Ulrich Hirn

An Implementation of Groebner Bases using Modular and Floating Point Arithmetic

1999
Diploma Thesis
Universtity of Technology Graz

Table of contents

Preface

In many fields of engineering it is necessary to determine the solutions of systems of nonlinear polynomial equations with several indeterminates.

Computing a lexicographic Groebner Basis from the original input polynomials, and subsequently the solutions points, is a promising procedure. Many commercially available software packages offer the computation of Groebner Bases. However, problems that involve polynomials of high degree and/or with many indeterminates cannot be solved. Allthough, in theory the algorithm eventually terminates, the demand of memory and computations to be performed often exceeds the capabilities of the computer.

The software package *gfloat*, that is introduced in this thesis, computes all solutions of a system of polynomials. Using gfloat the computational effort to compute a lexicographic groebner basis is significantly reduced, thus it is often possible to obtain results, where commercial software fails. Two specific approaches are used to improve the performance:
(1) The usage of floating point/modular coefficients for the polynomials
(2) Computing a lexicographic Groebner Basis indirectly by conversion from a total degree reverse lexicographic Groebner Basis.

I wish to thank Prof. Peter Dietmaier for his patience, advice and support during the long process of work on this thesis. Also I want to thank Dr. Siegfried Lösch, the original creator of *gfloat* [Lösch 1996] who introduced me to the world of Groebner Bases and gave me a great deal of inspiration.

1. Groebner Bases

This section will NOT dig into the mathematics of Groebner Bases, it will just provide the mathematical techniques that were required to make the software gfloat. From a mathematical point of view it will not be of any interest, because no formal proofs are given - the focus is strictly on explaining general ideas of the algorithms that are used. It is the goal that a reader with no experience in abstract algebra could actually use this paper as a manual to rewrite gfloat. So all algorithms used in gfloat are explained and lots of examples are given. Over the whole section, the paper largely follows the ideas developed in [Cox, Little, O'Shea 1992] and [Becker, Weispfennig 1993].
Still abstract description is being used: it is the main advantage of abstract algebra, that it describes properties of strictly defined objects, no matter what the actual nature of the object is. In the case of this paper we will see that the concept of residue class rings finds its application on modular arithmetic as well as the transformation of the monomial ordering of Groebner Bases.
First we will define some objects that will be used over and over again.

1.1 Basics

1.1.1 Algebraic objects

(1.1.1.1)

A commutative ring consists of a set R and the binary operations $\cdot$ + and – (i.e. multiplication, addition and subtraction). The operations are defined on R and satisfy the following conditions for all a,b,c $\in$ R

(i) $(a \pm b) \pm c = a \pm (b \pm c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ - the operations are associative
(ii) $a \pm b = b \pm a$ and $a \cdot b = b \cdot a$ - the operations are commutative
(iii) $(a \pm b) \cdot c = a \cdot c \pm b \cdot c$ - the operation $\cdot$ is distributive
(iv) There are neutral elements $0,1 \in R$ such that $a = a \pm 0$ and $a = a \cdot 1$
(v) The operation - is the inverse of + such that a - b = 0.

The most common commutative ring is the set of integers **Z**, multiplication addition and subtraction are defined for **Z**, zero and one are its neutral elements. In this paper we will mainly find the commutative ring $k[x_1, x_2, ... x_n]$ i.e. the set of all polynomials in the indeterminates $x_1, x_2, ... x_n$ with the monomial coefficients from the field k.

(1.1.1.2)

A field consists of a set R and the binary operations · / + and – (i.e. multiplication, division, addition and subtraction). The operations are defined on R and satisfy the following conditions for all a,b,c $\in R$

(i) (a±b)±c = a±(b±c) and (a·b)·c = a·(b·c) - the operations are associative
(ii) a±b = b±a and a·b = b·a - the operations +-· are commutative
(iii) (a±b)·c = a·c±b·c and (a±b)/c = a/c±b/c - the operations /· are distributive
(iv) There are neutral elements 0,1 $\in$*R such that a = a±0 and a = a·1*
(v) The operation - is the inverse of + such that a - a = 0.
(vi) The operation / is the inverse of · such that a / b = 1.

So a field has the same properties (i) to (v) as the ring - so every field is a ring - additionally division is defined over R. The set of rational numbers **Q**, the real numbers **R** or the complex numbers **C** are fields. The set of integers **Z** is not a field because a division of two integers not necessarily yields another integer.

Dealing with algebraic equations polynomials are the principal objects being used. Polynomials in just one variable x_1 are called a **univariate** polynomials, ones with using more variables $x_1, x_2, \ldots\ldots x_n$ are called **polyvariate** polynomials.

(1.1.1.3)

*A **monomial** in n variables* $x_1, x_2, \ldots\ldots x_n$ *is a product of the form*

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot x_3^{\alpha_3} \cdot \ldots\ldots \cdot x_n^{\alpha_n} = x^{\alpha}$$

where all of the exponents α *are non - negative integers.*
*The **total degree*** $|\alpha|$ *of the monomial* x^{α} *is defined as* $|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_n$.

(1.1.1.4)

*k being a field, a **polynomial** f in* $x_1, x_2, \ldots\ldots x_n$ *is a finite linear combination of monomials in the form*

$$f = \sum_m a_{\alpha_m} \cdot x^{\alpha_m} \qquad a \in k$$

$a_{\alpha m}$ *are the coefficients of the polynomial, each expression* $x^{\alpha m}$ *is called a **term**,*
and $a_{\alpha m} x^{\alpha m}$ *is a monomial.*
*The **total degree** deg(f) of the polynomial f is defined as the maximium* $|\alpha_m|$.

The poynomial g = $5x^2y^3z - 3y^2z^2 + 7xz^2 - 3$ for example lies in k[x,y,z] with k =**Z**.
It has four monomials and the total degree deg(g) = 6.
For the time being just use integer coefficients will be used, later we will introduce polynomials with modular coefficients and floating point coefficients.

1.1.2 Monomial Orderings

Taking an univariate polynomial that consists of the terms *2, -3x^2, 5x* and *8x^3* we would intuitively write it in the form *f = 8x^3 - 3x^2 + 5x + 2.* What we do is sorting the terms of the polynomial by decreasing degree of the monomials.
Taking a polynomial *g* in *k[x,y,z]* consisting of the terms *3xy^2, -7z^3* and *5xz* one can easiliy realize that the monomials *3xy^2 and -7z^3* both have a total degree of 3. Which one should be written first in the polynomial? We obviously need more complex rules to sort the monomials of polyvariate polynomials.

(1.1.2.0)

*A **monomial ordering** on $k[x_1, x_2, ... x_n]$ is any relation > on the set of monomials x^α that satisfies the following:*
(i) > is a total ordering.

That means that every strictly decreasing sequence of terms x^α

$$\alpha(1) > \alpha(2) > \alpha(3) > .....$$

eventually terminates.

(ii) If $x^\alpha > x^\beta$ and $\gamma \in Z_0^+$ then $x^{\alpha\gamma} > x^{\beta\gamma}$.

(iii) > is a well ordering.

That means that every set of terms x^α has a smallest element under >

There are several different monomial orderings. For our purposes we need two of them, the lexicographic ordering denoted as $>_{lex}$ and the total degree reverse lexicographic ordering, denoted as $>_{tdegrevl}$

1.1.2.1 Lexicographic Ordering

(1.1.2.1)

Let x^α and x^β be terms in $k[x_1, x_2, ... x_n]$ and $\alpha = (\alpha_1, \alpha_2, ... \alpha_n)$, $\beta = (\beta_1, \beta_2, ... \beta_n)$ be the exponent vectors of the terms.
We say $x^\alpha >_{lex} x^\beta$ if, in the vector difference $\alpha - \beta$, the left-most nonzero entry is positive.

Some Examples in k[x,y,z] with the ordering x $>_{lex}$ y $>_{lex}$ z:

(1) $xy^2 >_{lex} y^3z^3$ since
α =(1,2,0), β = (0,3,3) and $\alpha - \beta$ = (1,-1,-3)
(2) $xy^2z^3 >_{lex} xy^2z$ since
α =(1,2,3), β = (1,2,1) and $\alpha - \beta$ = (0,0,2)

Please note that there are many different $>_{lex}$ orderings, corresponding how the variables are set e.g. z $>_{lex}$ x $>_{lex}$ y is a different ordering from x $>_{lex}$ y $>_{lex}$ z. In fact for n variables there are n! lexicographic orderings corresponding to the permutation of the position of the variables.

Taking the monomials from the above example with the ordering z $>_{lex}$ x $>_{lex}$ y we get

(1) $z^3y^3 >_{lex} xy^2$ since
α =(3,3,0), β = (0,1,2) and $\alpha - \beta$ = (3,2,-2)
(2) $z^3xy^2 >_{lex} zxy^2$ since
α =(3,1,2), β = (1,1,2) and $\alpha - \beta$ = (3,0,0)

The lexicographic ordering has its name because it is like the alphabetic ordering used in dictionaries and telephone books. The exact definition of alphabetic ordering would be a lexicographic ordering with a > b >c > ..... > y > z.

1.1.2.2 Total Degree Reverse Lexicographic Ordering

(1.1.2.2)

Let x^α and x^β be terms in k[x_1, x_2, ... x_n] and α =(α_1, α_2, ... α_n), β = (β_1, β_2, ... β_n) be the exponent vectors of the terms.
We say $x^\alpha >_{tdegrevl} x^\beta$ if

(i) $|\alpha| > |\beta|$
or
(ii) If $|\alpha| = |\beta|$, the right-most nonzero entry in the vector difference $\alpha - \beta$, is negative.

Some Examples in k[x,y,z] with the ordering x $>_{tdegrevl}$ y $>_{tdegrevl}$ z:

(1) $xy^2z^3 >_{tdegrevl} x^3yz$ since
rule (i) applies: The total degree $|\alpha| = 6 > |\beta| = 5$
(2) $x^5yz >_{tdegrevl} x^4yz^2$ since
rule (ii) applies because $|\alpha| = |\beta| = 7$.and furthermore
α =(5,1,1), β = (4,1,2) and $\alpha - \beta$ = (1,0,-1)

As for the lex there are n! different tdegrevl orderings for n variables.
The question may arise why we need different monomial orderings at all. It will be shown that applying the same algorithms -for example the Normalform algorithm and Buchberger's algorithm- to polynomials sorted with respect to different monomial orderings returns very different results.
Only a lexicographic Groebner Basis allows to calculate the roots of system of polynomial equations allthough, on the way to find a lexicographic Groebner Basis it can be useful to compute a total degree reverse lexicographic Groebner Basis.

All monomial orderings determine the unique position of the terms within the polynomial, consequently one term in the polynomial will be maximimum term with respect to >. This term will be written first in a polynomial sorted w.r.t a certain monomial ordering >. Now this first term in a polynomial plays such an important role that it has a specific name: leading term (also calles headterm).

(1.1.2.3)

Let $f = \Sigma x^{\alpha}$ *be a polynomial in* $k[x_1, x_2, ... x_n]$ *and > be a monomial ordering.*

(i) The leading term or headterm of a polynomial f is
LT(f) = HT(f) = the maximum term w.r.t >

(ii) The leading coefficient of f is
LC(f) = the coefficient ot the leading term LT(f)

(iii) The leading monomial of f is
$LM(f) = \frac{LT(f)}{LC(f)}$ *i.e. the leading term with coefficient = 1.*

1.1.3 Division of Polynomials - Normalform Algorithm

In this section an algorithm to divide a polyvariate polynomial $g \in k[x_1, x_2, ... x_n]$ by a set of other polynomials $f_1, f_2, ... f_s \in k[x_1, x_2, ... x_n]$ will be described . We will see that the division algorithm for univariate polynomials is just a special case of the general division algorithm to be described.
In order to develop analogies between the division algorithms we want to revise the frequently used division algorithm for a univariate polynomial $g \in k[x_1]$ by another univariate polynomial $f \in k[x_1]$ before we look at the generalized one.

(1.1.3.1)

Taking the univariate polynomials g,f $\in$ k[x] one can write every g as a product g = qf + r. Following the algorithm to find q and r is presented.

Input: g, f
Output: q, r

q:= 0; r := f;
WHILE r $\neq$ 0 **AND** LT(f) divides LT(r) **DO**

$$q := q + \frac{LT(r)}{LT(f)}$$

$$r := r - \frac{LT(r)}{LT(f)} \cdot g$$

In (1.1.3.1) *g* is the dividend, *f* is the divisor, *q* stands for quotient and *r* stands for remainder. So every polynomial can be written as quotient times divisor plus remainder.
Now we will give an example for the division algorithm taking $g=3x^5-4x^3+x^2-2$ and $f=x^2+x-2$. After the first run of the WHILE loop we have:

$$(3x^5 - 4x^3 + x^2 - 2) \div (x^2 + x - 2) = 3x^3$$
$$\underline{-3x^5 - 3x^4 + 6x^3}$$
$$-3x^4 + 2x^3 + x^2 - 2$$

where $q=3x^2$, and $r=-3x^4+2x^3+x-2$. Obviously the term $\frac{LT(r)}{LT(f)} = 3x^2$ is designed in that way that the expression $\frac{LT(r)}{LT(f)} \cdot g = -3x^5-3x^4+6x^3$ cancels the leading term of *r.* In the next run of the WHILE loop the term $-3x^2$ will be added to the quotient and consequently the expression $\frac{LT(r)}{LT(f)} \cdot g = 3x^4+3x^3-6x^2$ again cancels the leading term of the remainder *r.*
So the degree of the remainder drops each time the WHILE loop is performed because the leading term - the one with the highest degree - is cancelled each time.
Repeating the WHILE loop two more times we have:

$$(3x^5 - 4x^3 + x^2 - 2) \div (x^2 + x - 2) = 3x^3 - 3x^2 + 5x - 10$$
$$\underline{-3x^5 - 3x^4 + 6x^3}$$
$$-3x^4 + 2x^3 + x^2 - 2$$
$$\underline{+3x^4 + 3x^3 - 6x^2}$$
$$5x^3 - 5x^2 - 2$$
$$\underline{\vdots}$$
$$20x - 22$$

The leading term of the remainder r=20x-22 is not divisible by the leading term of the divisor LT(f) the condition of the WHILE loop evaluates to false and the algorithm terminates with the result $q=3x^3-3x^2+5x-10$ and r=20x-22. We can write

$$g = 3x^5-4x^3+x^2-2 = (3x^3-3x^2+5x-10)\cdot(x^2+x-2) + 20x-22 = fq + r.$$

Please note that the remainder comes to its final value r=20x-22 when the leading term of the divisor can not divide r (or r becomes zero).
The generalized division algorithm will be given now.

(1.1.3.2)

Given a a monomial ordering >, a polynomial $g \in k[x_1, x_2, ... x_n]$ sorted w.r.t > and a set set of polynomials $f_1, f_2, ... f_s \in k[x_1, x_2, ... x_n]$ sorted w.r.t > g can be written as

$$g = a_1f_1 + a_2f_2 + ... + a_sf_s + r$$

where $r, a_1, a_2, ..., a_s \in k[x_1, x_2, ... x_n]$. We say g is divided by $f_1, f_2, ... f_s$. Furthermore $r, a_1, a_2, ..., a_s$ are found using the following algorithm:

Input: $g, f_1, f_2, ... f_s$
Output: $a_1, a_2, ..., a_s, r$

$a_1 := a_2 := ..... a_s := r := 0$
$p := g$
WHILE $p \neq 0$ **DO**
    i := 1
    divisionoccurred := false
    WHILE $i \leq s$ **AND** divisionoccurred = false **DO**
        IF $LT(f_i)$ divides LT(p) **THEN**
            $a_i := a_i + \frac{LT(p)}{LT(f_i)}$
            $p := p - \frac{LT(p)}{LT(f_i)} \cdot f_i$
            divisionoccurred :=true
        ELSE
            i:= i +1
    IF divisionoccurred = false **THEN**
        r:= r + LT(p)
        p:=p - LT(p)

The generalized division algorithm follows the same principals as the one for univariate polynomials.

- As a result the dividend *g* will be written as a linear combination of the divisors $f_1, f_2, \ldots\ldots f_s$ plus a remainder *r.*
- The algorithm consequently eliminates the leading terms of the dividend *p* using the headterms of the dividends.
- The remainder *r* is changed if none of the leading terms of the divisors divides the leading term of the dividend, when this happens the headterm of the dividend *p* is moved to the remainder.

As there are several divisors, it may be possible that several of their laeding terms divide the leading term of the dividend. If this happens the first polynomial in the list of divisors will be used to eliminate the headterm of the dividend.

We will take $g=x^2y^2-x^2y-3xy+6x^2$ and $f_1=y^2-2x$, $f_2=x^2y+3xy$ as an example, the polynomials are sorted w.r.t. tdegrevl ordering.
Both, the leading terms of f_1 and f_2 divide the headterm of the divisor, so we take the first polynomial f_1. After elimination of the headterm by multiplying f_1 with the factor x^2 and subtracting this from g we find the situation:

$$\begin{array}{l} x^2y^2 - xy^2 - 3xy + 6x^2 = \begin{matrix}(y^2 - 2x)\cdot(x^2)\\ (xy^2 + 3xy)\cdot 0\end{matrix} \\ \underline{-x^2y^2 + 2x^3} \\ -x^2y + 2x^3 - 3xy + 6x^2, \quad r = 0 \end{array}$$

The headterm of *p, LT(p)=-x^2y* is not divisible by *LT(f_1)=y^2,* so we take f_2 multiplied by *–1* and subtract it again:

$$\begin{array}{l} x^2y^2 - xy^2 - 3xy + 6x^2 = \begin{matrix}(y^2 - 2x)\cdot(x^2)\\ (xy^2 + 3xy)\cdot(-1)\end{matrix} \\ \underline{-x^2y^2 + 2x^3} \\ -x^2y + 2x^3 - 3xy + 6x^2, \quad r = 0 \\ \underline{+x^2y \qquad +3xy} \\ 2x^3 \qquad +6x^2, \quad r = 0 \end{array}$$

The new leading term $2x^3$ is not divisible by LT(f_1) or LT(f_2) so it is added to the remainder. However, the new leading term of p, LT(p)=$6x^2$ is still not divisible by one of the headterms of f_i so we also have to add it to the remainder. Now p is zero and the algorithm terminates. We obtain the final result:

$$x^2y^2 - xy^2 - 3xy + 6x^2 = \begin{matrix}(y^2 - 2x)\cdot(x^2)\\ (xy^2 + 3xy)\cdot(-1)\end{matrix}$$

$$\begin{array}{l} -x^2y^2 + 2x^3 \\ \hline -x^2y + 2x^3 - 3xy + 6x^2, \quad r = 0 \\ +x^2y \quad\quad + 3xy \\ \hline \quad 2x^3 \quad\quad + 6x^2, \quad r = 0 \\ \quad\quad\quad\quad\quad\quad\quad\quad \\ \hline \quad\quad\quad\quad\quad 6x^2 \quad , r = 2x^3 \\ \hline \quad\quad\quad\quad\quad\quad\quad\quad 0 \quad , r = 2x^3 + 6x^2 \end{array}$$

Note that the remainder is a sum of monomials, none of which is divisible by the leading terms $LT(f_1)$, $LT(f_2)$.
In accordance with (1.1.3.2) we can write $g=x^2y^2-x^2y-3xy+6x^2$ as a sum of products a_1f_1, a_2f_2 and a remainder r, $g=x^2y^2-x^2y-3xy+6x^2 = (x^2)\cdot(y^2-2x) + (-1)\cdot(x^2y+3xy) + 2x^3 + 6x^2$

For the next example we will take the same monomial ordering, the same dividend g and even the same divisors f_1,f_2. The only difference is in the order in the list of the divisors, now $f_1=x^2y+3xy$ and $f_2=y^2-2x$.
Surprisingly the result is totally different:

$$x^2y^2 - x^2y - 3xy + 6x^2 = \begin{matrix}(x^2y + 3xy)\cdot(y - 1)\\ (y^2 - 2x)\cdot(-3x)\end{matrix}$$

$$\begin{array}{l} -x^2y^2 - 3xy^2 \\ \hline -3xy^2 - x^2y - 3xy + 6x^2 \quad\quad , r = 0 \\ +3xy^2 \quad\quad\quad\quad - 6x^2 \\ \hline \quad\quad -x^2y - 3xy \quad\quad , r = 0 \\ \quad\quad +x^2y + 3xy \\ \hline \quad\quad\quad\quad 0 \quad\quad , r = 0 \end{array}$$

This time we can write g as a sum of products of f_1, f_2 with a zero remainder.
$g=x^2y^2-x^2y-3xy+6x^2 = (y-1)\cdot(x^2y+3xy) + (-3x)\cdot(y^2-2x)$.
The generalized division algorithm has obviously some properties that are different from the one for univariate polynomials.

- The order of the divisors $f_1, \ldots, f_s$ matters, both in the results of the remainder r and the factors $a_1, \ldots, a_s$.
- The remainder may be nonzero, even if the dividend can be written as a sum $g=a_1f_1+a_2f_2+\ldots\ldots+a_sf_s$ of multiples of the divisors.

The Normalform algorithm is a version of the generalized division algorithm that is a cornerstone of the theory and applications of Groebner Bases. In principle it performs exactly the same operations as the generalized division algorithm, the only difference is, that it just delivers the remainder of the division, the factors a_1 a_2,...., a_s are not determined. Another difference in the algorithm is, that it does not only check the leading terms, it considers all the terms of the dividend. Starting from the first monomial it looks for a term t that is is a multiple of of one of the LT(f_1),....,LT(f_s), which is eliminated subsequently. If no more term in r can be eliminated the algorithm terminates.

(1.1.3.3)

Given a a monomial ordering >, a polynomial $g \in k[x_1, x_2, ... x_n]$ sorted w.r.t > and a set set of polynomials ***B***$=\{f_1, f_2, ... f_s\} \in k[x_1, x_2, ... x_n]$ *the remainder r of a division g by $f_1, f_2, ... f_s$ called the* ***Normalform*** *of g modulo B will be denoted as*

r = NF(g) mod B.
The NF(g) mod B is found by the following algorithm:

Input: g, f_1, f_2, ... f_s
Output: r

```
r := 0
p := g
WHILE p ≠ 0 DO
        i := 1
        divisionoccurred := false
        WHILE i ≤ s AND divisionoccurred = false DO
                IF LT(f_i) divides LT(p) THEN
                        p := p - (LT(p) / LT(f_i)) · f_i
                        divisionoccurred :=true
                ELSE
                        i:= i +1
        IF divisionoccurred = false THEN
                r:= r+ LT(p)
                p:=p - LT(p)
```

It is a key question in the theory of ideals and Groebner bases to determine if a polynomial g can be written as a sum $g=a_1f_1+a_2f_2+..... +a_sf_s$ of multiples of polynomials f_1, ... ,f_s. The normalform of g mod {f_1, ... ,f_s} provides an answer to that problem. As we have seen in the example above NF(g) mod {f_1, ... ,f_s} =0 is a **sufficient** condition for $g=a_1f_1+a_2f_2+..... +a_sf_s$.

1.2 Varieties, Ideals and Groebner Bases

1.2.1 Varieties

The final goal of this section is to describe a way to find the solutions of a system of algebraic equations. We need to define an object that describes the set of all the solutions.

(1.2.1.1)

Let k be a field, and let $f_1, f_2, ... f_s$ be polynomials in $k[x_1, x_2, ... x_n]$. Then we set the ***affine variety*** $V(f_1, f_2, ..., f_s)$

$$V(f_1, f_2, ..., f_s) = \{(a_1, ..., a_n) \in k^n \ : \ f_i(a_1, ..., a_n) = 0\} \text{ for all } 1 \le i \le s.$$

Thus an affine variety $V(f_1, f_2, ..., f_s) \subset k^n$ is the **geometric object** in the n-dimensional space k^n that is defined by the system of equations $f_1 (x_1,... x_n) = f_2(x_1,... x_n) = ... = f_s (x_1,... x_n) = 0$. This geometric object consists of the set of n-tuples $(a_1,... ,a_n) \in k^n$ - these are the solution points of the system of equations $f_i = 0$.
The trivial solution $x_1 = x_2 = ... = x_n = 0$ always exists. Please note that apart from the trivial solution there can be an infinite number of solution points (if the dimension of the variety is greater zero) or no solution point at all (if the equations $f_i = 0$ are contradictory), however if there is a finite number of non-trivial solution points we say that the variety is **zero dimensional**.

Visualizing the geometric object that is being defined by the variety using the field of rational numbers R makes it easier to understand varieties. In the following some examples of varieties in R^n are given. Allthough the affine space of the variety can have arbitrary dimension we will only use examples in the plane R^2 and the 3-dimensional space R^3 .

(1)
The first example is set in $R^2[x,y]$ the variety $V = (y - x^3 + 3x^2 + 1)$. This is equivalent to the equation $y = x^3 - 3x^2 - 1$. So this variety obviously can be represented as the graph of the polynomial function $y = x^3 - 3x^2 - 1$. The dimension of the variety is one and the number of solution points thus is infinite.

(2)
The second example is set in $R^3[x,y,z]$ the variety $V = (x^2 + y^2 + z^2 - 9)$, which is the sphere of radius = 3 with the center located at the origin; the number of solution points infinite, the dimension is two.

(3)
The next example has more than one polynomial, again $k = R^3[x,y,z]$, $V = (z, x^2+y^2+z^2-4)$. The first equation $z = 0$ represents the plane that is spanned by the x- and y- coordinate axis, $x^2 + y^2 + z^2 = 9$ is again a shpere. The variety is the intersection of the plane with the sphere which is a circle located in the plane z = 0 with the radius = 3 and the center in the origin.

(4)
Adding another polynomial we have the variety $V = (x^2-y^2-z^2-1, z, x^2+y^2+z^2-9) \subset R^3[x,y,z]$. The equation $x^2-y^2-z^2 = 1$ describes a hyperboloid of revolution obtained by rotating the hyperbola $x^2-y^2 = 1$ around the x-axis. It is easy to see that the solution is zero dimensional, the variety consists of four points.
$V = (x^2-y^2-z^2-1, z, x^2+y^2+z^2-9) = \{ x = \pm\sqrt{5}, y = \pm 2, z = 0 \}$.

1.2.2 Ideals

In this section the basic algebraic object of this paper will be introduced.

(1.2.2.1)

*Let R be a ring. Taking $f_1, f_2, \ldots f_s \in R$ we set the **ideal I** = $\langle f_1, f_2, \ldots, f_s \rangle$*

$$I = \langle f_1, f_2, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i : h_1, h_2, \ldots, h_s \in R \right\}$$

This is equivalent to the following definintion:

(1.2.2.2)

*Let R be a ring. Taking the f,g and $h \in R$ the subset $I \subset R$ is an **ideal** if it satisfies:*

(i) $0 \in I$
(ii) if $f,g \in I$ the $f + g \in I$
(iii) If $f \in I$ then $hf \in I$.

We will now give an interpretation for ideals as a set of polynomials in the polynomial ring $k[x_1 , ... ,x_n]$ and call this a polynomial ideal. That set can be derived from a number of generating polynomials. Taking an ideal $\mathbf{I} = \langle f_1,f_2,...,f_s \rangle \subset k[x_1 , ... ,x_n]$ and the polynomials $h_1,h_2,...,h_{s+1} \in k[x_1 , ... ,x_n]$ we find an element g_1 of the ideal using 1.2.2.2 (iii) :

$g_1 = h_1 f_1 + h_2 f_{2+} ,..., + h_s f_s \in \mathbf{I}$

and further elements $g_{2, ... ,} g_t$

$g_2 = h_1 f_1 + h_2 f_2 + ,..., + h_s f_s + h_{s+1} g_1 \in \mathbf{I}$
.....
$g_t = h_1 f_1 + h_2 f_2 + ,..., + h_s f_s + h_{s+1} g_1 + ... + h_{s+t} g_{t-1} \in \mathbf{I}$

As we see above, every element $g_1 , ..., g_t$ itself is a linear combination $\sum_{i=1}^{s} h_i f_i$.

Thus a polynomial ideal $\mathbf{I} = \langle f_1,f_2,...,f_s \rangle$ can be seen as the set of all polynomial consequences generated from the basis polynomials $f_1,f_2,...,f_s$.

For example we take the ideal $\mathbf{I} = \langle 3x-y^2+xy,\ xy^2-x^2y+1 \rangle$ and the polynomials $h_1 = x-y$, $h_2 = -2$. Generated by the basis we can write one element of the ideal as

$(x-y)(3x-y^2+xy) + (-2)(xy^2-x^2y+1) = 3x^2y-4xy^2+3x^2-3xy+y^2-2 \in \langle 3x-y^2+xy,\ xy^2-x^2y+1 \rangle$.

Another example of an ideal in the ring of integers Z see section 1.4.1.
In order to get a better picture about the concept of polynomial ideals an analogy with linear algebra can be made. Ideals and subspaces are similar: A subspace is formed by a span of the vectors $v_1,v_2,.....,v_n$ the elements of the subspace are linear combinations $a_1v_1+a_1v_2+.....+a_1v_n$ with $a_1, ... ,a_n$ being scalars. Both, subspaces and polynomial ideals are linear combinations of 'generators'. In the case of ideals the generators are polynomials $f_1,f_2,...,f_s$ instead of the vectors $v_1,....,v_n$ and the generators are multiplied by polynomials $h_1,h_2,...,h_t$ instead of scalars $a_1, ... ,a_n$.

(1.2.2.3)

Given an ideal $\mathbf{I} = \langle f_1,f_2,..., f_s \rangle \in k[x_1 , ... ,x_n]$ *that is generated by the polynomials* $f_1,f_2,...,f_s$. *We will call* $f_1,f_2,...,f_s$ *the* ***basis*** *of* ***I.***

Note that one ideal can have many different bases, a Groebner Basis is just one that has special properties. The crucial fact is now, that it is possible to change the basis of an ideal without changing the ideal itself. As we will see in later sections, Buchberger's Algorithm produces a Groebner Basis by continuously changing the basis of a given ideal.
Furthermore the Hilbert Basis Theorem states, that every ideal has a finite basis i.e. a finite number of generating polynomials.

(1.2.2.4)

A Variety that is determined by all the polynomials of an ideal $I = \langle f_1,f_2,...,f_s \rangle \in k[x_1 , ... ,x_n]$ *will be denoted as* $V(I)$.
Given an ideal $I = \langle f_1,f_2,...,f_s \rangle$ *and the variety* $V(I)$ *then* $V(I) = V(f_1,f_2,...,f_s)$

In other words: the variety determined by an ideal is equivalent to the variety determined by the basis of the ideal.
This is the most important observation, because this links varieties, which are geometric objects, to ideals, which are algebraic objects. Every variety $V(f_1,f_2,...,f_s)$is not only determined by its polynomials $f_1,f_2,...,f_s$ but equivalently by the **ideal** $I = \langle f_1,f_2,...,f_s \rangle$ generated by these polynomials.
As it is possible to change the basis of an ideal without changing the ideal itself, we now have a powerful tool to determine the solution points of a variety $V(f_1,f_2,...,f_s)$: We change the basis of the ideal to a groebner basis $g_1,g_2,...,g_t$ and determine the variety $V(g_1,g_2,...,g_t)$ which is equivalent to the variety $V(f_1,f_2,...,f_s)$.

$$V(f_1,\ldots f_s) \xrightarrow{\text{is equivalent to}} V(I = \langle f_1,\ldots f_s \rangle) \xrightarrow{\text{is equivalent to}} V(I = \langle g_1,\ldots g_t \rangle) \xrightarrow{\text{is equivalent to}} V(g_1,\ldots g_t)$$

1.2.3 Groebner Bases and their Properties

Following a definition for S-polynomials will be given. In later sections these polynomials will play a key role in computing a Groebner Basis.

(1.2.3.1)

Let $f,g \in k[x_1 , ... ,x_n]$ *be nonzero polynomials. The least common multiple of the leading terms of f and g will be denoted as LCM(LM(f),LM(g)). An S-Polynomial S(f,g) is defined as*

$$S(f,g) = \frac{LCM(LM(f),LM(g))}{LT(f)} \cdot f - \frac{LCM(LM(f),LM(g))}{LT(g)} \cdot g$$

We take $f=2x^3y -3xy+1$, $g=3 x^2y^2+5x^2-y$ as an example in R[x,y] with tdegrevl ordering.
We find $LM(f)= x^3y$, $LM(g)= x^2y^2$, $LCM(x^3y, x^2y^2)=x^3y^2$, $LT(f)=2x^3y$, $LT(g)=3x^2y^2$.

$$S(f,g) = \frac{y}{2} \cdot f - \frac{x}{3} \cdot g = x^3y^2 - \frac{3xy^2}{2} + \frac{y}{2} - x^3y^2 - \frac{5x^3}{3} - \frac{x}{3} = -\frac{3xy^2}{2} - \frac{5x^3}{3} + \frac{y}{2} - \frac{x}{3}$$

The point about S-Polynomials is, that they are cancelling the leading terms of the polynomials *f* and *g*. In the example the leading terms multiplid with the factors both yield x^3y^2 and thus are cancelled.

Now we are ready for the definition of Groebner Bases.

(1.2.3.2)

Given a polynomial $f \in k[x_1 , \dots ,x_n]$ and an ideal $\mathbf{I} = \langle g_1,g_2,\dots,g_s \rangle \in k[x_1 , \dots ,x_n]$ that is generated by the polynomials $g_1,g_2,\dots,g_s$.
*We will call $\mathbf{G} = \{g_1,g_2,\dots,g_s\}$ a **Groebner Basis** of **I** if all the remainders $r_{ij} = NF(S(g_i,g_j)) = 0$ where $S(g_i,g_j)$ are the S-polynomials of the pairs $i \neq j$ and $i,j \leq s$.*

This definition can be used to determine if the basis of a given ideal is a Groebner Basis. All we need to do is check if all possible S-polynomials reduce to zero. Considering the ideal $\mathbf{I}=\langle y-x^2, z-x^3 \rangle$ we want to check if $\mathbf{G}=\{y-x^2, z-x^3\}$ is a Groebner Basis w.r.t. to lex ordering y>z>x of **I**. The only possible S-polynomial is $S(y-x^2, z-x^3)= -zx^2+yx^3$ because the pairs i=j are not to be checked.
Using the Normalform algorithm one finds that $NF(-zx^2+yx^3)$ mod **G** yields 0 and thus **G** is a Groebner Basis.

(1.2.3.3)

Let $\mathbf{G} = \{ g_1,g_2,\dots,g_s \}$ be a Groebner Basis of an ideal $\mathbf{I} \in k[x_1 , \dots ,x_n]$ and f a polynomial.

*Then r = NF(f) mod **G** is uniquely determined. Particularly r is determined no matter how $g_1,g_2,\dots,g_s$ are listed when using the Normalform algorithm.*
*Furthermore NF(f) mod **G** = 0 if and only if $f \in \mathbf{I}$*

Now the ideal membership problem for a polynomial f can be solved. One needs to fix a monomial ordering, find a Groebner Basis and compute the normalform of f mod the Groebner Bases. If f reduces to zero f is an element of the ideal.

Still every ideal has many different Groebner Bases, there are even different Groebner Bases with respect to a specific monomial ordering. These Groebner Bases are sometimes larger than necessary, some of the basis polynomials can be eliminated using the following criterion.

(1.2.3.4)

Let $\boldsymbol{G} = \{ g_1,g_2,...,g_s \}$ *be a Groebner Basis of a polynomial ideal* $\boldsymbol{I} \in k[x_1 , ... ,x_n]$ *and* $g_i \in \boldsymbol{G}$.
If $LT(g_i)$ *is divisible by one of the other leading terms of* $\boldsymbol{G}\text{-}\{g_i\}$ *then* $\boldsymbol{G}\text{-}\{g_i\}$ *is also a Groebner Basis.*

In other words, if the leading term of a base polynomial g_i *in a Groebner Basis is divisible by one of the leading terms of the other basis polynomials then* g_i *can be eliminated from the Groebner Basis.*

(1.2.3.5)

Given a an ideal $\boldsymbol{I} \neq \{0\}$ *with the Groebner Basis* $\boldsymbol{G} = \{ g_1,g_2,...,g_s \}$ *we will call*
$\boldsymbol{G'} = \{ g'_1,g'_2,...,g'_s \}$ *with* $g'_i = NF(g_i) \bmod \boldsymbol{G}\text{-}\{g_i\}$ *a* ***reduced Groebner Basis****.*

For a given monomial ordering > every ideal $\boldsymbol{I} \neq \{0\}$ *has a unique reduced Groebner Basis.*

So the reduced Groebner Basis of an ideal is obtained by reducing every base polynomial $g_1,g_2,...,g_s$ modulo the other base polynomials.
The fact that every ideal has a unique Groebner Basis for a given monomial ordering allows to determine if two sets of polynomials $\{ f_1,f_2,...,f_s \}$ and $\{g_1,g_2,...,g_t \}$ generate the same ideal. Fixing a monomial ordering and computing the reduced Groebner Basis for $\langle f_1,f_2,...,f_s \rangle$ and $\langle g_1,g_2,...,g_t \rangle$, the ideals are equal if and only if the Groebner Bases are the same.

(1.2.3.6)

Given an ideal $\boldsymbol{I} = \langle f_1,f_2,...,f_s \rangle \subset k[x_1 , ... ,x_n]$ *we will call* $\boldsymbol{I_k} \subset k[x_{k+1}, ... ,x_n] = \boldsymbol{I} \cap k[x_{k+1}, ... ,x_n]$ *the* ***kth elimination ideal*** *of* $\boldsymbol{I}$.
$\boldsymbol{G} = \{ g_1,g_2,...,g_s \}$ *is a Groebner Basis of* $\boldsymbol{I}$ *with respect to lex ordering where* $x_1 > x_2 > .... > x_n$.
Then for every $0 \leq k \leq n$ *every set of polynomials*

$\boldsymbol{G_k} = \boldsymbol{G} \cap k[x_{k+1} , ... ,x_n] = \boldsymbol{G} - \{ g \in k[x_1 , ... ,x_k] \}$

is a Groebner Basis of the kth elimination ideal $\boldsymbol{I_k}$.

This is called the elimination theorem.
Elimination ideals $\boldsymbol{I_k}$ *are subsets of the original ideal, precisely the intersection of the original ideal* $\boldsymbol{I}$ *with the subspace* $k[x_{k+1} , ... ,x_n]$. *This subspace -and thus the subset* $\boldsymbol{I_k}$*- does not contain any monomials with the variables* $x_1,x_2,....,x_k$, *in other words* $\boldsymbol{I_k}$ *consists of all consequeces of* $f_1,f_2,...,f_s$ *which eliminate the variables* $x_1,x_2,....,x_k$.
We obtain a Groebner Basis $\boldsymbol{G_k}$ *of a kth elimination ideal* $\boldsymbol{I_k}$ *simply by removing all poynomials that contain variables* $x_1,x_2,....,x_k$ *from the Groebner Basis* $\boldsymbol{G}$.

For example we take the Ideal $\mathbf{I}=\langle x^2+y+z-1,\ x+y^2+z-1,\ x+y+z^2-1\rangle$ that has a Groebner Basis w.r.t $x>y>z$ $\mathbf{G} = \{ g_1,g_2,g_3,g_4 \}$ with

$$g_1=x+y+z^2-1 \qquad g_2=y^2-y-z^2+z$$
$$g_3=2yz^2+z^4-z^2 \qquad g_4=z^6-4z^4+4z^3-z^2$$

With the elimination theorem we find the elimination ideals:
$\mathbf{I_1} = \mathbf{I} \cap \mathbf{R}[y,z] = \langle y^2-y-z^2+z,\ 2yz^2+z^4-z^2,\ z^6-4z^4+4z^3-z^2\rangle$
$\mathbf{I_2} = \mathbf{I} \cap \mathbf{R}[z] = \langle z^6-4z^4+4z^3-z^2 \rangle$

Zero dimensional ideals, that is ideals of zero dimensional varieties (see 1.2.1) have special properties. One can show, that for every zero dimensional ideal $\mathbf{I} = \langle f_1,f_2,...,f_s \rangle \subset k[x_1 , ... , x_n]$, there exists an univariate elimination ideal $\mathbf{I_{n-1}} \subset k[x_n]$ and further a univariate polynomial g_s that generates the elimination ideal $\mathbf{I_{n-1}} = \langle f_s \rangle$. Following (1.2.3.6) f_s is the Groebner basis of $\mathbf{I_{n-1}}$ and thus f_s is the univariate polynomial with the minimum degree, because all other polynomials in the ideal $\mathbf{I_{n-1}}$ are multiples of the Groebner base polynomial f_s. So by looking at the degree of the univariate polynomial of a lexicographic Groebner Base one can immediately determine the maximium number of solutions for the variable x_n.
In our example the degree of $g_4= z^6-4z^4+4z^3-z^2$ is 6 and there is a maximum of 6 different (maybe complex) solutions for z.

In summary one can say Groebner Bases w.r.t a lex ordering $x_1>x_2>....>x_n$ deliver a system of equations, that allows successive elimination of the variables. Provided that a variety is zero dimensional, computing a reduced lex Groebner Basis $\mathbf{G} = \{ g_1,g_2,...,g_n \}$ is a systematic approach to find all the solution points because the Groebner Basis will be in the form:

$g_1(x_1)$,
$g_2(x_1,x_2)$,
.....,
$g_n(x_1,x_2,....,x_n)$.

We find the solution values for x_1 using g_1. Substituting the solution values for x_1 into $g_2,...,g_n$ we find the solution values for x_2 using g_2 etc ..... until we have all the solution points.

In the above example they are (1,0,0), (0,1,0), (0,0,1), $(-1+\sqrt{2},-1+\sqrt{2},-1+\sqrt{2})$ and $(-1-\sqrt{2},-1-\sqrt{2},-1-\sqrt{2})$.

1.3 Computation of Groebner Bases

1.3.1 Buchberger's Algorithm

In this section systematic ways to find Groebner Bases will be explored. A Groebner Basis **G**=$\{g_1,g_2,...,g_t\}$ for a given Ideal **I** =$\langle f_1,f_2,...,f_s\rangle$ can be produced following Buchberger's algorithm.
A rudimentary version of the algorithm will be presented now, an improved, more efficient version that was used for the implementation of *gfloat* will be described later.

(1.3.1.1)

Given an ideal ***I*** $\subset k[x_1 , ... ,x_n] = \langle f_1,f_2,...,f_s \rangle$ *the following algorithm (Buchberger's Algorithm) produces a Groebner Basis* ***G*** $= \{ g_1,g_2,...,g_s \}$ *of the ideal* ***I****.*

Input: A basis **B** = $\{f_1, f_2, ..., f_s\}$ of the ideal **I**
Output: A Groebner Basis **G =** $\{g_1,g_2,...,g_t\}$ of **I.**

P = a set of polynomial Pairs P := $\{f_i,f_j\}$ where $i\neq j$ and $i,j\leq s$
G := B
t := s
WHILE P $\neq$ {} DO
    f_i,f_j :=a pair in P
    P = P - f_i,f_j
    w = NF(Spoly (f_i,f_j))
    IF w $\neq$ 0 **THEN**
        t := t + 1
        f_t := w
        G = G $\cup$ $\{f_t\}$
        P = P $\cup$ $\{f_i,f_t\}$ where $i<t$

The following very small example will illustrate the algorithm. Given is the ideal $\langle x^3-2, x^2y-2y^2+x\rangle$ using tdegrevl ordering. Before the iteration (in the WHILE loop) of the algorithm starts, the pairset P has to be initialized. P := $\{f_i,f_j\}$ where $i\neq j$ and $i,j\leq s$, so P consists of all possible combinations of polynomials f where $f_i\neq f_j$. In the given example there are only two polynomials in the input basis B, so there is just one pair f_1f_2 in the pairset. So before entering the while loop, P=$\{f_1f_2\}$ and G= $\{f_1,f_2\}$.

In the first loop of the iteration, the pair f_1f_2 is taken from the pairset, the S-polynomial made from f_1f_2 is $Spoly(f_1f_2) = 2y^2x-x^2-2y$. Now the polynomial w is determined, where w is the normalform of $Spoly(f_1f_2)$ modulo the base G. As none of the headterms of the polynomials in G divides any of the terms of $Spoly(f_1f_2)$, it is already in normalform. So $w=2y^2x-x^2-2y\neq 0$ and the new polynomial $f_3=w$ is added to the basis G. Also the pairs f_1f_3 and f_2f_3 are added to the pairset. At the end of the first iteration $P=\{f_1f_3, f_2f_3\}$ and $G= \{f_1,f_2,f_3\}$.

In the second loop pair f_1f_3 is taken from the pairset to make the S-polynomial $Spoly(f_1f_3) = 1/2x^4+x^2y-2y^2$. Please note that any pair can be taken from the pairset P; f_1f_3 is just an arbitrary choice. In the first step of the normalform algorithm f_1 is used for reduction. With $1/2x^4+x^2y-2y^2 - 1/2x\cdot f_1$ w reduces to $w = x^2y-2y^2+x$. In another reduction step with $x^2y-2y^2+x - f_2$ one finds that w reduces to zero. So at the end of the second iteration of Buchberger's algorithm the basis $G= \{f_1,f_2,f_3\}$ has remained unchanged, the pairset $P=\{f_2f_3\}$ just became smaller by the pair that was taken out to build the S-polynomial.

In the third loop the remaining pair f_2f_3 is taken for $Spoly(f_2f_3) = -2y^3+1/2x^3+2xy$. This polynomial can be reduced once using $w = -2y^3+1/2x^3+2xy - 1/2\cdot f_1 = -2y^3+2xy+1$. So the situation is $P=\{f_1f_4,f_2f_4,f_3f_4\}$ and $G= \{f_1,f_2,f_3,f_4\}$ where f_4 is the polynomial $-2y^3+2xy+1$ that was added.

In the following loop one finds that the normalform $NF(Spoly(f_2f_4))$ modulo G yields zero reducing $Spoly(f_2f_4)=-2y^4+x^3y+xy^2+1/2x^2+y$ using $(-y)\cdot f_4$, $y\cdot f_1$ and $1/2\cdot f_3$. It is easy to verify that the remaining two pairs $f_1f_4$and f_3f_4 also produce S-polynomials that reduce to zero modulo the base G. As there is no more pair left in the pairset P, the algorithm terminates, the Basis $G=\{x^3-2,x^2y-2y^2+x,2y^2x-x^2-2y,-2y^3+2xy+1\}$ is a tdegrevl Groebner Basis of the ideal $\langle x^3-2, x^2y-2y^2+x\rangle$.

Concluding it can be said, that a Groebner Basis is obtained from an abitrary base by making all possible S-polynomials from the original base and applying the normalform algorithm to these S-polynomials. If the normalform of an S-polynomial does not reduce to zero it is added to the base and new S-polynomials are made with the base polynomial just added. The algorithm terminates when no more S-polynomials are left to be reduced.

1.3.2 Selection Strategies and Syzygy Criteria

This simple version of Buchberger's Algorithm shows some deficiencies: It does not produce a reduced Groebner Basis and it is not very efficient. The further can be resolved easily, in order to obtain a reduced Groebner Basis of an ideal one has to reduce every base polynomial modulo the other base polynomials. This is performed by the subalgorithm REDUCEALL. In this algorithm every basepolynomial g_i of a Basis $G=\{g_1,g_2,...,g_S\}$ is being reduced modulo the other polynomials $G-\{g_i\}$ until none of the base polynomials is reduceable by $G-\{g_i\}$.

During the iteration of Buchberger's Algorithm pairs f_i,f_j are selected from the pairset P and the S-polynomials made from the pairs are being reduced. The Algorithm produces a Groebner Basis no matter which of the pairs is selected, so some optimization can be achieved by choosing 'optimal' pairs in order to accelerate the algorithm. It has bee suggested to choose pairs f_i,f_j in P such that LCM (f_i,f_j) is as small as possible w.r.t the chosen ordering. The normalform reduction of these S-polynomial are more likely to yield nonzero, so new elements of the Groebner Basis are found sooner in the process of the algorithm.
Another selection strategy for pairs that has been proposed, was to choose pairs f_i,f_j where the leading monomial of the headterm of the corresponding S-Polynomial is minimal w.r.t. the chosen ordering.
Very effective but somewhat more complex is the so called sugar strategy [Giovini, Mora, Niesi, Robbiano, Traverso 1991]
In *gfloat* the first pair f_i,f_j in P is selected for S-Polynomial. The pairs in P are being sorted with their LCM's being minimal w.r.t. to the choosen ordering >, so always the pair with the least LCM(f_i,f_j) is taken.

Zero reductions of polynomials during Buchbergers algorithm are a waste of time, because they do not have any effect on the result. The so called syzygy criteria are being used to eliminate superflous zero reductions of S-Polynomials. [Gebauer, Möller 1988]

(1.3.2.1)

Given an ideal $I \subset k[x_1 , \ldots ,x_n] = \langle f_1,f_2,\ldots,f_t \rangle$ *w.r.t. > and a set of Pairs* $P = \{ f_i,f_j \}$ *where* $f_i,f_j \in \{f_1, f_2, \ldots\ldots, f_{t-1}\}$ *the normalform NF(S-poly*(f_if_j)*) of the S-poly of any pair* f_i,f_j *fulfilling one of the following criteria is zero.*

Criterion M: *M holds for a pair* f_i,f_k *if there exists a* $j<k$ *such that* f_j,f_k *divides properly* f_i,f_k*. M stands for multiple.*

Criterion F: *F holds for a pair* f_i,f_k *if there exists a* $j<i$ *such that* $f_j,f_k = f_i,f_k$*. F stands for first (the first pair that is equivalent).*

Criterion B: B_k *holds for a pair* f_i,f_j *if* $j<k$*, HT(*,f_k*) divides properly* f_i,f_j *and* $f_i,f_k \neq f_i,f_j \neq f_j,f_k$*. B stands for backward as the criterion eliminates pairs that were built in a previous loop of Buchberger's algorithm*

For examples see 1.3.3.

In the subalgorithm UPDATE_PAIRS each time a new polynomial f_t is added to the Base **G** the new pairs f_i,f_t with $i \leq t$ are added to the pairset P. Then all pairs are elimited, where one of the syzygy criteria hold. Finally the remaining pairs in the pairset are sorted w.r.t. >.

(1.3.2.2)

Given an ideal $\mathbf{I}=\langle f_1,f_2,\ldots, f_t\rangle \subset k[x_1 , \ldots ,x_n]$ *w.r.t. > and a set of Pairs* $P = \{f_k,f_l\}$ *where* $f_k,f_l \in \{f_1, f_2, \ldots\ldots, f_{t-1}\}$ *the algorithm UPDATE_PAIRS*(f_t) *produces a set of pairs* $P = \{f_i,f_j\}$ *where* $f_i,f_j \in \{f_1, f_2, \ldots\ldots, f_t\}$ *, none of the criteria M,F or B hold and* $\{f_i,f_j\}$ *are sorted w.r.t. >.*

Input: A basis **B** = $\{f_1, f_2, \ldots\ldots, f_{t-1}\}$ of the ideal **I** and an additional polynomial generator f_t of the ideal **I**
A set of pairs $P = \{f_k,f_l\}$ where $f_k,f_l \in \{f_1, f_2, \ldots\ldots, f_{t-1}\}$
Output: An updated pairset $\{f_i,f_j\}$ where $f_i,f_j \in \{f_1, f_2, \ldots\ldots, f_t\}$

N := $\{f_i,f_t\}$ the set of new pairs where i<t
IF criterion M holds for any pair f_i,f_t where i<t
    N := N - f_i,f_t
IF criterion F holds for any pair f_i,f_t where i<t
    N := N - f_i,f_t
P := P $\cup$ N
IF criterion B holds for any pair f_i,f_j where $i,j \leq t$
    P := P - f_i,f_j
sort P w.r.t. >

Another subalgorithm, REDUNDANT, is being used to remove redundant elements from the base. In REDUNDANT the criterion given in (1.2.3.4) is applied to the base G.
Adding REDUCEALL, REDUNDANT and UPDATE_PAIRS leads to an improved version of Buchbergers algorithm. [Gebauer, Möller 1988]

(1.3.2.3)

Given an ideal $I \subset k[x_1, \ldots, x_n] = \langle f_1, f_2, \ldots, f_s \rangle$ *the following improved version of Buchberger's Algorithm produces a reduced Groebner Basis* $G = \{ g_1, g_2, \ldots, g_s \}$ *of the ideal* I.

Input: *A basis* $B = \{f_1, f_2, \ldots\ldots, f_s\}$ of the ideal I
Output: A Groebner Basis $G = \{g_1, g_2, \ldots, g_t\}$ *of* I.

preprocessor:
B := REDUCEALL(**B**)

initialization:
P is a set of polynomial Pairs $\{f_i, f_j\}$
P := {}
G := $\{f_1\}$
FOR t=2 **TO** s
    UPDATE_PAIRS(f_t)
    G = **G** $\cup$ $\{f_t\}$

iteration:
WHILE P $\neq$ {} DO
    f_i, f_j := the first pair in P
    P = P - f_i, f_j
    w = NF(Spoly (f_i, f_j))
    IF w $\neq$ 0
        t := t + 1
        f_t := w
        G = **G** $\cup$ $\{f_t\}$
        REDUNDANT (f_t)
        UPDATE_PAIRS(f_t)

postprocessor:
G := REDUCEALL(**G**)

This is the version of Buchberger's Algorithm that is implemented in *gfloat*. In order to give a good understanding about it a detailed example is given now. Like in section 1.3.1 p.20 the ideal $\langle x^3-2, x^2y+x-2y^2 \rangle$ is given, but this time a Groebner Basis is computed using lexicographic ordering.
Apart from some additional comments, this example uses output that was created by *gfloat*. Because of that the polynomials have floating point coefficients.

1.3.3 Example

```
                    Input Base: (real)

g_{f1}  =  +0.1e1 [29953.0] * x ^ 3
-0.2e1 [29953.0]
;

g_{f2}  =  +0.1e1 [29953.0] * x ^ 2  * y
+0.1e1 [29953.0] * x
-0.2e1 [29953.0] * y ^ 2
;

Preprocessor:
reduce: Selected for reduction: f1
f1 is not reduceable
reduce: Selected for reduction: f1
f1 is not reduceable
reduce: Selected for reduction: f2
f2 is not reduceable

New pairs:
(f1,f2) = x^3y
pairs after update:
{(f1,f2)}

Buchberger main loop: run 1
Basis: [f1, f2, ]

Spoly with f1 and f2:
-0.1e1 [29953.0]* x ^ 2
+0.2e1 [29953.0]* x * y ^ 2
-0.2e1 [29953.0]* y
;
f3 is not reduceable

New pairs:
(f1,f3) = x^3, (f2,f3) = x^2y

pairs after update:
{(f1,f3) (f2,f3)}

Buchberger main loop: run 2
Basis: [f1, f2, f3, ]

Spoly with f2 and f3:
+0.2e1 [29953.0]* x * y ^ 3
+0.1e1 [29953.0]* x
-0.4e1 [29954.0]* y ^ 2
;

f4 is not reduceable

New pairs:
(f1,f4) = x^3y^3,  (f2,f4) = x^2y^3, (f3,f4) = x^2y^3
critM cancelled: (f1,f4)
critF cancelled: (f3,f4)
pairs after update:
{(f1,f3) , (f2,f4)}
f1 is redundant
f2 is redundant

Buchberger main loop: run 3
Basis: [f3, f4, ]

Spoly with f2 and f4:
-0.5e0 [29953.0]* x ^ 2
+0.3e1 [29953.0]* x * y ^ 2
-0.2e1 [29953.0]* y ^ 4

NForm: using f3  curr_mon = 1
First Poly (base poly):
g_{f3}  =  +0.1e1 [29953.0] * x ^ 2
-0.2e1 [29953.0] * x  * y ^ 2
+0.2e1 [29953.0] * y;

Second Poly (to be reduced):
```

```
f5 := -0.5e0 [29953.0]* x ^ 2
+0.3e1 [29953.0]* x * y ^ 2
-0.2e1 [29953.0]* y ^ 4
;

 f5 reduced to:
+0.2e1 [29953.0]* x * y ^ 2
-0.2e1 [29953.0]* y ^ 4
+0.1e1 [29953.0]* y

f5 is not reduceable

New pairs:
(f3,f5) = x^2y^2, (f4,f5) = xy^3
pairs after update:
{(f1,f3), (f3,f5)(f4,f5)}

Buchberger main loop: run 4
Basis: [f3, f4, f5, ]

Spoly with f4 and f5:
+0.5e0 [29953.0]* x
+0.1e1 [29953.0]* y ^ 5
-0.25e1 [29953.0]* y ^ 2
;
f6 is not reduceable

New pairs:
(f3,f6) = x^2, (f4,f6) = xy^3, (f5,f6) = xy^2
critB canceled: (f3,f5)
critM cancelled: (f4,f6)
pairs after update:
{(f1,f3), (f3,f6)(f5,f6)}
f4 is redundant

Buchberger main loop: run 5
Basis: [f3, f5, f6, ]

Spoly with f5 and f6
-0.2e1 [29953.0]* y ^ 7
+0.4e1 [29953.0]* y ^ 4
+0.5e0 [29953.0]* y
;
f7 is not reduceable

New pairs:
(f3,f7) = x^2y^7, (f5,f7) = xy^7, (f6,f7) = xy^7
critM cancelled: (f3,f7)
critF cancelled: (f6,f7)
critF cancelled: (f5,f7)
pairs after update:
{(f1,f3), (f3,f6)}
f3 is redundant
f5 is redundant

Buchberger main loop: run 6
Basis: [f6, f7, ]

S-polynomial of f3 and f6
 -0.2e1 [29953.0]* x * y ^ 5
 +0.3e1 [29953.0]* x * y ^ 2
 +0.2e1 [29953.0]* y
;

Groebner: run 6 NormalForm: run 1      using f6
Groebner: run 6 NormalForm: run 2      using f6
Groebner: run 6 NormalForm: run 3      using f7
Groebner: run 6 NormalForm: run 4      using f7
f8 reduced to zero

Buchberger main loop: run 7
Basis: [f6, f7, ]

Spoly with f1 and f3:
+0.2e1 [29953.0]* x ^ 2 * y ^ 2
-0.2e1 [29953.0]* x * y
-0.2e1 [29953.0];

Groebner: run 7 NormalForm: run 1      using f6
 f8 reduced to:
```

```
-0.4e1 [29953.0]* x * y ^ 7
+0.1e2 [29953.0]* x * y ^ 4
-0.2e1 [29953.0]* x * y
-0.2e1 [29953.0]

Groebner: run 7 NormalForm: run 2      using f6
 f8 reduced to:
+0.1e2 [29953.0]* x * y ^ 4
-0.2e1 [29953.0]* x * y
+0.8e1 [29953.0]* y ^ 12
-0.2e2 [29953.0]* y ^ 9
-0.2e1 [29953.0]

Groebner: run 7 NormalForm: run 3      using f6
 f8 reduced to:
-0.2e1 [29953.0]* x * y
+0.8e1 [29953.0]* y ^ 12
-0.4e2 [29954.0]* y ^ 9
+0.5e2 [29953.0]* y ^ 6
-0.2e1 [29953.0]

Groebner: run 7 NormalForm: run 4      using f6
 f8 reduced to:
+0.8e1 [29953.0]* y ^ 12
-0.4e2 [29954.0]* y ^ 9
+0.54e2 [29953.0]* y ^ 6
-0.1e2 [29953.0]* y ^ 3
-0.2e1 [29953.0]

Groebner: run 7 NormalForm: run 5      using f7
 f8 reduced to:
-0.24e2 [29953.0]* y ^ 9
+0.56e2 [29953.0]* y ^ 6
-0.1e2 [29953.0]* y ^ 3
-0.2e1 [29953.0]

Groebner: run 7 NormalForm: run 6      using f7
 f8 reduced to:
+0.8e1 [29951.0]* y ^ 6
-0.16e2 [29954.0]* y ^ 3
-0.2e1 [29953.0]

f8 is not reduceable
New pairs:
(f6,f8) = xy^6, (f7,f8) = y^7
critF cancelled: (f6,f8)
pairs after update:
{(f7,f8)}

Buchberger main loop: run 8
Basis: [f6, f7, f8, ]

S-polynomial of f7 and f8
;

Postprocessor:
reduce: Selected for reduction: f1
f1 is not reduceable
reduce: Selected for reduction: f1
f1 is not reduceable
reduce: Selected for reduction: f2
f2 is not reduceable


       Fully reduced Groebner Base:

g_{f6}  =  +0.1e1 [29953.0] * x
+0.2e1 [29953.0] * y ^ 5
-0.5e1 [29953.0] * y ^ 2

g_{f8}  =  +0.1e1 [29951.0] * y ^ 6
-0.2e1 [29951.0] * y ^ 3
-0.25e0 [29951.0]

The ideal is zero dimensional!
Set of reduced Terms [1,y,y^2,y^3,y^4,y^5]
The number of solutions is less or equal 6
```

1.4 Converting the Monomial Ordering of Groebner Bases

Only a lexicographic Groebner Basis allows to evaluate the actual roots of a system of polynomial equations. For zero dimensional ideals it is possible to transform a Groebner Basis w.r.t. a certain monomial ordering to a Groebner Basis w.r.t. a different monomial ordering. It turns out, that it is much more efficient in terms of number and size of the polynomials involved to use total degree reverse lexicographic ordering to compute a Groebner Basis. This already beginning to show in the extremely small examples from the previous section, the bigger the problems get (in terms of polynomial degree and number of indeterminates) the bigger grows the difference in the computational effort.
That is why in most cases it is faster to compute a lexicographic Groebner Basis indirectly by first finding the total degree basis and subsequently converting it to a lexicographic basis. Unfortunately the conversion algorithm only works for zero dimensional ideals, but still its advantages are huge. Many problems where it is totally impossible to find a Groebner Basis using Buchberger's algorithm with lexicographic ordering can easily be solved using total degree reverse lexicographic ordering and conversion of the basis.

1.4.1 Residue Class Rings

Before it is possible to outline the idea of the conversion algorithm it is necessary to explain the concept of residue class rings and its connection with Groebner Bases. We will start by defining residue class rings and providing two examples for them.

(1.4.1.1)

*Given a ring **R** and an ideal **I** $\subset$ **R**. For each b $\in$ I the **residue class** of b modulo I is defined as the set [b]=b+I={b+s $|$ s $\in$ I} where b is called the **representitive** of the residue class b+I. The set {b+I $|$ b $\in$ R} containing all residue classes determined by I will be denoted as **R/I** (R modulo I).*
*In the case that **I** is a **principal ideal** (i.e. the ideal is of the kind I=$\langle$a$\rangle$ where a $\in$ **R** is solely generated by the element a) the generator a of the ideal is called the **modulus** of **R/I**.*

Please note, that that up to now we have used ideals only in the special case of polynomial ideals. Referring to (1.2.2.1) the definition of ideals is a much more general one. An ideal can be a subset of *any* ring, yet we have only regarded the special case where **I** was a subset of the polynomial ring $k[x_1, x_2, ... x_n]$.

For a more general example of an ideal consider the ring R = **Z** where **Z** is the set of integers. An ideal $I \subset Z = \langle 7 \rangle$ consists of all integers that are multiples of 7,
I = { ...., -14, -7, 0, 7, 14, ...}.
Following 1.4.1.1 the set of all residue classes Z modulo 7 is $Z/\langle 7 \rangle = \{b+\langle 7 \rangle \mid b \in Z\}$. With the ideal $\langle 7 \rangle$ consisting of all integer multiples of 7 we obtain a total of seven residue classes and $Z/\langle 7 \rangle = \{b+\langle 7 \rangle \mid b \in Z\} = \{0+\langle 7 \rangle, 1+\langle 7 \rangle, ... , 5+\langle 7 \rangle, 6+\langle 7 \rangle\}$.
Minimal representitives of $Z/\langle 7 \rangle$ are {0,1,2,3,4,5,6} what explains the term 'residue class', the minimal representitives are the possible remainders of a reduction a/I where $a \in R$ (in the example $a/\langle 7 \rangle$ where $a \in Z$).
A visualization of the residue classes of $Z/\langle 7 \rangle$ ={[0],[1],[2],[3],[4],[5],[6]} may illustrate this, here [0] denotes the residue class $0+\langle 7 \rangle$, [1] is equivalent to $1+\langle 7 \rangle$ etc... .

		...	...	...	...	...	...	...
		14	15	16	17	18	19	20
		7	8	9	10	11	12	13
$Z/\langle 7 \rangle =$	{	**[0],**	**[1],**	**[2],**	**[3],**	**[4],**	**[5],**	**[6]**	}
		-7	-6	-5	-4	-3	-2	-1
		14	-13	-12	-11	-10	-9	-8
		...	...	...	...	...	...	...

$Z/\langle 7 \rangle$ is generated by a pricipal ideal since 7 is its only generator, so 7 is called the modulus of this residue class ring. This special case of a residue class ring over the ring of integers N, which is generated by an ideal with a single, prime gerator - the modulus - will play an important role in the next section where it will be used to define modular arithmetic.

In the following operations (+ - ·) i.e. addition, subtraction multiplication will be defined for residue classes. It can be shown that under these operations the set of residue classes R/I forms a ring, the so called residue class ring R/I.

(1.4.1.2)

*Given a ring **R** and an ideal $I \subset R$ and the set of all residue classes as R/I. Taking the residue classes $[a],[b] \in R/I$ and definig the operations*

(i) $[a] \pm [b] = [a \pm b]$
(ii) $[a] \cdot [b] = [a \cdot b]$

One can show that R/I is a ring, with unity [1]=1+I and zero element [0]=I. It is called the residue class ring of R modulo I.

Using our example Z/⟨7⟩ for two small examples we find

[8] + [10] = [1] + [3] = [4] is equivalent to [8 + 10] = [18] = [4] . ... rule (i).
[-4] · [9] = [3] · [2] = [6] is equivalent to [-4 · 9] = [-36] = [6] ... rule (ii).

This is a remarkable step: Now we are able to perform computations with residue classes in R/I, we can use any representitive of the residue class for these computations. Again: all this is not restricted to integer ideals - it applies to all ideals. For the rest of this chapter we will look at polynomial residue class rings R/I over the polynomial ring $k[x_1, ..., x_n]$ that is defined by the ideal $\boldsymbol{I} \subset k[x_1, ..., x_n]$.

(1.4.1.3)

*Given is a field $k[x_1, ..., x_n]$ and the ideal $\boldsymbol{I} \subset k[x_1, ..., x_n]$. The set of residue classes $k[x_1, ..., x_n]/\boldsymbol{I}$ is called residue class ring of $k[x_1, ..., x_n]$ modulo the ideal **I**.*

*(i) Taking $c,d \in k[x_1, ..., x_n]$ then c and d are only in the same residue class iff NF(c) mod **I** = NF(d) mod **I**.*
*(ii) NF(c) mod **I** and NF(d) mod **I** are the **standard representitives** of c and d.*

In other words: two polynomials c,d are in the same residue class if they have the same normalform modulo the ideal I.
Combining the fact that representitives of the same residue class have the same normal form with the rules on (+ - ·) in residue class rings leads to the following useful correspondences.

(1.4.1.4)

*Given is a field $k[x_1, ..., x_n]$ and the ideal $\boldsymbol{I} \subset k[x_1, ..., x_n]$. The set of residue classes $k[x_1, ..., x_n]/\boldsymbol{I}$ is called residue class ring of $k[x_1, ..., x_n]$ modulo the ideal **I**.*

*(i) NF(c) mod **I** + NF(d) mod **I** = NF(c + d) mod **I***
*(ii) NF(NF(c) mod **I** · NF(d) mod **I**) mod **I** = NF(c · d) mod **I***

1.4.2 The Set of Reduced Terms

In order to illustrate the set of reduced terms it is useful to take a look at the normalform algorithm (1.1.3.3). We can see that the algorithm terminates when all the monomials in the reduced polynomial can not be divided by any of the headterms of the basis polynomials. The set of terms that cannot be divided by any of the headterms of a Groebner Basis is called the set of reduced terms.
It is clear that a polynomial that is in normal form (i.e. it has been fully reduced modulo a Groebner Basis) only consists of monomials with reduced terms - if there were a non-reduced term the polynomial is not in normal form. So every polynomial in normal form is a linear combination of reduced terms.
Is this set of reduced terms finite? It can be shown, that a Groebner Basis of a zero dimensional ideal has to contain polynomials that have a leading term that is a power of each one of the indeterminates in the ideal (criterion 1.4.2.1 given below). These univariate headterms divide terms with higher exponents, so exponents of each indeterminate of the reduced terms are limited. If the exponents are limited the number of headterms must be finite.

(1.4.2.1)

Let $\mathbf{I} \subset k[x_1, \ldots, x_n]$ *be an ideal and* $\mathbf{G} = \{g_1, g_2, \ldots, g_s\}$ *be a Groebner Basis of* $\mathbf{I}$ *w.r.t to any monomial ordering* $>$ *. Then the Ideal is zero dimensional iff there is a* g_i *for each* $1 \leq i \leq n$ *with* $LT(g_i) = (x_n)^{\alpha}$ *where* $0 < \alpha \in N$.

The criterion can be used to check if an ideal (and the according variety) is zero dimensional. Taking the tdegrevl Groebner Basis $G=\{x^3-2, x^2y-2y^2+x, 2xy^2-x^2-2y, 2y^3-2xy-1\}$ from section 1.3.1 we find that the ideal is zero dimensional because for each of the indeterminates x,y there is a headterm of the basepolynomials with a power of x and y, namely x^3 and y^3. So the set of reduced terms of this Groebner Basis is finite and the ideal is zero dimensional.

Next the algorithm REDTERMS will be given that allows to determine the set of reduced terms. This algorithm is extremely useful, because it will turn out, that the number of reduced terms is equivalent to the number of solution points.

(1.4.2.2)

Given the Groebner Basis $\mathbf{G} = \{ g_1,g_2,...,g_s \}$ *of the ideal* $\mathbf{I} \subset k[x_1 , ... ,x_n]$ *and* $k_1 , ... ,k_n \in N$. *The algorithm REDTERMS produces the set of the reduced terms* $t \in RT(\mathbf{I}(\mathbf{G}))$ *such that* $x_1^{k_1} , ... ,x_n^{k_n}$ *does not divide t.*

Input: A Groebner Basis $\mathbf{G} = \{g_1,g_2,...,g_t\}$ of the ideal **I** and $k_1 , ... ,k_n \in N$
Output: The set R of reduced terms

```
R  := {1}
FOR  i = 1 TO n DO
        T := R
        WHILE   T≠ {} DO
                t := an element in T
                T := T \ {t}
        FOR l = 1 TO k_i
                        t := t · x_i
                        IF t is in normalform modulo G THEN
                                R := R ∪ {t}
                END
        END
END
```

As an example we will use the tdegrevl Groebner Basis from section 1.3.1 p.21 $G=\{x^3-2,x^2y-2y^2+x,2xy^2-x^2-2y,2y^3-2xy-1\}$ and $k_1,k_2 = (3,3)$. k_1,k_2 are the exponents from the univariate Headterms x^3 and $2y^3$. Choosing $k_1 , ... , k_n$ like the exponents of the univariate headterms necessarily produces all reduced terms of the ideal.

The three leftmost columns contain the loop variables of the algorithm's three loops. The outermost FOR loop's counter is i, the while loop's condition is an expression of T and the innermost loop's counter is l. The value of the counter is only printed when it was changed. The rightmost column shows the status and some comments.

i	T	l	Computations
-	-	-	R = {1}
1	-	-	Start of the outermost FOR loop T = R = {1}
	{1}	-	WHILE loop t = 1, T = T \ t = { }
	{ }	1	inner FOR loop - l goes from 1 to $k_1 = 3$ t = 1·x = x, x is in NF mod G because no headterm of G divides x $\Rightarrow$ R = R $\cup$ t = {1, x}
		2	t = x·x = x^2 , x^2 is in NF mod G $\Rightarrow$ R = R $\cup$ t = {1, x,x^2 }
		3	t = x^3 , x^3 is not in NF mod G because the headterm of x^3-2 divides x^3 $\Rightarrow$ R remains unchanged {1, x,x^2 } inner FOR loop stops
			WHILE LOOP stops
2			T = R = {1, x,x^2 }
	{1,x,x^2}		WHILE loop t = 1, T = T \ t = {x,x^2}
	{x,x^2}	1	inner FOR loop - l goes from 1 to $k_2 = 3$ t = y, y is in NF mod G $\Rightarrow$ R = R $\cup$ t = {1, x,x^2,y}
		2	t = y·y = y^2 , y^2 is in NF mod G $\Rightarrow$ R = R $\cup$ t = {1, x,x^2,y,y^2 }
		3	t = y^3 , y^3 is not in NF mod G $\Rightarrow$ R remains unchanged {1, x,x^2,y,y^2} inner FOR loop stops
			WHILE loop t = x, T = T \ t = {x^2}
	{x^2}	1	inner FOR loop - l goes from 1 to $k_2 = 3$ t = x·y, xy is in NF mod G $\Rightarrow$ R = R $\cup$ t = {1, x,x^2,y,y^2,xy}
		2	t = xy^2 , xy^2 is not in NF mod G $\Rightarrow$ R = {1, x,x^2,y,y^2,xy}
		3	t = xy^3 , xy^3 is not in NF mod G $\Rightarrow$ R = {1, x,x^2,y,y^2,xy}
			WHILE loop - t = x^2, T = { }
	{ }	1	t = x^2·y, x^2y is not in NF mod R = {1, x,x^2,y,y^2,xy}
		2	t = x^2y^2 , x^2y^2 is not in NF mod G $\Rightarrow$ R = {1, x,x^2,y,y^2,xy}
		3	t = x^2y^3 , x^2y^3 is not in NF mod G $\Rightarrow$ R = {1, x,x^2,y,y^2,xy}
			inner FOR loop stops WHILE loop stops outer FOR loop stops $\Rightarrow$ the algorithm terminates

The returned set of reduced terms is R = {1, x,x^2,y,y^2,xy}. So all polynomials that are in normalform modulo **I** are a linear combination of these reduced terms, for example NF($2y^6+y^5-3x^2$) mod **I** = $5y^2 + 8xy - 6x^2 - x + 5$.

1.4.3 Converting Groebner Bases

The conversion algorithm only works for zero dimensional ideals, that is because of a fundamental property only they have. We know that for every zero dimensional ideal there exists a univariate polynomial in each indeterminate with its degree equivalent to the number of solutions - we can find it using Buchberger's algorithm with lexicographic ordering. Non-zero-dimensional ideals do not contain univariate polynomials, so the following will not work for them.
Taking again the above Groebner Basis G=$\{x^3-2, x^2y-2y^2+x, 2xy^2-x^2-2y, 2y^3-2xy-1\}$ of the ideal ***I*** we want to find the univariate polynomial g_1 that is of degree n and has y as its only indeterminate.

$$g_1 = C_{n+1}y^n + C_ny^{n-1} + C_{n-1}y^{n-2} + ... + C_2y + C_1$$

where $C_1, ... , C_{n+1}$ are the constant rational coefficients we want to find. Please note that in this section for NF(x) mod **I** will be written NF(x). Using (1.4.1.4) we know that

$$NF(g_1) = C_{n+1}NF(y^n) + C_nNF(y^{n-1}) + ... + C_2NF(y) + C_1NF(1).$$

Furthermore every polynomial in normal form modulo a Groebner Basis can be written as a linear product of the reduced terms of the ideal. Denoting the set of reduced terms as $R_1,R_2, ... , R_s$ a polynomial in normal form is a linear product $a_1R_1 + a_2R_2 + ... + a_sR_s$. We get

$$\begin{aligned} NF(g_1) = \quad & C_{n+1}(a_1R_1 + a_2R_2 + ... + a_sR_s) \\ & + C_n(b_1R_1 + b_2R_2 + ... + b_sR_s) \\ & + ... \\ & + C_2(c_1R_1 + c_2R_2 + ... + c_sR_s) \\ & + C_1(d_1R_1 + d_2R_2 + ... + d_sR_s) . \end{aligned}$$

The expression represents the normal form of the univariate polynomial g_1. Now it is rewritten and sorted such that we get a linear combination of the reduced terms $R_1,R_2,...,R_s$

$$\begin{aligned} NF(g_1) = \quad & (a_1C_{n+1} + b_1C_n + ... c_1C_2 + d_1C_1)R_1 \\ & + (a_2C_{n+1} + b_2C_n + ... c_2C_2 + d_1C_1)R_2 \\ & + ... \\ & + (a_{s-1}C_{n+1} + b_{s-1}C_n + ... c_{s-1}C_2 + d_{s-1}C_1) R_{s-1} \\ & + (a_sC_{n+1} + b_sC_n + ... c_sC_2 + d_sC_1) R_s \end{aligned}$$

The univariate polynomial g_1 is an element of the ideal **I**, consequently $NF(g_1) = 0$ (see 1.2.3.3). So the expressions in parenthesis, that represent the coefficients of the reduced terms $R_1, R_2, \ldots, R_s$, must yield zero. One gets s linear equations, one for each reduced term. Norming these equations by setting $C_{n+1} = 1$ makes the system solvable, the final equations to find the coefficients $C_1, C_2 \ldots C_n$ is

$$
\begin{aligned}
&a_1 + b_1C_n + \ldots c_1C_2 + d_1C_1 = 0 ,\\
&a_2 + b_2C_n + \ldots c_2C_2 + d_1C_1 = 0 ,\\
&\ldots.\\
&a_{s-1} + b_{s-1}C_n + \ldots c_{s-1}C_2 + d_{s-1}C_1 = 0 ,\\
&a_s + b_sC_n + \ldots c_sC_2 + d_sC_1 = 0 .
\end{aligned}
$$

Finally we get an answer about the degree of the univariate polynomial. As there is a linear expression for every reduced term $R_1, R_2, \ldots, R_s$ we need s indeterminates $C_1, C_2 \ldots C_n$ to obtain a solvable system of linear equations, n = s

(1.4.3.1)

*Given the Groebner Basis **G** of the zero dimensional ideal $\mathbf{I} \subset k[x_1, \ldots, x_n]$ the following is equivalent*

*(i) The number of reduced terms RT(**I**(**G**)) is n.*
(ii) The univariate polynomial of minimum degree $g_1 \in \mathbf{I}$ is of degree n.
(iii) The number of solution points of the variety determined by the ideal is n.

Similar to the procedure to find a univariate polynomial

$$g_1 = C_{n+1}y^n + C_ny^{n-1} + C_{n-1}y^{n-2} + \ldots + C_2y + C_1$$

it is possible to find another polynomial

$$g_2 = x + f(y) = C_{n+1}x + C_ny^{n-1} + C_{n-1}y^{n-2} + \ldots + C_2y + C_1$$

which is the next lexicographic base polynomial. Equivalently one can determine the coefficients of ANY polynomial in the ideal that has the same number of monomials as the number of reduced terms.
This method to convert a Groebner Basis w.r.t. a certain monomial ordering to a Groebner Basis w.r.t. a different monomial ordering was first published by [Faugère, Gianni, Lazard, Mora 1993]. The algorithm that was implemented in gfloat is called CONVGROEBNER [Becker, Weispfennig 1993].

(1.4.3.2)

Given a zero dimensional ideal $\boldsymbol{I} \subset k[x_1, \ldots, x_n]$ *and its Groebner Basis* ***G*** *w.r.t. a monomial ordering* $>$ *. The algorithm CONVGROEBNER produces the reduced Groebner Basis* ***F*** *of* ***I*** *w.r.t. a different monomial ordering* $>'$ *and the set of reduced terms R of* ***F****.*

Input: A Groebner Basis **G** of the ideal **I** w.r.t to a monomial ordering $>$
the set U of reduced terms of **G**
Output: the Groebner Basis **F** of **I** w.r.t to a monomial ordering $>'$
the set R of reduced terms of **F**

F := {}; H:= {}
t := 1; R := {1}
create a new indeterminate C_1
$\underline{C} = \{C_1\}$; q:= C_1
WHILE MINTERM(H,t) $\neq$ false **DO**
t := MINTERM(H,t)
h := NF (t) mod **G(I)**
p := h + q
S := the system of linear equations in $\underline{C}$ from p = 0 where
each equation s_j is the linear expression representing the coefficient
of the reduced term $u_j \in U$
IF S has a solution **THEN**
g := t + $\sum C_i r_i$ with r $\in$ R
H := H $\cup$ {t}
F := F $\cup$ {g}
ELSE
R := R $\cup$ {t}
create a new indeterminate C_t
$\underline{C} = \underline{C} \cup \{C_t\}$
q = $C_t \cdot$h + q
END

The algorithm basicly takes terms t with increasing monomial order w.r.t. to $>'$. The normalform of these terms is multiplied with a new indeterminate C_t and added to p which represents the normalform of the new base polynomial (i.e. the univariate polynomial). Rearranging the coefficients of the reduced terms delivers the expressions that makes up the system of linear equations S. Solving this system delivers a polynomial of the Groebner Basis w.r.t. the new ordering $>'$. H is the set of headterms w.r.t. the new ordering $>'$.
The WHILE loop stops when the subalgorithm MINTERM delivers false.

The algorithm is a generalization of the ideas introduced for the univariate polynomial. It builds the new basis polynomials taking terms that are minimal w.r.t. the new term ordering **and** cannot be divided by any of the headterms h $\in$ H of the new Groebner Basis. These inevitably must be the reduced terms in the new Groebner Basis. The auxiliary algorithm MINTERM finds these terms. It is clear that MINTERM differs, depending on the new term order >' . For our purposes we want to obtain a lexicographic Groebner Basis so we use a version called LMINTERM.

(1.4.3.3)

Given is the set of all terms T in [x_1 , ... ,x_n] , its subset S $\subset$ T and a term t $\in$ T. Furthermore there is a set of terms M = {v $\in$ T | t $<_{lex}$ v and any s $\in$ S does not divide v}. The algorithm LIMINTERM returns u where
***if M≠{}** u is the minimal element $\in$ M w.r.t. lexicographic ordering*
***if M = {}** u := false .*

Input: A finite set of terms S $\subset$ k[x_1 , ... ,x_n] and a term t.
Output: If it exists, the minimal term u where u $>_{lex}$ t and u is not divisible by any s$\in$S. If u does not exist, false.

u := t
FOR i = 1 **TO** n
    u := u · x_i
    IF u is not divisible by any s $\in$ S **THEN**
        return u
    ELSE
        $u = \frac{u}{x_i^{\nu}}$ where $\nu = \text{degree}(u) \text{ in } x_i$
return false

The variable n is the number of indeterminates. Now consider t = xy^2, S={x^6,y} $\in$ [x,y,z] and z>y>x as an example.

i	Computations
-	u = xy^2
1	u = x_1·u = x·xy^2
	s_2 = y divides u
	u = y^2 = x^2y^2/x^2 = u/x_1^{ν} where ν = degree of x_1
2	u = x_2·u = y·y^2
	s_2 = y divides u
	u = 1 = y^3/y^3 = u/x_2^{ν} where ν = degree of x_2
3	u = x_3·u = z
	No term $s_k \in$ S = {x^6,y} divides u $\Rightarrow$ the algorithm returns u=z and terminates

LMINTERM ({x^6,y}, xy^2) returns z as minimal term > xy^2 w.r.t. lexicographic ordering that is not divisible by any of the terms $s_k \in \{x^6,y\}$.

Finally here is a thorough example for CONVGROEBNER. The Groebner Basis w.r.t. total degree reverse lexicographic ordering **G** = {x^3-2,x^2y-2y^2+x,2xy^2-x^2-2y,2y^3-2xy-1} from section 1.3.1 will be converted into a lexicographic Groebner Base with x>y.
Using criterion 1.4.2.1 one can see that the ideal of the Basis is zero dimensional (see p. 31), the transformation algorithm is applicable. The algorithm REDTERMS (1.4.2.2) yields the set of reduced terms T = {1,x,x^2,y,y^2,xy} (see example p. 33) , so the degree of the univariate polynomial will be six, the system has 6 solution points.
Below a very detailed description of every step in the algorithm is given.

WHILE Loop	**Computations**
-	Initialization: F := {}; H:= {} t := 1; R := {1} create a new indeterminate C_1, the set of indeterminates $\underline{C}$ = {C_1}; q:= C_1
WHILE Loop 1	LMINTERM (H={}, t=1) := y, the while loop is entered and t := y h := Normalform of t modulo the tdegrevl Groebner Basis **G**; so h := y p := h + q = y + C_1 The system S consists of two equations C_1= 0 and 1 = 0, one indeterminate and two equations, obviuosly unsolvable R := R ∪ {t} = R ∪ {y} = {1,y} a new indeterminate C_2 is created and $\underline{C}$ = $\underline{C}$ ∪ {C_2} = {C_1,C_2} q = C_t·h + q = C_2·y + C_1
WHILE Loop 2	LMINTERM (H={}, t=y) := y^2, the while loop is entered and t := y^2 h := Normalform of t modulo the tdegrevl Groebner Basis **G**; so h := y^2 p := h + q = y^2 + C_2·y + C_1 The system S consists of three equations C_1= 0, C_2= 0 and 1 = 0, it has two indeterminates and three equations, obviuosly unsolvable R := R ∪ {t} = R ∪ {y^2} = {1,y,y^2} a new indeterminate C_3 is created and $\underline{C}$ = $\underline{C}$ ∪ {C_3} = {C_1,C_2,C_3} q = C_t·h + q = C_3·y^2 +C_2y + C_1
WHILE Loop 3	LMINTERM (H={}, t=y^2):= y^3, so t = y^3 h := NF(y^3)= xy + ½ p := h + q = xy + 1/2 + $C_3$$y^2$ + C_2y + C_1 S := C_1 + 1/2= 0, C_2= 0, C_3= 0 and 1 = 0, System is unsolvable R := R ∪ {y^3} = {1,y,y^2,y^3} C_4 is created and $\underline{C}$ = {C_1,C_2,C_3,C_4} q = C_4·(xy + 1/2) + C_3·y^2 +C_2y + C_1

WHILE Loop	Computations
WHILE Loop 4	t = LMINTERM (H={}, t=y^3):= y^4, so h := NF(y^4)= $1/2x^2 + 3/2y$ p := $1/2x^2 + 3/2y + C_4 \cdot (xy + 1/2) + C_3y^2 + C_2y + C_1$ S := $C_1+1/2 = 0$, $C_2+3/2= 0$, $C_3= 0$, $C_4= 0$ and $1/2= 0$, System unsolvable R := {1,y,y^2,y^3,y^4}; C_5 is created and $\underline{C} = \{C_1,C_2,C_3,C_4,C_5\}$ q = $C_5 \cdot (1/2x^2 + 3/2y) + C_4 \cdot (xy + 1/2) + C_3 \cdot y^2 + C_2y + C_1$
WHILE Loop 5	t = LMINTERM ({}, y^4):= y^5, so h := NF(y^5)= $5/2y^2 - 1/2x$ p := $5/2y^2 - 1/2x + C_5 \cdot (1/2x^2 + 3/2y) + C_4 \cdot (xy + 1/2) + C_3y^2 + C_2y + C_1$ S := $C_1+ 1/2 \cdot C_4 = 0$, $C_2 + 3/2 \cdot C_5 = 0$, $C_3 + 5/2 = 0$, $C_4= 0$, $1/2 \cdot C_5= 0$ and $-1/2 = 0$, 5 indeterminates for 6 equations - the System is unsolvable R := {1,y,y^2,y^3,y^4,y^5}; $\underline{C} = \{C_1,C_2,C_3,C_4,C_5,C_6\}$ q = $C_6 \cdot (5/2y^2 - 1/2x) + C_5 \cdot (1/2x^2 + 3/2y) + C_4 \cdot (xy + 1/2) + C_3 \cdot y^2 + C_2y + C_1$

Up to now the crucial step - the creation of system S of linear equations – has not been described in detail.

WHILE Loop	Computations
WHILE Loop 6	t = LMINTERM ({}, y^5):= y^6, so h := NF(y^6)= $2xy + 5/4$ p := $2xy + 5/4 + C_6 \cdot (5/2y^2 - 1/2x) + C_5 \cdot (1/2x^2 + 3/2y) + C_4 \cdot (xy + 1/2) +$ $C_3y^2 + C_2y + C_1$ p := 0 and the right side of the equation is rewritten with linear expressions of $\underline{C}$ as coefficients of the reduced terms T = {1,y,x,y^2,xy,x^2} $0 = 1 \cdot (C_1+ 1/2 \cdot C_4 + 5/4) + y \cdot (C_2 + 3/2 \cdot C_5) + x \cdot (-1/2C_6) + y^2 \cdot (C_3 + 5/2C_6) +$ $xy \cdot (C_4+ 2) + x^2 \cdot (1/2 \cdot C_5)$ The right side of the equation can only yield zero if all the equations in parenthesis are zero, we get the system of linear equations S := $C_1+ 1/2 \cdot C_4 + 5/4 = 0$, $C_2 + 3/2 \cdot C_5 = 0$, $-1/2C_6 = 0$, $C_3 + 5/2C_6 = 0$, $C_4+ 2 = 0$ and $1/2 \cdot C_5 = 0$.

WHILE Loop	Computations
WHILE Loop 7	Rewriting this as a matrix gives a clear description of the situation. Each column represents one of the indeterminates in C, each row contains coefficients of one of the reduced terms of G.

$$\begin{array}{l} \\ RT = 1 \\ RT = y \\ RT = x \\ RT = y^2 \\ RT = xy \\ RT = x^2 \end{array} \begin{array}{c} \begin{array}{cccccc} C_1NF(1) & C_2NF(y) & C_3NF(y^2) & C_4NF(y^3) & C_5NF(y^3) & C_6NF(y^5) \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{3}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & \frac{5}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix} \end{array} = \begin{array}{c} NF(y^6) \\ \begin{bmatrix} -\frac{5}{4} \\ 0 \\ 0 \\ 0 \\ -2 \\ 0 \end{bmatrix} \end{array}$$

This system is solvable, the solution is $C_1 = 1/4$, $C_2 = 0$, $C_3 = 0$, $C_4 = -2$, $C_5 = 0$ and $C_6 = 0$.

With these solutions we are able to build the first polynomial g_1 of the lexicographic Groebner Basis:

$g := t + \sum C_i r_i = y^6 + C_6 \cdot y^5 + C_5 \cdot y^4 + C_4 \cdot y^3 + C_3 \cdot y^2 + C_2 \cdot y + C_1 \cdot 1$
$= y^6 - 2\, y^3 - ¼$

$H := H \cup \{t\} = \{ y^6\}$

$F := F \cup \{g\} = \{y^6 - 2\, y^3 - 1/4\}$

WHILE Loop	Computations
WHILE Loop 8	t = LMINTERM ({ y^6}, y^6):= x, so h := NF(x)= x p := x + C_6·(5/2y^2 - 1/2x) + C_5·(1/2x^2 + 3/2y) + C_4·(xy + 1/2) + C_3y^2 + C_2y + C_1 The corresponding system of linear equations is $$\begin{array}{l} \\ RT = 1 \\ RT = y \\ RT = x \\ RT = y^2 \\ RT = xy \\ RT = x^2 \end{array} \begin{array}{c} C_1NF(1) \quad C_2NF(y) \quad C_3NF(y^2) \quad C_4NF(y^3) \quad C_5NF(y^3) \quad C_6NF(y^5) \\ \begin{bmatrix} 1 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{3}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & \frac{5}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix} \end{array} = \begin{array}{c} NF(x) \\ \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{array}$$ This system is also solvable: C_1= 0, C_2= 0, C_3= -5, C_4= 0, C_5= 0 and C_6= 2. This leads to the second lexicographic basepolynomial: g := t + $\sum C_i r_i$ = x + C_6·y^5 + C_5·y^4 + C_4·y^3 + C_3·y^2 + C_2·y + C_1·1 = x +2 y^5 – 5 H := H $\cup$ {t} = {x,y^6} F := F $\cup$ {g} = {x +2 y^5 - 5,y^6 -2 y^3 - 1/4}
WHILE Loop 9	t = LMINTERM ({ x,y^6}, x):= false $\Rightarrow$ The WHILE clause is false and the algorithm terminates returning the lexicographic Groebner Basis F = {x +2 y^5 - 5,y^6 -2 y^3 - 1/4} and the set of reduced terms R = {1,y,y^2,y^3,y^4,y^5} of the new Basis.

This result is equivalent to the lexicographic Groebner Basis that was computed in the example at the end of section 1.3.3

2. Modular and Floating Point Arithmetic

One of the problems with Buchberger's algorithm is the handling of the integer coefficients of the basis polynomials. In order to eliminate certain monomials in the NORMALFORM algorithm, the monomials must be multiplied with factors that lead to coefficients that are relatively prime – which in turn leads to higher factors for subsequent reduction steps. As a result the size of the integer coefficients constantly grows.
Consider for example the input equations:
$-919\,z^2 + 705zy + 526zx - 312y^2 + 202yx - 761x^2 + 804z - 996y - 252x + 343 = 0,$
$-175z^2 - 58zy + 544zx + 893y^2 - 174yx + 915x^2 + 443z - 335y - 566x - 869 = 0$
$98z^2 + 527zy - 380zx - 880y^2 - 768yx - 465x^2 + 642z - 213y - 743x - 494 = 0.$
Fixing a monomial ordering x>y>z the lexicographic Groebner Basis can be found quite easily – only 16 loops in Buchbergers algorithm (i.e. 16 calls of NORMALFORM) with all in all 183 reduction steps have to be performed. Still the Groebner Basis looks like this:

+ 34105595830687624609579291821223031 7 z^8
- 14320874278926302151786380672021660 88 z^7
+ 31402435003293556509378480244956154 86 z^6
- 71125284124814383169067503275821445 95 z^5
+ 16910069706909592774384412990438658596 z^4
- 28863025932078164149243369438006214231 z^3
+ 28072531806464518807591314465507762769 z^2
-12525008535845455607958821781995996574 z
+ 14230905540227965494570655218439038 50

79873330314625290307567765616992977774931004436390134979052462683503465636167713131596959195828047186167316084880878022160559057541898678203945252225177198799351608330071305 9 y
+45297186596635372248843409944683967640744493854777010108043686015439356578486122174973180453151913469415144759073726921955101492353513010499110578395835574860068477433932251 96 z^7
−12705873921609838057274440213509111295642540637888600140134835948231725926794849052711406866393459094241022663296879860962453865801630363570752773151187830553529522654457868576 z^6
+24074210220854535192564503670885241519807304728693945157014219326481275655974757359291225417492327360618095037627539894925805545223200731215219923875320731812463266807299185672 z^5
−61021718178959306987620604468410964952747390609056119321330709106334726904480230441723112030998718568531089918584968249452735890827119000393213001109677498975492551940830798475 z^4
+13990108166382515972848365710738603953490429635593857966381490247237058355073206603824341680874579177138746046123730958544743639519061110284336795800963827048729294303459241587 6 z^3
−18922482709149771018018522238447311643908858474316551133863470112882607650406192252970334950642661554981797220239146770025247556248300264696308568695793099560617049869688527492 1 z^2
+11008623840370858097017392082451107836188234027241173801565900861051334563958186629046701283378130792924183908257206938723040400609298233668063370532810765896560902616545302592 7 z
+13760854054783190291514372755398400583672606068134002248554788273282087478484813801460259953783757456734199595638908452835949335225135356722147518822969134425112648355138272652

79873330314625290307567765616992977774931004436390134979052462683503465636167713131596959195828047186167316084880878022160559057541898678203945252225177198799351608330071305 9 x
+87000244395656512149885354665870915128596416911666134782739952551184531989105887967291458237548742791104679434468090791469738841810059166589576772885919226447669773339912200 00 z^7
−24359730723465369010572637651039911663256900067560753484312576300801680749783299825564652100199256404153021370158348160218292758596015539583176340676048714323540064180281974990 z^6
+46034468832417368411320593617781708291273629068915294882528394713706107266092556951510448732227921782597441317927812435778348590056378506570046730099587798935321611882219467 4 z^5
−11700805277962026276596295007010903664316855832955734972006845342133979050679295885709662927470714737041372727462156211047572991459065781016461543216127298405599085518753990777 3 z^4
+26750296433199653551004328915639283768361534716153082457069555831067016835019609802407600351314678374274702793402408297958750571755868307509695360590558517754451882888636207346 z^3
−36172387932173070700274626391875325046446135587571567104728238361609959585168968908052314521091999668732894440238377540450631630520521807251949193475593869328940011512600095117 2 z^2
+20935691771561572598332333522446656811369515875889747491011080048671116088416603422952090295369258655772074557161042311390917490597769586888015970792150959060974834880672453250 5 z
−25300786711117364941467035796821295903926516943242249466186648867628456983508137381561310270172727175645193786997896174299607718335581494827784731954248804798019355014232574579

It is obviuos that these large numbers can not be stored in regular integer data types like `int` or `long int`. They can only be processed using multiple precision integer arithmetic, which consumes a lot of processor performance as well as memory.
In Buchberger's algorithm the overwhelming majority of execution time is spent in the NORMALFORM and SPOLY routines, which are basicly multiplying and adding monomials. Compared to multiple precision integer computations, the term operations are fast. So in terms of consumption of memory and processor time Buchberger's algorithm is largely a task of manipulating very big integers (see section 4.2).
It is the main idea of *gfloat* to use floating point arithmetic instead of integer arithmetic, which limits the size of the polynmial coefficients. Thus the effort to compute Groebner Bases is reduced drastically.

2.1 Multiple Precision Floating Point Arithmetic

Floating point arithmetic represents numbers with a limited precision, they are inherently inaccurate. This inaccuracy varies, depending on the operations that were previously executed with this floating point number.
Modifying an existing software for computation of Groebner Bases simply by changing the integer polynomial coefficients to hardware floating point data types like `float` or `double` does not produce the desired result. Even for small examples the coefficients of the resulting Groebner Bases are very inaccurate, even totally incorrect. That makes it necessary to monitor the inaccuracy during floating point operations.

(2.1.1)

An integer n can be represented by a floating point number $f = m \cdot b^x$ *where* $0 \leq m < 1$ *is the mantissa, b is the base and x is the exponent of the floating point number.*
The number of digits in the mantissa is called mantissa length.

As an example take an integer 451614; representing it by a floating point number with base=10 and a mantissa length of 4 decimal digits one finds that using m=0.4516, and x=6

$$f = 0.4516E6 = m \cdot b^x = 0.4516 \cdot 10^6 = 451600$$

The floating point representation of 451614 is not accurate, this is the case because the mantissa is not large enough to store all the digits of the integer. The number has to be truncated to fit into the mantissa, that produces inaccuracy. The size of the mantissa determines the precision of the floating point representation. A floating point with infinitely large mantissa is equivalent to an integer.

Taking a base b=2 leads to binary floating point numbers. Different representations are possible for mantissa and exponent. The method used for the following examples was chosen because it is equivalent to the representation of multiple precision software floats that are being used in *gfloat*. Hardware floating point numbers are usually different, e.g. ANSI/IEEE 754-1985 standard.

Taking a look at the accuracy of floating point numbers two main reasons for inaccuracy can be identified:

- rounding error
- subtractive cancellation

Rounding takes place whenever the result of an arithmetic operation needs to be truncated to fit into the mantissa. This starts already during the conversion from integer to floating point format if the integer cannot be represented exactly. The maximum rounding error for a binary float is ½ bit. For every arithmetic operation (except for addition and subtraction of floats with equivalent exponents) the result is rounded, so in the worst case there is a loss of accuracy of n/2 bits after n operations. Fortunately rounding errors have a tendency to even out over a large number of operations so the actual situation is much better [Knuth 1981].

Subtractive cancellation takes place whenever two floating point numbers of nearly the same size are subtracted. As an example take two binary floating point numbers
a = 0.11011000110000111011001110110E111000 = 61013772159942656 and
b = 0.11011000110000111010011010101E111000 = 61013716258258994.
Performing the subtracion a-b = c leads to the following situation:

	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	0	0	1	1	1	1	0	1	1	0	E	1	1	1	0	0	0	a
-	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1	1	0	1	0	1	E	1	1	1	0	0	0	-b
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	E	1	1	1	0	0	0	c

During subtraction some of the leading bits of the mantissa are cancelled to zero. The crucial point is, that floating point numbers are normalized, that means the leading bit of the mantissa is always 1 [Knuth 1981]. Consequently the mantissa is shifted to the left for each leading zero and the exponent is decremented for every shift. During the shifting the bits on the right side of the mantissa are filled with arbitrary values, usually zeros.

1	1	0	1	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	E	1	0	0	1	0	0	c

So after a subtractive cancellation of n leading bits in the mantissa, the last n bits of the result contain arbitrary values (indicated with x). The precision loss of the mantissa can be calculated from the exponent of the floating point number, because it is decremented for every shifting step. In the above example the precision loss is

$$111000_{bin} - 100100_{bin} = 56 - 36 = 20 \text{ bits.}$$

It is important to realize, that the effect of subtractive cancellation depends on the nature of the algorithm. If no subtractions of nearly equal floats are performed, no precision loss will occur. In this case rounding errors will be the predominant source of inaccuracy. However, Buchberger's algorithm is very sensitive towards subtractive cancellation, because it is designed to eliminate certain terms in the polynomials. During that process the coefficients of the other terms are also subtracted what causes a permanent precision loss. Hardware floating point arithmetic mantissa length is by far not suficcient to preserve enough accuracy through the whole algorithm, is it necessary to use software floating point arithmetic.

Software floating point arithmetic uses the same techniques as hardware, but it does not use the hardware logic that is provided for the standard floating point data types `float` and `double`. It rather uses software to implement the same arithmetic operations. It has the advantage over harware arithmetic that it is very flexible, it is possible to use arbitrariliy long mantissas. The disadvantage of course is, that it is very much slower than hardware logic.

It uses the same representation (2.1.1) as binary and decimal floating point arithmetic, the only difference is, that it uses so called processor words instead of decimal- or binary digits. A word is the basic processing unit in the arithmetic-logical unit (ALU) of the processor, a 32bit processor has a word length of 32 bits. A word is usually represented by the data type `long int`. Now how can a processor word be a digit? Taking a look at decimal digits one finds, that there are ten different symbols 0,1,2,…,9 for a digit. Basically the symbols 0,1,2,…,9 only indicate one of the ten possible **states** of the digit. The point is, that other symbols could also be used to represent the states 0,1,…,9 of the digit, for instance every number could be represented by a set of 4 bits. So 784 could be represented as

$$784 = 7\cdot10^2\ 8\cdot10^1\ 4\cdot10^0 = [0111]\cdot10^2 + [1000]\cdot10^1 + [0010]\cdot10^0.$$

Remember that the position of a digit represents a multiplication of the digit with the base raised by the position.

Now that is exactly the idea of using words as digits. A word with a length of w-bits can be in 2^w different states. So a word can represent 2^w different numbers. Summarizing one can say that w-digit (word digit) representation uses ′digits′ with 2^w ′symbols′ represented by the bit states. Everything else is equivalent to binary or decimal floating point representation.

For an example consider a w-digit floating point representation of 10^{37}. w=64, so the base b of this representation is 2^{64}. First we will rewrite 10^{37} as multiples of the powers of the base.

$$10^{37} = 54210108624275221 \cdot (2^{64})^1 + 12919594847110692864 \cdot (2^{64})^0$$

The factors could be taken as w-digits for an integer w-digit representation of 10^{37}:

$$10^{37} = [54210108624275221][12919594847110692864]$$

[w-digit in position 1] [w-digit in position 0]

The w-digit is the expression in brackets. It does not matter how the digit is represented, binary decimal or hexadecimal, as long as it defines the 2^{64} states of the w-digit. Taking (2.2.1) the floating point representation would be

$$10^{37} = [0].[54210108624275221][12919594847110692864]E[2]$$

Software floating point representation uses w-digits because they make it convenient to build a mantissa. The mantissa digits are simply represented by an array of words (long integers). Additionally a lot of arithmetic (addition and subtraction of words for example) is directly supported by the hardware.
In *gfloat* the software floating point arithmetic was not written by ourselves, it was taken from the GNU MP multiple precision library. Nevertheless it is necessary to understand w-digit floating point representation to implement the subtractive cancellation controll.

Now we are ready for a few examples of subtractive cancellation. First we will again consider the example from the beginning of section 2 (p.42). This time the Groebner Basis is computed using multiple precision floating point arithmetic, two different mantissa lengths were chosen. First a mantissa length of 300 bits was chosen, for the second computation the mantissa had 500 bits. The third equation of the basis is printed below, in order to point out the differences the same monomials are always printed together. The first coefficient is from the floating point number with 300 bit mantissa, the second with 500 bit.

0.1e1 * x
0.1e1 * x

+0.10892277065818831987537531307425425569418288185237789583741447413183230066966592013924407062e2 * z ^ 7
+0.10892277065818831987537531307425425569418288185237789583741447413183230066912838738552646932e2 * z ^ 7

−0.30497952980689669389284569446815075987812642940443130072368184969039206041109962272231473561e2 * z ^ 6
−0.30497952980689669389284569446815075987812642940443130072368184969039206040875759568377090353e2 * z ^ 6

+0.57634342591056552565606994611195859806848595526198418124232316991544340183286297298535523568e2 * z ^ 5
+0.57634342591056552565606994611195859806848595526198418124232316991544340182990604844947323288e2 * z ^ 5

−0.14649201719612707819398044057557401560322411526924870810761150772447567554031195173399573823e3 * z ^ 4
−0.14649201719612707819398044057557401560322411526924870810761150772447567554060434667489372780e3 * z ^ 4

+0.33490899062088453242520429428117077381194370479713906158828567708920376897758483908549205017e3 * z ^ 3
+0.33490899062088453242520429428117077381194370479713906158828567708920376897892801532814290414e3 * z ^ 3

−0.45287191343703979116686737397749226068111933923430218981217607119994847431066921216899452149e3 * z ^ 2
−0.45287191343703979116686737397749226068111933923430218981217607119994847431227249595855742799e3 * z ^ 2

+0.26211116638175434506884433401113198250852979863532769158414427683716780734489209021509365844411e3 * z
+0.26211116638175434506884433401113198250852979863532769158414427683716780734564982314814655844460e3 * z

−0.31676138470070325503871091452156644934341392516242709295353793233200561308325379784694213715530 5e2
−0.31676138470070325503871091452156644934341392516242709295353793233200561308412716229603033528028 3e2

In the shaded positions, the results are different. Obviously the last digits of the first (300 bit) coefficient have been currupted by subtractive cancellation and rounding errors. Allthough only about 200 multiplication and addition operations were performed the loss of accuracy is quite sizeable.
From this example one might get the impression, that it is possible to determine the number of correct bits from a floating point operation, just by reperforming it with larger mantissa and taking the results as correct if they do not differ.
However, that is NOT so. When leading mantissa bits are cancelled, the bits that are entered on the right side during the shifting are usually constant (zeros). Whatever mantissa length is chosen, the incorrect bits in the result will hold the same values. So even if all bits of two floats are incorrect, they still have the same uncorrect value. That shows the following example given by Rump [Rump 1988].

$$f(a,b) = 33.75b^2 + a^2(a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/2b$$

Evaluating f(a,b) for a=77617 and b=33096 one finds the following wrong results for the hardware data type `float` and `double` and the result for a 128 bit software float which is correct for the first 11 decimal digits.

```
float           f= 1.17260304005317
double          f= 1.17260304005317863185
SW float   f= -0.827396059903165622456064540059894416999033115786
```

Still it is possible to find a correct result by constantly raising the mantissa length, because eventually the floats will be large enough to store correct bits. That certainly is an inconvenient method and not suited for bigger problems. It is necessary to find some estimate for the floating point error.
Only interval arithmetic can produce truly reliable error estimates because it uses strictly worst case criteria. The application of worst case criteria on the other hand is what makes interval arithmetic very pessimistic, the real error is often much smaller than the estimate [Knuth 1981]. That is because interdependencies of errors are neglected.
Applying interval arithmetic to floating point arithmetic huge mantissas would be needed to obtain a significant result, that would leave no advantage over integer arithmetic. So *gfloat* uses only an error estimate instead of exact error tracking.

Considering the fact, that rounding errors are small compared to the error due to subtractive cancellation, a usable estimate for the error of a floating point number is the number of cancelled digits. In *gfloat* 10 percent of the mantissa are reseved for rounding errors. That is a purely empirical value. If 90 percent of the mantissa has been cancelled, it is assumed that the float has no significant bits any more.
Cancellation controll is performed on bit level, every single bit that is cancelled is recorded.

(2.1.2)

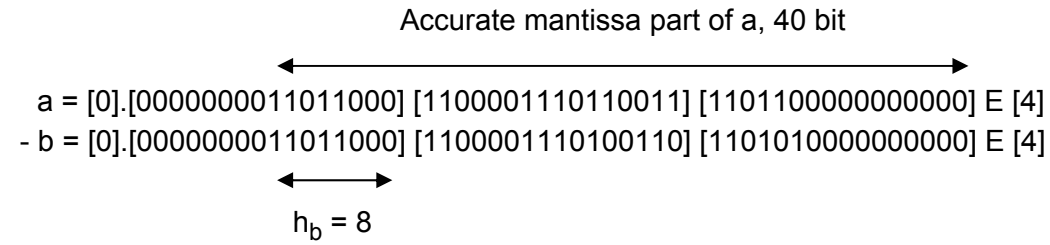
Using multiple precision floating point variables a,b with a wordlength of w bits and performing an addition or subtraction r=a±b the number of bits that are cancelled is CB. h_i *is the position of the leading bit in the highest digit of the multiple precision float i,* $expo_i$ *is its exponent.*

$$CB = max(h_a, h_b) - h_r + w(max(expo_a, expo_b) - expo_r)$$

We will again take the example a-b for a=61013772159942656 and b=61013716258258994 but this time the numbers will be represented as 48bit mantissa floating point numbers with w=16 bit wordlength. The 48 bit mantissa is split into a 48/16 = 3 word w-digit mantissa, the base is 2^{16} so

a = [0].[0000000011011000] [1100001110110011] [1101100000000000] E [4]
b = [0].[0000000011011000] [1100001110100110] [1101010000000000] E [4]

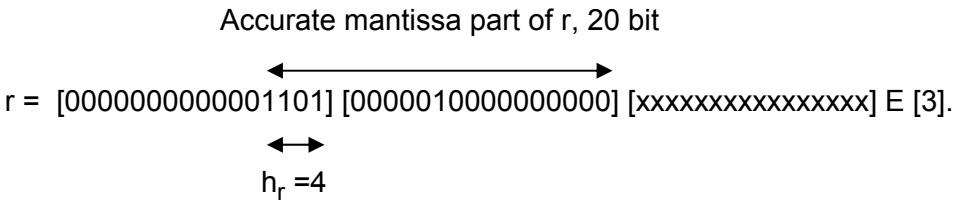
If we want to check this we take (2.1.1) and find that
$a = 11011000_{bin}\cdot(2^{16})^3 + 1100001110110011_{bin}\cdot(2^{16})^2 + 1101100000000000_{bin}\cdot(2^{16})^1$ what is equivalent to $a = 216\cdot2^{48} + 50099\cdot2^{32} + 55269\cdot2^{16} = 61013772159942656$. Now we come to the subtraction.



The unshifted result r = a-b is

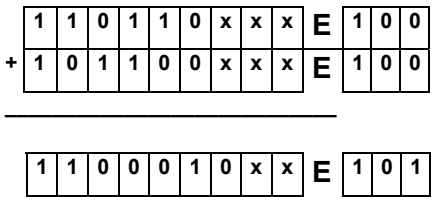
r = [0000000000000000] [0000000000001101] [0000010000000000] E [4].

The first w-digit is zero so the mantissa is shifted to the left once and the exponent is decremented, after that we have the final result



A and b have the same leading bit position $h_a = h_b = 8$ and the leading bit position of the result $h_r = 4$. The exponents $expo_a = expo_b = 4$ and $expo_r = 3$. So with wordlength w=16 the number of cancelled bits is $CB = \max(h_a, h_b) - h_r + w(\max(expo_a,expo_b) - expo_r)$
$CB = \max(8,8) – 4 + 16(\max(4,4) – 3) = 20$bits.

Please note that expression (2.1.2) not only indicates loss of bits (for CB>0), but also may return a gain (for CB<0). This can actually happen if two floats are added such that the mantissa of the result needs to be shifted to the right in order to be able to store it. Then one incorrect bit on the right side of the mantissa is eliminated.



This case can be detected by an increment of the exponent. Please note that this reflects only the very special view of error estimate based on mantissa shifting operations. Interval arithmetic delivers a different error estimate.

Obviously subtractive cancellation leads to polynomial coefficients with a different number of accurate mantissa bits. In following computations it happens, that two operands have different mantissa errors. Apart from the additional cancellation that may occur, the error of the result needs to be defined.

(2.1.3)

Using multiple precision floating point variables a,b,r with a wordlength of w (in bits) the number of accurate bits is ab_a,ab_b. The position of the leading bit in the highest digit of the multiple precision float i is h_i , its exponent is $expo_i$. Expressions (2.1.3.1) to (2.1.3.3) define the accurate bits ab_r of the result for the following arithmetic operations.

Please note that it is assumed that the exponent of a is greater equal exponent of b.

(2.1.3.1) For addition and subtraction $r = a \pm b$

$ab_r = min\ (ab_a,\ ab_b + ab_c)$ *where* $ab_c = w(expo_a - expo_b) + h_a - h_b$

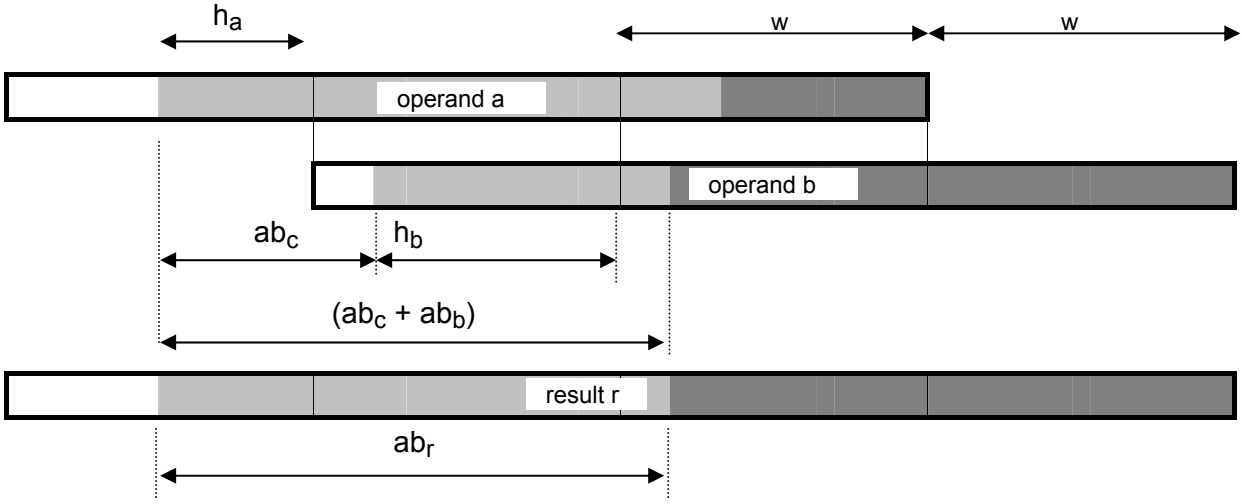
(2.1.3.2) For multiplication and division $r = a \cdot b$, $r = a/b$

$ab_r = min\ (ab_a, ab_b)$

(2.1.3.3) For square root $r = \sqrt{a}$

$ab_r = ab_a$

Expression (2.1.3.1) looks somewhat confusing, the following may illustrate the situation. A floating point addition is performed by adding the mantissa bits that represent the same integer bit, so the mantissas have to be shifted relative to each other before adding.



White indicates zeros, the gray area is the accurate mantissa part and black are corrumped bits. One can see that ab_c can be interpreted as the number of accurate bits in operand a that have a higher position after shifting than the highest nonzero bit of operand b.

Evaluating the correct digits for multiplication and division is difficult to handle precisely. That is because the exact arithmetic computations are very costly, for multiplication for instance one shift and add operation for each mantissa bit is required. In order to avoid this the GNU-mp library uses numerical algorithms in highly optimized assembler code. For our purposes exact arithmetic was assumed, what leads to (2.1.3.2).
Finding the exact square root of a number with interval arithmetic reveals a reduction of the relative error to the half. However in GNU-mp arithmetic Newton interation is used for square roots, it consists of multiplication and addition operations. Consequently the error is estimated to be constant (2.1.3.3).

Now we have a tool to monitor the precision loss in multiple precision floating point operations. For rounding errors 10 percent of the mantissa are reserved, but what happens if the rounding error exceeds this part of the mantissa?
And there are even more problems: it is impossible to determine if an expression r=a-b represented by two floats yields exactly zero. We will again take the example from the beginning of section 2.1 where an integer a=451614 is represented by a four decimal digit float 0.4516E6. Taking b=451638 it will be represented by the same floating point number! Now consider a polynomial operation where p= 79415 x^2 + 451614xy and q =9353y^2 + 451638xy. Performing r = p – q with integer coefficients we obtain

$$r = p–q = 79415\ x^2 + 451614xy – 9353y^2 - 451638xy = 794115\ x^2 - 9535y^2 –24xy$$

The same operation in floating point arithmetic yields

$$r = p–q = 0.7941e5x^2 + 0.4516e6xy – 0.9353e4y^2 - 0.4516e6xy = 0.7941e5x^2 - 0.9535e4y^2$$

The term xy was has been cancelled falsely. Of course with a larger mantissa this would not have happened, but it is impossible to find out if which mantissa length has to be used such that false cancellation of terms cannot occur.
Buchberger's algorithm reacts unpredictably upon false cancellation of terms. It may happen that this cancellation changes the structure of the ideal what in turn affects the solutions. The consequences may be disastrous, it has to be made sure that this cannot happen.

In order to address these two problems, checking if the rounding errors are exceeding the reserved part of the mantissa and determination if a subtraction yields exactly zero, a combination of floating point and modular arithmetic is introduced.

2.2 Modular Arithmetic

It will be the aim of this section to introduce a 'modular arithmetic' in order to create polynomials that have 'modular' coefficients - coefficients where the basic arithmetic operations (+ - · /) are defined differently than for integers. Using these modular coefficients for the polynomials it will also be possible to compute Groebner Bases. These so called modular Groebner Bases are then used to find important properties about their varieties. Also modular arithmetic will be irreplaceable on the way to compute Groebner Bases with floating point coefficients.

Modular arithmetic is another application of the concept of residue class rings that was already introduced in section 1.4.1. Basically modular arithmetic uses the residue classes of a residue class ring $Z/\langle m\rangle$ where the ideal lies in the ring of integers and is of the kind $I=\langle m\rangle$ with $m \in N^+$. Consequently **I** is a principal ideal, the generator m of the ideal is called the **modulus** of $Z/\langle m\rangle$. Modular coefficients are the standard representitives of $Z/\langle m\rangle$.
As an example we will again use the residue class ring $Z/\langle 7\rangle$ with its seven residue classes [0],[1],[2],[3],[4],[5] and [6].

	...	...	...	...	...	...	...
	14	15	16	17	18	19	20
	7	8	9	10	11	12	13
$Z/\langle 7\rangle$ =	**{ [0],**	**[1],**	**[2],**	**[3],**	**[4],**	**[5],**	**[6] }**
	-7	-6	-5	-4	-3	-2	-1
	14	-13	-12	-11	-10	-9	-8
	...	...	...	...	...	...	...

The first step to obtain a modular arithmetic will be to define a way to determine the residue class membership of an integer and its standard representitive.

(2.2.1)

Given a set of integer residue classes $Z/\langle m\rangle$, with a prime modulus $m \in Z$ and two integers $c,d \in Z$. The remainder upon division c/m will be denoted rem(c/m).

(i) Taking c>0 and d<0 then c and d are only in the same residue class iff rem(c/m) = rem(d/m) + m.

(ii) Taking c>0 and d>0 respectively c<0 and d<0 then c,d are only in the same residue class iff rem(c/m) = rem(d/m).

*(iii) The rem(c/m) is the **standard representitive** of c>0 and rem(d/m) + m is the standard representitive of d<0. The standard representitive is called the **modular coefficient** of the residue class [c]*

Applying this to our example Z/⟨7⟩ we want determine the residue class membership of 13, -13,8 and -8. Rem(13/7)=6 and rem(8/7)=1, rem(-13)+7= -6+7=1 and rem(-8)+7=6 so we find that 13∈[6] and -8∈[6] furthermore -13∈[1] and 8∈[1] what can also be seen in the illustration above.

(2.2.1) (iii) also provides a way to find the standard representitives of each residue class of Z/⟨m⟩ because all the remainders r are within -m < r < m (m being the modulus) if the the remainder is negative just add the modulus needs to be added. One immediate consequence is that modular coefficients all are in the range $0 \leq r < m$, they cannot grow larger.

Please recall that for two residue classes a and b the operations (+ - ·) are defined as [a]±[b] = [a±b] and [a]·[b] = [a·b] with unity [1] and zero element [0]. Also given a ring **R** and an ideal **I** ⊂ **R** the set of all residue classes denoted as R/I is a ring under the operations (+ - ·).

The following examples show integer operations and its modular eqiuvalents in Z/⟨7⟩. The representitives of the residue classes are modular coefficients (standard representitives).

integer operation	**modular operation modulus = 7**
17+23=40	[3]+[2]=[5]
13·23=299	[6]·[2]=[5]
15-31= -16	[1]-[3]=[5]

In the previous sections it has been stated, that the polynomial ring $k[x_1, x_2, ... x_n]$ is formed over a field *k*. In a field the four basic algebraic operations (+ - · /) are defined. In the residue class ring only (+ - ·) were defined. In order to be able to use modular arithmetic for polynomial coefficients it is necessary to find a modular operation for division of two modular coefficients.

This can be achieved using the so called extended euclidean algorithm which was first described around 300 B.C. by the famous ancient Greek mathematician Euclid [Becker, Weispfennig 1993].

(2.2.2)

*Given a ring **R** and a,b∈R the following algorithm computes d=gcd(a,b), the greatest common divisor of a and b as well as s and t satisfying d=as+bt.*
QUOT,REM := A/B will denote a division A/B where QUOT is the quotient and REM is the remainder.

input: *a,b*
output: d,s,t

A:=a, B:=b
S:=1, T:=0
U:=0, V:=1
WHILE B≠0 **DO**
    QUOT,REM := A/B
    A:=B, B:=REM
    S1:=S, T1:=T
    S:=U, T:=V
    U:=S1 - QUOT·U
    V:=T1 - QUOT·V
d:=a
s:=S
t:=T

As an example take a=25, b=31991. The following table shows the situation at the end of each WHILE loop.

	QUOT	REM	A	B	S	T	S1	T1	U	V
Loop1	0	25	31991	25	0	1	1	0	1	0
Loop2	1279	16	25	16	1	0	0	1	-1279	1
Loop3	1	9	16	9	-1279	1	1	0	1280	-1
Loop4	1	7	9	7	1280	-1	-1279	1	-2559	2
Loop5	1	2	7	2	-2559	2	1280	-1	3839	-3
Loop6	3	1	2	1	3839	-3	-2559	2	-14076	11
Loop7	2	0	1	0	-14076	11	3839	-3	31985	-25

At the end of loop7 the condition of the WHILE loop B≠0 no longer holds, the loop stops and the algorithm returns d=1, s= -14076, t=11. So the greatest common divisor of 25 and 31991 is 1 which is not a surprise considering that 31991 is a prime. One can easily verify that
1 = d = as + bt = 25·(-14076) + 31991·11 .

Before we continue to develop the idea of modular arithmetic it may be worth to point out some interesting relationships between greatest common divisors and Groebner Bases. Per definition the Euclidean algorithm is not restricted to the ring of integers Z, it works in any ring R. Considering the ring of univariate polynomials *k[x]* the algorithm computes the greatest common divisor of two univariate polynomials $f_1(x)$, $f_2(x)$. Cascading the algorithm one can determine the gcd of any set of univariate polynomials $f_1(x) \ldots f_i(x)$. In order to develop this the following relationship is used.

$$gcd\ (f_1(x) \ldots f_i(x)) = gcd(f_1(x), gcd\ (f_2(x) \ldots f_i(x)))$$

The gcd of a set of polynomials is equivalent to the gcd of the first polynomial and the gcd of all other polynomials. This can be repeated over and over again until the 'innermost' expresion is a gcd of the two polynomials $f_{i-1}(x), f_i(x)$ which can be computed with the Euclidean algorithm.

$$gcd\ (f_1(x) \ldots f_i(x)) = gcd(f_1(x), gcd\ (f_2(x), gcd(f_3(x), (\ \ldots .\ , gcd(f_{i-1}(x), f_i(x)))))$$

The crucial point is, that the greatest common divisor of a set of polynomials gcd($f_1(x) \ldots f_i(x)$) is a Groebner Basis of the ideal $\langle f_1(x) \ldots f_i(x) \rangle$. Looking at this aspect of gcd's of polynomials in one variable Buchberger's algorithm can be seen as a generalization of the Euclidean algorithm for the ring of polyvariate polynomials $k[x_1 \ldots . x_n]$. Furthermore it makes sense to consider a Groebner Basis of a polyvariate a ideal as a set of greatest common divisors of the generating polynomials of the ideal.

Back to modular arithmetic: The extended Euclidean algorithm delivers d=gcd(a,b) and additionally two factors s,t that satisfy d = as + bt. In the following the modular operations will take place in the residue class ring Z/⟨m⟩ where m is the prime modulus. Now if we choose modulus m and an arbitrary number a as inputs for the algorithm - like in the above example where m=b=31991 and a=25 - the gcd(a,m) is always one. This can also be expressed in modular arithmetic.

$$as + mt = gcd(a,m) = 1 \quad \Rightarrow \quad [a]\cdot[s] + [m]\cdot[t] = [1]$$

Applying the modular operations defined above we find [m]·[t] = [mt]. The expression mt obviously is a multiple of the modulus m and thus belongs to the residue class [0], so [mt]=[0]. Per definition [0] is the neutral element in the ring so finally the above expression can be written as

$$[as] + [mt] = [as] + [0] = [as] = 1$$
$$[as] = 1 \Leftrightarrow [s] = [1/a]$$

As the expression [as] equals 1, s obviously is the multiplicative inverse of a. So

$$[as] = 1 \quad \Leftrightarrow \quad [s] = [1/a]$$

Again taking the above example and a modular arithmetic with modulus=31991 we have [a]=[25] and [1/a] = [-14076]=[-14076+31991]=[17915]. So the modular coefficient of 1/25 is [17915]. In order to verify we have to check

$$[1]=[25][-14076]=[25\cdot(-14076)]=[25\cdot 17915]$$

Now it is easy to find the modular coefficients of rational numbers. For instance

$$[8568/25] = [8568]\cdot[1/25]=[8568 \bmod 31991]\cdot[-14076 \bmod 31991]=[2902].$$

Finally we have a definition for the division operation in modular arithmetic. This makes $Z/\langle m\rangle$ a field and thus it is possible to define the polynomial ring $Z/\langle m\rangle[x_1, x_2, ... x_n]$ over the field $Z/\langle m\rangle$. In other words we can use modular arithmetic for coefficients in a polynomial ring.

(2.2.3)

Given a ring Z then $Z/\langle m\rangle$ forms a field defining the following operations for $[a],[b]\in Z/\langle m\rangle$. [1/a] will be called the multiplicative inverse element of a.

(i) [a]+[b] = [a+b]
(ii) [a]·[b] = [a·b]
(iii) [1/a] = [s] where [s] is delivered by the extended Euclidean algorithm eucl_ex(a,m)

The unity is [1] and the zero element is [0]. There is no multiplicative inverse for [0].

Now we will apply arithmetic with modular coefficinets to Groebner Basis computation. Taking the same input equations as in the example p.42, namely $-919 z^2 + 705zy + 526zx - 312y^2 + 202yx - 761x^2 + 804z - 996y - 252x + 343 = 0$, $-175z^2 - 58zy + 544zx + 893y^2 - 174yx + 915x^2 + 443z - 335y - 566x - 869 = 0$ and $98z^2 + 527zy - 380zx - 880y^2 - 768yx - 465x^2 + 642z - 213y - 743x - 494 = 0$. We will first convert the integer polynomials to modular polynomials. Taking the modulus 31991 we find

$$h1 = x^2 + 27871xy + 30436y^2 + 18412xz + 8743yz + 6391z^2 + 7357x + 18414y + 29846z + 10425,$$
$$h2 = x^2 + 9230xy + 3742y^2 + 560xz + 24404yz + 19579z^2 + 3181x + 11887y + 22097z + 3810,$$
$$h3 = x^2 + 8051xy + 28553y^2 + 26144xz + 14928yz + 6054z^2 + 6331x + 1858y + 7016z + 27589$$

As expected, all the coefficients are greater equal zero and smaller equal than 31990, depending on which residue class the integer coefficient belongs to.

A monomial ordering x>y>z is fixed and the total degree reverse lexicographic Groebner Basis is computed. The result consists of six polynomials

$$f1 = 1\,z^4 + 10459z^3 + 18694xz + 8873yz + 13360z^2 + 17046x + 16299y + 5684z + 26444,$$
$$f2 = xz^2 + 20375z^3 + 17516xz + 20807yz + 30836z^2 + 22281x + 13846y + 22143z + 5192,$$
$$f3 = yz^2 + 13384z^3 + 23800xz + 17262yz + 29087z^2 + 17725x + 7145y + 11312z + 15230,$$
$$f4 = x^2 + 14389xz + 9959yz + 8506z^2 + 28318x + 23660y + 21071z + 19134,$$
$$f5 = xy + 29467xz + 22446yz + 31546z^2 + 18666x + 9405y + 15315z + 2068,$$
$$f6 := y^2 + 5615xz + 4594yz + 12907z^2 + 3101x + 266y + 31011z + 28801,$$

It is one of the main benefits of modular arithmetic, that the monomial coefficients cannot grow. On the other hand the information about the actual solutions of the input equations is lost, because the modular coefficient gives only information about the residue class of the according integer coefficient. It is impossible to find out *which* integer in the residue class was the original coefficient [Lazard 1992].

Still, only the structure (i.e. the terms in the polynomials) of a Groebner Basis can tell a lot about the solution set. In earlier sections criterion (1.4.1.4) was introduced, that allowed to determine if the ideal of a given Groebner Basis is zero dimensional – it is, if there is a univariate headterm for every basis polynomial. Applying this to the above example we find that HT(f1) = z^4, HT(f4) = x^2 and HT(f6) = y^2 so the ideal is zero dimensional, there exists a finite set of solution points.
If the ideal turned out to be zero dimensional, we can even find out the number of actual solutions because the number of reduced terms of this Groebner Basis is equivalent to the number of discrete points in the according variety (see 1.4.3.1). In order to find all reduced terms of a Groebner Basis the algorithm REDTERMS (1.4.2.2) can be used. For the example REDTERMS returns the reduced terms

$$\text{Reduced Terms } (f1,f2,f3,f4,f5,f6) = \{1,x,y,yz,xz,z,z^2,z^3\}$$

There are eight reduced terms and therefore eight solutions, the total degree of the univariate polynomial is also eight, what can be verified with the lexicographic Groebner Basis that was given at the beginning of section 2.1 (p.42)

In summary we found that modular Groebner Bases can be used to find the dimension of the solution set of a given problem, additionally one can determine the number of solutions if it is finite. Please note, that it is NOT necessary to compute a lexicographic Groebner Basis in order to get this information (example!), the criteria work on every Groebner Basis.
That is important because the size and number of polynomials involved in Buchberger's algorithm makes it often impossible to compute a lexicographic Groebner Basis, even with modular arithmetic.
So for many problems, where it is sufficient to find the number of solutions, for example position analysis in kinematics, computing a modular total degree reverse lexicographic Groebner Basis can deliver the desired information. Additionally the computation of modular Groebner Bases takes only a tiny fraction of time, compared to integer or floating point arithmetic (see section 4.2).

In the previous section it turned out to be impossible to find out if an integer subtraction a-b yielded zero when the integers were in floating point representation. In modular arithmetic this is no problem: For an integer subtraction that yields zero, the corresponding modular operation also yields zero, because the integer zero is always a member of the residue class [0].
Unfortunately all multiples of the modulus are also member of [0] (see 2.2.1). This has undesired consequences on polynomials with modular coefficients: all monomials with integer coefficients that are multiples of the modulus have the modular coefficient [0] and are thus cancelled.
For example take the input polynomial h3 = $98z^2 + 527zy - 380zx - 880y^2 - 768yx - 465x^2 + 642z - 213y - 743x - 494$ from above example. Converting it to a modular polynomial with modulus m=107 we obtain

$$h3 := x^2 + 41xy + 18y^2 + 71xz + 6yz + 87z^2 + 49x + 26y + 103$$

The monomial 697z has vanished. Considering that the integer coefficient 642 can be expressed as $6 \cdot 107$ it is clear that the modular coefficient is [0], the term has been cancelled allthough its integer coefficient is nonzero.

This of course raises serious problems; it was already mentioned in the previous section that false cancellation must not occur in Buchberger's Algorithm. But is it possible to find out if a modular monomial had been cancelled falsely?

It may help to recompute the modular Groebner Basis with several different moduli. If the result is equivalent, meaning that the structure of the terms in the bases are identical (the coefficients will of course be different) that is a strong hint that it is correct. Still it is not very secure and not satisfying.
Combining modular with multiple precision floating point arithmetic will open new ways to address this problem.

2.3 Combined Modular and Floating Point Arithmetic

In both, modular and floating point represenatation of integers some specific information about the original number is lost.
In floating point representation the ‘big picture‘ about the represented integer is still correct. Only a limited part of digits of the integer is represented, these are the most significant digits (highest places). So floating point arithmetic is accurate within the part of the mantissa that has not been currupted by rounding errors or cancellation. It gives a coarse idea about the absolute value of the represented integer, but no information about the exact value.
In contrast to that, modular representation is perfectly able to distinguish integers that are in close neighbourhood. Integers within plus/minus half of the modulus will be assigned to different residue classes thus having different modular coefficients. So the information about the exact value in the least significant digits of the integers is preserved. Modular arithmetic gives a good idea about the detailed location of the represented integer but no information about its order of magnitude.

So combining them gives us the best of two worlds. Floating point information is suited to have an approximation of the integer value of the monomial coefficient. Modular information is suited to find out, if the integer value of a coefficient is exactly zero and the term has to be cancelled.

These properties allow it to solve the problems mentioned in previous sections.

- It is possible to determine when all the significant digits in the mantissa have been cancelled (cancellation failure). This is indicated by a subtraction where the floating point coefficient is zero and the modular coefficient is nonzero. In that case the mantissa length needs to be increased and the computation must be restarted.
  Please note that only the result of the modular operation decides, if a floating point zero is interpreted as a real zero or as a cancellation failure.

- In order to find out if an expression yields zero the modular **and** the floating point coefficient must be zero. Rounding error and subtractive cancellation make it possible that a floating point operation yields nonzero even if the corresponding integer result is zero. So a new definition for a zero result is given here: A subtraction of two floating point numbers is zero if all significant digits in the mantissa were cancelled. Significant are these digits, that are neither reseved for rounding errors nor have been corrupted by subtractive cancellation.

- It is possible to detect false cancellations of monoms in modular arithmetic. That happens when the integer represented by the modular coefficient is a multiple of the modulus. The modular result of the operation is zero, the floating point result is nonzero.A false cancellation makes it necessary to reperform the modular Groebner Basis computation with a different modulus.

- Still there exists a very rare case where a modular result is zero and the float result is nonzero, that was NOT caused by a multiple of the modulus. It happens, when the rounding errors exceed the reserved area of the mantissa (rounding failure) **and** the modular zero is a real zero. This is very unusual, because normally all mantissa bits are cancelled in subtractions before the rounding errors becomes so big.
  It is a strong indicator for rounding failure if computations with several different moduli yield modular zero and floating point nonzero at the very same operation
  (i.e. the same monom and the same NORMALFORM reduction step). Interpreting this as a false modular cancellation would assume, that the integer coefficient was a multiple of of each of the moduli, which is extremely unlikely.
  After rounding error failure the computation has to be reperformed with increased mantissa length.

For combined modular and floating point arithmetic remains only the very unlikely occurrence, that the integer coefficient is a multiple of the modulus and all significant mantissa bits are cancelled in the floating point operation (cancellation failure). Then the corresponding monomial would be cancelled falsely.
This can be summarized in the coefficient test [Lösch 1996]

(2.3.1)

Two integers a,b are represented by a pair a multiple precision floating point coefficient af,bf and a modular coefficient am,bm. A subtraction r=a-b is represented by rf=af-bf and rm=am-bm. The floating point result rf is defined to be 0.0 (zero) if all significant bits in the mantissa are cancelled.
The following coefficient test interprets the result of the subtraction:

IF rm = [0] **THEN**
    IF rf = 0.0 **THEN**
        r = 0 or r is a multiple of the modulus and cancellation failure
    ELSE
        r ≠ 0
        repeat the computation with a different modulus
        IF several computations with different moduli fail at the same monom **THEN**
            r = 0,
            rounding failure, repeat computation with longer mantissa
ELSE
    -- REMARK: rm ≠ [0] --
    IF rf = 0.0 **THEN**
        cancellation failure, repeat computation with longer mantissa

Repeating the computation with different moduli lowers the likelihood of a false monomial cancellation. As it was described for rounding failure, the integer coefficient would have to be a multiple of each of the different moduli.

As a final check *gfloat* automatically substitutes the solution points back in the input equations. At this point – to the latest - a wrong result due to false momial cancellation should be identified.

Finally here is the combined modular / floating point Groebner Basis of the example that has been used allthrough this chapter.

It was computed with floating point mantissas of 8*64 = 512 bits. Please note that the number enclosed in brackets at the end of the floating point coefficient indicates the estimated number of correct bits, i.e. the number of bits that have not been affectd by subtractive cancellation minus the 10% of the mantissa that is reserved for rounding error. One can see that only about half of the mantissa digits is estimated to be correct.

x	0.1e1 [454.0]x
+ 28044z ^ 7	+0.10892277065818831987537 53e2 [249.0]z ^ 7
+ 27472z ^ 6	-0.30497952980689669389284 57e2 [249.0]z ^ 6
+ 2050z ^ 5	+0.57634342591056552565606 99e2 [248.0]z ^ 5
+ 15052z ^ 4	-0.14649201719612707819398 04e3 [249.0]z ^ 4
+ 22332z ^ 3	+0.33490899062088453242520 43e3 [249.0]z ^ 3
+ 29078z ^ 2	-0.45287191343703979116686 74e3 [249.0]z ^ 2
+ 13105z	+0.26211116638175434506884 43e3 [251.0]z
+ 17394	-0.31676138470070325503871 09e2 [250.0]

y	+0.1e1 [321.0]y
+8054z ^ 7	+0.56711278242947111933155 34e1 [265.0]z ^ 7
13072z ^ 6	-0.15907529924645342633261 71e2 [265.0]z ^ 6
+ 19237z ^ 5	+0.30140486350105776876250 64e2 [265.0]z ^ 5
+ 14625z ^ 4	-0.76398114287449287667392 7e2 [266.0]z ^ 4
+ 6301z ^ 3	+0.17515368535748712203195 48e3 [266.0]z ^ 3
+ 4208z ^ 2	-0.23690614419873450404340 43e3 [266.0]z ^ 2
+ 7905z	+0.13782602774927881565050 13e3 [267.0]z
+ 19986	-0.17228346433757620108127 41e2 [267.0]

z ^ 8	0.1e1 [268.0]z ^ 8
+ 13246z ^ 7	-0.41989808212177977648321 58e1 [268.0]z ^ 7
+ 1523z ^ 6	+0.92074142786381668220572 89e1 [268.0]z ^ 6
+ 26822z ^ 5	-0.20854432357055343981749 08e2 [268.0]z ^ 5
+ 8953z ^ 4	+0.49581510878323975329252 66e2 [268.0]z ^ 4
+ 21630z ^ 3	-0.84628417211546596592033 03e2 [268.0]z ^ 3
+ 30387z ^ 2	+0.82310632970104397789048 65e2 [268.0]z ^ 2
+ 6218z	-0.36724203846266394181449 58e2 [268.0]z
+ 17318	+0.41726013557643942667785 68e1 [268.0]

3. The software *gfloat*

Gfloat finds the solution points of systems of nonlinear polynomial equations by evaluating a lexicographic Groebner Base of the input system. The key point is, that gfloat uses combined modular and floating point arithmetic for the polynomial coefficients, what results in a drastic reduction of computation time.
The error in the floating point computations is estimated by monitoring subtractive cancellation of mantissa digits. Multiple precision Floating point arithmetic with arbitrary mantissa length is used, what makes it possible to increase the precision of the computations if necessary.
The solution of the input equations consists of four steps (Figure 3a). First a **modular** Groebner Basis is computed. Usually that is a total degree reverse lexicographic basis. During this process, all the steps, that were necessary to produce the Groebner Basis are recorded in a trace file [Traverso 1988]. Superfluous computations occur in Buchberger's algorithm, S-polynomials that reduce to zero have no effect on the result. These operations are omitted in the trace file. At the end of the first step the number of solution points can already be determined from the structure of the modular Groebner Basis.

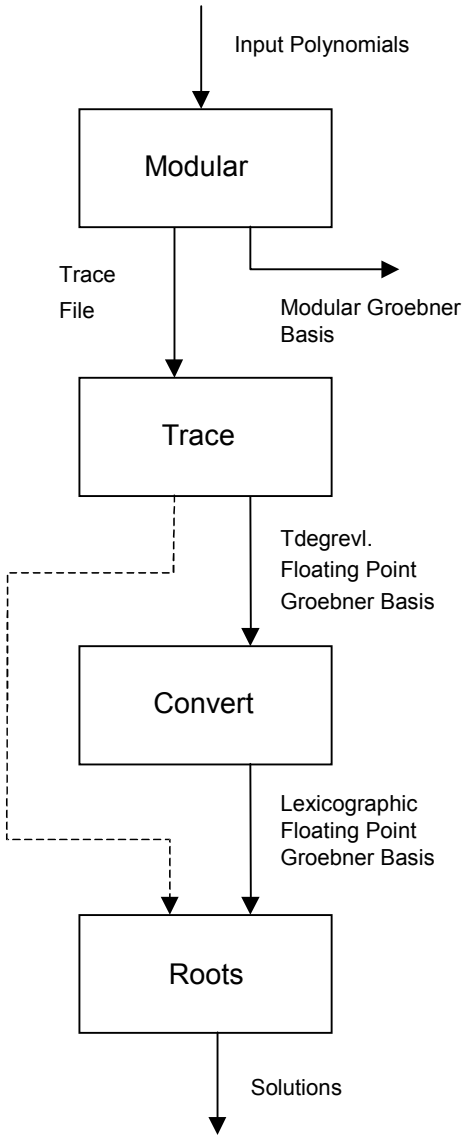


Fig. 3a
Flow-Chart of *gfloat*

The second step, **trace** performs exactly the operations that have previously been recorded in the trace file. It uses modular/floating point arithmetic so the output of step two is a floating point Groebner Basis. Usually one will first compute a total degree reverse lexicographic Groebner Basis and then convert it to a lexicographic Basis, because this is much faster than directly making a lexicographic Basis. But it the other is also possible as indicated by the dashed arrow in Fig. 3a.
The total degree reverse lexicographic Basis is then **convert**ed into a lexicographic Groebner Basis. This is only possible for systems with a finite number of solution points (zero dimensional solution).
The final routine **roots** numerically finds the actual solutions of the lexicographic base using Laguerre's method.
For a detailed description of installation and operation of gfloat see appendix A.

The structure of *gfloat* is shown in Figure 3b. The central routine is gfloat. It invokes the modules modular, trace, convert and roots which actually compute the Groebner Basis and the solutions. These four modules are in fact independent executables that are built seperately, they are activated by a system call. The floating point arithmetic routines are compiled seperately and linked during the built.
The modules are not directly controlled by the gfloat, they are rather selecting some options according to the ‘options‘ file in the ‘gfloat‘ directory. For details about the options see appendix A.

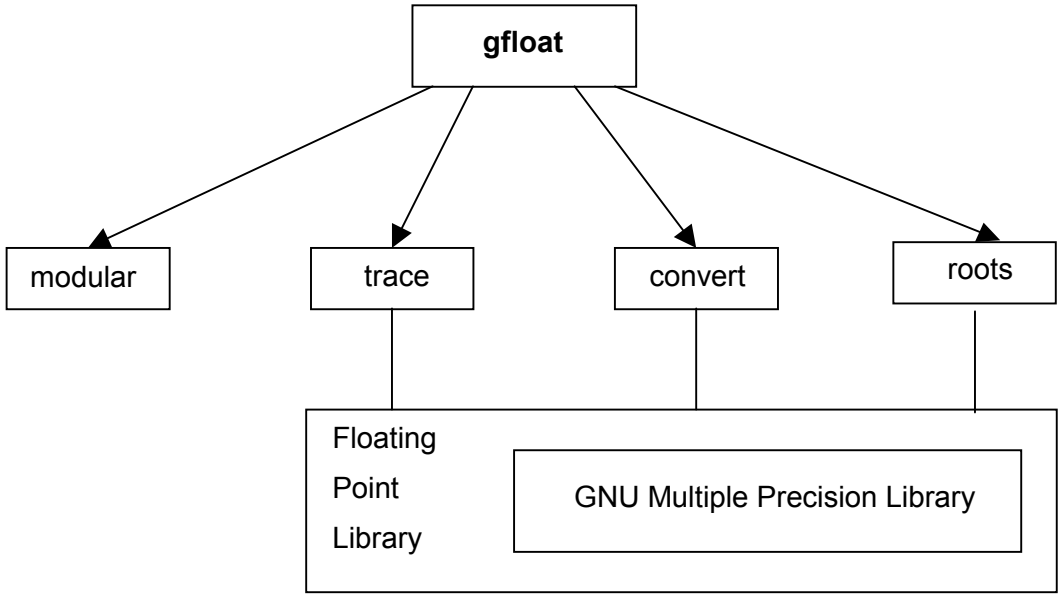
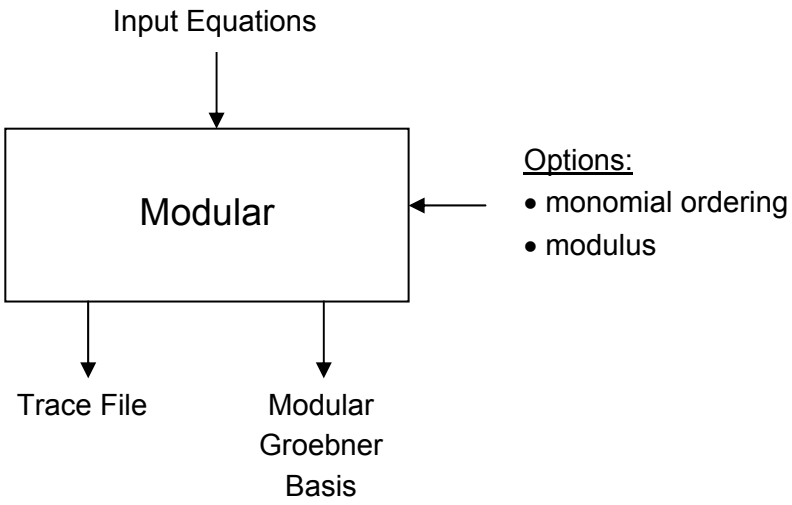


Fig. 3b
The structure of *gfloat*

Each of the routines modular, trace and convert produces an output file that the subsequent routine uses as input. That may look inefficient, but there are reasons for that: The four modules of gfloat are fully encapsulated. The tasks performed by each of the modules are not similar enough, that the benefit of sharing code would have outweighed the additional overhead. Error checking is very simple, if the file with the result from the previous step was not generated an error is assumed and the execution is aborted.

Gfloat keeps track of some indicators (see 4.2) by regularely writing them to a status file with the extension .buchb_info. That makes it possible to see the progress of the algorithm in the modular run. Additionally one can estimate the memory necessary for the trace by multiplying the mantissa size with the maximum number of monoms.
The most important results of the different modules are written to a file with the extension .result. It is a kind of summary of every run of gfloat.

3.1 Modular

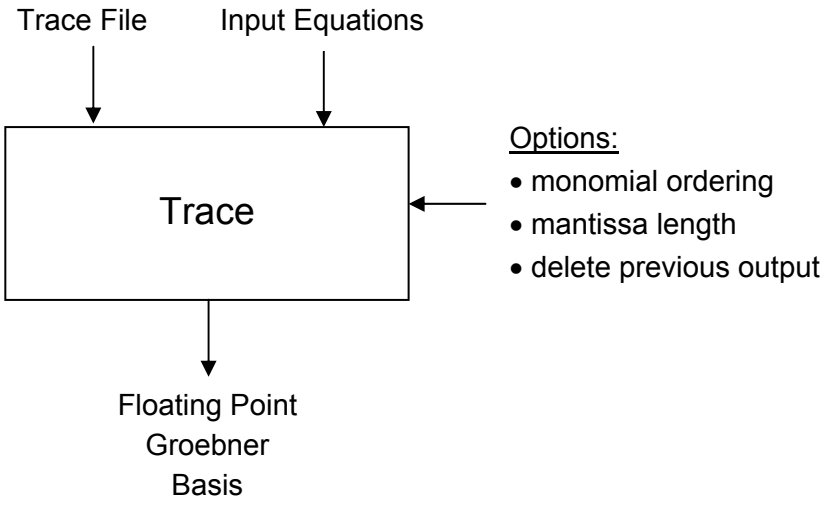


The first module produces a reduced modular Groebner Basis and records the operations that were performed to obtain this Basis. Buchberger's algorithm is implemented exactly like in (1.3.2.3). Criterion (1.4.3.1) and the algorithm REDTERMS (1.4.2.2) are applied to the Groebner Basis to find out, if the solution of the input equations is zero dimensional and the number of solutions (if finite).
The options (that are set in the 'options' file) allow the following selections:

- The modulus of the modular arithmetic.
- The monomial ordering (tdegrevl or lex) that is used for the polynomials. Consequently the according Groebner Basis is found. Please note that the subsequent trace run will use the same ordering.

All the decisions in Buchberger's Algorithm are taken in the modular run, the trace simply repeats the operations that modular writes to the trace file. Particularely the selection of the pairs (see section 1.3.2) the decision which monom to eliminate in the NORMALFORM reduction (1.1.3.3) and, most important, the application of the syzygy criteria (1.3.2.1) is performed in this module.
Some statistics about the status of the algorithm are automatically logged into a .buchb_info file (see A.2). That makes it possible to check during runtime if the algorithm is already terminating (see section 4.2).

3.2 Trace



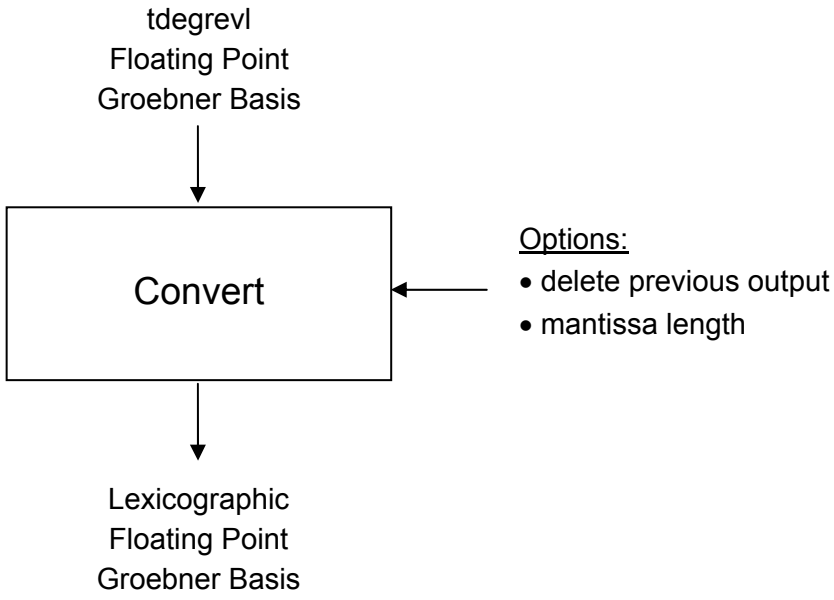
The second module takes the input equations and uses the information in the trace file to build the same Groebner Basis that was already found in the modular run, only with modular/floating point coefficients.
The following options are available:

- The monomial ordering (tdegrevl or lex) of the basis to be computed.
- The mantissa length for the multiple precision floating point arithmetic.
- The option ‘delete previous output’ is normally set to ‘YES’. However, sometimes the modules trace and convert have to be rerun, usually because of a failure in the coefficient test (2.3.1). In this case previous results could be reused when setting ‘delete previous output’ to ‘NO’.

All the actions will be taken in trace are recorded in the trace file. Each instruction is encoded in a record, which takes one line in the file. A record starts with one character that identifies instruction type and additional characters with informations about the operands.

s …… Sort the base
c …… Number of polynomials that reduced to zero
0 …… NORMALFORM reduction step in PREPROCESSOR
1 …… Construction of S-Polynomial (see 1.2.3.1)
2 …… NORMALFORM reduction step in POSTPROCESSOR
3 …… NORMALFORM reduction step of S-Polynomial
4 …… Polynomials in irredundant Groebner Basis
5 …… Cancellation of redundant basis polynomial

3.3 Convert



The Convert module is only executed if previously a total degree reverse lexicographic Groebner Basis was computed. Then the tdegrevl Basis is converted into a lexicographic Base following the CONVGROEBNER algorithm (1.4.3.2). Substantial speeding up of CONVGROEBNER was achieved by using correspondence (1.4.1.4 (ii)) for NORMALFORM computations.
The output Basis is stored in a file with the extension .lex[mant] where [mant] is the mantissa length in bits. The options are the same as in the trace.

Please note that if a coefficient test (2.3.1) fails in this module, the trace run has to be repeated also. The same applies if longer mantissas are selected in order to obtain a higher precision of the result.
The floating point coefficient with the lowest precision is detected automatically and printed to the .result file (see A.2), that makes it possible to find out if the estimated error is already larger than the numerical precision of the roots module.

3.4 Roots

In order to find the actual solution points of the lexicographic Groebner Basis first of all the roots of the univariate polynomial $p_n(x)$ have to be found (n being the degree of the polynomial). Laguerre's method [Press et al. 1990] is used to find a root r_i.

Then the degree of the polynomial is reduced to $p_{n-1}(x)$ by dividing $p_{n-1}(x) = \frac{p_n(x)}{x - r_i}$.

So the next root can be evaluated in the reduced polynomial $p_{n-1}(x)$ and so on until all are found. These roots are then substituted to the other basis polynomials to obtain the remaining coordinates. Note that each root of the univariate polynomial represents just one solution point, because all other lexicographic base polynomials substituted with the value of r_i become linear expressions.
Laguerre's method is a numerical algorithm that iterates in the complex plane and approximates one root of a polynomial. There are other methods that are converging faster, but Laguerre's method is extremely stable, because it always finds its way to a root, even if the starting point of the iteration is very far away.

(3.4.1)

Taking a univariate polynomial $p_n(x)$ of degree n the following algorithm (Laguerre's method) finds an iteration value r_i approximating a root R_i of $p_n(x)$ such that $|R_i - r_i| < |eps|$.

Input: a univariate polynomial $p_n(x)$ of degree n
a starting point r_s of the iteration
an approximation error eps.
Output: an approximation r_i of a root R_i

i=1
$r_i = r_s$
DO

$$b = \sqrt{(n-1)\cdot[(n-1)p'(r_i)^2 - np(r_i)p''(r_i)]}$$

$$a = \frac{n \cdot p(r_i)}{\max(\,|p'(r_i)+b|\,,\,|p'(r_i)-b|\,)}$$

$r_{i+1} = r_i - a$
WHILE $|a| \geq |eps|$

It is extremely effective to combine Horner's Scheme with Laguerre's method. It is very fast to evaluate the value of the polynomial and the derivatives in the iteration point, because it is not necessary to compute expressions with high powers. Additionally it can be used to perform the division $p_{n-1}(x) = \frac{p_n(x)}{x - r_i}$.

A problem arises with Laguerre's method. The expression for b in (3.4.1) includes a square root. The standard formula for finding the root $\sqrt{z}$ =x+yi of a complex number z=c+di is

$$x = \sqrt{\frac{\sqrt{c^2 + d^2} + c}{2}} \text{ and } y = \text{sgn}(d) \cdot \sqrt{\frac{\sqrt{c^2 + d^2} - c}{2}}.$$

When the iteration converges towards a real solution, the imaginary part of b goes towards zero and in the evaluation of the square root the expressions $\sqrt{c^2 + d^2} - c$ becomes $\sqrt{c^2} - c$ what leads to subtractive cancellation of all mantissa bits if c<0. That can be avoided for the formula $x = \sqrt{\frac{\sqrt{c^2 + d^2} + c}{2}}$ by multiplying it with $\sqrt{\frac{\sqrt{c^2 + d^2} - c}{\sqrt{c^2 + d^2} - c}}$. The analog procedure for the formula of y avoids cancellation for c>0.

(3.4.2)

Given a complex number z=c+di and its square root $\sqrt{z}$ = x+yi. The following expressions compute $\sqrt{z}$ avoiding subtractive cancellation.

(3.4.2.1) for c>0: $x = \sqrt{\frac{\sqrt{c^2 + d^2} + c}{2}}$, $y = \text{sgn}(d) \cdot \frac{b}{\sqrt{2\sqrt{c^2 + d^2} + 2c}}$

(3.4.2.2) for c<0: $x = \frac{b}{\sqrt{2\sqrt{c^2 + d^2} - 2c}}$, $y = \text{sgn}(d) \cdot \sqrt{\frac{\sqrt{c^2 + d^2} - c}{2}}$

As an example consider the univariate polynomial p_5=256x^5-1024x^4+992x^3-752x^2+265x-75 with a starting point r_1=0. In the first loop of the iteration p(0)=-75, p'(0)=265, p''(0)=-1504. With n=5 we get b= $\sqrt{4[4 \cdot 265^2 - (-75)(-1504)}$ =820, the maximum of the absolute value max(|265+820| , |265-820|) is 1085, so $a = \frac{5 \cdot (-75)}{1085}$ = -0,34562. The value after the first iteration is $r_2 = r_1 - a = 0.34562$. In the second iteration step the value turns complex, r_3= 0.2193+0.7473i. In the following iterations r converges towards r_∞=0.25+0.5i.

3.5 The Floating Point Library

The floating point library contains all routines that are necessary to controll the floating point arithmetic. All these routines are compiled into a big object file and linked to the modules that use multiple precision floats.

The arithmetic part of the library is the GNU multiple precision library gmp-2.0.2 [(1)]. We just added the part that monitores the subtractive cancellation. Consequently a design was chosen that seperates the arithmetic part strictly from the error tracking. The code of gmp-2.0.2 was not modified, instead the data structure and the functions of the FP arithmetic are just wrappers with some additional functionality.These wrappers have exactly the same interface as the GNU-mp functions and structures.The header file is even equivalent, only the the prefix ‘a’ was added to the function names. Thus two main goals were achieved. First the total encapsulation of the arithmetic part makes it easy to incorporate eventual later versions of the arithmetic library. Additionally maximum protection of the existing GNU code was realized, of course for the cost of some extra overhead.
Second the equality of interfaces would make it easy to implement the new multiple precision floating point arithmetic in applications that already use the gmp-2.0.2.

Each multiple precision float operation completes three steps:

- First the number of correct result mantissa bits without cancellation is evaluated using (2.1.3.1) to (2.1.3.3).
- Then the arithmetic operation is performed by a call of the GNU-mp floating point library.
- In a third step that is only necessary for addition and subtraction the number of lost bits is determined with expression (2.1.2).

[(1)] GMP is one of many software products provided by GNU (Free Software Foundation). Everyone is free to use and redistribute the sourcecode, for details on licensing see http://www.gnu.org

4. A Problem from Kinematic Synthesis

This section demonstrates the capabilities of the software gfloat and points out some characteristic behaviour of Buchberger's Algorithm. For that a problem from kinematic synthesis was chosen. Figure 4a shows the mechanism to be synthesized. It is a planar four bar linkage for rigid bodies connected by revolute pairs with parallel axes of rotation.

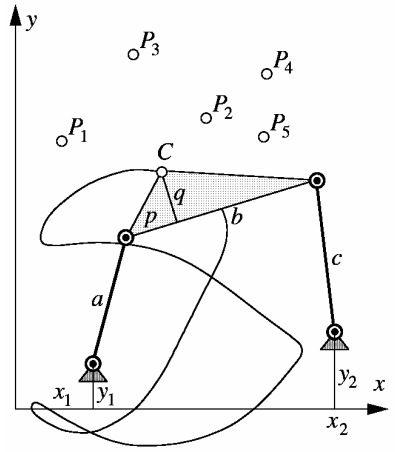


Fig. 4a
The mechanism to be synthesized

The coupler curve is traced by the coupler point C during the motion of the mechanism, it is defined by the nine design parameters a,b,c,p,q, x_1,y_1,x_2 and y_2. For this example the specific synthesis problem is considered, where the positions of the pivots are fixed, the according values for x_1,y_1,x_2 and y_2 are given. So the five remaining design parameters a,b,c,p and q need to be found. We want to synthesize all mechanisms such that their coupler curves pass through a set of five specific points P_1,P_2…P_5. The coordinates x_1,y_1,x_2,y_2 of the pivots and the points P_1,P_2…P_5 are printed in the tables below.

i	P_i (x,y)
1	(0.25 | 0.25)
2	(0.25 | 0.5)
3	(0.25 | 0.75)
4	(0.75 | 0.25)
5	(0.75 | 0.75)

Pivot Coordinates	
x_1	0
y_1	0
x_2	1
y_2	0

4.1 Mathematical Description and Solution

The coupler curve of the planar four bar mechanism is described by an algebraic equation of degree six with the parameters a,b,c,p,q,x_2 (and the indeterminates x and y). It can be derived by algebraization of the kinematic structure of the mechanism [Hiesleitner 1991].

(4.1)

Equation 4.1.1 describes the coupler curve of the mechanism to be synthesized

(4.1.1)

$$b^2(x^2 + y^2)^3 - 2bd(bx + px + qy)(x^2 + y^2)^2 + (A_1x^2 + 4bd^2qxy + A_2y^2)(x^2 + y^2) - 2d(A_3x + A_4y)(x^2 + y^2) + A_5x^2 - 4bd^2(a^2 - b^2 + f^2)qxy + A_6y^2 - 2d(e^2 - a^2)(A_7x + A_8y) + d^2(e^2 - a^2)f^2 = 0$$

with the expressions

(4.1.2) $d = x_2$

(4.1.3) $e = \sqrt{p^2 + q^2}$

(4.1.4) $f = \sqrt{(b - p)^2 + q^2}$

(4.1.5) $A_1 = b^2(-2a^2 + d^2) + (-2b^2 + d^2)e^2 + (a^2 + b^2 - c^2 + 2d^2)(b^2 + e^2 - f^2)$

(4.1.6) $A_2 = b^2(-2a^2 + d^2) + (-2b^2 + d^2)e^2 + (a^2 + b^2 - c^2)(b^2 + e^2 - f^2)$

(4.1.7) $A_3 = -2a^2b^2 + a^2 - b^2 - c^2 + d^2)e^2 + ½(a^2 + 2b^2 - c^2 + d^2 - e^2 - f^2)(b^2 + e^2 - f^2)$

(4.1.8) $A_4 = -bq(a^2 - 2b^2 + c^2 - d^2 + e^2 + f^2)$

(4.1.9) $A_5 = a^2b^2(a^2 - 2d^2) + (-2a^2c^2 + (b^2 - c^2)^2 + 2a^2d^2 - 2c^2d^2 + (a^2 + d^2)^2)e^2 + (b^2 - 2d^2)e^4 + (b^2 + e^2 - f^2)(-((a^2 + b^2 - c^2 + d^2)(a^2 + e^2)) + (a^2 + d^2)(b^2 + e^2 - f^2))$

(4.1.10) $A_6 = a^2b^2(a^2 - 2d^2) + (-2b^2c^2 + (a^2 - c^2)^2 + 2b^2d^2 - 2c^2d^2 + (b^2 + d^2)^2)e^2 + (b^2 - 2d^2)e^4 + (b^2 + e^2 - f^2)(-((a^2 + b^2 - c^2 - d^2)(a^2 + e^2)) + (a^2 - d^2)(b^2 + e^2 - f^2))$

(4.1.11) $A_7 = -a^2b^2 + (-a^2 + c^2 - d^2)e^2 + ½(b^2 + e^2 - f^2)(2a^2 - c^2 + d^2 + f^2)$

(4.1.12) $A_8 = b(-c^2 + d^2 + f^2)q$

Taking (4.1.1) and substituting the expressions (4.1.2) to (4.1.12) one obtains a polynomial function f(x,y) with the parameters a,b,c,p,q and x_2. A value for x_2 is given, so five parameters remain in the function.
Taking the points P_1,P_2…P_5 and substituting their coordinates for the indeterminates x and y delivers five equations with the design parameters a,b,c,p,q as unknowns. The solution values for a,b,c,p,q of this system of equations define the mechanisms with their coupler curves passing through the given points.

Buchberger's algorithm shows better terminating behaviour on input polynomials of lower degree, so the expressions a^2, b^2 and c^2 are substituted with a^2=p1, b^2=p2 and c^2=p3, finally renaming p=p4 and q=p5 the system of input eqations for gfloat is ready. Each of the polynomials is of total degree eight and consists of 78 monomials.

Equation 1:
5 - 80*p1 + 320*p1^2 - 20*p4 + 240*p1*p4 - 640*p1^2*p4 - 16*p2*p4 + 128*p1*p2*p4 + 16*p3*p4 - 128*p1*p3*p4 + 40*p4^2 - 160*p1*p4^2 + 320*p1^2*p4^2 + 80*p2*p4^2 - 1024*p1*p2*p4^2 + 64*p2^2*p4^2 - 96*p3*p4^2 + 128*p1*p3*p4^2 - 128*p2*p3*p4^2 + 64*p3^2*p4^2 - 192*p2*p4^3 + 1664*p1*p2*p4^3 - 384*p2^2*p4^3 + 384*p2*p3*p4^3 + 128*p2*p4^4 - 768*p1*p2*p4^4 + 1088*p2^2*p4^4 - 256*p2*p3*p4^4 - 1280*p2^2*p4^5 + 512*p2^2*p4^6 - 20*p5 + 160*p1*p5 - 32*p2*p5 + 256*p1*p2*p5 + 32*p3*p5 - 256*p1*p3*p5 + 192*p2*p4*p5 - 512*p1*p2*p4*p5 - 320*p2*p4^2*p5 + 256*p1*p2*p4^2*p5 - 256*p2^2*p4^2*p5 + 256*p2*p3*p4^2*p5 + 512*p2^2*p4^3*p5 - 256*p2^2*p4^4*p5 + 40*p5^2 - 160*p1*p5^2 + 320*p1^2*p5^2 + 144*p2*p5^2 - 768*p1*p2*p5^2 + 64*p2^2*p5^2 - 96*p3*p5^2 + 128*p1*p3*p5^2 - 128*p2*p3*p5^2 + 64*p3^2*p5^2 - 192*p2*p4*p5^2 + 1664*p1*p2*p4*p5^2 - 384*p2^2*p4*p5^2 + 384*p2*p3*p4*p5^2 + 256*p2*p4^2*p5^2 - 1536*p1*p2*p4^2*p5^2 + 1664*p2^2*p4^2*p5^2 - 512*p2*p3*p4^2*p5^2 - 2560*p2^2*p4^3*p5^2 + 1536*p2^2*p4^4*p5^2 - 320*p2*p5^3 + 256*p1*p2*p5^3 - 256*p2^2*p5^3 + 256*p2*p3*p5^3 + 512*p2^2*p4*p5^3 - 512*p2^2*p4^2*p5^3 + 128*p2*p5^4 - 768*p1*p2*p5^4 + 576*p2^2*p5^4 - 256*p2*p3*p5^4 - 1280*p2^2*p4*p5^4 + 1536*p2^2*p4^2*p5^4 - 256*p2^2*p5^5 + 512*p2^2*p5^6 = 0

Equation 2:
325 - 2080*p1 + 3328*p1^2 - 520*p4 + 3744*p1*p4 - 6656*p1^2*p4 + 160*p2*p4 - 512*p1*p2*p4 - 160*p3*p4 + 512*p1*p3*p4 + 1040*p4^2 - 1664*p1*p4^2 + 3328*p1^2*p4^2 - 416*p2*p4^2 - 5120*p1*p2*p4^2 + 1280*p2^2*p4^2 - 1920*p3*p4^2 - 512*p1*p3*p4^2 - 2560*p2*p3*p4^2 + 1280*p3^2*p4^2 + 768*p2*p4^3 + 11776*p1*p2*p4^3 - 4608*p2^2*p4^3 + 4608*p2*p3*p4^3 - 512*p2*p4^4 - 6144*p1*p2*p4^4 + 9472*p2^2*p4^4 - 2048*p2*p3*p4^4 - 10240*p2^2*p4^5 + 4096*p2^2*p4^6 - 1040*p5 + 3328*p1*p5 - 1280*p2*p5 + 4096*p1*p2*p5 + 1280*p3*p5 - 4096*p1*p3*p5 + 1536*p2*p4*p5 - 8192*p1*p2*p4*p5 - 3584*p2*p4^2*p5 + 4096*p1*p2*p4^2*p5 - 4096*p2^2*p4^2*p5 + 4096*p2*p3*p4^2*p5 + 8192*p2^2*p4^3*p5 - 4096*p2^2*p4^4*p5 + 1040*p5^2 - 1664*p1*p5^2 + 3328*p1^2*p5^2 + 3936*p2*p5^2 - 6144*p1*p2*p5^2 + 1280*p2^2*p5^2 - 1920*p3*p5^2 - 512*p1*p3*p5^2 - 2560*p2*p3*p5^2 + 1280*p3^2*p5^2 + 768*p2*p4*p5^2 + 11776*p1*p2*p4*p5^2 - 4608*p2^2*p4*p5^2 + 4608*p2*p3*p4*p5^2 - 1024*p2*p4^2*p5^2 - 12288*p1*p2*p4^2*p5^2 + 14848*p2^2*p4^2*p5^2 - 4096*p2*p3*p4^2*p5^2 - 20480*p2^2*p4^3*p5^2 + 12288*p2^2*p4^4*p5^2 - 3584*p2*p5^3 + 4096*p1*p2*p5^3 - 4096*p2^2*p5^3 + 4096*p2*p3*p5^3 + 8192*p2^2*p4*p5^3 - 8192*p2^2*p4^2*p5^3 - 512*p2*p5^4 - 6144*p1*p2*p5^4 + 5376*p2^2*p5^4 - 2048*p2*p3*p5^4 - 10240*p2^2*p4*p5^4 + 12288*p2^2*p4^2*p5^4 - 4096*p2^2*p5^5 + 4096*p2^2*p5^6 = 0

Equation 3:
225 - 720*p1 + 576*p1^2 - 180*p4 + 1008*p1*p4 - 1152*p1^2*p4 + 240*p2*p4 - 384*p1*p2*p4 - 240*p3*p4 + 384*p1*p3*p4 + 360*p4^2 - 288*p1*p4^2 + 576*p1^2*p4^2 - 432*p2*p4^2 + 320*p2^2*p4^2 - 480*p3*p4^2 - 384*p1*p3*p4^2 - 640*p2*p3*p4^2 + 320*p3^2*p4^2 + 576*p2*p4^3 + 1152*p1*p2*p4^3 - 896*p2^2*p4^3 + 896*p2*p3*p4^3 - 384*p2*p4^4 - 768*p1*p2*p4^4 + 1344*p2^2*p4^4 - 256*p2*p3*p4^4 - 1280*p2^2*p4^5 + 512*p2^2*p4^6 - 540*p5 + 864*p1*p5 - 480*p2*p5 + 768*p1*p2*p5 + 480*p3*p5 - 768*p1*p3*p5 - 192*p2*p4*p5 - 1536*p1*p2*p4*p5 - 192*p2*p4^2*p5 + 768*p1*p2*p4^2*p5 - 768*p2^2*p4^2*p5 + 768*p2*p3*p4^2*p5 + 1536*p2^2*p4^3*p5 - 768*p2^2*p4^4*p5 + 360*p5^2 - 288*p1*p5^2 + 576*p1^2*p5^2 + 912*p2*p5^2 - 768*p1*p2*p5^2 + 320*p2^2*p5^2 - 480*p3*p5^2 - 384*p1*p3*p5^2 - 640*p2*p3*p5^2 + 320*p3^2*p5^2 + 576*p2*p4*p5^2 + 1152*p1*p2*p4*p5^2 - 896*p2^2*p4*p5^2 + 896*p2*p3*p4*p5^2 - 768*p2*p4^2*p5^2 - 1536*p1*p2*p4^2*p5^2 + 2176*p2^2*p4^2*p5^2 - 512*p2*p3*p4^2*p5^2 - 2560*p2^2*p4^3*p5^2 + 1536*p2^2*p4^4*p5^2 - 192*p2*p5^3 + 768*p1*p2*p5^3 - 768*p2^2*p5^3 + 768*p2*p3*p5^3 + 1536*p2^2*p4*p5^3 - 1536*p2^2*p4^2*p5^3 - 384*p2*p5^4 - 768*p1*p2*p5^4 + 832*p2^2*p5^4 - 256*p2*p3*p5^4 - 1280*p2^2*p4*p5^4 + 1536*p2^2*p4^2*p5^4 - 768*p2^2*p5^5 + 512*p2^2*p5^6 = 0

Equation 4:
25 - 80*p1 + 64*p1^2 - 60*p4 + 176*p1*p4 - 128*p1^2*p4 - 80*p2*p4 + 128*p1*p2*p4 + 80*p3*p4 - 128*p1*p3*p4 + 40*p4^2 - 96*p1*p4^2 + 64*p1^2*p4^2 + 272*p2*p4^2 - 512*p1*p2*p4^2 + 320*p2^2*p4^2 - 160*p3*p4^2 + 128*p1*p3*p4^2 - 640*p2*p3*p4^2 + 320*p3^2*p4^2 - 320*p2*p4^3 + 640*p1*p2*p4^3 - 1408*p2^2*p4^3 + 1408*p2*p3*p4^3 + 128*p2*p4^4 - 256*p1*p2*p4^4 + 2368*p2^2*p4^4 - 768*p2*p3*p4^4 - 1792*p2^2*p4^5 + 512*p2^2*p4^6 - 20*p5 + 32*p1*p5 - 160*p2*p5 + 256*p1*p2*p5 + 160*p3*p5 - 256*p1*p3*p5 + 448*p2*p4*p5 - 512*p1*p2*p4*p5 - 320*p2*p4^2*p5 + 256*p1*p2*p4^2*p5 - 256*p2^2*p4^2*p5 + 256*p2*p3*p4^2*p5 + 512*p2^2*p4^3*p5 - 256*p2^2*p4^4*p5 + 40*p5^2 - 96*p1*p5^2 + 64*p1^2*p5^2 + 208*p2*p5^2 - 256*p1*p2*p5^2 + 320*p2^2*p5^2 - 160*p3*p5^2 + 128*p1*p3*p5^2 - 640*p2*p3*p5^2 + 320*p3^2*p5^2 - 320*p2*p4*p5^2 + 640*p1*p2*p4*p5^2 - 1408*p2^2*p4*p5^2 + 1408*p2*p3*p4*p5^2 + 256*p2*p4^2*p5^2 - 512*p1*p2*p4^2*p5^2 + 3200*p2^2*p4^2*p5^2 - 1536*p2*p3*p4^2*p5^2 - 3584*p2^2*p4^3*p5^2 + 1536*p2^2*p4^4*p5^2 - 320*p2*p5^3 + 256*p1*p2*p5^3 - 256*p2^2*p5^3 + 256*p2*p3*p5^3 + 512*p2^2*p4*p5^3 - 512*p2^2*p4^2*p5^3 + 128*p2*p5^4 - 256*p1*p2*p5^4 + 832*p2^2*p5^4 - 768*p2*p3*p5^4 - 1792*p2^2*p4*p5^4 + 1536*p2^2*p4^2*p5^4 - 256*p2^2*p5^5 + 512*p2^2*p5^6 = 0

Equation 5:
405 - 720*p1 + 320*p1^2 - 540*p4 + 1200*p1*p4 - 640*p1^2*p4 + 432*p2*p4 - 384*p1*p2*p4 - 432*p3*p4 + 384*p1*p3*p4 + 360*p4^2 - 480*p1*p4^2 + 320*p1^2*p4^2 - 1008*p2*p4^2 + 512*p1*p2*p4^2 + 576*p2^2*p4^2 - 288*p3*p4^2 - 384*p1*p3*p4^2 - 1152*p2*p3*p4^2 + 576*p3^2*p4^2 + 960*p2*p4^3 + 128*p1*p2*p4^3 - 1920*p2^2*p4^3 + 1920*p2*p3*p4^3 - 384*p2*p4^4 - 256*p1*p2*p4^4 + 2624*p2^2*p4^4 - 768*p2*p3*p4^4 - 1792*p2^2*p4^5 + 512*p2^2*p4^6 - 540*p5 + 480*p1*p5 - 864*p2*p5 + 768*p1*p2*p5 + 864*p3*p5 - 768*p1*p3*p5 + 576*p2*p4*p5 - 1536*p1*p2*p4*p5 - 192*p2*p4^2*p5 + 768*p1*p2*p4^2*p5 - 768*p2^2*p4^2*p5 + 768*p2*p3*p4^2*p5 + 1536*p2^2*p4^3*p5 - 768*p2^2*p4^4*p5 + 360*p5^2 - 480*p1*p5^2 + 320*p1^2*p5^2 + 720*p2*p5^2 - 256*p1*p2*p5^2 + 576*p2^2*p5^2 - 288*p3*p5^2 - 384*p1*p3*p5^2 - 1152*p2*p3*p5^2 + 576*p3^2*p5^2 + 960*p2*p4*p5^2 + 128*p1*p2*p4*p5^2 - 1920*p2^2*p4*p5^2 + 1920*p2*p3*p4*p5^2 - 768*p2*p4^2*p5^2 - 512*p1*p2*p4^2*p5^2 + 3712*p2^2*p4^2*p5^2 - 1536*p2*p3*p4^2*p5^2 - 3584*p2^2*p4^3*p5^2 + 1536*p2^2*p4^4*p5^2 - 192*p2*p5^3 + 768*p1*p2*p5^3 - 768*p2^2*p5^3 + 768*p2*p3*p5^3 + 1536*p2^2*p4*p5^3 - 1536*p2^2*p4^2*p5^3 - 384*p2*p5^4 - 256*p1*p2*p5^4 + 1088*p2^2*p5^4 - 768*p2*p3*p5^4 - 1792*p2^2*p4*p5^4 + 1536*p2^2*p4^2*p5^4 - 768*p2^2*p5^5 + 512*p2^2*p5^6 = 0

For the floating point arithmetic a mantissa length of 5700 bit was chosen. The computation time for the lexicographic Groebner Basis took 109 seconds on a 600 MHz, 64 bit machine (AlphaStation).
The system has 36 solutions, 18 of them are real, so there exist 18 mechanisms with the given pivot coordinates and their coupler curve going through the five given points.
The solutions are printed in the following table where the real solutions are in the gray lines.

P5	P4	P3	P2	P1
-0.7119895e0 – 0.1138211e-247*I	0.5e0 +0.1302076e-228I	0.9628363e0 +0.1035683e-228I	0.965678e0 +0.1552071e-227I	0.9628363e0 +0.1302527e-227I
-0.4543939e0 -0.2285656e-244I	0.5e0 +0.5157446e-229I	0.1442238e1 +0.4108944e-229I	0.2014508e1 +0.6157161e-228I	0.1442238e1 +0.5168069e-228I
-0.2424575e0 - 0.2089495e-241I	0.5e0 +0.199743e-229I	0.4569168e1 +0.159465e-229I	0.1098491e2 +0.2389315e-228I	0.4569168e1 +0.200592e-228I
-0.2932242e-1 +0.4203301e-241I	0.9869704e0 +0.5635832e-230I	0.1008133e1 +0.4514939e-230I	0.6119911e3 +0.676385e-229	0.6117821e3 +0.568052e-229I
0.3980263e0 - 0.2534512e0I	0.5e0- 0.1747453e-230I	-0.4105941e-1 +0.3817516e-1I	0.7144235e0 +0.1276755e1I	-0.4105941e-1 +0.3817516e-1I
0.9716476e0 - 0.5544045e0I	0.5e0 - 0.2896811e-230I	0.321692e0 +0.1581857e0I	-0.1211619e-1 +0.7403326e-1I	0.321692e0 +0.1581857e0I
-0.3397962e0 +0.6780031e-242I	0.9199856e0 +0.3172832e-229I	0.5103059e0 +0.2530339e-229I	0.5899654e1 +0.3791476e-228I	0.7227844e1 +0.3182739e-228I
-0.3030579e0 - 0.1117672e-240I	0.5e0 +0.2680747e-229I	0.2518219e1 +0.2138702e-229I	0.4973225e1 +0.3204588e-228I	0.2518219e1 +0.2690183e-228I
-0.295357e0 +0.1215105e-240I	0.600751e0 +0.2585381e-229I	0.2113919e1 +0.2062789e-229I	0.5656269e1 +0.309083e-228I	0.3534801e1 +0.2594707e-228I
-0.6798624e0 +0.3384061e-247I	0.3647073e0 +0.1171762e-228I	0.1309681e1 +0.9321856e-229I	0.9563646e0 +0.1396958e-227I	0.6745428e0 +0.1172374e-227I

P5	P4	P3	P2	P1
0.4375e0 -0.25e0I	0.5e0 - 0.625e-1I	0.3889523e-231 - 0.9769482e-231I	0.6153846e0 +0.9230769e0I	0.487692e-230 - 0.1229657e-229I
0.5913277e0 - 0.4514945e0I	0.6634732e0 - 0.179996e0I	0.2122194e0 - 0.9751928e-1I	-0.1751174e-1 +0.3078004e0I	0.6521953e-1 - 0.7003233e-1I
0.3980263e0 +0.2534512e0I	0.5e0 +0.1596476e-231I	-0.4105941e-1 - 0.3817516e-1I	0.7144235e0 - 0.1276755e1I	-0.4105941e-1 - 0.3817516e-1I
0.7947702e-1 +0.4068757e0I	0.5e0 +0.6278324e-231I	0.8034421e0 - 0.3629949e-1I	0.5642767e-1 +0.7536877e-1I	0.8034421e0 - 0.3629949e-1I
0.3967213e0 +0.7830428e-232I	0.9105479e0 - 0.1701428e-228I	0.5047077e0 - 0.1394856e-228I	0.6521956e1 - 0.2104124e-227I	0.4774117e1 - 0.1769499e-227I
0.34375e0 - 0.9375e-1I	0.15625e0 +0.15625e0I	-0.1475644e-230 -0.591246e-231I	0.1169231e1 +0.1046154e1I	-0.1859005e-229- 0.7426958e-230I
0.7947702e-1 - 0.4068757e0I	0.5e0 - 0.1506067e-229I	0.8034421e0 +0.3629949e-1I	0.5642767e-1 - 0.7536877e-1I	0.8034421e0 +0.3629949e-1I
0.4773832e0 - 0.2403675e0I	0.5259869e0 - 0.143291e0I	0.6626049e-1 - 0.5054367e-1I	0.4925619e0 +0.6548805e0I	0.1563529e-1 - 0.1692409e-1I
0.3969307e0 - 0.8096596e-232I	0.4708243e0 +0.1641263e-228I	0.1442457e0 +0.1345299e-228I	0.1083213e1 +0.2030305e-227I	0.2004798e0 +0.1707291e-227I
0.6578229e0 - 0.4562129e0I	0.5e0 +0.2527322e-230I	0.8673513e-1 - 0.7726016e-1I	0.2921105e-1 +0.3742502e0I	0.8673513e-1 - 0.7726016e-1I
0.34375e0 +0.9375e-1I	0.15625e0 - 0.15625e0I	-0.7636332e-231 -0.2358899e-231I	0.1169231e1 - 0.1046154e1I	-0.9613386e-230 -0.2980129e-230I
0.4375e0 +0.25e0I	0.5e0 +0.625e-1I	-0.6849788e-232 +0.6286125e-232I	0.6153846e0 - 0.9230769e0I	-0.8627516e-231 +0.7896704e-231I
0.4043948e0 +0.2718634e-233I	0.5e0 +0.4220932e-230I	0.2132792e0 +0.349323e-230I	0.872905e0 +0.519313e-229I	0.2132792e0 +0.4379479e-229I
0.5291342e0 +0.1557882e-233I	0.6888236e0 +0.2926627e-232I	0.2813733e0 +0.2417981e-232I	0.8359968e0 +0.2823988e-231I	0.2002445e0 +0.243652e-231I
0.4902368e0 - 0.1348154e-234I	0.5e0 +0.10053e-231I	0.1251908e0 +0.8551922e-232I	0.1020083e1 +0.1283154e-230I	0.1251908e0 +0.1083651e-230I
0.4773832e0 +0.2403675e0I	0.5259869e0 +0.143291e0I	0.6626049e-1 +0.5054367e-1I	0.4925619e0 - 0.6548805e0I	0.1563529e-1 +0.1692409e-1I
0.7268108e0 - 0.132925e-239I	0.5e0 - 0.8718622e-233I	0.4319559e2 - 0.6374137e-233I	0.6144426e2 - 0.9600833e-232I	0.4319559e2 - 0.7986247e-232I
0.5299309e0 - 0.1475939e-233I	0.692948e0 +0.126484e-232I	0.2841671e0 +0.1275431e-232I	0.8385634e0 +0.2609398e-231I	0.2044369e0 +0.2177919e-231I
0.614548e0 - 0.2917544e-2I	0.206577e0 +0.4944974e0I	-0.1088966e-1 +0.4783234e-3I	0.8065282e0 +0.57444e0I	-0.1253375e0 - 0.1266832e0I
0.614548e0 +0.2917544e-2I	0.206577e0 - 0.4944974e0I	-0.1088966e-1 - 0.4783234e-3I	0.8065282e0 - 0.57444e0I	-0.1253375e0 +0.1266832e0I
0.6479371e0 +0.242742e-236I	0.263625e0 - 0.162189e-232I	0.1696275e1 - 0.9683111e-233I	0.2849934e1 - 0.1519393e-231I	0.6871755e0 - 0.124411e-231I
0.6265086e0 - 0.486555e-235I	0.5e0 - 0.2467973e-232I	0.1626831e0 - 0.1238645e-232I	0.8718604e0 - 0.1736948e-231I	0.1626831e0 - 0.1366024e-231I
0.5913277e0 +0.4514945e0I	0.5913277e0 +0.4514945e0I	0.2122194e0 +0.9751928e-1I	-0.1751174e-1 - 0.3078004e0I	-0.1751174e-1 - 0.3078004e0I
0.77104e0 +0.4222508e-241I	-0.5724022e0 - 0.8941979e-233I	0.2151177e0 - 0.6679445e-233I	0.1703965e0 - 0.1004748e-231I	0.4413301e0 - 0.8376711e-232I
0.6578229e0 +0.4562129e0I	0.5e0 - 0.8924435e-233I	0.8673513e-1 +0.7726016e-1I	0.2921105e-1 - 0.3742502e0I	0.8673513e-1 +0.7726016e-1I
0.9716476e0 +0.5544045e0I	0.5e0 - 0.6035043e-233I	0.321692e0 - 0.1581857e0I	-0.1211619e-1 - 0.7403326e-1I	0.321692e0 - 0.1581857e0I

Taking the real sets of solutions for P_1,P_2…P_5 the design parameters a,b,c,p,q for the four bar mechanisms are easily computed. Figure 4b shows the corresponding mechanisms and their coupler curves. The mechanisms are always depicted in such a position that the coupler point C coincides with point (0.25|0.25).
For mechanisms with two closure modes, the curve of one of the modes is printed with a dashed line.

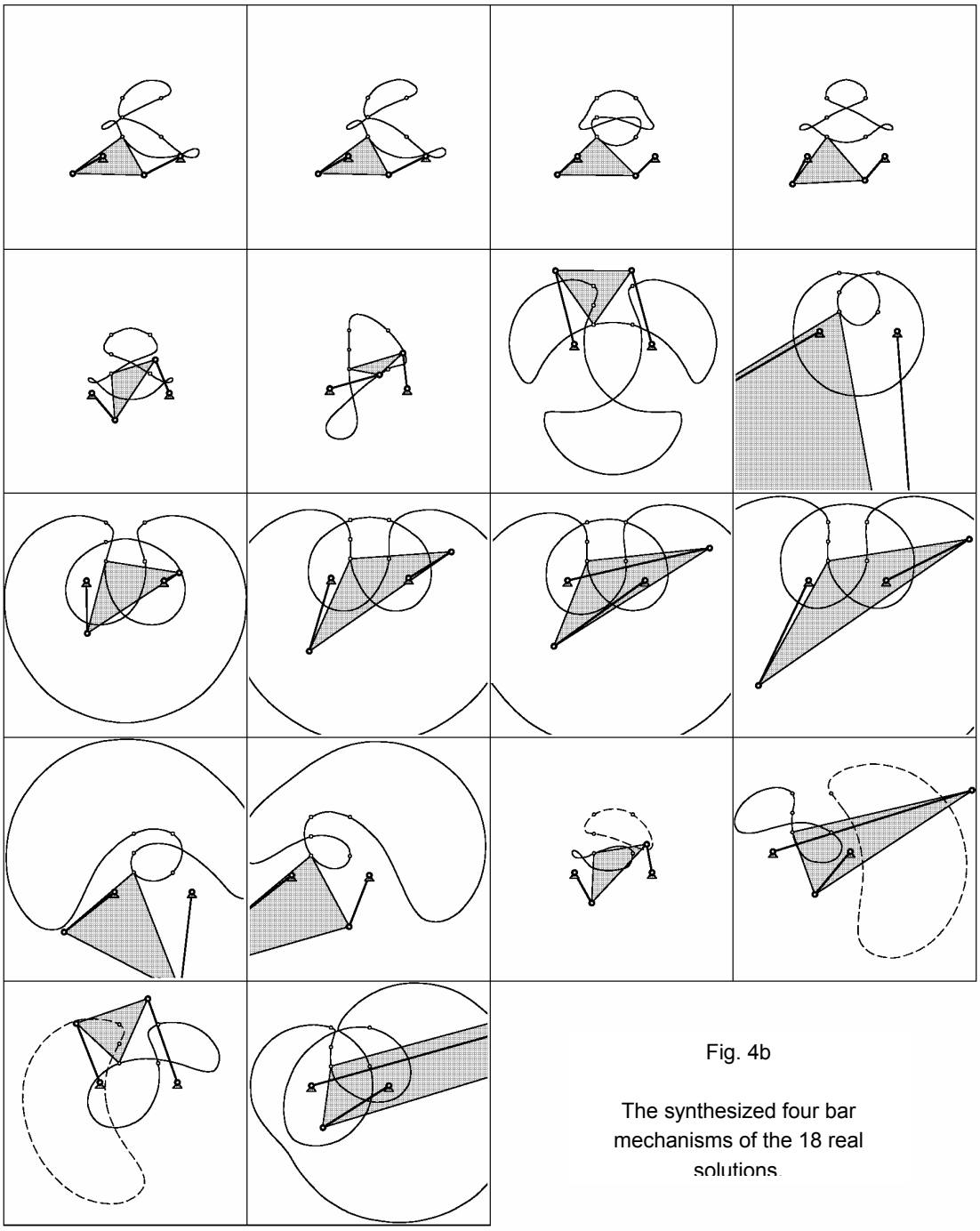


Fig. 4b

The synthesized four bar mechanisms of the 18 real solutions

4.2 Computational Aspects of Buchberger's Algorithm

In this section some of the specific properties of Buchberger's algorithm will be demonstrated. The data is taken only from the above example (section 4.1), still the behaviour is very typical for this algorithm. All the following diagrams show the calculation of the total degree reverse lexicographic Groebner Basis, the characteristics for directly making a lexicographic Groebner Basis would be worse by several orders of magnitude.

The most important characteristic of Buchberger's Algorithm is, that the number and size of the polynomials to be stored during the computation of the Groebner Basis becomes extremely large. Figure 4c shows the number of monomials in memory during the whole algorithm. The x-axis shows the progress of the Groebner Basis computation by the number of S-Polynomials that were already reduced modulo the Basis. Buchberger's algorithm terminates when all S-Polys are processed which is indicated here with 100%. The y-axis shows the number of monomials in memory during the algorithm. This is taken in relation to the starting size of the basis. At the beginning just the input equations have to be stored, so the starting value is 100% for the 390 monomials from the five input equations (p.72-73).

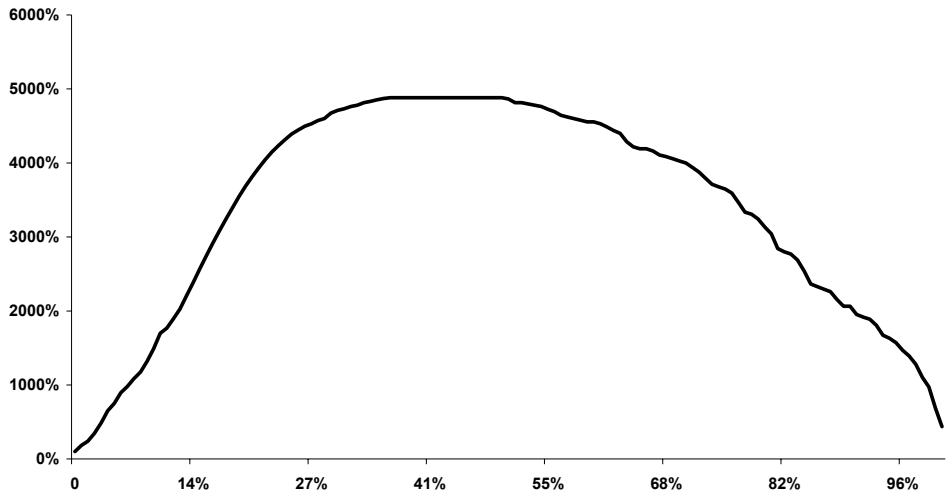


Fig. 4c
Memory demand in relation to input

Memory demand rises enormously, up to 50 times the initial amount, before it starts to fall. The tdegrevl Groebner Basis finally is just somewhat larger than the input.
For floating point arithmetic the size of the coefficients is fixed, so the number of monomials is proportional to the used memory. For our example a maximum of 14.8 megabytes of polynomial data was stored.
Using integer arithmetic memory demand would be a lot worse because the size of the coefficients also keeps growing.

In figure 4d the gray line again shows the number of monomials over the whole algorithm. This number of all monomials is a sum of the two black curves. The solid one shows the number of monomials in the actual base, the dashed line shows the number of monomials in redundant polys. As in the previous picture the number of monomials is shown in relation to the input.
Redundant polynomials are not part of the ideal basis any more, but they still must be kept in memory. Recall, that the implemented version of Buchberger's algorithm (1.3.2.3) uses a subroutine called REDUNDANT. In REDUNDANT criterion (1.2.3.4) is used to eliminate superfluous polynomials from the ideal basis. However, these polynomials cannot be deleted from memory, when there still is a pair within the set of pairs, that uses this polynomial. So this polynomial will be needed later to build an S-poly, but it is redundant in the base.

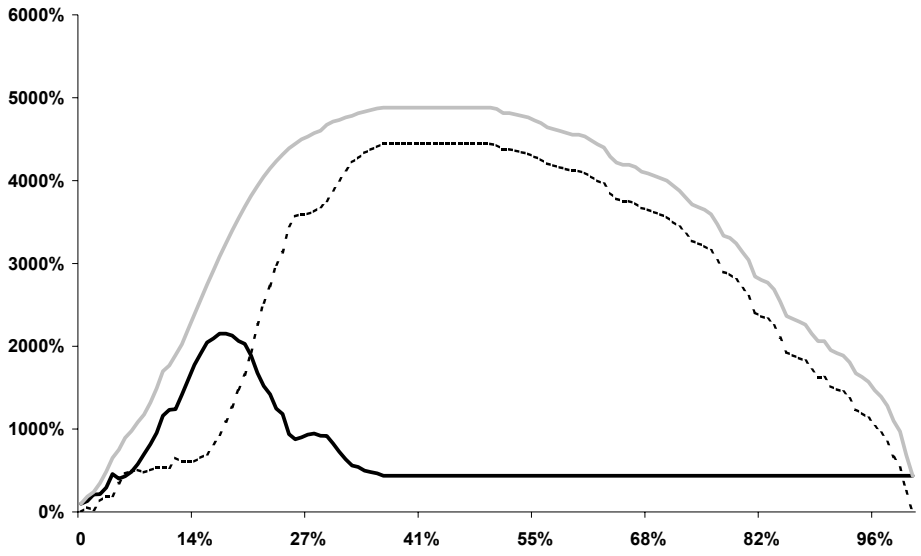


Fig. 4d
Memory demand of active (black) and redundant (dashed) polynomials

In the starting point and at the end of the computation there can be no redundant polynomials. In the beginning, the active base as well as the redundant polynomials grow. Surprisingly,very soon the major part of the memory is NOT needed to store the active base, but rather for old, redundant polynomials that are only needed to build S-polynomials.
For very large problems, where hardware memory is not sufficient and extensive swapping to the harddrive is necessary this particular property leaves an opportuninty for optimization. The redundant polynomials are not part of the base, they are not needed for NORMALFORM reductions. So they are used very rarely, namely just for building of specific S-polys. Substantial speedup of the computation could be achieved by keeping the active base polynomials in memory and writing the redundant polynomials to the hardrive.
This idea is not implemented in gfloat but it may be a suggestion for further optimization.

The terminating behaviour of Buchberger's algorithm is very specific, it is depicted in figure 4e. Especially during computations of very large problems it can be interesting for the user to check the status of the algorithm in order to see if there is a chance to obtain a result. Gfloat keeps track of these indicators, for details see sections (3.1) and (A.2).
Three characteristic indicators give a rather good idea about the progress that has been made. The black line is the number of polynomials in the base. The dark gray line is the number of pairs in the pairset i.e. the number of S-polynomials that need to be reduced. Finally the light gray line is the maximum number of monoms that may occur intermediately during the NORMALFORM reduction. (Usually during a normalform reduction the polynomials become longer for some reduction steps, later they shrink).
Again the x-axis shows the progress of the algorithm in percent. The y-axis shows the above values in percent of their maximum. So each of the lines reaches 100 percent at the peak value of the specific indicator.

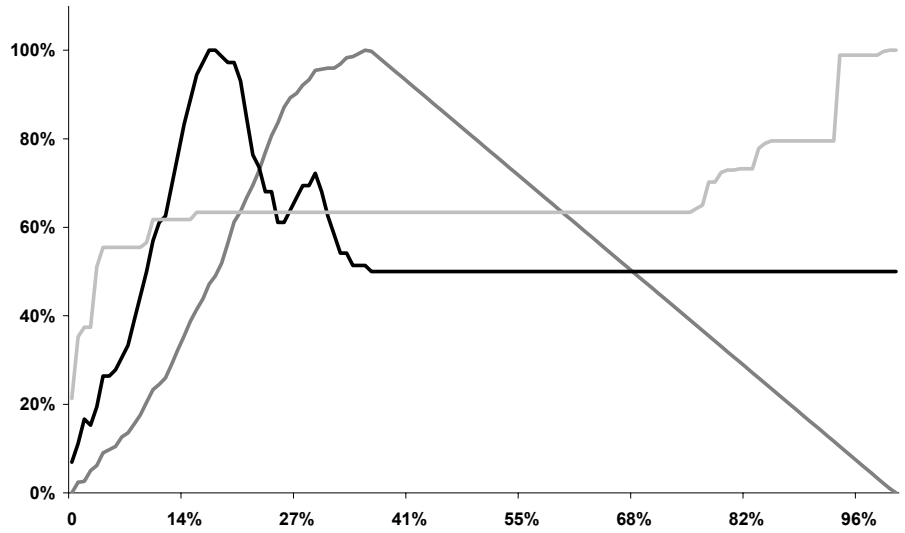


Fig. 4e
Terminating behaviour of Buchberger's algorithm.

Number of base polynomials – black
Number of pairs – dark gray
Maximum intermediate polynomial size – light gray

The most secure indicator is the top of the number of pairs (dark gray line). At this point the basis is usually already a Groebner Basis and will remain unchanged, which can be seen from the horizontal black line. For termination the algorithm has to reduce the remaining S-polynomials to zero. The memory demand also peaks at this point (see fig. 4c), so it is certain from that moment on that the algorithm will terminate.
It is a fairly good indicator for termination, when the number of base polynomials starts to drop substantially. Unfortunately there are usually several local peaks, which makes it hard to decide if the absolute maximum was already reached. But ist usable to some extent, especially because it shows earlier than the peak of the pairs.

The third indicator, maximum intermediate size of a polynomial (light gray line) is very vague. Here the signal is stabilization, not a peak. At first it grows very fast, then the growth slows and eventually stops totally. It shows very early, a long time before the actual peak of the pairs and before the maximum of base polynomials.
It can also be used as somewhat of a negative indicator in the sense that termination is out of sight, as long as the maximum intermediate size of the polynomials keeps growing at the same pace.

Figure 4e also very well shows the benefits of the trace. At about 40% of the computations the final Groebner Basis is obtained (peak of pairs, unchanged number of base polynomials), the following zero reductions of the remaining S-polynomials are only performed in the modular run. For the much more time consuming trace run they are omitted.

Finally figure 4f shows some statistics about the runtime (in seconds) of the different modules.

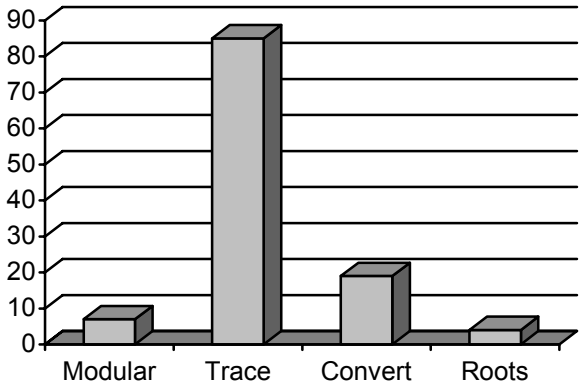


Fig. 4f
Runtime of the modules

Clearely the trace is the bottleneck of the computation. The conversion is a lot faster but still takes twice the time of modular and roots together.

Please note, that allthough the trace performs only 40% of the operations of the modular it takes 15 times longer. That difference can only be explained by the more time consuming arithmetic for the floating point coefficients. That very much strengthens the assumption made in section 2 (p.43) that from a computational point of view, finding Groebner Bases is largely a task of manipulating the monomial coefficients.

Appendix A: Installation and Usage of *gfloat*

Installation and usage of *gfloat* is described in the files INSTALL and MANUAL which are included in the software. These files are expanded together with the sourcecode of *gfloat* and will be located in the uppermost directory gfloat-1.1 .

A.1 Installation

```
Please note that the GNU C Compiler gcc is required to install
gfloat.
If you have not installed the gcc on your machine you have to do so.
See the GNU Website www.gnu.org for download if necessary.

Three make targets are available

1.) make
2.) make clean
3.) make remove

1.) make
The built of gfloat requires the GNU C-compiler gcc and the
make utility.
In order to perform the installation enter the directory
gfloat-1.1 and type

make

The built will be finished when the message 'installation completed
sucessfully !!' appears.
In order to perform a little test enter the directory
gfloat-1.1/gfloat and type

gfloat sigi

The solutions from the input system stored in sigi.inp are being
computed.
No error message should appear, the last output on the screen should
consist
of the six solution points. The detailed output is stored in the
file
gfloat-1.1/data/sigi.result


2.) make clean
If you want to remove all files created by the built,
enter the directory gfloat-1.1 and type

make clean

Not only the objects and executables are being removed,
the configuration files for the floating point arithmetic
are also deleted. Now it is possible to rebuild the project,
the fp-arithmetic will then be reconfigured.
```

```
3.) make remove
In order to remove all files and directories that were created
for installation of gfloat enter the directory gfloat-1.0
and type

make remove

Everything, all sourcefiles and all executables as well as the
directory structure are being removed. Only use this make
target if do not want to use gfloat again and remove it
completely.
```

A.2 Users Guide

```
Table of contents:
==================


0. Introduction
1. Directory Structure
2. Operating gfloat
3. Output created by gfloat
4. Error Handling



0.
Introduction
============

The software computes the solution points of systems of algebraic
equations (polynomials). However, when the set of solution points
is infinite (the solution body is NOT zero dimensional) no solutions
can be found.
Thus gfloat is suited for solution of two kinds of mathematical
problems:
(1)  Determination if the set of solutions from a given system of
     input polynomials is finite (zero dimensional) or infinite
     (non-zero dimensional)
(2)  Find the solution points of a given system of input polynomials
     (if thenumber of solution points is finite)

The mathematical approach used in gfloat is to compute a
lexicographic
Groebner Basis and subsequently the solutions. The lexicographic
Groebner Basis is obtained from a total degree reverse lexicographic
Groebner
Basis via conversion of monomial ordering. Floating point/modular
arithmetic
is used to represent the coefficients of the polynomials.
```

Gfloat is free software, you are welcome to redistribute and/or use parts
of the code for your own purposes. The software is written in C, it was developped by

Institute of General Mechanics
University of Technology Graz
Kopernikusgasse 24
A-8010 Graz, Austria (Europe)

If you have any questions, remarks or --> bugs <-- to report please feel free to contact: dietm@mech.tu-graz.ac.at

1. The directory structure of gfloat

gfloat-1.1 contains three subdirectories.

The subdirectory gfloat-1.1/gfloat contains input files and the executable 'gfloat'.

The subdirectory gfloat-1.1/data contains output files created by the program. Enter the directory gfloat-1.1/data to see the output created by gfloat. See 3. for details about the naming conventions for the output files.

The subdirectory gfloat-1.1/groebner_fl contains the source code, objects and executables.

2. Operating gfloat

Gfloat is operated from the directory gfloat-1.1/gfloat. Two files are required for input in gfloat. These two files are the input file and the options file, these files have to be in the gfloat-1.1/gfloat directory.

2.0

In order to compute the solutions of a system of equations stored in the input file [name].inp type as command line

```
gfloat [name]
```

The input file [name].inp will be opened and the lexicographic Groebner Base and the solution points of the input equations are computed.
Example: the comand line

```
gfloat sigi
```

will compute the solutions of the input file sigi.inp that is delivered with gfloat.

2.1
The input file '*.inp'
It is created by the user and is used to feed the input equations to the program. The input file follows the following format:
The first line contains the number of indeterminates.
The second line contains the indeterminates seperated by commas, they also must be enclosed in braces or brackets.
Please note that NO WHITE SPACE is allowed between the variable names and the commas. Next come the input equations each one preceded by a equals sign (=) and terminated by a colon (;).
The input files sigi.inp and dietm.inp that are delivered in gfloat-1.1/gfloat, can be used as examples.

2.2
The 'options' file
It is edited by the user. Several options can be set in this file, these options are altering the behaviour of gfloat. Comment lines are allowed in this file, each line that starts with an asterisk * will be regarded as a comment line and will subsequently be ignored by the program.
Options.bak is a backup 'options' file with default settings.
Copying options.bak to options will reset the options.
The desription of the options follows below.

2.2.1
output in file
Can be set to YES or NO.
If this option is set to YES the screen output will be redirected to the file gfloat-1.1/data/*.out
Set to YES the out put will appear on the screen.
Example and default setting:
output to file = NO

2.2.2
total degree reverse lex ordering
Can be set to YES or NO
If this option is set to NO the lexicographic groebner base will be computed directly. This is not suitable for 'big' input equations (i.e. many different indeterminates and/or high total degree of the input eaqutions). This setting is not efficient and is used to create examples.
Set to YES gfloat will first compute a total degree reverse lex base and subsequently transform it to a lexicographic Groebner base.
This setting is the standard setting.
Example and default setting:
total degree reverse lex ordering = YES

2.2.3
new input file (deletes previous output)
Can be set to YES or NO
Set to YES all previously computed output is deleted. Please note that this options has to be set to YES if the input file was changed, otherwise severe errors may occur.
However if the input file has not been changed and float is rerun (especially when it has been necessary to increase the mantissa length of the floats) it may be efficient to set it to NO
Example and default setting:
new input file (deletes previous output) = YES

```
2.2.4
modulus
Can be set to any positive prime integer
This sets the modulus used to perform the modular arithmetic.
Every 'large' prime integer should work fine.
When running a big problem it is useful to choose a high modulus.
However it is useful to run a problem with several moduli to
check if the results are identical.
Example and default setting:
modulus =  1073741827

2.2.5
mantissa length (in bits)
Can be set to any positive integer
Specifies the mantissa length for the floating point arithmetic.
If a 'precision failure' occured during execution of gfloat
the mantissa length can be increased via this option. It
needs some experience to find the 'optimal'mantissa length.
If chosen too small 'precision failures' will occur, if
chosen too big the performance of suffers (because large
mantissas consume time for the floating point arithmetic).
For smaller problems 1000 bits may be enough, for very big problems
10000 bits and more are not uncommon.
Example and default setting:
mantissa length (in bits) = 3000

2.2.6
numerical precision (in decimal digits)
The numerical precision used for the laguerre solver
used to find the solution points of the lexicographic
Groebner Base. A numerical precision of [n] decimal
places provides iteration until the first [n] decimal
places of the floating point number are not changing
any more.
Example and default setting:
numerical precision (in decimal digits) = 50

2.2.7
output level
This option changes the amount of screen output that is
created by the program. Every output level creates
additional output to the level below.
output level = 0 ... minimum output
output level = 1 ... + output of final results
output level = 2 ... + status output
output level = 3 ... + detailed + intermediate results
output level = 4 ... + very detailed status
                     + more intermediate results
output level = 5 ... + all other possible output
Please not that an output level of 3 or higher
substantially affects the performance of the program and
may create very large amounts of output. High output_level
from 3 to 5 in combination with 'output to file = YES'
gives profound insight in the algorithms used by gfloat.
Still output_level of more than 2 should only
be used for small examples.
Example and default setting:
output level = 0
```

```
3.
Output created by gfloat
=========================

Several different files are created by gfloat. These files
are identified by the file extention. The filename [name] is the
same as the input file [name].inp
All these files are created in the /gfloat-1.1/data directory

[name].result ....... A file that contains all results of gfloat.
                      1.) the input data
                      2.) all solution points
                      3.) result of bachsubstitutions of solution
                      points in input equations.
                      4.)  The smallest precision in the
                      lexicographic Groebner base.
                      5.) Time needed for computation
                      6.) Option setting chosen for computation

[name].out .......... file that contains redirected screen output

[name].buchb_info ... information about the base size during
                      buchberger's algorithm.

[name].tdgr[mant] ... total degree reverse lex ordering Groebner
                      base, stored  in internal format. Do not edit
                      this file !!!
                      [mant] is the mantissa length.
                      example: sigi.tdgr1000 is a tdgr base were
                      floating point coefficients have a mantissa
                      of 1000 bits. The input file was sigi.inp

[name].lex[mant] ...  lexicographic groebner Base, stored  in
                      internal format. Do not edit this file !
                      [mant] is the mantissa length.

[name].lextrace ..... Tracefile for lexicographic Groebner Base
                      of the file [name].in, stored in
                      internal format, do not edit !!!!

[name].tdgrtrace .... Tracefile for total degree reverse lex
                      Groebner Base of the file [name].inp
                      Stored in internal format, do not edit !!!!
```

```
4.
Error handling
==============

4.1
Unable to compile gfloat
Please note that the GNU C-Compiler is required to compile gfloat.
It is available over the internet, e.g. www.gnu.org

4.2
'Raise precision'
A precision failure occures, if the mantissa of the floating
point numbers is not big enough. Increase the mantissa length
in the 'options' file and restart.

4.3
'Choose a different modulus'
A multiple of the modulus was hit during the computation. Choose
different
modulus in 'options' file and restart. If this error occurs with
several
different moduli raise precision, and restart.
Note that it is useful to use high moduli, because the probability
of hitting
a multiple of a high modulus is smaller.
```

References

[Becker, Weispfennig 1993] Gröbner Bases, A Computational Approach to Commutative Algebra. Graduate Texts in Mathematics Vol 141, Springer.

[Cox, Little, O'Shea 1992] Ideals, Varieties and Algorithms, An Introduction to Computational Algebraic Geometry and Commutative Algebra. 1st edition, Springer.

[Faugère, Gianni, Lazard, Mora 1993] Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. J. Symbolic Computation 16 (1993) p.329-344.

[Gebauer, Möller 1988] On an Installation of Buchberger's Algorithm. J. Symbolic Computation 6 (1988) p.275-286.

[Giovini, Mora, Niesi, Robbiano, Traverso 1991] One Sugar Cube, Please or Selection Strategies in the Buchberger Algorithm. Proceedings of the 1991 International Symposion on Symbolic and Algebraic Computation ISSAC, ACM Press 1991.

[Hiesleitner 1991] Entwicklung einer Software zur graphischen Darstellung der Kinematik des Viergelenkes, Diplomarbeit, Institut für Mechanik, Technische Universität Graz.

[Knuth 1981] The Art of Computer Programming, Vol.2 Seminumerical Algorithms. 2nd edition, Addition Wesley.

[Lazard 1992] Steward platforms and Gröbner Bases. Proceedings of the 3rd international Workshop on Advances in Robot Kinematics (3ARK, Ferrara, Italy, Sept. 1992).

[Lösch 1996] Gröbner Basen in Fließkommazahlen, Anwendungen in der Kinematik. Dissertation, Institut für Mechanik, Technische Universität Graz.

[Press et al. 1990] Numerical Recipies in C, The Art of Scientific Computing. Cambridge University Press 1990.

[Rump 1988] Algebraic Computation, Numerical Computation and Verified Inclusions. Jaußen. R. (ed.), Trends in Computer Algebra, Springer LNCS 296, p.177-197.

[Traverso 1988] Gröbner Trace Algorithms. ISSAC 1988, Lecture Notes in Computer Science, Springer.