Computer Algebra, Theory and Practice

Arjeh M. Cohen
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
and
Mathematical Institute, State University of Utrecht
P.O. Box 80010, 3508 TA Utrecht, The Netherlands

1. Introduction

The space and time capacities of modern computers (work stations) do not only allow for numerical arithmetic, polynomial arithmetic, the finding of exact solutions of (differential) equations, differentiation and integration, but also for computations in user-defined abstract algebraic structures (like groups, rings and algebras) and computing with elements thereof. I expect that software packages realizing these possibilities will become quite common in the near future. The discipline of mathematics and computer science concerned with the development, use and theoretic background of these packages, is referred to as *computer algebra.*

Graphical facilities also become widely available: it will be possible for work station users to construct, rotate and deform graphs, polytopes and surfaces at will. A suitable analogue for activities in this direction might be *computer geometry* (the name computational geometry is more common); at any rate I will pay no attention to them in the rest of this paper.

In a way, it is no surprise that large portions of algebra can be 'computerised'. For, algebra being the study of operations on sets (cf. [11]), it is possible to examine an algebraic structure on computer provided the elements of the set underlying the structure can be exactly represented and the operations can be described by means of algorithms.

Research in computer algebra addresses (at least) three issues: Firstly, given a structure (e.g. an algebra, ring, semi-group) and a problem related to that structure (for instance: does there exist a unit?), determine whether there exists an algorithm to solve that problem; and if so, give one. Secondly, in the cases where it is clear that an algorithm exists, compare it with other ones, prove or disprove that there are faster ones, that there are alternative ones using less memory; these questions have been studied for about two decades under the

name of 'complexity theory'. Thirdly: what can be achieved with practical implementations?

These three issues cover the whole range from abstract theory to technological practice. Below, in Section 8, I will pay little attention to the third issue and, in Section 7, even less to the second one. I will deal with the first (most theoretical) issue in Sections 3-6, thereby specifying the algebraic structures to groups and polynomial rings, and the problem to the so-called 'word problem'. The main theme is that this classical theoretic setting is quite relevant to questions of everyday practice.

However, before starting the treatment of these three issues, I will present an example of how a computer algebra package can be used to perform straightforward computations with polynomials (some other examples appeared in [3]).

2. On the usage of a computer algebra package—an example

Dealing with representations of Lie groups, I wanted to carry out some computations of the following nature. Given a polynomial map $a,b \mapsto F(a,b)$ determine the rational function $f(x,y)$ in x,y whose formal power series expansion around $(x,y)=(0,0)$ satisfies

$$f(x,y)=\sum_{a,b\geqslant 0} F(a,b)x^a y^b. \tag{1}$$

The paramount polynomials F for which I would like to know the answer were $A2$, $B2$ and $G2$ given by

$$A2(a,b)=\frac{1}{2}(a+1)(b+1)(a+b+2),$$

$$B2(a,b)=\frac{1}{3!}(a+1)(b+1)(a+b+2)(2a+b+3) \text{ and}$$

$$G2(a,b)=\frac{1}{5!}(a+1)(b+1)(a+b+2)(a+2b+3)(a+3b+4)(2a+3b+5).$$

The readers interested in Lie groups might recognise these expressions as the dimensions of representations of Lie groups of the type suggested by the notation, cf. [8, p. 140]. Realising that the identity

$$\frac{c!}{(1-x)^{c+1}}=\sum_{a\geqslant 0}(a+1)\cdots(a+c)x^a \tag{2}$$

can be used to break F down to a polynomial G of smaller degree with control over the rational function for the difference $F-G$, I wrote the following Pascal-like program recursively defining the function `mkrat`. This program is suitable for use in MAPLE†.

† MAPLE is the name of a computer algebra package developed in Waterloo.

```
readlib(coeftayl):
mkrat := proc(F)
local adeg,bdeg,corr,i,antw,r,d,tt;
tt := expand(F);
adeg := degree(tt,a):
if tt=0 then antw := 0
  else
  d := expand(coeftayl(tt,a=0,adeg));
  bdeg := degree(d,b):
  d := coeftayl(d,b=0,bdeg);
  corr := 1;
  for i to adeg do corr := corr*(a+i) od;
  for i to bdeg do corr := corr*(b+i) od;
  corr := corr*d;
  tt := tt-corr;
  r:=((adeg!)*(bdeg!)*d/(((1-y)^(bdeg+1))*((1-x)^(adeg+1))))
  antw := mkrat(tt)+r
fi;
antw
end:
```

The first line of this program reads in the library-function `coeftayl`, which, upon the call `coeftayl(tt,a=0,adeg)`, returns the coefficient of a^{adeg} in the Taylor series expansion of *tt* with respect to *a* around 0. Next, the function `mkrat` is defined, whose sole argument is a polynomial F in a and b. The recursive part of the definition is in the end: the polynomial *corr* with the same highest degree term $d\,a^{adeg}b^{bdeg}$ as F (beforehand, by use of `expand`, F has been written as a sum of terms) plays the role of the expression in (2) above; it corresponds to the rational function r and satisfies $mkrat(F) = mkrat(F - corr) + r$.

Now, imagine the above program has been written in file `m`. The following MAPLE session then leads to the result that I was after:

```
    |\^/|        McMaple V4.0 --- January 1987
._|\|   |/|_.    Licensed by Univ of Waterloo
 \  MAPLE  /     Dept ZW, CWI, Amsterdam
 <_____>         Do not redistribute
    |            For on-line help, type  help( );

>
read m: # the function mkrat is read in from file m#
```

```
>
A2 := (a+1)*(b+1)*(a+b+2)/2: # the example A2 above #

>
B2 := (a+1)*(b+1)*(a+b+2)*(2*a+b+3)/6:

>
G2 := (a+1)*(b+1)*(a+b+2)*(a+2*b+3)*(a+3*b+4)*(2*a+3*b+5)/120:

>
mkrat(A2);
```

$$-\frac{1}{(1-y)^2(1-x)^2}+\frac{1}{(1-y)^3(1-x)^2}+\frac{1}{(1-y)^3(1-x)^2}$$

```
>
a2 := normal(''); #brings the preceding result in normal form#
```

$$a2 := -\frac{-1+yx}{(-1+y)^3(-1+x)^3}$$

```
>
b2 := normal(mkrat(B2));
   #the same as for A2, but now as a compound statement#
```

$$b2 := \frac{1+x-4yx+y^2x+y^2x^2}{(-1+y)^4(-1+x)^4}$$

```
>
g2 := normal(mkrat(G2));  #same for G2 #

g2 :=
```

$$(1+x+8y+8y^2x-6y^4x+15y^3x^2-26y^3x^3+8y^3x^4+y^4x^3 +y^4x^4-26yx-41y^2x+78y^2x^2+8y^2+y^3+15yx^2-6yx^3 +yx^4-41y^2x^3) /((-1+y)^6(-1+x)^6)$$

The result can be easily checked by substituting values for a and b. Here is a function suitable for the check of a single coefficient:

```
poll := proc(f,a,b )
coeftayl(coeftayl(f,x=0 ,a) ,y=0,b)
end:
```

Thus the test

```
poll(g2,1,1);
                                        64

>
subs(a=1,b=1,G2)  ;
                                        64
```

verifies that indeed the coefficient of xy in the power series expansion of $g2$ equals $G2(1,1)$. (Of course, the package knows about **for** loops, so that many more values can be checked with hardly more effort.)

The gist of the above example is that, in a high level programming language like MAPLE, mathematical objects (here: rational functions) are incorporated in such a way that mathematicians can carry out standard computations with these objects in a quite natural and much more efficient way than by pad and pencil.

3. Undecidability

As mentioned above, an important issue in computer algebra is: Given a question one could ask of any algebraic structure of given type (say, group), does there exist a method to find the answer? Much attention has been paid to this topic, long before the name computer algebra became widely accepted. See the Russian encyclopeadia [11] for a short introduction. The (major) part of the theory of algorithms that is based on Church's Thesis points out a specific, natural class $\mathfrak{A}$ of *algorithms*. This class can be defined in various ways, for instance by use of Turing machines. The notions *(un)solvable* and *(un)decidable* for a collection $\mathfrak{B}$ of problems are then defined as the (non-)existence of an algorithm in $\mathfrak{A}$ that, when applied to a problem from $\mathfrak{B}$, gives the right answer. For example, consider the problem $b_{T,w}$: 'Given a Turing machine T and an input word w that can be read by T, does T terminate when w is fed to it?'. Here *termination* stands for stopping after a finite number of steps. Then $\{b_{T,w} | T$ a Turing machine, w an input word for $T\}$ is a standard example of an undecidable collection, called the *Halting Problem*. There even exist particular machines T for which the collection $\{b_{T,w} \mid w$ an input word for $T\}$ is undecidable.

Naturally, algebra too has its unsolvable problems. Here, like in daily life, the word 'problem' is often used to refer to a class of problems. A very fundamental one is the word problem in group theory.

There are three basic ways to present a group: as a permutation group, as a matrix group, or by means of generators and relations. Here, we shall work with the latter. See [12] or [13] for an extensive treatment. Let X be an abstract set. The *free monoid* on X is the set $\mathfrak{M}$ of all words in the alphabet X (including the empty word ϵ), together with the multiplication coming from concatenation. Such a free monoid can be realized on a computer without any problem: it is nothing but the set of words over the alphabet X. The idea behind representing other objects rests on the fact that any monoid on $|X|$ generators can be obtained as a quotient (i.e., a homomorphic image) of $\mathfrak{M}$.

A notion similar to 'free monoid' exists for groups. Let X^{-1} stand for the set of formal elements $\{x^{-1}|x\in X\}$ and stipulate $(x^{-1})^{-1}=x$. The *free group on* X is the group F whose elements are equivalence classes of the free monoid $\mathfrak{M}$ on $X\sqcup X^{-1}$ with respect to the congruence relation $\equiv$ generated by $xx^{-1}\equiv\epsilon$ (for all $x\in X\sqcup X^{-1}$), so that $uxx^{-1}v\equiv uv$ (for all $u,v\in\mathfrak{M}$, $x\in X\sqcup X^{-1}$). The product of two elements $[u]$, $[v]$ of F is $[uv]$, and the inverse of $[x_1\cdots x_m]$ is $[x_m^{-1}\cdots x_1^{-1}]$ (for all $x_1,\ldots,x_m\in X\sqcup X^{-1}$). Thus, F is the image of the natural monoid morphism $u\mapsto[u]$ ($u\in\mathfrak{M}$). In other words, F is the group generated by X in which no relations occur. Each element of F corresponds to a unique *simple word* in $\mathfrak{M}$, that is a word in which no substring of the shape xx^{-1} or $x^{-1}x$ occurs for $x\in X$. There is an algorithm for reducing an arbitrary word to a simple word by successive deletions of occurrences of xx^{-1} or $x^{-1}x$. As a consequence, we can effectively identify F with the subset of $\mathfrak{M}$ consisting of all simple words. Moreover, the problem 'given two words u and v in $\mathfrak{M}$, decide whether $[u]=[v]$' (read: the collection of problems...) is solvable.

Let G be a group. A *presentation* (X,R) *for* G is an abstract set X (of generators) together with a set R (of relations) of pairs of words in $X\sqcup X^{-1}$, usually written in the form

$$x_1\cdots x_m=y_1\cdots y_n$$

where $x_i,y_j\in X\sqcup X^{-1}$ for each i,j, with the property that G is isomorphic with the quotient F/N of the free group F on X (viewed as the subset of $\mathfrak{M}$ of simple words), by the smallest normal subgroup N containing R (viewed as the set of elements $x_1\cdots x_m y_n^{-1}\cdots y_1^{-1}$ of F for each of the relations of R).

Suppose G has presentation (X,R). Then, up to isomorphism, we may take $G=F/N$, with F and N as above. By the above, an element of G can be presented on a computer by a (simple) word over the alphabet $X\sqcup X^{-1}$. The *word problem for* G, or rather (X,R), is the collection of problems: given two words u and v over $X\sqcup X^{-1}$, does $[u]N=[v]N$ hold?

As we have seen above, the word problem for F is solvable. According to the Boone-Novikov Theorem there are groups for which the word problem is unsolvable. Below is an instance of this result in a relatively simple shape (much simpler than the original examples!). It is due to Collins [5] who used a method of Borisov to turn an unsolvable semi-group presentation into an unsolvable group presentation.

For $x,y\in G$, the commutator $xyx^{-1}y^{-1}$ of x and y is denoted by $[x,y]$.

THEOREM (Boone-Novikov; cf. [5]). *There is no algorithm that solves the word problem for the group presentation*

generators: a,b,c,d,e,p,q,r,t,k;

and relations: $[a,p]=[b,p]=[c,p]=[d,p]=[e,p]=p^9$,

$[q^{-1},a^{-1}]=[q^{-1},b^{-1}]=[q^{-1},c^{-1}]=[q^{-1},d^{-1}]=[q^{-1},e^{-1}]=q^9$,

$[a,r]=[b,r]=[c,r]=[d,r]=[e,r]=1$,

$$p\,a\,c\,q\,r = r\,p\,c\,a\,q,\ p^2 a\,d\,q^2 r = r\,p^2 da\,q^2,\ p^3 b\,c\,q^3 r = r\,p^3 c\,b\,q^3,$$
$$p^4 b\,d\,q^4 r = r\,p^4 d\,b\,q^4,\ p^5 c\,e\,q^5 r = r\,p^5 caq^5,\ p^6 d\,e\,q^6 r = r\,p^6 e\,d\,b\,q^6,$$
$$p^7 c\,d\,c\,q^7 r = r\,p^7 c\,d\,c\,e\,q^7,\ p^8 ca^3 q^8 r = r\,p^8 a^3 q^8,$$
$$p^9 d\,a^3 q^9 r = r\,p^9\ a^3 q^9,\ k\,a^{-3} t a^3 = a^{-3} t\,a^3 k,$$
$$[t,p]=[t,q]=[k,p]=[k,q]=1.$$

The proof is based on a reduction to the semi-group from which this group has been built. Proofs of unsolvability of the word problem for semigroups can be given by reduction to the Halting Problem for Turing machines.

4. Coset enumeration for groups

The previous section shows that there are limits to the possibilities of analysing groups presented by generators and relations. Nonetheless there are algorithms providing satisfactory results in many cases. One of these is the so-called Todd-Coxeter algorithm, which takes as input a presentation for G and a set of generators for a subgroup H, and tries to construct the permutation representation $H \setminus G$ of G on the collection of all *cosets* Hg ($g \in G$) of H in G. We recall that the *permutation representation* of G on $H \setminus G$ is the morphism from the group G to the group of permutations of $H \setminus G$ given by $x \mapsto (Hg \mapsto Hg\,x)$.

Let G be a group presented by means of a set X of generators and a set R of relations. Furthermore, let $w_1,\ldots,w_t$ be a collection of words over the alphabet $X \sqcup X^{-1}$ and let H be the subgroup of G generated by the elements represented by the words $w_1,\ldots,w_t$. The *Cayley graph* of G with respect to (X,R) and $w_1,\ldots,w_t$ is the graph whose vertex set is $H \setminus G$ and whose edges are directed and labelled with elements from X: an edge labelled $x \in X$ goes from Hg to Hg' if and only if $Hgx = Hg'$.

The Todd-Coxeter algorithm attempts to construct the Cayley graph starting from the vertex H, which gets the name 1. See [13] for a thorough description of the method. In short, for each vertex drawn and each generator $x \in X$, the edges labelled x emanating from and ending in that vertex are drawn. If it is not (immediately) clear where for instance an edge with a given starting vertex ends, the end vertex will be a newly introduced vertex. The relations however force vertices to coincide every so often. So, after the creation of sufficiently many new vertices, a check on vertex collapse should be run. Three kinds of collapses are to be taken into account:

(i) If i is a vertex from which $x \in X$ points towards both j and k, while $j<k$, then k is identified with j (for x, being a permutation of $H \setminus G$, can send i to only one vertex of Γ; this vertex will be denoted by $i\,x$). The same rule applies to two edges both labelled x and ending in i (yielding a collapse of vertices $i\,x^{-1}$).

(ii) If $x_1 \cdots x_r = y_1 \cdots y_s \in R$ and i is a vertex, then $i\,x_1 \cdots x_r$ coincides with $i\,y_1 \cdots y_s$.

(iii) If $w_i = x_1 \cdots x_r$, then $1\,x_1 \cdots x_r = 1$ (because $w_i \in H$ and $1 = H$ imply $1\,w_i = H = 1$).

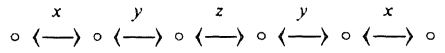
If no more collapses occur and all edges emanating and ending at newly created vertices are drawn, then the Cayley graph of G with respect to (X,R) and $w_1,\ldots,w_t$ has been found. If the Cayley graph is infinite, we cannot expect the algorithm to terminate. In this light, the following theorem is best possible.

THEOREM (Mendelsohn, cf. [13]). *If the index $|H \setminus G|$ of H in G is finite, then the Todd-Coxeter Algorithm terminates.*

From a practical point of view, it would be useful to know how long it would take in terms of the input and/or size of $G \setminus H$. However, very little is known about the time it takes to terminate.

EXAMPLES

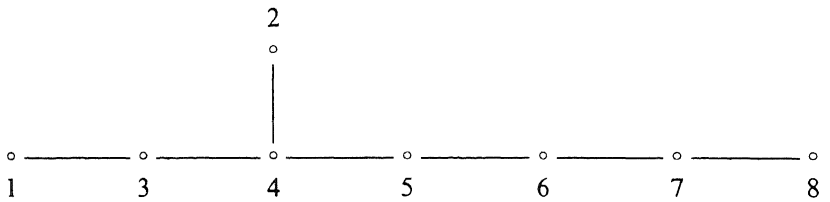
(i) **The group G with presentation (X,R) where $X=\{x,y,z\}$ and $R=\{x^2=y^2=z^2=1,(xy)^3=(xz)^2=(yz)^4=1\}$.** In constructing the Cayley graph we start with the vertex corresponding to the subgroup H generated by y and z. Leaving out the 'loops', i.e., edges whose end and starting vertices coincide, we get



where H is an end vertex.

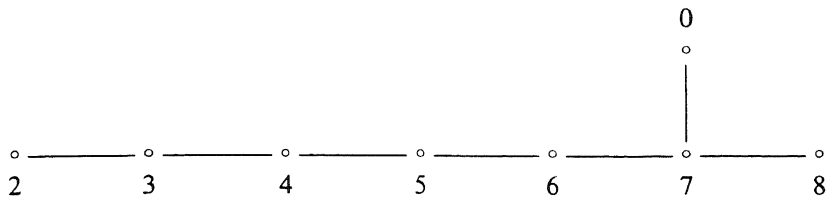
(ii) **Collins' group.** Todd-Coxeter applied to the group presentation in the Boone-Novikov Theorem as formulated in Section 3 and the words p,q,r,t,k gives a single coset; this means that each of a,b,c,d,e can be expressed in terms of p,q,r,t,k.

(iii) **The Weyl group of type E_8.** Let W be the Weyl group of type E_8. That is, W has the presentation with generators $r_1,\ldots,r_8$ and relations visualised by the Dynkin diagram of type E_8:



The relations are to be read from the diagram by means of the rules: $(r_ir_j)^3=1$ if $\{i,j\}$ is an (unlabelled) edge of the diagram and $(r_ir_j)^2=1$ otherwise (also if $i=j$). We shall frequently write i instead of r_i $(1\leqslant i\leqslant 8)$. We select the words $2,3,4,5,6,7,8,w^{-1}8w$ where $w=765432415635276145341324567853421345678$, and let D denote the subgroup of W generated by them (read: their images in W). From the theory of Weyl groups it can be derived that D is isomorphic to the Weyl group of type D_8. (The argument I have in mind is that the generators of D are the reflections with roots $\alpha_2,\alpha_3,\ldots,\alpha_8,\alpha_0$, where α_0 is the longest root.) At any rate, D has the presentation with generators $2,3,4,5,6,7,8,0$,

where 0 stands for $w^{-1}8w$, given by the diagram below in the same manner as for W above:



As a permutation of $D \setminus W$, the so-called *Coxeter element* $t = 14683257$ consists of 9 cycles of length 15. Write $c = 134354365768$ and $d = 134254316542347$. The Todd-Coxeter algorithm, as implemented in e.g. CAYLEY† and a little extra work, yield that

$$Y = \{1, c, c^2, c^3, c^4, c^5, d, d^3, d^4\}\{t^i \mid 0 \leqslant i \leqslant 14\}$$

is a full set of coset representatives of D in W, that is,

$$W = \coprod_{y \in Y} Dy.$$

5. Knuth-Bendix

In order to study an algebraic structure A by means of a computer we need to find an effective way of representing its elements. The general strategy is to consider A as the quotient of an object $\mathfrak{M}$ whose elements can easily be represented on a computer. For instance, if $\mathfrak{M}$ is the free monoid on the finite set (alphabet) X and we have a surjective morphism $\phi: \mathfrak{M} \to A$, an element $a \in A$ will then be *represented* by any $w \in \phi^{-1}(a)$. We have seen instances (where A is a group) of this principle in the previous section, and we shall see another in the next one (where A is a quotient ring and $\mathfrak{M}$ a polynomial ring, which has again the property that its elements can be easily represented on computer).

In this section we shall take A to be a group G with presentation (X, R) (both X and R finite). For convenience, we assume that X is closed under taking inverses (that is, $x^{-1} \in X$ whenever $x \in X$, in which case $(x^{-1})^{-1} = x$).

In this and the following section, we shall concentrate on a method to solve the word problem for A which, in a very general form is attributed to Knuth & Bendix. In [9] and [12] more elaborate introductions to the subject can be found than the one given here.

In order to compare elements of $\mathfrak{M}$, we introduce a linear ordering $<$ on $\mathfrak{M}$ refining the relation '|' (that is 'is a divisor of'). For all $m, m', m'', n \in \mathfrak{M}$ we require

(i) if $m \neq 1$ then $1 < m$;

(ii) if $m' < m''$ then $mm'n < mm''n$.

† CAYLEY is a software package for group theoretic computations developed by J.J. Cannon in Sydney.

A linear ordering with these properties is called a *reduction ordering* (in other literature also *admissible*). The *total degree ordering* (first according to total degree, then lexicographically) is an example. The ordering is *well founded*. That means that each decreasing sequence terminates (again meaning: stops after finitely many steps). Now view each of the relations $x_1 \cdots x_n = y_1 \cdots y_m \in R$, if necessary, rewritten so as to obtain $x_1 \cdots x_n > y_1 \cdots y_m$, as an *elementary rewrite rule*

$$x_1 \cdots x_n \Rightarrow y_1 \cdots y_m.$$

The word problem can be conveniently reformulated by use of this ordering. A *rewrite rule* has the form $umv \Rightarrow um'v$ with $m = m' \in R$ and $m > m'$. By definition, two words $w, w' \in \mathfrak{M}$ correspond to the same element of G if w' can obtained from w by a sequence of rewrite rules and their inverses (an *inverse rewrite rule* is a replacement of the shape $umv \Rightarrow um'v$ with $m = m' \in R$ and $um'v \Rightarrow umv$ a rewrite rule). Given $w \in \mathfrak{M}$, determining whether w represents the identity in G, could be an impossibly hard task because it is not clear how far 'up' with respect to the ordering (using inverse rewrite rules) one has to go before the word collapses to ϵ with rewrite rules. We say that w *reduces* to w', notation $w \Rightarrow^* w'$, if there is a sequence of rewrite rules that, when successively applied to w, yields w'.

Each subset of $\mathfrak{M}$ has a unique smallest element with regard to $<$. So if we can find an algorithm that, upon input $w \in \mathfrak{M}$, returns the smallest word of the subset consisting of all words representing the same element of G as w, this algorithm will solve the word problem for (X,R). The *Knuth-Bendix Algorithm*, when applied to (X,R), attempts to create a new set R' of relations with the same quotient group G, but with the additional property that, if u, v, w are words in $\mathfrak{M}$ such that both $u \Rightarrow^* v$ and $u \Rightarrow^* w$, then there is $x \in \mathfrak{M}$ with $v \Rightarrow^* x$ and $w \Rightarrow^* x$. This property is called *confluence*. It will be clear that by use of R' every word w can be rewritten to a unique smallest word w_0, and that w_0 is the unique smallest word in $\mathfrak{M}$ corresponding to the same element of G as w. Thus, if the Knuth-Bendix algorithm succeeds, the word problem for the presentation (X,R) is solvable.

EXAMPLES

(i) **Example (i) of Section 4 revisited.** The following system of 10 rewrite rules for the presentation (X,R) of that example is confluent with regard to the total degree-reduction ordering satisfying $x < y < z$.

$$x^2 \Rightarrow 1;\ y^2 \Rightarrow 1;\ z^2 \Rightarrow 1;$$

$$x^{-1} \Rightarrow x;\ y^{-1} \Rightarrow y;\ z^{-1} \Rightarrow z;$$

$$y\,x\,y \Rightarrow x\,y\,x;\ z\,x \Rightarrow x\,z;\ z\,y\,z\,y \Rightarrow y\,z\,y\,z;$$

$$z\,y\,x\,z\,y\,x \Rightarrow y\,z\,y\,x\,z\,y.$$

(ii) **Coxeter groups.** If $(m_{ij})_{1 \leq i,j \leq n}$ is an $n \times n$ matrix with $m_{ii} = 1$ and $m_{ij} = m_{ji}$ for all i,j, then the *Coxeter system* over $(m_{ij})_{1 \leq i,j \leq n}$ is the group presentation

with generators $r_1, \ldots, r_n$ and relations $(r_i r_j)^{m_{ij}} = 1$ for all $i,j \in \{1, \ldots, n\}$. (Examples (i) and (iii) of Section 4 are Coxeter systems.) The corresponding group G is called a *Coxeter group*. Tits has given a nice set of rewrite rules for these systems that is neither confluent nor obeys a reduction ordering, but still leads to an algorithm for determining whether two words represent the same element of G. Let $\mathfrak{M}$ be the free monoid over $r_1, \ldots, r_n$ and consider the rewrite rules

$$r_i r_i \Rightarrow 1 \quad (1 \leqslant i \leqslant n), \tag{1}$$

$$r_i r_j r_i \cdots \Rightarrow r_j r_i r_j \cdots \quad (1 \leqslant i,j \leqslant n) \qquad \text{both sides of length } m_{ij}. \tag{2}$$

As $r_i^{-1} = r_i$, there is no need for additional formal symbols $r_i^{-1} \in \mathfrak{M}$. Tits has shown that the word problem 'does w represent 1 in W?' is solvable by means of the algorithm: continue to apply rewrite rules (2) till either two consecutive occurrences of the same generators appear (so that $r_i r_i$ is a subword for some i) or no more new words of the same length can be obtained by means of (2). In the former case, remove $r_i r_i$ from the word and iterate the procedure from the beginning with the newly obtained word (the length has decreased, so this case occurs a finite number of times). In the latter case stop: the answer is *yes* if the word is empty and *no* otherwise.

Better algorithms are obtained by means of a presentation of G as a matrix group. Nevertheless, an interesting, as yet unsolved problem (1989) is to find a confluent (possibly infinite) set of rewrite rules for any Coxeter group.

Instead of discussing the Knuth-Bendix algorithm in full generality, I will switch to a famous instance where the algorithm always succeeds: quotients of polynomial rings.

6. Gröbner bases

Let R be a field. For most of the remarks below on solvability to make sense, it is needed that R itself can be effectively presented on a computer, but we shall not worry about that here. (The problems would dissolve if we were to take for R the field of rational numbers.) Put $T = R[X_1,\ldots,X_n]$—the polynomial ring with indeterminates $X_1,\ldots,X_n$. The ring T plays a similar role for rings as the free group F for groups. In particular, we imagine the elements of T to be representable on a computer (each element of T having a unique presentation) and use it to study its quotient rings. To this end we specify a set B of elements of T (polynomials over R in $X_1,\ldots,X_n$, and denote by (B) the ideal of T generated by B; the elements f and f' of T represent the same element of the quotient ring $S = T/(B)$ if (and only if) $f+(B) = f'+(B)$. The *word problem for S (with respect to B)* is: Given B and two polynomials f, f' in T, does $f \equiv f' \bmod (B)$ hold (equivalently, does $f+(B) = f'+(B)$ hold in S)?

A *monomial* in T is an element of T of shape $X_1^{a_1} \cdots X_n^{a_n}$, often abbreviated to X^a, where a stands for $(a_1, \ldots, a_n) \in \mathbb{N}^n$. By $\mathfrak{M}$ we denote the set of all monomials. Thus, $m = X^a \in \mathfrak{M}$ can be identified with the element a of $\mathbb{N}^n$, and we shall do so whenever convenient. An element $f \in T$ can be uniquely written

$f=\sum_{m\in\mathfrak{M}} f_m m$. Now $f_m m$ is called a *term* of f.

We choose a reduction ordering $<$ on $\mathfrak{M}$ (i.e., an ordering satisfying the properties (i) and (ii) of Section 5). The lexicografical ordering is an example. The highest monomial occurring in $f\in T$ is denoted by lm_f (short for *leading monomial*); thus

$$lm_f := \max \mathfrak{M}_f, \quad \text{where } \mathfrak{M}_f := \{m\in\mathfrak{M} \mid f_m \neq 0\}.$$

The coefficient of this monomial in f is denoted by lc_f.

When confronted with $f\in T$, we want a canonical representative of $f+(B)\in T/(B)$. This representative g should have minimal lm_g. If, for $f,g\in T$ there exist $m\in\mathfrak{M}_f$ and $b\in B$ with

$$lm_b | m \quad \text{and} \quad g=f-\frac{f_m}{lc_b}\frac{m}{lm_b}b$$

(so that $g_m=0$), then we write $f\Rightarrow_B g$, or, more succinctly, $f\Rightarrow g$. Observe that g is smaller than f in the sense that $\mathfrak{M}_f > \mathfrak{M}_g$ (with respect to a natural ordering on subsets induced by $<$) and that $g+(B)=f+(B)$. Let $g,h\in T$. We say that g *reduces to* or *can be rewritten to* h modulo B, notation $g\Rightarrow^* h$, if there is a (possibly trivial) sequence of reductions starting with g and ending with h. Now, g is called a *normal form* with respect to B and $<$ if there is no $h\in T$ with $g\Rightarrow_B h$. Furthermore, $h\in T$ is called a *normal form* of $g\in T$ if h is in normal form and $g\Rightarrow^* h$. Now the notion of Gröbner basis can be introduced. B is called *Gröbner basis* with respect to $<$ if each element of T has a unique normal form with respect to B.

EXAMPLE. A normal form need not be unique. Take $T=\mathbb{Q}[X,Y]$ and $B=\{X^2Y-1,\ XY^2-1\}$. Then X and Y are both normal forms of X^2Y^2 with respect to the lexicografical ordering. The element $X-Y=Y(X^2Y-1)-X(XY^2-1)\in(B)$ is in normal form but cannot be rewritten to 0.

The ordering $>$ induces a well-founded ordering on the collection of subsets of $\mathfrak{M}$ (an easy consequence of the fact that $>$ on $\mathfrak{M}$ is well founded). Consequently, there is an algorithm **normal** (B,f) reducing any $f\in\mathfrak{M}$ to a normal form with respect to B: Find $b\in B$ and $m\in\mathfrak{M}_f$ such that $lm_b|m$; if b,m are found, take

$$f := \textbf{normal}(B, f-f_m m(lm_b lc_b)^{-1} b)$$

and invoke recursion; if b and m cannot be found, f is in normal form.

THEOREM (Gröbner bases characterisation, cf. [2]). *Suppose $<$ is a reduction ordering on the set $\mathfrak{M}$ of monomials in T. The following statements concerning a finite subset B of T are equivalent.*

(i) *B is a Gröbner basis;*

(ii) *for all $f,g,h\in T$ with $h\Rightarrow_B f$ and $h\Rightarrow_B g$, there exists $k\in T$ with $f\Rightarrow_B^* k$ and $g\Rightarrow_B^* k$;*

(iii) *each element of* (B) *reduces to* 0;
(iv) 0 *is the unique normal form of each element in* B;
(v) $\{lm_f | f \in (B)\} = \{t\, lm_b | b \in B, t \in \mathfrak{M}\}$.

These characterisations are not effective: they do not show how—in a finite number of steps—it can be checked whether a given subset B of T is a Gröbner basis. To reach that goal, we need the following concept. The *S-polynomial* of two polynomials $f, g \in T$, notation $S(f,g)$, is

$$S(f,g) = \begin{cases} 0 & \text{if } f = 0 \text{ or } g = 0 \\ \operatorname{lcm}(lm_f, lm_g)(lc_g lm_f^{-1} f - lc_f lm_g^{-1} g) & \text{otherwise} \end{cases}.$$

THEOREM (Effective characterisation of Gröbner bases). *Let* $<$ *be a reduction ordering on the set of monomials of the polynomial ring* T. *The following are equivalent:*
(i) B *is a Gröbner basis;*
(ii) *if* $b, b' \in B$ *then* 0 *is a normal form of* $S(b,b')$.

A Gröbner basis of a (finitely generated) ideal I of $T = R[X_1, \ldots, X_n]$ enables us to uniquely represent elements T/I by elements of T. But it also helps solving quite a few other problems: for instance in deciding whether two subsets generate the same ideal. There is no bijective correspondence between ideals and Gröbner bases.

EXAMPLE (Several Gröbner bases with respect to the same ordering and generating the same ideal). Take $T = R[X,Y]$, and $B^{(i)} = \{X - Y^{1+3i},\ Y^3 - 1\}$. Then $I = (B^{(i)})$ is independent of $i \in \mathbb{N}$. Each $B^{(i)}$ is a Gröbner basis of I with respect to the lexicografical ordering on $\mathfrak{M}$ with $X > Y$:

$$\begin{aligned} S(X - Y^{1+3i},\ Y^3 - 1) &= -(X - Y^{1+3(i+1)}) \Rightarrow_{B^{(i)}} (X - Y^{1+3(i+1)}) - (X - Y^{1+3i}) \\ &= Y^{1+3i}(1 - Y^3) \Rightarrow^*_{B^{(i)}} 0. \end{aligned}$$

A slight refinement rids us of this ambiguity: let I be an ideal of T. We call B a *reduced Gröbner basis* of I if it is a Gröbner basis and if each $b \in B$ has $lc_b = 1$ and is in normal form with respect to $B \setminus \{b\}$. Then (cf. [2]) I has a unique reduced Gröbner basis B. This basis is a minimal Gröbner basis of I in the sense that no subset of B is also a Gröbner basis of I. Moreover, $|\{lm_b | b \in B\}| = |B|$.

The above theorem also provides an algorithm for finding Gröbner bases.

ALGORITHM GBASIS: Given B return a Gröbner basis of (B)

$P := \{\{b,b'\} | b, b' \in B\}$;
while $P \neq \varnothing$
do choose $\{b,b'\} \in P$;
$P := P \setminus \{\{b,b'\}\}$;

$c := \mathbf{normal}(B, S(b,b'))$;

if $c \neq 0$

then $P := P \cup \{\{b,c\} | b \in B\}$;

$B := B \cup \{c\}$;

fi

od;

return B

There is a paucity of applications of the Gröbner basis theory. We only mention

THEOREM (A. van Essen [7]). *Let* f: $R^n \to R^n$ *be a polynomial mapping given by* $f(X) = (f_1(X),...,f_n(X))$, *with* $f_i(X) \in T$ *for each* i. *Choose a reduction ordering on the monoid of monomials in* $X = (X_1,...,X_n)$ *and* $Y = (Y_1,...,Y_n)$ *with* $Y_1 < Y_2 < \cdots < Y_n < X_1 < X_2 < \cdots < X_n$. *Write* $B = \{Y_i - f_i(X) | 1 \leqslant i \leqslant n\}$. *Then* f *is invertible in* $R[X]$ *if and only if there exist* $g_i(Y)$ $(1 \leqslant i \leqslant n) \in R[X,Y]$ *such that* $\{X_i - g_i(Y) | 1 \leqslant i \leqslant n\}$ *is the reduced Gröbner basis of* (B). *If, moreover,* f *is invertible then its inverse is given by* $Y \mapsto (g_1(Y),...,g_n(Y))$.

The practical performance of Gröbner bases algorithms is rather limited: the algorithms require a lot of space and time. Nevertheless, the algorithms are (and should be) available in most general purpose computer algebra software packages.

7. Complexity

So far, we have been looking at problems for which it is not clear at the outset whether a solution *can* be found. For problems like multiplying two integers, such questions are not even asked. There, the issue is to determine how *fast* they can be multiplied. This is the domain of complexity theory; cf. [1] and [10] for excellent introductions. Various results in this area, originating from pure mathematics have been implemented on computers: fast (probabilistic) primality tests and factorisations of integers, the Discrete Fourier Transform for fast multiplication of polynomials, the so-called L(ower) U(pper-diagonal) P(ermutation matrix)-decomposition for fast multiplication and inverting of matrices see e.g. [4] for an introduction. According to my taste, complexity theory is a fascinating subject as it often requires original and creative work to come up with algorithms that are better than the known ones.

8. Practice

Apart from the theoretic issues discussed so far, computer algebra also deals with practical aspects like actually building a software package and making it available for easy use to a large audience. Usually a compromise has to be made between the two goals of having the 'fastest implementation possible'

and providing 'user-friendliness'.

There are inspiring interactions between the actual building of packages and theoretic problems. A good example is the algorithmic variant of classical 'integration'. See [6] for this and other examples.

In the year 1989, at least two packages have been constructed in The Netherlands. One of them is called **FORM**; it has been constructed by J. Vermaseren and specialises in fast computations with huge expressions of use to high energy physicists and mathematicians, such as rational functions. In limiting to a restricted area of mathematics, it is able to perform considerably better than the big general purpose packages.

The other package is **LiE**. It has been developed at CWI to perform computations of Lie group and Weyl group theoretic nature. Simple complex Lie group representations can be parametrised by vectors with non-negative integer entries—the so-called *dominant weights*. For instance, for the Lie group of all 3×3 matrices having determinant equal to 1, the dimension of the representation belonging to dominant weight $[a,b]$ amounts to the value $A2(a,b)$ of the polynomial $A2$ defined in Section 2. In **LiE**, a standard formula, the so-called Weyl's Dimension Formula (cf. [8]) has been used to compute these numbers. The rational generating function $a2$ of Section 2 provides a more compact way to store the map

$$x,y \mapsto \sum_{a,b \geqslant 0} A2(a,b)\, x^a y^b.$$

Now, on the one hand, it is known that many more functions than the dimension have rational generating functions (examples being tensor product decompositions, 'branchings', etc.), on the other hand, the only algorithms known to evaluate these functions are rather space and time consuming. Therefore, for a given Lie group, the question arises, whether there is an effectively computable bound on the number of values that have to be computed for the function in question in order to be able to determine the corresponding rational generating function. For, once it has been found, more values of the function can be computed by mere polynomial arithmetic. Thus, a practical observation raises a question of highly theoretical nature (involving the homology of certain rings related to the Lie group).

We finish with an example of the other way around, where the theory (of Weyl groups) provides tricks to improve standard algorithms in a software package (**LiE**). For the sake of exposition, we restrict attention to the complex simple Lie group G of type E_8. Each simple Lie group corresponds to a Weyl group; the one for G is the Weyl group W of type E_8 given in Example (ii) of Section 4. The Weyl group W can be represented as a set of invertible square matrices of size 8 with integer entries, so that it transforms vectors with integral entries of size 8. An *orbit* of W on the vector space is a set of the form $\{wv \mid w \in W\}$ for some vector v. Standard methods to compute the trace of a diagonisable element in G require enumerating all elements of orbits of W. But there are orbits of 696,729,600 vectors, so enumeration should not involve storing all orbit elements. Since the subgroup D as in Example (ii) of Section

4 is *monomial* (that is, after a suitable change of bases, all matrices have exactly one non-zero entry in each column and in each row), it is straightforward to write an algorithm that enumerates all vectors of the orbit $\{wv|w \in D\}$ for any given vector v but does not store more than two vectors. Now using the coset representatives of Example (ii) of Section 4, this algorithm for D could be extended to a relatively fast algorithm for W, enumerating the elements of any orbit of W in a very limited amount of space.

REFERENCES

1. A. BORODIN, I. MONRO (1975). *The Computational Complexity of Algebraic and Numeric Problems,* Elsevier, Amsterdam.
2. B. BUCHBERGER (1985). Gröbner bases: an algorithmic method in polynomial ideal theory. N.K. BOSE (ed.). *Multidimensional System Theory,* Reidel, Dordrecht, 184-232.
3. ARJEH M. COHEN (1988). Mathematical formula manipulation from a user's point of view. *CWI Quarterly 1(3),* 53-63.
4. ARJEH M. COHEN, BERT RUITENBURG (1989). *Algebra and Computer Algorithms,* lecture notes.
5. D. COLLINS (1989). *Unsolvable Decision Problems,* a letter, 1-8.
6. J.H. DAVENPORT, Y. SIRET, E. TOURNIER (1988). *Computer Algebra, Systems and Algorithms for Algebraic Computation,* Acad. Press, London.
7. A. VAN ESSEN (1986). *A Criterion to Deduce if a Polynomial Map is Invertible and to Compute its Inverse,* Report Kath. Univ. Nijmegen, 1-6.
8. J.E. HUMPHREYS (1972). *Introduction to Lie Algebras and Representation Theory,* Springer, New York.
9. J.W. KLOP, A. MIDDELDORP (1988). An introduction to Knuth-Bendix Completion. *CWI Quarterly 1(3),* 31-52.
10. D.E. KNUTH (1973). *The Art of Computer Programming Vol 3,* Addison-Wesley, Reading MA.
11. M. HAZEWINKEL (1988). *Encyclopaedia of Mathematics, Vol 1 A-B, An updated and annotated translation of the Soviet 'Mathematical Encyclopaedia',* Reidel, Dordrecht.
12. C.C. SIMS (1989). *Computation with Finitely Presented Groups,* preliminary draft of a book.
13. M. SUZUKI (1982). *Group Theory I,* Springer Verlag, Berlin.