, which reduces to zero. Then Section 4.6 is devoted to collecting all the ideas coming from the previous parts and describing how to compute in the "computational zone" of the so called "resolution matrix". All the algorithms described in the chapter were implemented in CoCoAL, the programming language of CoCoA, before being coded in the kernel of CoCoA 3 [CNR96]. A set of examples, almost all of which come from specific problems in Algebraic Geometry, is the basis for our table of timings. It is described in Section 4.7, and shows the excellent behaviour of our algorithm, which, for instance, performs with many orders of magnitude better than the old CoCoA.

4.2 Some preparatory results

Let us start by considering the following situation. Let $R := k[X_1, \ldots, X_n]$ be the polynomial ring over a field, graded over $\mathbf{N}$ by $deg(X_i) > 0$ for $i = 1, \ldots, n$ and let L denote a finitely generated graded free R-module. Then $L \cong \bigoplus_{i=1}^{r} R(-a_i)$, where the a_i's are the degrees of a set of homogeneous generators of L.

Let σ be a term-order on the set of power products of L, i.e. elements of the form $(0, \ldots, 0, t, 0, \ldots, 0)$, where t is a power product in R.

If M is a submodule of L, we denote by $LT_\sigma(M)$ the leading term module of M, i.e. the submodule of L generated by $\{LT_\sigma(m) \mid m \in M, m \neq 0\}$.

If M is an $\mathbf{N}$-graded submodule of L, then $M = \bigoplus_{d \in \mathbf{N}} M_d$ and the elements of M_d are called homogeneous elements of degree d of M; we make the following

CONVENTION 1: Homogeneous generators of a graded module are written in non decreasing order with respect to their *degree*.

In practice it is clear that the generators, however given, can be rearranged to meet the requirements of Convention 1.

Suppose now that we want to compute a Gröbner Basis of M by means of Buchberger's Algorithm.

Definition 4.2.1 *A strategy which computes a Gröbner Basis in non decreasing degree is called a degree-compatible strategy. A strategy which always makes full reduction, i.e. it stops the reduction of every element when no power product in the support can be reduced, is called a fully-reductive strategy.*

CONVENTION 2: We assume that we always use degree-compatible fully-reductive strategies.

While the use of a degree-compatible strategy is essential in the following, the use of a fully reductive strategy is only optional.

Definition 4.2.2 *We use the expression "d-truncated Gröbner Basis" of M to mean the subset of all the elements of a Gröbner Basis of M having degree not greater than d and we denote it by $G_{\leq d}(M)$. We call $M_{\leq d}$ the module generated by the elements of M of degree not greater than d.*

It is very well-known that if M is a graded module and a degree-compatible strategy is used to compute a Gröbner Basis of M, then, after passing a degree d, a d-truncated Gröbner Basis of M is already computed.

Lemma 4.2.3 *Let M be a graded submodule of a graded free module L and let $G_{\leq d}(M)$ be a d-truncated Gröbner Basis of M. Let $\{n_1, \ldots, n_s\}$ be homogeneous elements of L of degree d, such that the leading terms of the n_i's are*
a) pairwise different;
b) not divided by any leading terms of the elements of $G_{\leq d}(M)$.
Let $M' := M + (n_1, \ldots, n_s)$. Then $G_{\leq d}(M) \cup \{n_1, \ldots, n_s\}$ is a d-truncated Gröbner Basis of M'.

Proof 4.2.4 *A simple analysis of the possible critical pairs leads to the conclusion. Q.E.D.*

Proposition 4.2.5 *Let $M, M', n_1, \ldots, n_s$ be as in Lemma 4.2.3; assume that M is minimally generated by $\{m_1, \ldots, m_r\}$. Then M' is minimally generated by $\{m_1, \ldots, m_r\} \cup \{n_1, \ldots, n_s\}$.*

Proof 4.2.6 *The only possibility against minimality is that there is a linear combination $\sum a_i n_i + \sum b_j m_j = 0$, with $a_i \in k$ for $i = 1, ..., s$ and some of the a_i's different from 0. But then the leading term of $m := \sum a_i n_i$ is the leading term of one of the n_i's, hence is not divided by any of the leading terms of elements of $G_{\leq d}(M)$. At the same time m can be written as $-\sum b_j m_j$, hence it reduces to 0 via $G_{\leq d}(M)$. This is a contradiction. Q.E.D.*

4.3 Computing a Minimal Set of Critical Pairs

Our next goal is to show that the results of the previous section can also be used to obtain a *minimal* set of critical pairs.

Let $\{m_1, \ldots, m_r\}$ be a set of homogeneous elements of a graded free module $L_0 \cong \bigoplus_{i=1}^{s} R(-a_i)$ and let $M := (m_1, \ldots, m_r)$. If $\{e_1, \ldots, e_s\}$ denotes the canonical basis of L_0, then $deg(e_i) = a_i$. We recall that the power products of L_0 are the elements of the form $t \cdot e_i$, where $t = X_1^{p_1} \cdots X_n^{p_n}$ and we explicitly remark that $deg(t \cdot e_i) = deg(t) + a_i$.

Suppose we want to compute a Gröbner Basis of M with respect to a degree-compatible term-order σ on L_0. We have to run Buchberger's Algorithm, so we are interested in the so called *critical pairs*.

We recall that two elements n_i and n_j of L_0 give rise to a critical pair if $LT(n_i)$ and $LT(n_j)$ are of type $s \cdot e_l$ and $t \cdot e_l$, with the same l. We introduce new labels ϵ_i's to indicate the position, so that the critical pair can be described by $-\frac{lcm(s,t)}{s}\epsilon_i + \frac{lcm(s,t)}{t}\epsilon_j$. We simply refer to it as $P(i,j)$ and we observe that its degree is $deg(P(i,j)) := deg(lcm(s,t)) + deg(e_l)$. In this way we embody the critical pairs in a graded module, which we call $\mathcal{M}$.

Given a critical pair $-a\epsilon_i + b\epsilon_j$, of course $LT(a \cdot n_i) = LT(b \cdot n_j)$, so we need a convention to break the tie.

CONVENTION 3: Let $i < j$ and $P(i,j) := -a\epsilon_i + b\epsilon_j$, we put

$$LT(P(i,j)) := b\epsilon_j \ .$$

Corollary 4.3.1 *With the above construction and Convention, we have a term-order on the R-module $\mathcal{M}$ generated by the critical pairs.*

To process the critical pair $P(i,j)$ means to compute the normal form of $-\frac{lcm(s,t)}{s}n_i + \frac{lcm(s,t)}{t}n_j$, where the normal form is taken with respect to the part of the Gröbner basis already computed. We observe that such a normal form is a homogeneous element of degree $deg(P(i,j))$. We also observe that such normal forms depend on the order chosen to process the critical pairs. This suggest the following

CONVENTION 4: We always use the following order on the critical pairs of a given degree: $P(i,j) < P(h,k)$ if *either* $j < k$ *or* $j = k$ and $i < h$.

For every new element n_k added to the Gröbner Basis, we need to add a new label ϵ_k, so that the new pairs $P(i,k)$ can be written in the form $-a\epsilon_i + b\epsilon_k$. This means that for every new element n_k added to the Gröbner Basis, we upgrade the module containing the critical pairs and we obtain a new module $\mathcal{M}_k$. We observe that $\mathcal{M} = \cup_k \mathcal{M}_k$. This does not cause any problem to the algorithm which computes a Gröbner Basis of the critical pairs since each computation among the currently given pairs keeps inside the module $\mathcal{M}_k$. Therefore we can use the results of the previous section to produce a minimal set of generators of the critical pairs. As we have already noted, two elements of the critical pairs produce a critical pair (of critical pairs) only if they are of type $P(i,k), P(j,k)$ (same k).

Now we are ready to state the main result

Theorem 4.3.2 *Let i,j,k be three indices, such that $i < j < k$. Let $P(i,k) := -a\epsilon_i + c_i\epsilon_k$, $P(j,k) := -b\epsilon_j + c_j\epsilon_k$, $P(i,j) := -A\epsilon_i + B\epsilon_j$. There are three possibilities.*
1) If $c_i | c_j$ then $P(j,k)$ can be deleted from the minimal set of pairs.
2) If c_j properly divides c_i then $P(i,k)$ can be deleted from the minimal set of pairs.
3) No divisibility occurs. Then $P(i,j)$ can be deleted from the minimal set of pairs if and only if $GCD(a,b) = 1$.

Proof 4.3.3 *In the first case we have $P(i,k) < P(j,k)$. Let $c_j = d \cdot c_i$. Then $P(j,k) = d \cdot P(i,k) + \alpha \cdot P(i,j)$. Therefore not only $P(j,k)$ can be deleted from the minimal set of pairs, but also it reduces to 0, hence it does not contribute to the Gröbner Basis of the*

pairs.
In the second case let $c_i = e \cdot c_j$ we have $P(i,k) = e \cdot P(j,k) + \alpha \cdot P(i,j)$. We observe that $deg(P(i,k)) > deg(P(j,k))$ and $P(i,k)$ reduces to 0, hence it can be deleted from the minimal set of pairs; moreover it does not contribute to the Gröbner Basis of the pairs.
In the third case we let $C := lcm(c_i, c_j)$ and we see that the critical pair associated to $(P(i,k), P(j,k))$ produces $-(C/c_i)a\epsilon_i + (C/c_j)b\epsilon_j$, which is therefore a multiple of $P(i,j)$, i.e. $((C/c_i)a\epsilon_i + (C/c_j)b\epsilon_j) = R \cdot (-A\epsilon_i + B\epsilon_j)$. If $GCD(a,b) \neq 1$, then the produced pair $-(C/c_i)a\epsilon_i + (C/c_j)b\epsilon_j$ is a strict multiple of $P(i,j)$, hence it reduces to 0 and does not contribute neither to the minimal set of pairs nor to their Gröbner Basis. If $GCD(a,b) = 1$, then the pair $P(i,j)$ is taken out from the minimal set of pairs. Of course it cannot be discarded from the Gröbner basis of $\mathcal{M}$; however the pair $P(i,j)$ was considered before when all the pairs of type $(-,j)$ were considered. Q.E.D.

Remark 4.3.4 *From the preceding Theorem we derive a procedure,* `New_Pairs(K)`*, which builds the new minimal pairs, when a new element is added to the Gröbner Basis, and removes the non minimal ones from the pairs to be processed. If the new element is the K^{th} one, then the computational cost of* `New_Pairs(K)` *is $\mathcal{O}(K)$ divisibility and $\mathcal{O}(K^2)$ coprimality tests between power products. Both the tests can be efficiently performed if the power-products in the components of the critical pairs keep the information of their "square-free part", as it is described in [B97].*

Remark 4.3.5 *The Procedure* `New_Pairs(K)` *can be interpreted as an improved version of Gebauer-Möller[GM89] criteria in the homogeneous case.*

4.4 Computing Minimal Syzygies

Let $\{m_1, \ldots, m_r\}$ be a set of homogeneous elements of L_0, with $d_i := deg(m_i)$, $i = 1, \ldots, r$; $d_1 \leq \cdots \leq d_r$, let $M := (m_1, \ldots, m_r)$, $L_1 := \bigoplus_{i=1}^r R(-d_i)$ and $\{\epsilon_1, \ldots, \epsilon_r\}$ the canonical basis of L_1. Then M can be represented as the image of the map of graded free modules

$$\Phi : L_1 \to L_0 \quad \text{where} \quad \Phi(\epsilon_i) = m_i;\ i = 1, \ldots r$$

We observe that, because of the chosen shifts, Φ turns out to be a graded homomorphism.

Now we consider a term-order σ on L_0; it induces a filtration on L_0. Classical results in the theory of associated graded objects to filtrations (see for instance [S80], [RV83], [Eis95]) suggest the following

Definition 4.4.1 *A term-order τ on L_1 is said to be induced by σ and Φ if the following condition holds: for every power products s and t, then*

$$t \cdot \epsilon_i >_\tau s \cdot \epsilon_j \quad \textit{if} \quad LT_\sigma(tm_i) >_\sigma LT_\sigma(sm_j)$$

A term-order τ on L_1 is said to be the term-order left-induced by σ and Φ if it is defined by:

$$t \cdot \epsilon_i >_\tau s \cdot \epsilon_j \quad \textit{if}$$
$$\textit{either} \;\; LT_\sigma(tm_i) >_\sigma LT_\sigma(sm_j) \quad \textit{or} \quad LT_\sigma(tm_i) = LT_\sigma(sm_j) \;\textit{and}\; i > j$$

CONVENTION 5: Henceforth we will use the term-order left-induced by σ and Φ.

Remark 4.4.2 *The use of left-induced term-orders τ was already suggested by [S80] in his Diploma Thesis. It is certainly not an original contribution of this chapter; however we want to remark that it is a "natural" choice. Namely it is the correct choice to make the morphism Φ compatible with the Gröbner filtrations induced on L_0 and L_1 by σ and τ respectively.*

Definition 4.4.3 *With the above notations and conventions, minimal critical pairs are of two different types, namely*

Type G *if the corresponding normal form is different from zero, hence it yields a new element in the Gröbner basis of M;*

Type S *if the corresponding normal form is zero, hence it yields a syzygy of the given set of generators of M.*

It is well-known (see for instance [AL94]) that *Type S* critical pairs yield generators of the module of syzygies of $\{m_1, \ldots, m_r\}$, but we get more.

Definition 4.4.4 *Let M be minimally generated by $\{m_1, \ldots, m_r\}$. Let be given a procedure which yields syzygies of the m_i's from the Type S critical pairs selected among a minimal set of critical pairs as described in Section 4.3 and processed in the order stated with Convention 4, while computing a Gröbner basis of M; we denote by S_d the set of the corresponding degree-d syzygies. Let G_d be a d-truncated Gröbner Basis of the module generated by $Syz(M_{\leq(d-1)})$. Then a Type S critical pair can be of two different types, namely*

Type S_{min} *if it yields a minimal generator for the module of syzygies*

Type S_0 *otherwise.*

If we apply the results of Section 4.2 to the above described situation, we get the following

Proposition 4.4.5 *Let M be a graded submodule of a graded free module L_0, assume that M is minimally generated by $\{m_1, \ldots, m_r\}$. Then the type S_{min} critical pairs described in Proposition 4.4.4 yield a minimal set of generators of $Syz(m_1, \ldots, m_r)$.*

Proof 4.4.6 *The proof follows from Proposition 4.2.5. Q.E.D.*

Corollary 4.4.7 *Let M be a graded submodule of a graded free module L_0, assume that M is minimally generated by $\{m_1, \ldots, m_r\}$. Then Proposition 4.4.5 yields an algorithm, which computes the minimal syzygies of the given minimal set of generators $\{m_1, \ldots, m_r\}$.*

Corollary 4.4.8 *Let M be a graded submodule of a graded free module L_0, assume that M is generated by $\{m_1, \ldots, m_r\}$. Then a combination of Proposition 4.2.5 and Proposition 4.4.5 yields an algorithm, which we call* `MinSyz` *and which computes the minimal syzygies of a minimal subset of the given generators.*

Remark 4.4.9 *This approach can be implemented by using a "horizontal" strategy, which does the computation degree by degree in both modules. The horizontal approach can be improved if we normalize a syzygy as soon as it is found. To make this remark effective, we must "go from left to right" in the sense that we have to process the pairs of the syzygies in degree d before normalizing the new incoming syzygies in the same degree.*

4.5 The use of the Hilbert Functions

In this section we show how to use information coming from numerical functions to possibly detect, hence discard, *Type* S_0 critical pairs.

Definition 4.5.1 *Let N be a graded submodule of a free graded module over $k[x_1, \ldots, x_n]$. We express the Hilbert-Poincaré series of N as $\mathcal{P}_N = \frac{<N>}{(1-\lambda)^n}$ and we call $< N >$ the "numerator" of N. We denote by $< N_{<d} >$ the numerator of the module generated by $N_{<d}$ and we denote by $< N_{<d} >^-$ the numerator computed via a (d-1)-truncated Gröbner Basis of $N_{<d}$. If $P(\lambda)$ is a series or a polynomial, we denote by $\mathrm{cf}_d(P)$ the coefficient of λ^d in $P(\lambda)$. Finally we denote by $\#(S_{min})_d$ the number of Type S_{min} critical pairs of degree d of N and by $\#(G)_d$ the number of Type G critical pairs of degree d of N.*

When the algorithm `MinSyz` enters a degree d, it computes the d-truncated Gröbner Basis of $Syz(M)_{<d}$. This *is not* the d-truncated Gröbner Basis of $Syz(M)$, since more syzygies can be added by *Type* S_{min} critical pairs. After having computed the d-truncated Gröbner Basis of $Syz(M)_{<d}$, the Algorithm requires to process the critical pairs of M of degree d. At that stage, a minimal set of generators of M of degree not greater than $d-1$ is already computed, say $m_1, \ldots, m_r$ of degrees $d_1, \ldots, d_r$. We denote by L the free graded module $\bigoplus_i R(-d_i)$.

Theorem 4.5.2 *With the preceding notation, the following equalities hold:*
(i) $\mathrm{cf}_d(< Syz(M)_{<d} >)$ + $\#(S_{min})_d$ = $\mathrm{cf}_d(< Syz(M) >)$
(ii) $\mathrm{cf}_d(< M_{<d} >^-)$ + $\#(G)_d$ = $\mathrm{cf}_d(< M_{<d} >)$
(iii) $\mathrm{cf}_d(< L >)$ - $\mathrm{cf}_d(< Syz(M)_{<d} >)$ - $\mathrm{cf}_d(< M_{<d} >^-)$ = $\#(S_{min})_d + \#(G)_d$

Proof 4.5.3 *We consider the following sequences*

$$0 \longrightarrow (Syz(M))_i \longrightarrow L_i \longrightarrow (M_{<d})_i \longrightarrow 0 \tag{4.1}$$

They are clearly exact for $i = 0, \ldots, d-1$*. But we know that* $M_{<d}$ *is presented through a minimal set of generators, hence the syzygies of* M *in degree d do not tie elements of* M *of degree d. Therefore also the* d^{th} *sequence is exact and the additivity of the Hilbert-Poincaré series implies that*

$$\mathrm{cf}_i(\mathcal{P}_L) - \mathrm{cf}_i(\mathcal{P}_{M_{<d}}) - \mathrm{cf}_i(\mathcal{P}_{Syz(M)}) = 0, \quad i = 0, \ldots, d \tag{4.2}$$

But

$$\begin{aligned}&(\mathcal{P}_L - \mathcal{P}_{M_{<d}} - \mathcal{P}_{Syz(M)}) = \\ &\quad (< L > - < M_{<d} > - < Syz(M) >)(1 + n\lambda + \cdots)\end{aligned} \tag{4.3}$$

hence

$$\mathrm{cf}_i(< L >) - \mathrm{cf}_i(< M_{<d} >) - \mathrm{cf}_i(< Syz(M) >) = 0, \quad i = 0, \ldots, d \tag{4.4}$$

Therefore, to prove (iii) it suffices to prove (i) and (ii). Arguing as before, it suffices to prove the corresponding statements for the series instead of proving them for the numerators. Now the proof is easy; namely, when the critical pairs of degree d of $Syz(M)_{<d}$ *have been computed, what is left out to complete the counting are precisely the minimal syzygies added by Type* S_{min} *and the new Gröbner Basis elements of M. Q.E.D.*

The Theorem allows to improve the algorithm `MinSyz` by possibly deleting some of the *Type* S_0 critical pairs (see the Tables at the end of the chapter).

4.6 Minimal Free Resolutions: the "top first" and the "reading pattern" strategy

Now we address the problem of computing minimal free resolutions of graded submodules of free graded modules over polynomial rings.

Of course a possible solution to this problem is to iterate the algorithm `MinSyz`, which computes a minimal set of syzygies. However, to improve the efficiency we have devised a "reading pattern" strategy. Let us explain what it is.

We represent the resolution as a matrix, whose columns are the modules of syzygies (the rightmost is the starting module) and whose rows are the degrees. We call *cell(i,d)* the computational zone of the i^{th} module of the syzygies in degree d and we partition each cell in the top part and the bottom part. The top part contains the minimal critical pairs and the bottom part the vectors. Before starting our computation we eliminate the cells, which are useless because of some extra knowledge, such as the Castelnuovo regularity. Then we start from the lowest degree and we proceed from left to right, computing first in the (i,d) cell, with lowest d and highest i.

Suppose we have completed all the top part of *cell(i+1,d)* i.e. we have processed all the critical pairs related to it. Then we enter *cell(i,d)*, where we find critical pairs in the top part and vectors in the bottom part. Some of the critical pairs (*Type G*) produce new elements in the Gröbner Basis of M. The others (*Type S*) produce syzygies, i.e. incoming vectors in the *cell(i+1,d)*. These vectors may reduce to 0 or produce new minimal generators of $Syz(M)$. In the last case they possibly produce new critical pairs in the cells *cell(i+1,δ)*, with $\delta > d$. Then *cell(i+1,d)* is completely done and we enter *cell(i-1,d)*, if not already discarded because of some extra knowledge.

As we have already pointed out, part of the *Type* S_0 critical pairs are discarded by means of Hilbert funcions. Moreover the above described scheme of computation may be changed, if we want to make full use of some extra pieces of knowledge.

4.7 Examples and Timing

We report the timing obtained with the examples described below. They have been computed with a **Pentium 133Mhz** with **32 Mb** of memory, running under **Linux**. The first column shows the performance of the algorithm itself. The second one, labeled **HDriven**. reports the performance, when the use of the Hilbert functions was enabled; the last one, labeled **Regularity**, shows the timing when we allowed the a priori knowledge of the Castelnuovo regularity. The "equal signs" mean that in the corresponding cases, the algorithm itself plus the Hilbert functions took care of the regularity, so that there was no extra improvement.

The Examples

```
--------------------------------------------------------------------
-- Catalecticant
Use R::=Z/(32003)[z[0..3,0..3,0..3]];
A:=Mat[
  [z[3,0,0], z[2,1,0], z[2,0,1]],
  [z[2,1,0], z[1,2,0], z[1,1,1]],
  [z[2,0,1], z[1,1,1], z[1,0,2]],
  [z[1,2,0], z[0,3,0], z[0,2,1]],
  [z[1,1,1], z[0,2,1], z[0,1,2]],
  [z[1,0,2], z[0,1,2], z[0,0,3]] ];
I:=Ideal(Minors(2,A));
```

$0 \longrightarrow R(-9) \longrightarrow R^{27}(-7) \longrightarrow R^{105}(-6) \longrightarrow R^{189}(-5) \longrightarrow R^{189}(-4) \longrightarrow R^{105}(-3) \longrightarrow R^{27}(-2)$

```
--------------------------------------------------------------------
-- 11 generic points in P^6
Use R ::= Z/(32003)[x[0..6]];
I := GenericPointsIdeal(11,6);
```

$0 \longrightarrow R^4(-8) \longrightarrow R^{18}(-7) \longrightarrow R^5(-5) \oplus R^{25}(-6) \longrightarrow R^{45}(-4) \oplus R(-5) \longrightarrow R^{46}(-3) \longrightarrow R^{17}(-2)$

```
--------------------------------------------------------------------
-- 2x2 Minors of a generic 3x5 matrix
Use R ::= Z/(32003)[x[1..3,1..5]];
A := Mat[[x[I,J]| J In 1..5]| I In 1..3];
I := Ideal(Minors(2,A));
```

$0 \longrightarrow R^6(-10) \longrightarrow R^{40}(-9) \longrightarrow R^{105}(-8) \longrightarrow R^{50}(-6) \oplus R^{120}(-7) \longrightarrow R^{168}(-5) \oplus R^{50}(-6) \longrightarrow R^{210}(-4) \longrightarrow R^{120}(-3) \longrightarrow R^{30}(-2)$

```
--------------------------------------------------------------------
-- Cyclic 5
Use R ::= Z/(32003)[abcdef];
I := Ideal(a+b+c+d+e, de+cd+bd+ae+ab,
cde+bcd+ade+abe+abc, bcde+acde+abde+abce+abcd, abcde-f^5);
```

$0 \longrightarrow R^2(-14) \longrightarrow R^2(-9) \oplus R(-11) \oplus R(-12) \oplus R^4(-13) \longrightarrow R(-6) \oplus R(-7) \oplus R^5(-8) \oplus R(-9) \oplus R^2(-10) \oplus R(-11) \oplus R^2(-12) \longrightarrow R(-3) \oplus R(-4) \oplus R^2(-5) \oplus R^2(-6) \oplus R^3(-7) \oplus R(-8) \oplus R(-9) \longrightarrow R(-1) \oplus R(-2) \oplus R(-3) \oplus R(-4) \oplus R(-5)$

```
-----------------------------------------------------------------
-- Five dense polys of deg 4
Use R ::= Z/(32003)[abcd];
I := Ideal([Randomized(DensePoly(4))| I In 1..5]);
```

$0 \longrightarrow R^{20}(-11) \longrightarrow R^{46}(-10) \longrightarrow R^{10}(-8) \oplus R^{20}(-9) \longrightarrow R^5(-4)$

```
-----------------------------------------------------------------
-- Commute 3x3
Use R::=Z/(32003)[x[1..3,1..3]y[1..3,1..3]];
A := Mat[ [ x[I,J] | J In 1..3 ] | I In 1..3 ];
B := Mat[ [ y[I,J] | J In 1..3 ] | I In 1..3 ];
C := A*B-B*A;
I := Ideal(Concat(C[1],C[2],C[3]));
```

$0 \longrightarrow R^4(-9) \oplus R(-10) \longrightarrow R^{32}(-8) \longrightarrow R^3(-6) \oplus R^{58}(-7) \longrightarrow R^{32}(-5) \oplus R^{28}(-6) \longrightarrow R^2(-3) \oplus R^{31}(-4) \longrightarrow R^8(-2)$

```
-----------------------------------------------------------------
-- Cyclic 6
Use R ::= Z/(32003)[gabcdef];
I := Ideal(Ideal(a+b+c+d+e+f ,
ab+bc+cd+de+ef+fa ,
abc+bcd+cde+def+efa+fab ,
abcd+bcde+cdef+defa+efab+fabc ,
abcde+bcdef+cdefa+defab+efabc+fabcd ,
abcdef+g^6 ));
```

$0 \longrightarrow R^2(-20) \longrightarrow R^2(-14) \oplus R^{10}(-19) \longrightarrow R^{10}(-13) \oplus R(-16) \oplus R^4(-17) \oplus R^{15}(-18) \longrightarrow R(-10) \oplus R^4(-11) \oplus R^{16}(-12) \oplus R(-13) \oplus R^2(-14) \oplus R^3(-15) \oplus R^5(-16) \oplus R^9(-17) \longrightarrow R(-6) \oplus R(-7) \oplus R^2(-8) \oplus R^4(-9) \oplus R^6(-10) \oplus R^{11}(-11) \oplus R^2(-12) \oplus R^2(-13) \oplus R^2(-14) \oplus R(-15) \oplus R^2(-16) \longrightarrow R(-3) \oplus R(-4) \oplus R^2(-5) \oplus R^2(-6) \oplus R^3(-7) \oplus R^3(-8) \oplus R^2(-9) \oplus R^3(-10) \oplus R(-11) \longrightarrow R(-1) \oplus R(-2) \oplus R(-3) \oplus R(-4) \oplus R(-5) \oplus R(-6)$

```
------------------------------------------------------------------
-- Generic Entries
Use R ::= Z/(32003)[xyzt];
M := 3; N := 4;  P := NewMat(M,N);
For I := 1 To M Do
  For J := 1 To N Do;  P[I,J] := Randomized(DensePoly(2));
  End;
End;
I := Ideal(Minors(2,P));
```

$$0 \longrightarrow R^{17}(-8) \longrightarrow R^{48}(-7) \longrightarrow R^{16}(-5) \oplus R^{32}(-6) \longrightarrow R^{18}(-4)$$

The Timing

Name	Normal	HDriven	Regularity
Catalecticant	5.59	5.31	=
11 points in P^6	55.88	34.58	=
2x2 of 3x5	6.84	7.57	=
Cyclic 5	6.94	1.28	0.93
5 Dense	9.01	7.49	=
Commute 3x3	3.70	2.31	=
Cyclic 6	351.54	134.89	=
Generic Entries	14.56	9.75	5.86

The Statistics

Name	MP	GP	MS	$\mathbf{M}S_{min}$	$\mathbf{M}S_0$	$\mathbf{HK}S_0$	$\mathbf{Hd}S_0$
Catalecticant	902	111	791	616	175	165	10
11 points in P^6	257	46	211	144	67	62	5
2x2 of 3x5	1350	152	1198	869	329	181	148
Cyclic 5	166	34	132	34	98	71	27
5 Dense	204	52	152	96	56	33	23
Commute 3x3	376	69	307	191	116	56	60
Cyclic 6	374	61	313	137	176	82	94
Generic Entries	185	39	146	113	33	29	4

MP := Number of Minimal Pairs, i.e the pairs which survived after minimalizing them;

GP:= Number of Gröbner Pairs, i.e the pairs which produced Gröbner (non minimal) elements;

MS:= Number of minimal *Type S* pairs, i.e. the pairs which produced syzygies. Of course **MP**= **GP**+ **MS**;

$\mathbf{M}S_{min}$:= Number of minimal *Type* S_{min} pairs, i.e. pairs which produced necessary minimal i^{th}-syzygies for every i;

$\mathbf{M}S_0$:= Number of minimal *Type* S_0 pairs, i.e. pairs, which produced useless syzygies. Of course $\mathbf{MS} = \mathbf{M}S_{min} + \mathbf{M}S_0$;

$\mathbf{HK}S_0$:= Number of H–Killed *Type* S_0 pairs, i.e pairs killed by the use of the Hilbert functions, as described in section 4.5;

$\mathbf{Hd}S_0$:= Number of Hard–*Type* S_0 pairs, i.e useless pairs, which were not detected by the Hilbert functions. Of course $\mathbf{M}S_0 = \mathbf{HK}S_0 + \mathbf{Hd}S_0$.

Chapter 5

The EEE representation

In this chapter it is presented a new data structure for the representation of the multivariate ordered polynomials which is particularly compact, scalable and efficient.

5.1 Introduction

The majority of the computer algebra systems use the following representations for the multivariate polynomials ([BGS96]).

Representation	Applications	Systems
Dense Distributive	Gröbner Bases	CoCoA, SINGULAR, PoSSo
Sparse Distributive	communication between systems	Macaulay 2 (front end)
Dense Recursive	factorization	FACTORY
Sparse Recursive	factorization	Macsyma, FACTORY
Expression Tree	simbolic computation in general	Maple, Mathematica
EEE	arithmetics, Gröbner Bases	CoCoA

As explained in Sect. 1.3.1 to compare two terms we need to compute the corresponding vectors of the scalar products, by multiplying them with the given ordering matrix.

This operation turns out to be very costly, and hence a possible approach for the representation of the support of a polynomial may be based on the direct representation of the terms with the corresponding scalar products. To come back to the representation with the exponents it is sufficient to multiply the scalar products by the inverse of the ordering matrix (which must be invertible to have a complete ordering).

The ordering matrix represent a morphism between rings which maps regular polynomials in Laurent polynomials (that is polynomials which may have negative exponents).

The Laurent polynomials result lexicographically sorted. Hence may be used a recursive representation.

In general the recursive representation results to be very efficient for the sorting of the monomials. But the overhead of the structure is too expensive.

In this chapter we present a new representation for multivariate polynomials, which uses a mixed approach. The recursion is stopped at a given *level* and then the polynomial has a tree representation for the first m indeterminates and has a distributive representation for the other $n - m$ indeterminates.

Even if EEE is quite complex to implement its has a mathematical structure and hence it is simple to describe and presents a stable behaviour,

An intensive experimentation has highlighted a very good spatial complexity; this fact suggests the use of EEE as a protocol for the trasmission of polynomials between computation systems [BGS96].

The time complexity results to be particularly good in some significative classes of problems, such has the computation of the Gröbner basis of *multi-homogeneous* ideals, and the problems which requires non standard orderings (and hence the ordering matrix has to be used for comparing terms).

5.2 Representation of the Support

There are different ways of representing the support. In this section we present a classification of different representations and we propose the new EEE representation.

5.2.1 Representation of Power Products: Dense vs Sparse

The power products (the monomials without the coefficients) may be represented as the vectors of all the exponents of the given indeterminates (dense representation) or as linked lists of the indeterminates with non-zero exponents (sparse representation). The second choice is useful for sparse problems in which the number of indeterminates with non-zero exponents is very small with respect to the number of the given indeterminates.

5.2.2 List Representation

The classic representation of the multivariate ordered polynomials is a linked list of monomials, where the power product in each monomial is represented by a vector of **exponents**. The list is ordered accordingly to the given ring ordering.

Example 5.2.1 *Let R be $Q[xyzt]$ and the* DegRevLex *ordering. Consider the polynomial*

$$\text{ExamplePoly } := y^2z^3 + 2x^2y + 3x^2t + y^2t + xt$$

The ordering is defined by the matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$$

The representation of P is:

```
o--[0,2,3,0]-->1
|
+--[2,1,0,0]-->2
|
+--[2,0,0,1]-->3
|
+--[0,2,0,1]-->1
|
+--[1,0,0,1]-->1
```

The cost M_i of the space used by the i-th monomial in the list representation is C_i+nE+H, where C_i is the cost of the coefficient (which in general is not constant), E is the cost of each exponent n is the number of the indeterminates of the ring and H is the cost of a pointer (each element of the list points to the successive one).

For simplicity we consider $E = H = 1$. So $M_i = C_i + n + 1$.

The overall cost of the list representation of a polynomial P is $\sum_i M_i = \sum_i C_i + N(n+1)$ where N is the cardinality of the support of P.

In this thesis we focus on the representation of the support. The total cost of the coeffients of the polynomial does not depend on the representation of the support and then it is discarded.

So lets define the cost of the list representation of the support of a polynomial P as follows:

$$\text{ListReprSpace}(P) = N(n+1)$$

where N is the cardinality of the support of P and n is the number of the indeterminates.

So

$$\text{ListReprSpace}(\text{ExamplePoly}) = 5(4+1) = 25$$

5.2.3 Scalar Representation

If the ordering is different from lexicographic then the cost of the monomial comparison is quite high.

For special orderings (like DegRevLex, DegLex and few others, which have sparse matrices) it is possible to compute the vector of the scalar products in economic way, but in general we have to build the vector of the scalar products by multiplying the vector of the exponents by the ordering matrix.

For these reasons it is useful to represent a term as the vector of the scalar products. We can retrieve the vector of the exponents by multipling the vector of the scalar products by the inverse of the ordering matrix (which is of course invertible).

Lemma 5.1 *It is easy to see that the representation of polynomials using the scalar products is equivalent to represent the Laurent polynomials in the ring $Q[\overline{x}, \overline{x^{-1}}]$, using the lexicographic ordering.*

The polynomial of the example 5.2.1 is represented as follows:

```
o--[5, 0,-3,-2]-->1
|
+--[3, 0, 0,-1]-->2
|
+--[3,-1, 0, 0]-->3
|
+--[3,-1, 0,-2]-->1
|
+--[2,-1, 0, 0]-->1
```

The space cost for the scalar representation is obviously the same of the list representation, because the number of the scalar products is equal to the number of the indeterminates:

$$\text{ScalarsReprSpace}(P) = (n + 1)N$$

where N is the cardinality of the support of P and n is the number of the indeterminates. So

$$\text{ScalarsReprSpace}(\text{ExamplePoly}) = (4 + 1)5 = 25$$

5.2.4 Recursive Representation

As explained in Sect. 5.2.3 we can always assume that the ordering is lexicographic. In this case we have that the monomials with the same first exponent are consecutive. Moreover, inside each list of monomials with a given first exponent, the monomials with the same second exponent are grouped together. In general the monomials with the same first i exponents are together, for each $i = 1..n$ (where n is the number of the indeterminates of the ring).

These considerations suggest the use a labelled tree for representing the polynomials. Each edge of the tree is labelled with the common exponent of the corresponding subtree.

This tree representation corresponds to the recursive definition of the polynomial ring: we can see a polynomial in $k[x_1, \ldots, x_n]$ as an univariate polynomial in the first indeterminate x_1 and coefficients in the polynomial ring $k[x_2, \ldots, x_n]$, which is in turn defined as an univariate polynomial in the indeterminate x_2 and coefficients in $k[x_3, \ldots, x_n]$, and so on.

The recursive representation of the polynomial of the example 5.2.1 is

```
o--[5]-->o--[0]-->o--[-3]-->o--[-2]-->1
|
+--[3]-->o--[0]-->o--[0]-->o--[-1]-->2
|        |
|        +--[-1]-->o--[0]-->o--[0]-->3
|                           |
|                           +--[-2]-->1
|
+--[2]-->o--[-1]-->o--[0]-->o--[0]-->1
```

Each non-leaf node of the tree consists of an exponent (the label of the subtree) a pointer to the subtree and a pointer to the next sibling. Thus the cost of each non-leaf node is $E + 2H = 3$. The cost of each leaf is $C_i + E = C_i + 1$, the cost of the coefficient plus the cost of the last scalar product. So the cost for the support in the tree representation is

$3I + N$, where I is the number of the internal nodes of the tree and N is the cardinality of the support of the polynomial, that is the number of the leaves.

$$\text{TreeReprSpace}(P) = 3I + N$$
$$\text{TreeReprSpace}(\text{ExamplePoly}) = 3 * 12 + 5 = 41$$

5.2.5 Leveled Representation

In Sect. 5.2.4 we have seen how we can represent a polynomial as a tree with exponents in the nodes and coefficients in the leafs.

This representation is often too expensive in terms of space and time Hence it is useful to "stop" the recursion at a given "level".

More precisely we can represent a polynomial in a ring $(k[x_{l+1}, \dots, x_n])[x_1][x_2] \dots [x_l]$ as a multivariate polynomial in the indeterminates $\{x_1, \dots, x_l\}$ with coefficients in $k[x_{l+1}, \dots, x_n]$. The representation of the coefficients is distributive, while the representation of the terms is recursive. In this case we say that representation is l-leveled.

Given a computation we can "tune" the polynomial representation by chosing different levels.

Example 5.2.2 *The polynomial of the example 5.2.1 can be represented as*

```
o--[5]-->o--[0,-3,-2]-->1
|
+--[3]-->o--[0,0,-1]-->2
|        |
|        +--[-1,0,0]-->3
|        |
|        +--[-1,0,-2]-->1
|
+--[2]-->o--[-1,0,0]-->1
```

Each non-leaf node of the l-leveled recursive representation consists of an exponent (the label of the subtree) a pointer to the subtree and a pointer to the next sibling. Thus the cost of each non-leaf node is $E+2H = 3$. The cost of each leaf is $C_i+(n-l)E = C_i+(n-l)+1$, that is the cost of the coefficient plus the cost of the last $n-l$ exponents, plus the cost of the pointer to the next leaf. So the cost for the support in the l-leveled recursive representation is $3I + (n - l + 1)N$, where I is the number of the internal nodes and N is the number of the monomials of the polynomial, that is the number of the leaves.

$$\text{LevelledTreeReprSpace}(P) = 3I + (n - l + 1)N$$
$$\text{LevelledTreeReprSpace}(\text{ExamplePoly}) = 3 * 3 + (4 - 1 + 1)5 = 29$$

5.2.6 EEE: Space-Optimized Leveled Representation

The cost of the l-leveled representation is very large when the level is close to the number of the indeterminates. On the other hand the spirit of the scalar recursive representation is to build an hierarchical hashing to minimize the cost and the number of the monomial comparisons. We minimize the cost of the comparisons by using the scalar representation; we minimize the number of the comparisons by (recursively) grouping in different buckets the monomials with the same first scalar.

Very often it happens that the coefficient of a monomial in $(k[x_{l+1}, \ldots, x_n])[x_h][x_{h+1}] \ldots [x_l]$ is in turn a single monomial in $(k[x_{l+1}, \ldots, x_n])[x_{h+1}] \ldots [x_l]$.

In that case we would like to join together the two monomials and consider a monomial in $(k[x_{l+1}, \ldots, x_n])[x_h, x_{h+1}] \ldots [x_l]$.

Using this idea we consider a new representation, called EEE, in which the polynomial is given as a sum of monomials in different rings.

The recursive representation of P with maximum level $l = 4$ is

```
o--[5,0,-3,-2]-->1
|
+--[3]-->o--[0,0,-1]-->2
|        |
|        +--[-1,0]-->o--[0]-->3
|                    |
|                    +--[-2]-->1
|
+--[2,-1,0,0]-->1
```

Let S be the number of the scalar products stored in the first l levels of the tree. Let I be the number of the internal nodes. Apart the scalar products, each internal node costs 2, one pointer to the next sibling and the pointer to the first child. Taking into account also the level, the total cost is:

$$\text{EEESpace}(P) = 2 * I + S + (n - l)N$$
$$\text{EEESpace}(\text{ExamplePoly}) = 2 * 7 + 16 = 30$$

5.3 The EEE Representation

In this section we study the EEE representation, used to represent polynomials in $R := C[x_1, \ldots, x_n]$, where C is the ring of the coefficients. This data structure consists in representing polynomials using trees; the word EEE can be pronounced as "triple e" which sounds like "tree poly", but is shorter (i.e. "optimized").

Definition 5.3.1 *A* EEE*-term is a tuple of integers, that is an element of* $T := \bigcup_{n \geq 1} \mathcal{Z}^n$*. We denote a* EEE*-term using square bracket. Example:* $[0, 6, -3]$*. Given two* EEE*-terms* $t = [s_1, \ldots, s_n]$ *and* $t' = [s'_1, \ldots, s'_n]$ *we define the* sum *of the tuples as* $t + t' = [s_1 + s'_1, \ldots, s_n + s'_n]$*. Given two* EEE*-terms* $t = [s_1, \ldots, s_n]$ *and* $u = [s'_1, \ldots, s'_m]$ *we define the* concatenation *of the tuples as* $[t; u] := [t_1, \ldots, t_n, u_1, \ldots, u_m]$*.*

Definition 5.3.2 *Given a coefficients ring C, we define the set M of the* EEE-monomials *over C and the set P of the sums of* EEE*-monomials over C and T. Consider the grammar defined by the following BNF rules (also known as productions).*

(1) $P ::= C \mid M \mid M + L$

(2) $M ::= TC \mid T(P)$

Then P is the language defined by the rule (1), and M is the language defined by the rule (2). Note that the coefficients are not EEE*-monomials: they are* EEE*-sums.*

Definition 5.3.3 *Given a* EEE*-monomial* $m = tp$ *we say that* $\mathrm{coeff}(m) = p$ *is the* coefficient *of m, and* $\mathrm{term}(m) = t$ *is the* term *of m.*

Definition 5.3.4 *The function* fs *retrieves the first scalar product of a* EEE*-monomial.*

- $\mathrm{fs}([s_1, \ldots, s_k]p) := s_1$

Example 5.3.1 *Consider the polynomial* $p := y^2z^3 + 2x^2y + 3x^2t + y^2t + xt$ *in the ring* $R := Q[xyzt]$ *with the "degrevlex" ordering given by the matrix*

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$$

If we drop the names of the indeterminates, we may represent p as EEE*-sum:*

$[0,2,3,0]1 + [2,1,0,0]2 + [2,0,0,1]3 + [0,2,0,1]1 + [1,0,0,1]1.$

We may also represent each term t of p with the corresponding tuple ot its scalar products, that is the new tuple obtained by multiplying the tuple of the exponents of t by the ordering matrix. In this case p can be represented by the following EEE*-sum:*

$[5,0,-3,-2]1 + [3,0,0,-1]2 + [3,-1,0,0]3 + [3,-1,0,-2]1 + [2,-1,0,0]1$

Now we can group together some (sub)terms which start with the same scalar products. Here is the a possible result.

$[5,0,-3,-2]1 + [3]([0,0,-1]2 + [-1,0]([0]3 + [-2]1)) + [2,-1,0,0]1$

The last representation is the sum $m_1 + m_2 + m_3$ of three EEE*-monomials; the term of m_2 is the tuple* $[3]$*; the coefficient of m_2 is in turn a* EEE*-sum, etcetera.*

5.3.1 Tree Representation of the EEE-sums

A EEE-sum can be represented as a tree labelled on the arcs with EEE-terms. A single EEE-monomial $m = tp$ is represented as follows:

```
o--t-->Tree(p)
```

where "o" is the root and Tree(p) denotes the tree representation of the coefficient p.

The representation of a EEE-sum is the following:

```
o--t1-->Tree(c1)
|
+--t2-->Tree(c2)
|
...
|
+--tr-->Tree(cr)
```

A tree is said to be *optimized* if each of its children have in turn zero or at least 2 children. This means that an optimized tree may not have a subtree of the form

```
o--t1-->o--t2-->Tree(c)
```

Example 5.3.2 *The third representation of the Example 5.3.1 is quite hard to read, but its corresponding tree representation is easier.*

```
o--[5,0,-3,-2]-->(1)
|
+--[3]-->o--[0,0,-1]-->(2)
|        |
|        +--[-1,0]-->o--[0]-->(3)
|                    |
|                    +--[-2]-->(1)
+--[2,-1,0,0]-->(1)
```

Expanded Tree Representation

We can see the tree representation of a EEE-sum as a syntetic representation of another tree representation, that we call *expanded tree representation* with labels in $\mathcal{Z}$.

A single EEE-monomial $m = [s_1, ..., s_h]p$ is represented as follows:

```
o--s1-->o--s2-->o--...-->o--sh-->Tree(p)
```

Example 5.3.3 *The expanded tree representation of the example 5.3.2 is*

```
o--[5]-->o--[0]-->o--[3]-->o--[2]-->(1)
|
+--[3]-->o--[0]-->o--[0]-->o--[-1]-->(2)
|        |
|        +--[-1]-->o--[0]-->o--[0]-->(3)
|                           |
|                           +--[-2]-->(1)
|
+--[2]-->o--[-1]-->o--[0]-->o--[0]-->(1)
```

5.3.2 EEE-polynomials

Not all the EEE-sums represent an ordered polynomial. For example $l = [2, 3, 5]3 + [4]8$ does not represent a polynomial because the second EEE-monomial has too few scalar products.

In this section we define the EEE-polynomials as the EEE-sums that satisfy the following properties. They are "balanced", "ordered", "minimal", and "optimized". The first property ensures that we are representing a polynomial, the "ordered" property says that we are representing an ordered polynomial. The "minimality" induces the need of hierarchy (trees). Finally the "optimality" says that all the internal nodes of the (synthetic) tree representation of a EEE-polynomial must have at least two children (in other words we have the minimum number of nodes).

Definition 5.3.5 *Lets now define the* height *of a* $p \in P$, *as the height of its corresponding full-tree (and not the syn-tree).*

- $\text{height}(c) = 0$
- $\text{height}(m + p) = \max(\text{height}(m), \text{height}(p))$
- $\text{height}(tp) = len(t) + \text{height}(p)$

Definition 5.3.6 *A* EEE-*sum is* balanced *if its corresponding full-tree is a balanced tree.*

- $\text{balanced}(c) = \text{true}$
- $\text{balanced}(m_1 + \ldots + m_r) = \text{true}$ *if and only if*

  $\text{balanced}(m_i)\, \forall m_i$ *and*

  $\text{height}(m_i) = \text{height}(m_j)\, \forall m_i, m_j$
- $\text{balanced}(tp) = \text{balanced}(p)$.

Definition 5.3.7 *We say that* $p \in P$ *is* minimal *if each* EEE-*monomial of* p *has a different first scalar product.*

More precisely we define the predicate minimal *as follows:*

- $\text{minimal}(m_1 + \ldots + m_r) = \text{true}$ *if and only if*

  $\text{minimal}(m_i)\, \forall m_i$ *and*

  $\text{fs}(m_i) \neq \text{fs}(m_j)\, \forall m_i, m_j, i > j$
- $\text{minimal}(c) = \text{true}$
- $\text{minimal}(sp) = \text{minimal}(p)$.

Definition 5.3.8 *We say that $p \in P$ is* ordered *if the monomials of p are ordered in decreasing order with respect to their first scalar product.*

More precisely we define the predicate ordered *as follows:*

- $\mathrm{ordered}(c) = \mathrm{true}$
- $\mathrm{ordered}(m_1 + \ldots + m_n) = \mathrm{true}$ *if and only if*

  $\mathrm{ordered}(m_i)\, \forall i \in \{1, \ldots, n\}$ *and*

  $\mathrm{fs}(m_1) \geq \ldots \geq \mathrm{fs}(m_n)$
- $\mathrm{ordered}(tp) = \mathrm{ordered}(p)$.

Definition 5.3.9 *The length of $p \in P$ is the number of monomials, that is:*

- $\mathrm{length}(m_1 + \ldots + m_r) = r$

Definition 5.3.10 *We say that a* EEE*-sum $p \in P$ is* optimized *if the coefficients of all the monomials of p are not in turn single monomials. In other words a* EEE*-sum is optimized if and only if its tree is optimized.*

Here is the definition of the predicate optimized*.*

- $\mathrm{optimized}(m_1 + \ldots + m_r) := \mathit{true}$ *iff* $\mathrm{optimized}(m_i)\, \forall m_i$
- $\mathrm{optimized}(c) := \mathrm{true}$
- $\mathrm{optimized}(tp) := \mathrm{optimized}(p) \wedge (\mathrm{length}(p) > 1)$.

Definition 5.3.11 *We say that a* EEE*-sum p is a* EEE*-polynomial if p is balanced, minimal, ordered, and optimized. We define the set* EEE *of the* EEE*-polynomials as follows.*

$\mathrm{EEE} := \{p \in P \mid \mathrm{balanced}(p) \wedge \mathrm{minimal}(p) \wedge \mathrm{ordered}(p) \wedge \mathrm{optimized}(p)\}$

5.3.3 Sum in EEE Representation

In this section we define the sum of EEE-polynomials. First we define the function opt which is necessary to keep the result optimized (a necessary property for the EEE representation). Then we define $\mathrm{sum}(p, q)$, by induction on the structure of p and q.

Definition 5.3.12 *The following function "optimizes" the* EEE*-monomial* m*, that is force* m *to have the form* tp*, which* p *being a coefficient or a* EEE*-sum of at least two* EEE*-monomials.*

(1) $\mathrm{opt}(t(m+p)) = t(m+p)$

(2) $\mathrm{opt}(t(t'p)) = tt'p$

Definition 5.3.13 *Let* p *and* q *two* EEE*-polynomials such that* $\mathrm{height}(p) = height(q)$*. Then the sum of the two polynomials is defined as follows.*

(0) $\mathrm{sum}(c, c') = c + c'$

(1) $\mathrm{sum}(0, p) = \mathrm{sum}(p, 0) = p$

(2) $\mathrm{sum}(m + p, m' + q) =$

$$\begin{array}{l} m + \mathrm{sum}(p, m' + q) \ \textit{if}\ \mathrm{fs}(m) > \mathrm{fs}(m') \\ m' + \mathrm{sum}(m + p, q) \ \textit{if}\ \mathrm{fs}(m) < \mathrm{fs}(m') \\ \mathrm{sum}(m, m') + \mathrm{sum}(p, q) \ \textit{if}\ \mathrm{fs}(m) = \mathrm{fs}(m') \ \textit{and}\ \mathrm{sum}(m, m') \neq 0 \\ \mathrm{sum}(p, q) \ \textit{if}\ \mathrm{fs}(m) = \mathrm{fs}(m') \ \textit{and}\ \mathrm{sum}(m, m') = 0 \end{array}$$

(3) $\mathrm{sum}(m, m') = \mathrm{sum}(m', m)$

(4) $\mathrm{sum}(tp, tq) = \mathrm{opt}(t\,\mathrm{sum}(p, q))$ *if* $\mathrm{sum}(p, q) \neq 0$ *(0 otherwise)*

(5) $\mathrm{sum}(tp, [t; u]q) = \mathrm{opt}(t\,\mathrm{sum}(p, uq))$ *if* $\mathrm{sum}(p, uq) \neq 0$ *(0 otherwise)*

(6) $\mathrm{sum}([t; u]p, [t; u']q) =$

$$\begin{array}{l} t(up + u'q) \ \textit{if}\ fs(u) > fs(u') \\ t(u'q + up) \ \textit{if}\ fs(u) < fs(u') \end{array}$$

(7) $\mathrm{sum}(tp, t'q) =$

$$\begin{array}{l} tp + t'q \ \textit{if}\ fs(t) > fs(t') \\ t'q + tp \ \textit{if}\ fs(t) < fs(t') \end{array}$$

Consider the rules (4) and (5). It is easy to prove (by induction) that it is sufficient to apply the opt operator to the monomial; it is not necessary to optimize the whole polynomial. This consideration is very important for the efficiency of the implementation of sum.

5.3.4 Product in the EEE Representation

In this section we define the product of two EEE-polynomials.

prod : $\mathrm{EEE}^2 \to \mathrm{EEE}$

(0) $\mathrm{prod}(c, c') = c \cdot c'$

(1) $\mathrm{prod}(0, p) = \mathrm{prod}(p, 0) = 0$

(2) $\mathrm{prod}(m + p, q) = \mathrm{sum}(\mathrm{prod}(m, q), \mathrm{prod}(p, q))$

prod : $M \times \mathrm{EEE} \to \mathrm{EEE}$

(3) $\mathrm{prod}(m, m_1 + \ldots + m_n) = \mathrm{prod}(m, m_1) + \ldots + \mathrm{prod}(m, m_n)$

NOTE: here we construct directly the result. We can do this since $\mathrm{fs}(\mathrm{prod}(m, m_1)) > \ldots > \mathrm{fs}(\mathrm{prod}(m, m_n))$.

prod : $M^2 \to M$

(4) $\mathrm{prod}(m, m') = \mathrm{prod}(m', m)$

(5) $\mathrm{prod}([t; u]p, t'q) = (t + t')\,\mathrm{prod}(up, q)$ if $\mathrm{length}(t) = \mathrm{length}(t') = n > 0$

(6) $\mathrm{prod}(tc, ud) = (t + u)(cd)$

5.3.5 Converting a polynomial to its EEE representation

- $\mathrm{repr}(m + p) = \mathrm{sum}(\mathrm{repr}(m), \mathrm{repr}(p))$
- $\mathrm{repr}(0) = 0$
- $\mathrm{repr}(cx_1^{e_1} \ldots x_n^{e_n}) = \left(M \begin{bmatrix} e_1 \\ \vdots \\ e_n \end{bmatrix} \right) \mathrm{repr}(c)$, where M is the ordering matrix

So the EEE representation (which is unique) is built monomial by monomial by using the sum operator.

It is easy to say that $\mathrm{repr}(p)$ is "balanced", "ordered", "minimal", and "optimized".

5.3.6 Converting a EEE representation to the corresponding polynomial

Consider the ring $R := C[x_1, \ldots, x_n]$. Let p be a EEE-polynomial over R. The representation of p as sum of scalars products is defined as follows:

- $\mathrm{scalars}([s_1, \ldots, s_k]c) := c x^{s_1}_{n-k+1} x^{s_k}_n$
- $\mathrm{scalars}([s_1, \ldots, s_k]p) := \mathrm{scalars}(p) x^{s_1}_{n-h-k+1} x^{s_k}_{n-h}$, where $h = \mathrm{height}(p)$
- $\mathrm{scalars}(m + p) = \mathrm{scalars}(m) + \mathrm{scalars}(p)$
- $\mathrm{scalars}(c) = c$

The following function transforms a sum of scalar products in the corresponding polynomial.

- $\mathrm{exponents}(m + p) = \mathrm{exponents}(\mathrm{exponents}(m), \mathrm{exponents}(p))$
- $\mathrm{exponents}(0) = 0$
- $\mathrm{exponents}(c x_1^{s_1} \ldots x_n^{s_n}) = \left(M \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} \right) \mathrm{exponents}(c)$, where $M^(- 1)$ is inverse of the the ordering matrix

We finally define the semantics of a EEE-polynomial as the polynomial $\mathrm{sem}(p) \in R$ defined as follows:

$$\mathrm{sem}(p) := \mathrm{exponents}(\mathrm{scalars}(p))$$

5.4 Considerations about the Complexity

The EEE representation has its strengt on the exploiting the fact that parts of terms are shared and hence the overall cost in terms of space is less than nN, where N is the number of the terms and n is the number of the indeterminates.

In the special (but relevant) case of the k-homogeneous problems, in which the terms of each polyonomial share the first k scalar products, the cost of the representation is $< (n - k)N$.

5.5 Levels in the EEE representation

In Sect. 5.2.4 we have seen that may be a good idea to stop the recursion at a certain level. The same considerations may be applied to the EEE representation.

Definition 5.5.1 *Let* $R := C[x_1, \ldots, x_n]$*. The* l-leveled EEE representation *of a polynomial* $p \in R$ *corresponds to the* EEE *representation of* p *as a member of* $(C[x_{l+1}, \ldots, x_n])[x_1, \ldots, x_l]$.

5.6 Benchmarks

Arithmetics

Test ID	Description
007-6	product of generic polys of deg 3 and 12 in 6 indets
007-7	product of generic polys of deg 12 and 3 in 6 indets
007-10	product of generic polys of deg 9 and 9 in 6 indets
012-2	determinant MatDim = 7, NumIndets = 7, EntryDegree = 1
012-4	determinant MatDim = 6, NumIndets = 6, EntryDegree = 2
012-5	determinant MatDim = 7, NumIndets = 6, EntryDegree = 2

Recursive Leveled Representation

cpu (seconds)		Levels							
Name	**N**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
007-6	6	3.09	3.39	1.98	1.73	1.57	1.74	2.46	-
007-7	6	21.77	19.34	4.78	2.17	1.66	1.75	2.59	-
007-10	6	67.90	72.34	30.51	21.05	17.53	17.83	26.30	-
012-2	7	19.04	17.31	13.28	11.59	11.60	13.15	16.49	24.05
012-4	6	102.89	93.23	40.20	24.58	22.26	26.14	39.72	-
012-5	6	1031.08	963.08	383.00	212.77	181.20	216.89	331.40	-

size (Kbytes)		Levels							
Name	N	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
007-6	6	682	620	558	497	445	430	554	-
007-7	6	682	620	558	497	445	430	554	-
007-10	6	1480	1345	1211	1079	960	913	1183	-
012-2	7	82	75	68	62	56	53	56	70
012-4	6	272	247	222	199	179	177	226	-
012-5	6	511	465	418	373	335	325	418	-

EEE Representation

cpu (seconds)		Levels							
Name	N	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
007-6	6	2.69	2.19	1.76	1.57	1.46	1.29	1.28	-
007-7	6	17.46	13.74	4.49	2.00	1.48	1.26	1.28	-
007-10	6	61.68	58.18	26.32	19.66	16.14	13.38	13.72	-
012-2	7	18.80	16.37	12.18	11.09	11.33	12.86	13.62	12.33
012-4	6	100.03	83.23	37.98	23.94	22.02	20.07	19.27	-
012-5	6	1060.48	911.82	378.48	213.02	187.12	163.83	158.17	-

size (Kbytes)		Levels							
Name	N	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
007-6	6	620	558	496	436	387	324	317	-
007-7	6	620	558	496	436	387	324	317	-
007-10	6	1345	1211	1077	945	832	678	666	-
012-2	7	75	68	61	55	50	48	46	43
012-4	6	247	222	198	174	167	136	132	-
012-5	6	465	418	372	327	292	246	240	-

Gröbner bases with standard orderings

Test ID	**Description**
002-3	commute 4 x 4, trunc = 6
006-1	Valla example
004-3	cyclic, deg = 6
004-4	cyclic, deg = 7

cpu (seconds)		Levels				
Name	**N**	**0**	**8**	**16**	**24**	**32**
002-3	32	124.35	105.83	108.62	114.55	128.03
006-1	21	3.52	4.14	4.32	-	-

cpu (seconds)		Levels								
Name	**N**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
004-3	7	2.86	2.93	2.86	2.95	3.43	3.99	4.61	3.52	-
004-4	8	361.08	361.64	341.60	336.92	343.00	416.53	466.77	547.79	410.89

size (Kbytes)		Levels									
Name	**N**	**0**	**4**	**8**	**12**	**16**	**20**	**21**	**24**	**28**	**32**
002-3	32	12773	11428	10257	9221	8084	7779	-	8188	8581	7222
006-1	21	110	102	97	93	103	100	87	-	-	-

size (Kbytes)		Levels								
Name	**N**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
004-3	7	166	153	140	136	144	167	180	143	-
004-4	8	4026	3698	3374	3108	2986	3138	3658	3958	3129

Gröbner Bases with Special Orderings

Test ID	Description
005-1	Rees, builtin elim matrix
005-2	Rees, given elim matrix
010-1	Segre
009-2	Chordal Variety, builtin elim matrix
009-3	Chordal Variety, given elim matrix

cpu (seconds)		Levels			
Name	**N**	**0**	**3**	**6**	**9**
005-1	9	24.58	14.72	12.84	12.60
005-2	9	34.58	15.87	11.16	11.02
010-1	6	169.70	148.51	103.14	-

cpu (seconds)		Levels						
Name	**N**	**0**	**3**	**6**	**9**	**12**	**15**	**18**
009-2	18	21.93	22.55	23.17	20.17	21.99	23.58	22.82
009-3	18	23.88	23.54	23.69	21.54	22.13	24.54	22.80

size (Kbytes)		Levels									
Name	**N**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
005-1	9	926	868	804	752	697	635	570	605	650	559
005-2	9	857	794	729	677	618	572	521	557	599	512
010-1	6	3035	2736	2434	2142	1852	1420	1406	-	-	-

size (Kbytes)		Levels						
Name	**N**	**0**	**3**	**6**	**9**	**12**	**15**	**18**
009-2	18	566	522	526	544	591	594	492
009-3	18	566	501	452	457	474	521	424

Chapter 6

The CoCoA Web Interface

In this chapter it is proposed a model for a general interface between people and Computer Algebra Systems (CAS). The main features in our CAS interface are data navigation and the possibility of accessing powerful remote machines. Our model is based on the idea of Session Management We implemented our model using the client-server capabilities offered by WWW. We envision our proposal as being useful for: educational purposes; computing; data navigation; communication of structured data; simulation of operations on distributed systems.

6.1 Introduction

Our goal is the construction of a common *World Wide Web* interface for different *Computer Algebra Systems* (CAS for short) particularly for systems devoted to *Commutative Algebra computations*. We need to use different systems because each of them is powerful for doing particular tasks. We can mention, among non–commercial systems developed by academic research groups, CoCoA[CNR96], *GB*, *Macaulay*[BS92], *Singular*[GPS95] and the *PoSSo server*. All of them are capable of performing basic arithmetic operations on algebraic objects like: multivariate polynomials; rational functions; polynomial matrices; ideals. At the same time, advanced operations like Gröbner bases[B85, Rob88] of ideals and modules, elimination, computation of syzygies and resolution of both ideals and modules are supported in all these systems. In addition some of these systems are designed to perform specific algorithms like, for instance, *Hilbert function*, *Hilbert driven Buchberger Algorithm*[GMRT94] or *Tangent Cone Algorithm*[P94] in a particularly efficient way. These systems are mainly used as research tools but some of them are also currently being used for teaching purposes.

Because of these considerations it is clear that a common way to interact with the previously mentioned systems would be advantageous.

A prototypal version of our interface has been already demonstrated during the Workshop "Calcolo Simbolico" (November 20–21, 1995 Genova Italy) and during the Conference "Algorithmic Algebraic Geometry and Singularity Theory" (Schloss Dagstuhl, Germany, Jan 22 - 26 1996).

6.1.1 Interface requirements

The starting point of our investigation was our experience both as users and as (co)-workers on the CAS system CoCoA. In our experience we noted that the interaction between users and systems is characterized by mixture of hard computations (e.g. operations that involve Gröbner basis computation) and data inspection. Data inspection (or better *data navigation*) is a key point in CAS interfaces since we usually work with huge and complex data. So we have to consider the possibility of different *views* of the same output.

Since many computations are quite hard it is sometimes necessary to use suitable computational resources that are not locally available. So we were challenged by the possibility of adding *client-server capabilities* to the interface.

Another good feature would be, of course, *independence of architectures* (machine, operating system, graphical interface) at least from the client side of the interface.

We focused our attention on the possible use of a WWW interface. This choice gives us an unconditioned independence from the architecture (a user simply runs some browser), hypertextual data navigation, automatic client-server capabilites (due to the use of HTTP servers) and some other advantages like a complete abstraction with respect to the computational resources we are interacting with.

To meet our requirements, we have organized the interactions between users and applications in sets of queries called *sessions.* The collection of all information about sessions and applications is called the Session Database (SDB for short).

The SDB is updated by a server, called the Session Manager (SM for short), via a set of services: creation of new sessions; submission of queries; execution of applications. Starting from the SDB, the user can create reports to organize the outputs of the queries.

The SDB is stored on a single machine. Each operation is performed by running a process on the machine of the SDB.

Its important to note that our interface is interactive, that is, each query submitted to an application process may depend on the previous queries submitted to the same process. Moreover we can start a computation, turn off our computer, and then, possibly after several days, reconnect to the session to see the results.

The WWW interface to the SM is called WebSM. The WebSM is is connected to the SM via a set of *cgi-applications*, one for each service. Furthermore, with the WebSM the reports can be hypertextual.

For use the hardest point to investigate was related to *client-server* capabilities of the interface. If we plan to build such a type of interface, first of all we have to solve some specific problems like *crash recovery*. We will distinguish between local-crash recovery (usually due to net problems or browser crash) and remote-crash recovery (roughly speaking, crash of the SM or failure of cgi-applications).

6.2 The Session Manager

The Session Manager (SM for short) is the main engine of our tool. It is a server that enables interactions with the Session Database (SDB). WebSM is a WWW interface to the SM.

Here is a bit of terminology.

Definition 6.2.1

- *An* application *is an executable program;*
- *An* application process *(process for short) is an application in execution;*
- *An* application process agent *(agent for short) is a process on the same machine of the* SM *which passes the queries of the* SM *to the application processes;*
- *a* query *is a tuple* $\langle i, o, \mathrm{Status}, \mathrm{Info}\rangle$*, where* i *and* o *are respectively the input and the output of the query,* Status *is the current status of the query (see Sect. 6.2.3) and* Info *contains auxiliary information (not specified here) used by the* SM*;*
- *a* session *is a set of queries;*
- *a* Sessions Database (SDB) *is a tuple* $\langle A, P, S\rangle$*, where* A *is a set containing the information related to the applications connected to the* SM *(i.e. the machine in which the application runs, the agent of the application, etc.),* P *is a set of application processes and* S *is a set of sessions.*

The user interacts with the application processes by means of the SM which offers the following services:

- NewSession() creates a new session;
- AddQuery(s, i) adds the query $\langle i, \text{Null}, \text{Input}, \text{Info} \rangle$ to the session s;
- StartAppl(s, a) starts the application a within the session s by running a new application process;
- SubmitQuery(s, q, p) enqueues the query q of the session s to the queries for the process p;
- RestartProcess(s, p) starts a new process p' for the same application of p and passes all the queries queued for p to the new process p' (used in case of an abnormal termination of p);
- RecoverProcess(s, p) is the same as RestartProcess(), except that it also submits the previously answered queries to the new process in order to restore the state of the process as it was at the moment of the crash;
- Report(Params) creates a report according to the specified parameters (see Sect. 6.2.4).

The SM solves the queries using the application processes. The SM takes a query q submitted by the user to a process p and sends the input to the agent of p. When a process has completed a computation, the SM puts the result in the output part of the associated query.

In the following sections we will examine, in more detail, how the SM and the WebSM work. We will support our presentation with a simple example, which we now introduce.

We start with an empty SDB, and we consider two different sessions s_1 and s_2. The two sessions use two different processes, p_1 and p_2, for the same application (e.g. CoCoA), but they share the process p_3 for running the application Singular.

In the following we present the status of the SDB for the example, in a non–graphical style.

We begin with an empty SDB.

```
=============================================================
SM report of [29 Feb, 11:13:28]
There are 0 processes and 0 sessions.
=============================================================
```

6.2.1 Sessions

The user submits requests to the SM within a session. Other users can share this session and exchange data within it; moreover different sessions can share the same process.

In the example below a new session has been created (service NewSession()).

```
=============================================================
SM report of [29 Feb, 11:21:26]
There are 0 processes and 1 sessions.
Last operation: Created session 1 [29 Feb, 11:21:26]

------------------------------
Session 1 (no queries)
=============================================================
```

The user opens the connection with the WebSM by entering a session. The home page of the WebSM presents a list of available sessions, together with links that can be followed to enter each session.

6.2.2 Applications, Processes and Agents

An application is an executable program. The command StartAppl(s, a) begins the execution of the application a using a new process p; it also stores in the Sessions Database (SDB) the information related to p. The applications available via the SM can be on machines different from the one in which the SM operates.

The interaction between the process and the session manager is realized using *agents* that send the inputs to, and receive the output from, the application process via sockets, nfs or other forms of communication. The agents run on the same machine of the SM and communicate with it using the filesystem, in the following way: the SM writes the input of the query on a file (say query.in), then the agent communicates the input to the (possibly remote) application, waits for the result and writes it on another file (say query.out).

After the execution of the command StartAppl$(1, \mathrm{CoCoA})$ the SDB looks as follows:

```
=============================================================
SM report of [29 Feb, 11:26:09]
There are 1 processes and 1 sessions.
Last operation: Created process 1 [29 Feb, 11:26:09]

Process 1 : (CoCoA) serving query 1.1 (history 1.1)

------------------------------
Session 1 (1 queries)

1.1 Input: START CoCoA, Output: NULL, PID: 1, submitted
=============================================================
```

As the application process starts, a new query is automatically prepared to save the output related to the initialization of the application.

The WebSM presents to the user a page on which are listed all the applications available. For each application there is some information about the machine in which it resides (power and load of the machine, geographic location, etc.) and the list of the processes that run that application. Each running process has a link; clicking on it the user adds the process to the list of the *active processes* of the session. Moreover, the user can start a new application (command StartAppl()) by activating the corresponding link ["START"]. The new process that runs the application will become, automatically, an active process of the session.

6.2.3 Queries

A query is a tuple $\langle i, o, \mathrm{Status}, \mathrm{Info}\rangle$. The user creates a query within a session s using the command AddQuery(s, i), where i is the input of the query. A query can be submitted to a process using the command SubmitQuery(s, q, p) where s is the session, q is the query and p is the process.

Each query of a session has a status. Here is the list of all statuses and their meaning:

- Input: the query has an input, but it is not submitted to any process;
- Submitted: the input of the query has been submitted to a process. The identifier of the process and the submission time is contained in the Info;
- Running: the process to which was the query submitted is now processing it;
- Done: the process has completed the computation and o contains its output;

```
==============================================================
SM report of [29 Feb, 11:27:34]
There are 1 processes and 1 sessions.
Last operation: Created query 1.2 [29 Feb, 11:27:34]

Process 1 : (CoCoA) serving query 1.1 (history 1.1)

-------------------------------
Session 1 (2 queries)

1.1 Input: START CoCoA, Output: NULL, PID: 1, submitted [960229,11:26:09]
1.2 Input: X := 1+1, Output: NULL, PID: NULL, input
==============================================================
```

The query 1.1 (the query 1 of the session 1) has been submitted to the process 1 which is now serving it. The query 1.2 is not yet submitted to any process.

Using the WebSM the creation of a query is done by writing down the input in a *text–area* and pushing the button ["NEW QUERY"]. The user can submit the query to a process by selecting it within the list of the active processes of the session.

```
=============================================================
SM report of [29 Feb, 12:01:39]
There are 3 processes and 2 sessions.
Last operation: Served query 2.4 by process 3 [29 Feb, 12:01:39]

Process 1 : (CoCoA) ready (history 1.1 1.2 1.4)
Process 2 : (CoCoA) ready (history 2.1 2.2 2.3)
Process 3 : (Singular) ready (history 1.3 1.5 2.4)

------------------------------
Session 1 (5 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done
1.2 Input: X := 1+1, Output: OK, PID: 1, done
1.3 Input: START Singular, Output: Wilkommen zu Singular, PID: 3, done
1.4 Input: X, Output: 2, PID: 1, done
1.5 Input: X := x+y, Output: OK, PID: 3, done
------------------------------
Session 2 (4 queries)

2.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 2, done
2.2 Input: X := 1+2, Output: OK, PID: 2, done
2.3 Input: X, Output: 3, PID: 2, done
2.4 Input: X, Output: x+y, PID: 3, done
=============================================================
```

In the query 1.2, the user asked the process 1 to compute the expression $1+1$ and to store the result in the variable X.

In the query 1.4 the user asked the process 1 to print the value of the variable X.

Something similar happened for the queries 2.2 and 2.3. The outputs of the two (identical) queries 1.4 and 2.3 are different because the queries have been submitted to two different processes, even if they run the same application.

The interaction with p_3 shows how two users can communicate data by sharing the same process. The first user puts in the variable X of p_3 the polynomial $x + y$ (query 1.5) and the other user looks at it by asking the process p_3 for the value of the variable X (query 2.4).

6.2.4 Reports

An important service of the SM is the production of reports. Each user can ask the Session Manager for a report on one or more sessions. Different users can ask for different reports on the same SDB and on the same sessions.

The reports may differ from the order of presentation of the queries (by submission time or by creation time), by the portion of the view (query of a process, query of all the processes of an application, query range). Furthermore the user can ask for a report containing just the output of a particular query. Often the outputs are very big and the user would prefer to navigate them in a hypertextual style. For example: a matrix of polynomials could be reported as a matrix (HTML table) of hyperlinks, each of which can be represented by a string that summarizes the polynomial in that entry (e.g.: it gives the length of the polynomial). The user interested in seeing the entire polynomial can click on the link and receive the document containing the polynomial.

As WebSM runs on a WWW client, the navigation on the data is easy to perform. Obviously the applications should give sufficient information to the WebSM for the production of hypertextual outputs (for example in HTML). It is interesting to note that the applications can cooperate with their agents in producing hypertextual documents. The HTML makes it easy to write documents in "pretty" style. For example the HTML 3 will support "mathematical environment".

6.2.5 Scheduling of the Queries

The SM communicates each submitted query to the associated process by running the agent of the process. Periodically the SM checks to see if some process has completed a query (i.e.: its agent has written the result on the output file); if so, the SM copies the result of the query in the field Output of the query itself, and sets the status of the query to Done.

Then, the SM gets the next query queued for that process and contacts the agent for its computation.

6.2.6 Crash Recovery

Since all the sessions are stored in the machine on which the SM runs, the crash of a client's machine is not a problem. The problem of crash and its recovery is only a problem on the SM machine and the machines which run the processes.

The use of different applications on different machines implies a higher probability of a crash. The SM model provides a simple way to do crash recovery of abnormally terminated processes. The list of all inputs computed by each process (i.e. the history of the process) is stored in the part of SDB devoted to the processes; when a process abnormally terminates its execution, we can restore it by submitting all the queries processed until its termination.

E.g.: suppose that after the computation of the query 1.2 the process 1 dies.

```
==============================================================
SM report of [29 Feb, 14:17:42]
There are 1 processes and 1 sessions.
Last operation: Process 1 abormally terminated [29 Feb, 14:17:42]
```

```
Process 1 : (CoCoA) defunct (history 1.1 1.2)

------------------------------
Session 1 (4 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done
1.2 Input: X := 1+1, Output: OK, PID: 1, done
1.3 Input: X, Output: NULL, PID: 1, submitted
=============================================================
```

We can restart the process by submitting the command `RestartProcess(1,1)`.

```
=============================================================
SM report of [29 Feb, 14:17:59]
There are 2 processes and 1 sessions.
Last operation: Served query 1.3 by process 2 [29 Feb, 14:17:59]

Process 1 : (CoCoA) defunct ... 1.1 1.2
Process 2 : (CoCoA) ready ... 1.4 1.3

------------------------------
Session 1 (5 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done
1.2 Input: X := 1+1, Output: OK, PID: 1, done
1.4 Input: START CoCoA, Output: Welcome to CoCoA, PID: 2, done
1.3 Input: X, Output: Undefined var X, PID: 2, done
=============================================================
```

We note that the query 1.4 has been moved to the new process 2; but that the process 2 has no memory of the assignment done in the query 1.2. We can perform crash recovery by using the command `RecoverProcess` instead of using the command `RestartProcess`.

```
=============================================================
SM report of [29 Feb, 14:18:35]
There are 2 processes and 1 sessions.
Last operation: Served query 1.3 by process 2 [29 Feb, 14:18:35]

Process 1 : (CoCoA) defunct (history 1.1 1.2)
Process 2 : (CoCoA) ready (history 1.4 1.2 1.3)

------------------------------
Session 1 (5 queries)

1.1 Input: START CoCoA, Output: Welcome to CoCoA, PID: 1, done
1.2 Input: X := 1+1, Output: OK, PID: 2, done
1.4 Input: START CoCoA, Output: Welcome to CoCoA, PID: 2, done
1.3 Input: X, Output: 2, PID: 2, done
=============================================================
```

Note that the process has now recomputed the query 1.2 and hence the query 1.3 is correctly answered.

Obviously this approach is not efficient but it is fundamental because it is transparent to the user. Remember that our goal was to produce a common, user-friendly interface to different applications.

Moreover, depending on the application used, the user can speed up possible crash recoveries by "saving the state" of the process and possibly "restoring it". Suppose for example that we do some big computation and store its result in a variable, using a command like `X := Hard(A,B,C)`. Also suppose that `V` is the final value of the execution. Then a simple way of helping the SM is to replace the query `X := Hard(A,B,C)` with the new one `X := V`. In this way if a crash occurs the hard computation will not be redone.

6.2.7 Educational issues

Since many users can share the same session, our model is useful for teaching purposes. E.g. we can suppose that one user (the teacher) writes down a session and that other users (the students) read it. In this way the WebSM can be used for giving a lesson.

Suppose now that the teacher proposes an exercise to the students. The teacher can then write down one query for each student and ask each student to write down the result of ther investigation. Then the session will contain all the exercises written by all the students.

We are certain that educators will find many other interesting ways to use the potentiality of a system constructed in this way.

6.2.8 Ownership and Protection of Data

The shared use of a session implies that considerations of ownership of sessions (and of queries inside a session) and of security must be made. For example one might prefer that the only person who can write the lesson is the teacher. Furthermore we want to avoid the possibility that a student could look at the exercise done by another student.

A simple solution to this problem is to add the possibility for users to declare themselves, with a password and thus protect their data. This aspect of the SM (and of the WebSM) is currently under development.

Conclusions

One of the main requirements, stated at the beginning of the CoCoA 3 project, to host advanced mathematical algorithms (such as the Buchberger Algorithm for the computation of the Gröbner Basis of an Ideal) in a computation environment as *confortable* as possible for the user, has been satisfied: the community of the CoCoA users is growing fast.

To design and develop of such a big system has been a very challenging and interesting experience; it has been a very good platform for the experimentation of most of the notions and the techniques learned during the degree studies in computer science.

It has been also the occasion in which to invent new techniques and methodologies, like the Gröbner Framework and the Session Management Framework, and advanced data structures, such as the EEE polynomial representation.

As requested at the beginning of the project the overall system architecture resulted to be *portable* (thanks to the combination of the CTL language and the CDevices) and *programmable* (thanks to the easy to use CoCoAL language).

Limits of the CoCoA Architecture

At the time in which I started to develop CoCoA 3 the choice of the C language was really the only possible one. In the 1993 the C++ language was not sufficently stable. The more advanced Java language was not yet invented!

Another limit is the fact the CoCoA is hard to extend at a low level. It is adequate for making extensions using the CoCoAL language, and for the rapid prototyping of new algorithms. On the contrary there are no facilities to help CoCoA users to add new low level functionalities, in the mathematical library and within the interpreter.

Future Directions

In the last few years the programming technologies have evolved in a rich environment in which:

- it is possible to develop *distributed applications* by assembling different *software components* possibly written in different languages and running on different platforms, thanks to the *CORBA integration technology*;
- the communication between different software modules benefits of the result of the standardization process that produced the *XML technology*;
- it is easy to access relational and object-oriented databases.

Now it would be better to implement the high level part of the system (roughly speaking the CoCoAL interpreter) in Java language and connect it via JNI (Java Native Interface) to the mathematical kernel, written in C++ and organized as a set of software libraries.

Another useful change would be to redesign the CTL low level language of CoCoA in a new language, say *XCL (eXtensible Computation Language)*, based on the XML markup language. This approach would give to the CoCoA developers a powerful context for extending the system capabilities within a human readable low level language. Furthermore the usage of the *XML Beans* (software components automatically assembled at runtime during the parsing of an XML document) would make it possible to "plug" the semantics of the language without the need to rebuild the system.

Bibliography

[A98] J. Abbott, *Univariate factorization over the integers*, Preprint (1998).

[AL94] W. W. Adams, P. Loustaunau, *An Introduction to Gröbner Bases* , Graduate Studies in Mathematics: Amer. Math. Soc., Providence, R.I. (1994).

[BGS96] O. Backmann, S. Gray, and H. Schönemann. MPP: A framework for distributed polynomial computations. In *International Symposium on Symbolic and Algebraic Computing*, 1996.

[Bro71] W.S. Brown. On euclid's algorithm and the computation of polynomial greatest common divisors. *Journal of ACM*, 18:478–504, 1971.

[BS92] D. Bayer and M. Stillman. *Macaulay: a System for Computation in Algebraic Geometry and Commutative Algebra*, 1992.

[B85] B. Buchberger *Gröbner bases: An algorithmic method in polynomial ideal theory* in *Recent Trends in Mathematical Systems Theory* (N.K. Bose, Ed.), D.Reidel, Dordrecht, 1985, 184-232.

[B65] B. Buchberger. *An algorithm for finding a basis of the residue class ring of a zero-dimensional polinomial ideal.* PhD thesis, Math. Inst. Univ. Innsbruck, 1965.

[BW93] T. Becker and Volker Weispfenning. *Gröbner Bases: A computational approach to communtative algebra.* Springer-Verlag, 1993.

[B97] A.M. Bigatti, *Computations of Hilbert-Poincaré Series*, J. Pure Appl. Algebra, 119/3, 237–253 (1997).

[BCNR98] A.M. Bigatti, A. Capani, G. Niesi, L. Robbiano, *Hilbert-Poincaré Series and Elimination Problems*, Preprint (1998).

[BLR98] A.M. Bigatti, R. La Scala, L. Robbiano, *Computing Toric Ideals*, Preprint (1998).

[BR97] A.M. Bigatti, L. Robbiano, *Borel Sets and Sectional Matrices*, Annals of Combinatorics,1, 197–213, (1997).

[CCT97] M. Caboara, P. Conti, and C. Traverso, *Yet Another Ideal Decomposition Algorithm*, AAECC-12, Springer LNCS 1255, 39–54, (1997).

[CD96] A. Capani, G. De Dominicis, *Web Algebra*, In Proc. of WebNet 96. Association for the Advancement of Computing in Education (AACE) Charlottesville, USA, (1996).

[CDNR97] A. Capani, G. De Dominicis, G. Niesi, L. Robbiano, *Computing Minimal Finite Free Resolutions*, J. Pure Appl. Algebra, 117/118, 105–117 (1997).

[CDR96] M. Caboara, G. De Dominicis, L. Robbiano, *Multigraded Hilbert Functions and Buchberger Algorithm*, In *Proc. ISSAC '96*, 72–78 (1996), Y.N. Lakshman, editor, New York. ACM Press.

[CLD92] D. Cox, J. Little, D. O'Shea, *Ideals, Varieties, and Algorithms* , Springer-Verlag, New York (1992)

[CN98] A. Capani, G. Niesi, *The* CoCoA *3 Framework for a Family of Buchberger-like Algorithms*, In Gröbner Bases and Applications (Proc. of the Conf. 33 Years of Gröbner Bases), London Math. Soc. Lecture Notes Series, Vol **251**, B. Buchberger and F. Winkler eds, Cambridge University Press, 338–350 (1998).

[CNR96] A. Capani, G. Niesi, L. Robbiano, *Some Features of* CoCoA *3*, Comput. Sci. J. of Moldova 4, 296–314 (1996).

[CT] M. Caboara and C. Traverso. The powerful buchberger algorithm for modules. preprint.

[CN95] A. Capani and G. Niesi. *CoCoA 3.0 User's Manual.* Dipartimento di Matematica, Università di Genova, Via Dodecaneso, 35, I-16146 Genova (Italy), 3.0b edition, 1995.

[DST88] J.H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra: Systems and Algorithms for Algebraic Computation.* Academic Press, 1988.

[Edw89] H.M. Edwards. *Kronecker's Views on the Foundations of Mathematics.* Courant Institute of Mathematical Sciences, 1989.

[Eis95] D. Eisenbud. *Commutative algebra with a view toward algebraic geometry.* Springer-Verlag, 1995.

[GGMNPPSS] H. Grassmann, G.M. Greuel, B. Martin, W. Neumann, G. Pfister, W. Pohl, H. Schönemann, T. Siebert, *Standard Bases, Syzygies and their Implementation in Singular*, `preprint`

[GMNRT] A. Giovini, T. Mora, G. Niesi, L. Robbiano, C. Traverso, *"One sugar cube, please" or selection strategies in the Buchberger algorithm*, In *Proc. ISSAC '91*, Stephen M. Watt, editor, New York, ACM Press, 49–54 (1991).

[GN] A. Giovini and G. Niesi, CoCoA*: a user-friendly system for commutative algebra*, In *Design and Implementation of Symbolic Computation Systems – International Symposium DISCO'90*, Lecture Notes in Comput. Sci., 429, 20–29, Berlin, Springer Verlag (1990).

[GMRT94] P. Gianni, T. Mora, L. Robbiano, C. Traverso. *Hilbert function and Buchberger algorithm* preprint (1994)

[GPT88] A. Galligo, L. Pottier, and C. Traverso. Greater easy common divisor and standard basis completion algorithms. In P. Gianni, editor, *Proc. ISSAC '91*, pages 162–176. LNCS Springer, 1988.

[GPS95] G. M. Greuel, G. Pfister, H. Schönemann, *Singular:*, A System for Computation in Algebraic Geometry and Singular Theory. Department of Mathematics, 1995. Available via anonymous ftp from `helios.mathematik.uni-kl.de`.

[GM89] R. Gebauer, M. Möller, *On an installation of Buchberger's Algorithm*, Computational Aspects of Commutative Algebra, Robbiano ed., Academic Press, 141–152 (1989)

[GS96] D. Grayson, M. Stillman, *Macaulay 2*, 1996. Available via anonymous ftp from `math.uiuc.edu`.

[H] D. Hilbert, *Über die Theorie von algebraiscen Formen* , Math. Ann. **36**, 473–534 (1890)

[Her26] G. Hermann. Die frage der endlich nielen schritte in der theorie der polynomideale. *Mathematische Annalen*, 95:736–788, 1926.

[HH80] V.R. Huskey and H.D. Huskey. Lady lovelace and charles babbage. *Annals of the History of Computing*, 2:229–329, 1980.

[KB70] D.E. Knuth and P.B. Bendix. *Simple word problems in universal algebras.* Pergamon Press, 1970.

[Knu69] D.E. Knuth. *The Art of Computer Programming, Vol. II, Semi–numerical Algorithms.* Addison-Wesley, 1969.

[Knu82] D.E. Knuth. *The Art of Computer Programming, Vol. I, Fundamental Algorithms.* Addison-Wesley, 1982.

[Kro82] L. Kronecker. Grundzüge einer arithmetischen theorie der algebraischen größen. *J. Reine Angew. Math.*, 92:1–122, 1882.

[L96] R. La Scala, *Un Approccio Computazionale alle risoluzioni libere minimali*, Quaderno 1/96, PhD Thesis, University of Bari

[Laz77] D. Lazard. Algèbre linéaire sur $k[x_1, \ldots, x_n]$ et élimination. *Soc. Math. France*, pages 165–190, 1977.

[Mis93] B. Mishra. *Algorithmic Algebra.* Springer-Verlag, 1993.

[MM82] E.W. Mayr and A.R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329, 1982.

[MM86] M. Möller and T. Mora. New constructive methods in classical ideal theory. *Journal of Algebra*, 100:138–178, 1986.

[MR88] T. Mora and L. Robbiano. The gröbner fan of an ideal. *J. Symbolic Computation*, 6(2-3):183–208, 1988.

[NT92] M. Noro and T. Takeshima *Risa/Asir – a computer algebra system.* In: Proceedings of ISSAC '92, ACM Press, New York, pp. 387–396

[P94] Pfister, G. *The tangent cone algorithm and some applications to local algebraic geometry.* In Mora, T. and Traverso, C. (eds.), *Effective Methods in Algebraic Geometry*, Progress in Mathematics **94**, Birkäuser Verlag, Basel, 401-409

[Rob85] L. Robbiano. On term orderings. *Lecture Notes in Computer Science*, 204:513–517, 1985.

[Rob86] L. Robbiano. On the theory of graded structures. *J. Symbolic Computation*, 2(2):139–170, 1986.

[Rob88] L. Robbiano *Introduction to the theory of Gröbner bases*, *Queen's Papers in Pure and Applied Mathematics*, 1988.

[Rob96] L. Robbiano. Reviw of the books: [BW93] and [AL94]. *Bull. Amer. Math. Soc.*, 33(4):477–482, 1996.

[Rob98] L. Robbiano, *Gröbner Bases and Statistics*, In Gröbner Bases and Applications (Proc. of the Conf. 33 Years of Gröbner Bases), London Math. Soc. Lecture Notes Series, Vol **251**, B. Buchberger and F. Winkler eds, Cambridge University Press, 179–204 (1998).

[RV83] L. Robbiano, G. Valla, *Free Resolutions for Special Tangent Cones*, Proceedings of the Trento Conference, Lecture Notes in Pure and Applied Math. **84** (1983)

[S80] F. O. Schreyer, *Die Berchnung von Syzygien mit dem verallgemeinerten Weierstrass'schen Divisionsatz*, Diplomarbeit, Hamburg (1980)

[Sei74] S. Seidenberg. Construction in algebra. *Trans. Amer. Math. Soc.*, 197:273–313, 1974.

[S77] D. Spear, *A constructive approach to commutative ring theory*, Proceedings of 1977 MACSYMA User's Conference, NASA CP-2012, 369–376 (1977)

[T96] C. Traverso *Hilbert functions and the Buchberger algorithm*, J. Symbolic Computation, 22:355–376, 1996.

[Tra88] C. Traverso. Metodi costruttivi e calcolo automatico in algebra commutativa. *Bollettino U.M.I.*, 7(2-A):145–167, 1988.

[Win96] F. Winkler. *Polinomial Algorithms in Computer Algebra.* Springer, 1996.