The Word Problem and Relations in Rings

John Wavrik

This is a report on current research into the word problem and relations in non-commutative rings. My interest in computational problems in ring theory predates my interest in computers and I have done work with computers on these problems for a long time. A major breakthrough occurred in about 1997 when I discovered an algorithm, similar to the Gröbner Basis algorithm, which applied to polynomials in non-commuting variables with integer coefficients. The main ideas and examples were given in an article presented at the ISSAC conference in 1999. This is a report on the current state of this research.

The Word Problem

This problem can be posed in several related forms. In the case of group theory it is formulated in terms of determining a group given generators and relations.

> Consider an alphabet, G, of letters we call generators. Consider a set, R, of pairs (l,r) of words in the generators. These are called relations. We define on the set of words in G an equivalence relation: we take the transitive closure of SlT ≈ SrT if (l,r)$\in R$,. The word problem is to find an algorithm for determining whether or not two words in G are equivalent.
>
> Example:
>
> Let G = { a, b, c } and R = { (ab,c), (bc,a), (ca,b) }
>
> Claim: ba and aaab are equivalent
> Proof: $\underline{a}aab \approx b\underline{ca}b \approx bb\underline{ab} \approx b\underline{bc} \approx ba$ [1]

The word problem is known to be undecidable. There can be no general algorithm which solves the word problem for arbitrary G and R .

A related problem is the ideal membership problem in rings. Let R be a ring (not necessarily commutative) and I an ideal. The problem is to find an algorithm for deciding if an element f is or is not in the ideal. The ideal membership problem is also undecidable. If R is the free algebra over the rationals generated by G and I is the ideal generated by { l-r | (l,r) $\in$ R } then $w_1 \approx w_2 \Leftrightarrow w_1 - w_2 \in I$. Thus a solution to the ideal membership problem would give a solution to the word problem for groups.

Saying that the ideal membership problem is undecidable does not preclude the possibility of an algorithm that allows one to decide membership in a particular ideal I of a particular ring R. Nor does it preclude the possibility of an algorithm that works for an entire class of rings; or that works for some ideals but not others.

---

[1] I have underlined what is replaced from one stage to the next. The subword to be replaced can either occur on the right or the left of one of the relations.

Buchberger (1965) presented an algorithm which works for any ideal in $R = k[x_1,\ldots,x_n]$ (polynomials in commuting variables over a computable field, e.g. $\mathbb{Q}$). This is regarded as a major breakthrough in computational algebraic geometry.

Mora (1986) presented an algorithm for $R = k\langle x_1,\ldots,x_n\rangle$ (polynomials in non-commuting variables over a computable field). Mora's algorithm solves the ideal membership problem for *certain* ideals.

Gröbner Basis Algorithms

Both Buchberger and Mora's algorithms are based on a generalization of the Euclidean division algorithm to several variables. The division algorithm allows us, given polynomials f, g, to write $f = gq + r$ where r=0 or has smaller degree than g. In the case of one variable this provides an algorithm for ideal membership: f is in the ideal generated by g if and only if r=0.

For 1 variable we have the natural ordering $1 < x < x^2 < \ldots$ Every polynomial can be written with its terms in descending order. There is a leading term which determines the degree. An analysis of the division algorithm shows that we use the leading term of g to "kill off" the terms of f of higher degree.

For several variables there are many choices for term ordering. The results of the algorithms depend on the choice of term ordering. Both Buchberger and Mora's algorithms assume that we have a finitely generated ideal.[2] If $I = \langle g_1,..,g_s\rangle$ the generalized division algorithm allows us to write any polynomial f in the form $f = g + r$ where g is in I and where no term of r is divisible by any of the leading terms of the g_i.

The division process is a genuine computational algorithm. In both the commutative and non-commutative case there is an efficient way to compute g and r. However, in distinction from the 1-variable case, r=0 does not usually give a test for ideal membership. Certainly if r=0 we can conclude that f is in the ideal. But we cannot include the converse.

Example: $g_1 = x^2 - y$; $g_2 = xy - x$

$f = y^2 - y = (1-y)*g_1 + x*g_2$

f is in the ideal but r = f (not zero) in the division process.

The division process is not well behaved. It cannot be used directly to test for ideal membership.

The remarkable discovery of Buchberger is an algorithm which replaces a given basis for an ideal by a new basis (he called a Gröbner Basis in honor of this thesis advisor). A Gröbner basis for an ideal has the property that $f \in I \Leftrightarrow r=0$. Thus Buchberger not only showed that every ideal *has* a basis for which the ideal membership problem can be solved computationally, he also gave an algorithm which can be used to compute such a basis.

As a consequence of Buchberger's algorithm, the word problem can be solved for commutative semigroups.

---

[2] In the work I have done, ideals are two-sided.

The process of computing a Gröbner basis can be explained in several ways. The most popular way involves certain syzygies on the current basis elements. If f and g are polynomials we write (using multi-index notation):

$f = ax^{\alpha} + \text{lower terms}$
$g = bx^{\beta} + \text{lower terms}$

$Spol(f,g) = (f/a)x^{\gamma-\alpha} - (g/b)x^{\gamma-\beta}$ where $\gamma = lcm(\alpha,\beta)$

In the non-commutative case the situation is more complicated since the monomials are words and cannot be expressed as x^{α}, and the concept of least common multiple cannot be generalized to pairs of words. For any pair of polynomials there can be more than one S-polynomial or there can be none. We can find common multiples of the leading terms which play the role of x^{γ}.

Example: yxy and xyx are both common multiples of yx and xy. If
f = xy – ax and g = yx – by we get two S-polynomials
s1 = fx – xg = xby – axx and s2 = yf – gy = yax - byy.

Teo Mora has shown that it is only necessary to consider S-polynomials resulting from "matches" (or non-trivial overlaps). Two monomials M1 and M2 have an overlap if there are monomials L,R, and U so that M1=LU and M2=UR or if M1 = UR and M2 = LU. The common multiple (taking the place of LCM(M1,M2)) is M = LUR.

Example: M1 = yxx and M2 = xxy have three overlaps. The resulting common multiples are y<u>xx</u>y, yx<u>x</u>xy, xx<u>y</u>xx where the U is underlined in each case. If f = yxx + lower terms, g = xxy + lower terms then we obtain 3 S-polynomials for f and g: (1) fy – yg, (2) fxy – yxg, (3) xxf – gxx

In any case, the S-polynomial "kills" the leading terms of f and g. Both Buchberger and Mora show that a basis G is a Gröbner basis if and only if $Spol(f,g)\rightarrow 0 \ \forall \ f,g\in G$ where $f\rightarrow r$ (f reduces to r) means that r is the remainder of f upon division by G.

The basis algorithm starts with a basis G=[g1,..,gs]. If $Spol(gi,gj)\rightarrow h \neq 0$ then h is added to the basis. [There are some technical details]. The process is repeated until we obtain a basis satisfying the condition above.

In the commutative case, Buchberger shows that the process always terminates (and produces a Gröbner basis). In the non-commutative case, Mora notes that the process does not always terminate – but, when it does, it produces a Gröbner basis.

Because of the word problem there must be, in the non-commutative case, ideals which do not have a Gröbner basis.

Integer Coefficients

My paper develops an algorithm for non-commutative polynomials with integer coefficients. Both Buchberger's and Mora's algorithms use, quite heavily, the fact that coefficients are in a field. The problem is visible even from the start: in the division algorithm. Remember that the division algorithm, the key ingredient, has the 1-variable Euclidean division algorithm as a special case.

Example: If f = 3xx + 2x + 1 and g = 2x + 1 we can divide f by g to get a quotient $q = \left(\frac{3}{2}x + \frac{1}{4}\right)$ and a remainder $r = \frac{3}{4}$

Our ability to "kill" the x^2 term requires division in the coefficient domain. What should be the quotient and remainder if we stay within the integers?

The solution I use involves extending the term ordering to use an ordering on the coefficients as well as the ordering on the monomials. We place a total ordering on the integers:

$$0 < -1 < 1 < -2 < 2 < \ldots.$$

used by Buchberger in the commutative case [Buch3]. We choose an ordering on the monomials (words) and we then order terms by:

aS < bT if S < T (in the ordering of words)
or S = T and a < b (in the ordering of integers)

We cannot hope to "kill" terms in the division process – but only to replace them by terms of lower order.

Example: If f = 3xx + 2x + 1 and g = 2x + 1 we can divide f by g over the integers to get a quotient q = 2x and a remainder r = -xx + 1.

This generalizes to a multi-variable division of a polynomial f by a list $G=[g_1,..,g_s]$. Whenever a monomial in f is divisible by the monomial part of the leading term of a g_i, we subtract a suitable multiple of g_i to reduce the order of the term in f. The result is f = g + r where g is in the ideal $<g_1,..,g_s>$ and where none of the terms of r can be reduced further by the leading term of any g_i.

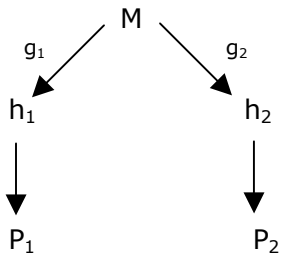
This generalized form of the division algorithm can be used to produce a generalized basis algorithm. In my implementation I used an alternative to S-Polynomials called "critical pairs". The idea is that a Gröbner basis can also be characterized by the property that the remainder upon reduction is unique.

The reduction process (even for coefficients in a field) depends upon choices. The reduction of f by $G=[g_1,..,g_s]$ can depend on the order in which we used the g_i and the position within a term in f where the leading term of g_i is matched to a substring.

Example: Let $G = \{g_1,g_2\}$ with g_1 = xxx – x and g_2 = yxx + xyx + xxy. We can reduce f = yxxx first by g_1 to obtain a remainder yx or we can reduce first by g_2 and (after several steps) obtain the remainder xy.

This observation is the key to the method for producing new basis elements (and hence is at the heart of the basis algorithm).

If a monomial M is a common multiple of the leading terms of g_1 and g_2, we can always reduce M in two ways: first we divide by g_1 (and then continue the division process to obtain a remainder P_1) or we first divide by g_2 (and then continue the division process to obtain a remainder P_2). We obtain a diagram



Notice that $P = P_1 - P_2$ will always be in the ideal generated by the current basis G. If G is a Gröbner basis they will always be equal. If $P \neq 0$ then it is appended to G. [There are some technical points about when it is appended and which basis is used for reduction.]

An issue is how to chose M. In the commutative case (coefficients in a field), M is chosen to be lcm(LT(gi),LT(gj)) where LT(f) is the leading term of f. In the non-commutative case (coefficients in a field) M is chosen to be a common multiple of LT(gi) and LT(gj) obtained from "matches" (see the discussion of Mora's algorithm above). We obtain a pair for each match.

We have found that, in the integer case, we should take M = cW (c the integer coefficient, W a monomial). W is chosen to be a common multiple of LM(gi) and LM(gj) where LM(f) is the leading monomial of f. The coefficient c must be chosen to be the smallest (in the integer ordering) so that division by LC(gi) and LC(gj) give non-trivial quotients (i.e. division gives a genuinely smaller remainder in both cases).

It is not clear, at the moment, whether or not we must allow even more possibilities for M. In some examples I have found that it is necessary to allow trivial matchs (where W is the product of LM(gi) and LM(gj)). This situation does not occur in the Mora algorithm (where coefficients are in a field).

Example: Let $G = \{g_1,g_2\}$ with $g_1 = 2x-a$ and $g_2 = 2y-b$
We know that $f = xb - ay$ is in the ideal: $f = g_1y - xg_2$
However, if we do not allow trivial overlaps, the basis algorithm gives G – and f does not reduce to 0.

If we allow trivial overlaps we get the basis {2x-a, 2y-b, xa+ax-aa, xb+ay-ab, ya+bx-ba, yb+by-bb} f does reduce to 0 with respect to this basis.

When viewed in terms of rewrite rules (see [Wav]) the condition that any M provides the same P_1 and P_2 is equivalent to the Gröbner condition on the basis. The issue, however, is that we are not using all possible M. Thus something remains to be proved if we wish to claim that we get a Gröbner basis in the case of termination. As in the case of coefficients in a field (Mora's Algorithm) the process may fail to terminate:

Example: Let $G = \{g_1,g_2\}$ with $g_1 = xy-ax$ and $g_2 = yx-by$
The algorithm produces two infinite sequences (alternating).
One is $-xb^ky + ax^{k+1}$ the other is $-ya^kx + by^{k+1}$.

Relations in Rings

Let R be a ring. R has an addition and multiplication satisfying the usual algebraic laws – but we do not assume that multiplication is commutative and we do not assume that R has a 1.

Example: A ring which often appears as a counterexample in the later work is the ring T of all strictly upper triangular 3 x 3 matrices:

$$\begin{pmatrix} 0 & a & b \\ 0 & 0 & c \\ 0 & 0 & 0 \end{pmatrix}$$

Notice that $\begin{pmatrix} 0 & a & b \\ 0 & 0 & c \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0 & d & e \\ 0 & 0 & f \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & af \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

Multiplication in T is not commutative, there is no 1, and if x,y,z are in T then xyz = 0. In particular $x^3 = 0$ for all x.

A <u>polynomial identity</u> on R is a polynomial $f \in \mathbb{Z}\langle x_1,..,x_k\rangle$ so that $f(a_1,..a_k)=0$ $\forall\ a_1,..a_k \in R$

Examples:

- xy – yx is a polynomial identity in any commutative ring.
- xyz is a polynomial identity in the ring T.

We imagine R to be a given ring and we look at those polynomial identities on R which lie in $\mathbb{Z}\langle x_1,..,x_k\rangle$. These are closed under (1) addition; (2) multiplication by arbitrary polynomials; (3) substitution of polynomials for the variables in an identity. We call a polynomial a <u>consequence</u> of a given set of identities if it can be obtained from the set by a finite sequence of these operations.

> The problem is to determine (computationally) if a polynomial identity is a consequence of a given set of identities.

Notice that if we ignore the operation of substitution for the variables, this is essentially the ideal membership problem. We expect this problem to be undecidable because the ideal membership problem is undecidable.

It is essential that we work over $\mathbb{Z}$ rather than $\mathbb{Q}$ To say that 2xy – 2yx is an identity on a ring is not the same as to say xy – yx is an identity. This justifies the development of algorithms for polynomials with integer coefficients.

Examples: Here are some consequences of $x^2 - x$

(1) xx - x
(2) yy - y
(3) yy + yx + xy + xx - y - x
(4) 4xx - 2x
(5) yx + xy + xx - x
(6) yx + xy
(7) 4xx - 4x
(8) 2x
(9) -2xy
(10) yx – xy

Notice that yx – xy is a consequence of $x^2 - x$ so we have a computational proof of the fact that a ring satisfying $x^2 - x$ is abelian.

A commutativity theorem asserts that, under some hypotheses on a ring R, R must be commutative. There are a number of theorems of this type in the literature. They provide a good testing ground for algorithms intended to deal with relations on rings. The hypotheses are usually that R satisfies a certain polynomial identity – and perhaps also some additional conditions (e.g. R has a 1, R has no non-zero nilpotent elements). Here are some examples (the hypotheses are given, the conclusion is R is commutative).

Theorem 1: $a^2 = a$
Theorem 2: R is a ring with 1 for which $(ab)^2 = a^2b^2$
Theorem 3: R is a ring with no nilpotents for which $(ab)^2 = a^2b^2$
Theorem 4: R has 1, 2x = 0 implies x=0. $(ab)^2 = (ba)^2$
Theorem 5: R no nilpotents and $(ab)^2 = (ba)^2$
Theorem 6: $a^3 = a$
Theorem 7: $a^4 = a$
Theorem 8: Let R be a ring which $(a^2 - a)$ is in the center
Theorem 9: Let R be a ring which $(a^3 – a)$ is in the center

Some of these theorems are special cases of more general theorems. A Theorem of Jacobson asserts that if R is a ring so that $\forall\ a \in R\ \exists$ integer $n(a)>1$ with $a^{n(a)}=a$, then R is commutative. The proof of this theorem given in the literature uses the structure theory for non-commutative rings. It is not a constructive proof, to say nothing of computational.

We have discussed how we have extended the Buchberger and Mora algorithms to non-commuting variables and integer coefficients. As noted, we do not yet have a proof of exactly what conditions guarantee that we obtain a Gröbner basis – so we will provisionally call the bases produced R-bases (for reduction bases). If G is an R-basis and $f \rightarrow 0$ (reduction by G) then we can assert $f \in \langle G \rangle$. This is how we use the R-bases in proofs of the commutativity theorems.

Starting Relations

The process we use to prove these theorems is to generate an R-basis from an initial set of relations and use it to test ideal membership (some modification is needed if there are side conditions). The initial relations are obtained by making substitution of variables in the hypothesis relations. The process of choosing suitable starting relations has not (yet) been automated. It is a very critical step both in the success

of a proof and in performance. At the moment there seems to be no obvious way to do this automatically.

Extended Example

We examine the proof that commutativity is a consequence of $x^3 - x$. This is one of the simpler theorems for machine proof. It is of moderate difficulty for proof by hand.

The simplest starting point is to make substitutions y and (x+y) for x. This gives consequences: xxx – x, yyy-y, - yyx - yxy - yxx - xyy - xyx – xxy, - yxyxx + xyxyx - xxyxy - xxyxx + yxy + xyx
xy – yx is not in the ideal.

If we add a substitution of x-y for x to those above, we get consequences:
xxx – x, yyy-y, - yyx - yxy - yxx - xyy - xyx – xxy, - 6xy,
2yx - 2xy, - yxyxx - xyxyx - xxyxy - xxyxx - yxy - 2xyy - xyx + 2xxy
This is interesting because we obtain some simpler consequences and one that looks close to the goal.

If we add xx + x and yy + y then we obtain 10 consequences including those above and 3xx + x, 3yy + y which seem simple and interesting.

If we make the previous substitutions and also replace x by xy and by yx then we get a list of 6 consequences: xxx – x, yyy – y, 3xx + 3x, 3yy + 3y, -6xy, and (at last) xy – yx. On a 450 Mhz computer the proof took .6 seconds. There were 115 critical pairs generated, but 110 of these reduced to 0. A total of 536 reduction steps were performed.

I should mention that an even quicker proof (.3 sec) can be produced with a less obvious choice of starting relations. Substitute for x the following: y, xy, yx, xxyx - yx, yyxy – xy, xyxx – xy, and yxyy – yx. We obtain the consequences xxx-x, yyy-y, and xy-yx.

Tracing

It is possible to trace an automated proof to see how new relations arise. In all but simple cases, however, the automated proof is too complicated to produce a simple "hand" proof. $X^3 - x$ is simple enough.

The starting relations we will use are $a^3 - a$ for a = x, y, xy, yx, xxyx - yx, yyxy - xy, xyxx - xy, yxyy – yx. The first step in the automated proof is "interreduction": we reduce each by all the others. The last four polynomials reduce to themselves.

F[4] - xxy F[0] yxxxyx + y F[0] yxxxyx + xxy F[0] yxyx -
y F[0] yxyx = - F[4]

Where $F[0]=x^3 - x$, $F[4]=(xxyx - yx)^3-(xxyx - yx)$. (We are saying that F[4] $\rightarrow$ -F[4] but have explicitly listed the steps in the reduction). The next step in the automated proof involves taking pairs of existing basis elements and forming appropriate common multiples of the

leading terms. We use the common multiple -1 xxyxx for –xyxx + xy and –xxyx + yx. This gives a new basis element yxx-xxy. This new relation is interesting. We have shown that in a ring with $x^3=x$, squares of elements are in the center.

The desired relation yx-xy comes from use of common multiple -yxyyxx for yxx-xxy and –yxyy+yx.

There were 134 critical pairs generated in this proof and 189 reduction steps. The above proof extracts just 2 of the critical pairs. We have also suppressed a list of the reduction steps needed. A complete "hand" proof would have to include these details.

Let me emphasize again that one does not normally trace an automated proof to extract a "hand" proof. The proof for x^4-x, for example, generates 2014 critical pairs and uses 47096 reductions. It takes only 2 minutes and would be considered of moderate difficulty for automated proofs.

Observations

A special case of the x^3-x theorem is used as a test case for the well known Otter automated theorem proving program produced by Argonne National Laboratories. Otter produces an automated proof that x^3-x implies 3xy+3yx = 0. This result takes about .5 sec to produce. Otter ran out of memory when an effort was made to prove xy-yx. Otter is a general and extensive proving program designed for "first order logic with equality". It is really not designed for theorem proving in algebra (although the special case of the x^3-x theorem is supplied with it an example. Our algorithm is designed for a specific area in algebra. It is interesting to note that it is competitive with other automated theorem proving programs and, in fact, performs well on this class of problems.

Side Conditions

Some of the classical commutativity theorems make use of side conditions in addition to the assumed polynomial identities. Here are some examples showing how such conditions can be handled.

$1 \in R$

Theorem 2: If R is a ring with 1 for which $(ab)^2 = a^2b^2$ then R is commutative.

This is one of the very simplest of commutativity theorems. It can be done by hand. The approach is to use 1 when making substitutions for the starting identities. Let $F(a,b) = (ab)^2 - a^2b^2$. The starting relations are F(x,y), F(x+1,y), F(x,y+1), and F(x+1,y+1). In this case no critical pairs need to be generated at all. The relation xy-yx is a result of interreducing the starting basis.

R has no nilpotents

Theorem 3: If R is a ring with no nilpotents for which $(ab)^2 = a^2b^2$ then R is commutative

Notice that the identity is the same as for the previous example. The side condition is different. We again Let $F(a,b) = (ab)^2 - a^2b^2$. We use the initial relations F(x,y), F(y+x,x), F(y+x,y), F(x,y+x), F(y,y+x).

In this case the first interreduction gives xyxy-xxyy, yyxx-yxyx, yyxy-yxyy, xyxx-xxyx. The basis algorithm examines 16 critical pairs, but all reduce to zero – thus these 4 polynomials are the R-basis. Notice that xy-yx is not in the basis, nor does it reduce to zero. In fact the ring T of strictly upper triangular 3 x 3 matrices does satisfy the identity $(ab)^2 - a^2b^2$ but is not commutative.

We can see, however, that $(xy-yx)^3 \rightarrow 0$ so that $(xy-yx)^3 = 0$ $\forall$ x,y $\in$ R. If R has no nilpotents then xy-yx is an identity.

Strategies for Initial Bases

1. Kitchen sink

It should not be assumed that one will achieve good results by adding every initial substitution that comes to mind. One can understand, on the basis of the summary we have just given, that a long list of initial relations can add considerably to the processing time. Each pair of basis elements must be examined and the results added to the list – whether or not they provide relations that actually are useful to the goal.

2. Adding one at a time

It is usually not helpful to add relations one at a time. In some cases the basis generated by a subset of relations is either infinite or time consuming to compute. It can take far more time to find a basis for a subset of relations than to find a basis for the entire set. This occurs because, in these cases, there are relations missing which would play an important role in simplifying the expressions that arise.

Example:

In Theorem 3, discussed above, We let $F(a,b) = (ab)^2 - a^2b^2$. The relations F(x,y) and F(y+x,y) are the first two polynomials used in a starting basis. These two polynomials taken alone generate an infinite basis. The critical pairs produce polynomials that would be reduced if we had added the other relations.

The choice of starting basis (or starting substitutions) is critical both in obtaining a result and in the performance of the algorithm. The fact that the starting basis is obtained by making substitutions for the variables in a given relation means that any element in the R-basis is a consequence of the given relation. Any polynomial, f, which reduces to zero using the R-basis is in the ideal generated by the R-basis, therefore a consequence of the given relation. If the process fails, however, it might be that the substitutions we tried are insufficient.

Example:

I cannot locate the file used for an early successful attempt for x^4-x. It took over 24 hours of computation to produce an R-basis. Eventually we used (see [ISSAC]) the following basis: First we took 2x, 2y, y^3x-xy^3, yx^3-x^3y (these were shown to be consequences of x^4-x as by an independent lemma). Then we added the polynomials obtained by substituting for a in a^4-a: x, y, y+x, yx+x. xy+y, yx+y, xy+x, xy, yx, xy-yx. This produces an R-basis in 129 seconds which contains xy-yx (actually –yx-xy). There were 1870 critical pairs generated, 1846 reduced to zero. 44185 reduction steps were needed. When proper subsets of this basis are used, the algorithm does not appear to terminate. [In general it is difficult to prove that the algorithm will not terminate. The situation which occurs in the example of infinite bases above (an infinite sequence can be shown to occur) is rare. In most cases the algorithm just runs for a long time and is manually aborted.]

Since $x = x^4 = (-x)^4 = -x$ in this ring, we could use Mora's algorithm over the field $\mathbb{Z}_2$. It as been proved that the Mora algorithm (over a field) does produce a Gröbner basis when it terminates. Here we obtain a Gröbner basis in 29 seconds. 792 critical pairs are tested and 759 reduce to zero. 14617 reductions steps were needed. In this test and those with subsets of the initial basis, the R-basis obtained by our algorithm and the basis obtained by Mora's algorithm over the field $\mathbb{Z}_2$ are the same – giving support to our suspicion that an R-basis is in fact a Gröbner basis.

A yet faster method uses a collection of relations known to be true of x^n-x for any n. Among them is $x^{n-1}y - yx^{n-1}$ which can be proved computationally. We then applied an operation which substitutes xy for y in every element of the current basis. It reduces the result and appends it to the basis if not zero. Finally we added $(x+y)^4$-(x+y) A basis containing xy-yx is obtained in 1.4 seconds.

Behavior of the Algorithm

Even the commutative algorithm (the Buchberger algorithm) is known to be badly behaved in some cases. Algorithms of this sort, in which new elements are added to a basis based on a complex system state, are very difficult to analyze. Cox, Little and O'Shea [IVA] give the following example: A Gröbner basis for

$$g_1 = x^5+y^4+z^3-1 , \ g_2 = x^3+y^2+z^2-1$$

Can be readily computed using either pure lexicographic order (called plex in Maple) or graded reverse lexicographic order (called tdeg in Maple). If, however, a single exponent is changed:

$$g_1 = x^5+y^4+z^3-1 , \ g_2 = x^3+y^3+z^2-1$$

The basis computed using plex has a basis element of total degree 25 with 282 terms and a largest coefficient of 167383594. I should note that I was unable to compute the basis using the algorithm in Maple – but it can be computed using either a larger computer or a more sophisticated implementation of the Buchberger

algorithm (I was able to compute it using Singular). The Buchberger algorithm has proven to be useful – but it is known that it can exhibit very bad behavior.
A problem of many algorithms in computer algebra is "intermediate expression swell". This is the name given for a situation in which the input and output are small or moderate in size, but data in the middle of the algorithm is very large. In the case of integer computations, an algorithm which starts with small integers and gives a result with small integers may require very large multi-precision computation.

Many examples of this situation have arisen in the R-basis algorithm we have discussed. The example mentioned above for x^4-x has

Starting relations		Ending basis
2x	F(yx+x)	+ 2x
2y	F(xy+y)	+ 2y
y^3x-xy^3	F(yx+y)	- yyyy - y
yx^3-x^3y	F(xy+x)	- yx - xy
F(x)	F(xy)	- xxxx - x
F(y)	F(yx)	
F(y+x)	F(xy-yx)	

This is a relatively short computation. In the midst of the computation the basis grows to 29 polynomials. Many of them have 20-30 terms. The largest intermediate polynomial which occurs has 50 terms and a total degree 8. This is by no means the worst behavior I have seen in this work. In some cases the algorithm makes use of consequences of the hypothesis relation which have several hundred terms.

Design Considerations

When to add a new element
Here is pseudo-code for the basic algorithm as it was when [ISSAC] was written (it has since been modified). For terminology and previous procedures see [ISSAC].

Procedure 2: R-Basis Algorithm (see [ISSAC])
Input: A finite set, G, of polynomials
Output: (If the algorithm terminates)
an R-Basis for the ideal generated by G.

```
H := G
WHILE not Empty(H) DO
    B := { (f1,f2,l1,r1,l2,r2) | f1∈ G, f2 ∈ H;
                (l1,r1,l2,r2) a match for (LM(f1),LM(f2)) }
    H := ∅
    WHILE not Empty(B) DO
        select m ∈B; B := B - {m}
        (p1,p2) := CritPair(m)
        p1 := NForm(p1,G∪H); p2 := NForm(p2, G∪H)
        f := p1 - p2
        IF f ≠ 0 THEN H := H∪{f}
    G := G∪H
G := Interreduce(G)
```

We have discussed the way in which potential new basis elements are obtained using pairs of elements of the existing basis. This algorithm is designed so that all pairs to be processed (together with information needed to process them) are stored in a list B. Pairs are selected from this list and processed. If a new (non-zero) element, f in the pseudo code above, is obtained notice that it is not immediately added to the basis G. It is stored in H, and $G \cup H$ is used in reductions. So the new element is used for reductions, but not for forming new pairs until all pending pairs (in the B list) are processed. Notice that after the B list is exhausted we do adjoin G to H. The new pairs, however, combine elements of the new G only with elements of H (since pairings between elements of the old G have already been processed).

Example:

We have mentioned that an infinite basis is obtained using g_1=xy-ax and g_2=yx-by. There are two common multiples of the leading terms which produce new basis elements - yax + byy and - xby + axx respectively. If the first were to be added to the basis immediately then it would pair with g_1 to produce - yaax + byyy, and so on. The result is that only the family $-ya^kx + by^{k+1}$ is produced and not any member of the family $-xb^ky + ax^{k+1}$.

Interreduction

It is common to want a reduced Gröbner basis. Reduced Gröbner bases are unique, usually shorter, and have smaller elements. Some implementers of Buchberger's algorithm make interreduction an option. It is often a separate command.

In the implementation used in the ISSAC99 paper, a basis was generated first and the algorithm concludes with interreduction. Each time a potential new basis element is produced (from critical pairs) it is reduced by all existing basis elements. However existing basis elements are not (in the algorithm above) reduced by the new basis element. This has the effect of tacking new basis elements on the end of the list, but making no attempt to reduce the size of the list at this stage.

Example:

For x^3-x we try the starting basis F(x), F(y), F(x+y), F(x-y), F(x+xx), F(y+yy) with F(a)=a^3-a. Using the algorithm above, without the interreduction step, we get

GB[0] = 1 xxx -1 x
GB[1] = 1 yyy -1 y
GB[2] = 1 yyx 1 yxy -1 yxx 1 xyy -1 xyx -1 xxy
GB[3] = 1 yxx -2 xyx 1 xxy 3 yx 3 xy
GB[4] = 3 xx 3 x
GB[5] = 3 yy 3 y
GB[6] = 3 yxy -9 xyx 6 yx 6 xy
GB[7] = -18 xyx 8 yx 10 xy
GB[8] = 6 xyx -6 yx -6 xy
GB[9] = -4 yx 4 xy
GB[10] = 1 xxyx 1 xyx -1 xxy -1 xy
GB[11] = -2 yx 44 xy
GB[12] = 1 xyxyx 1 yxyx 1 xyxy 1 xxyy -1 xyy -1 xyx
1 xxy -45 xy
GB[13] = -1 yxyx 1 xxyy -1 yxy 1 xyy -108 xy
GB[14] = 1 yxy -1 xyy 1 xyx -1 xxy 138 xy
GB[15] = -18 xy

GB[16] = 12 xy
GB[17] = 6 xy

If there is no interreduction, new basis elements are tacked on to the end of the list. Notice that there are some elements which occur later in the list which would simplify (or eliminate) earlier elements.

Now we apply interreduction: every polynomial is repeatedly reduced by the others. This gives
GB[0] = 1 xxx -1 x
GB[1] = 1 yyy -1 y
GB[2] = -1 yyx 1 xyy -1 yx 1 xy
GB[3] = -1 yxx 1 xxy -1 yx 1 xy
GB[4] = 3 xx 3 x
GB[5] = 3 yy 3 y
GB[6] = 0
GB[7] = 0
GB[8] = 0
GB[9] = 0
GB[10] = -1 xxyx -1 xyx 1 xxy 1 xy
GB[11] = -2 yx 2 xy
GB[12] = 0
GB[13] = 0
GB[14] = -1 yxy 1 xyy -1 xyx 1 xxy
GB[15] = 0
GB[16] = 0
GB[17] = 6 xy

I have used here a form of interreduction which does not remove the basis elements which have been reduced to zero. The numbering, therefore, corresponds to the numbering of the original basis. In practice, however, we do remove polynomials which become zero and renumber the rest:
GB[0] = 1 xxx -1 x
GB[1] = 1 yyy -1 y
GB[2] = -1 yyx 1 xyy -1 yx 1 xy
GB[3] = -1 yxx 1 xxy -1 yx 1 xy
GB[4] = 3 xx 3 x
GB[5] = 3 yy 3 y
GB[6] = 6 xy
GB[7] = -1 yxy 1 xyy -1 xyx 1 xxy
GB[8] = -2 yx 2 xy
GB[9] = -1 xxyx -1 xyx 1 xxy 1 xy

Discerning good starting relations
The R-basis in this case does not prove x^3-x. The xy-yx is not in the ideal generated by the starting relations. However, this is instructive. It suggests what needs to be (or could be) added to prove the theorem. If we know that squares are in the center, either GB[2] or GB[3] would reduce to xy-yx. We do have a computational proof that squares are in the center, so we can enlarge the original starting basis to obtain one which proves the x^3-x theorem.

Interreduction clearly has the effect of simplifying a basis. Rather than wait until the end, it is possible to revise the algorithm so that interreduction occurs after each new basis element is added. Notice that interreduction potentially changes all the existing basis elements (and possibly the numbering). Thus this has three disadvantages: (1) the algorithm becomes much harder to trace (say for the purpose of extracting a "hand" proof); (2) the optimization used with the B-list (where one only examines pairs consisting of an existing element with a new element) will not work; (3) we miss seeing some interesting consequences of the starting relations.

The idea of trying interreduction after each new basis element arose when some examples produced unreduced bases which suffer from severe intermediate expression swell: polynomials with hundreds of terms having enormous integer coefficients. A modified form of the algorithm, with the following pseudo code, was developed.

Procedure 2: R-Basis Algorithm (modified version)
Input: A finite set, G, of polynomials
Output: (If the algorithm terminates)
an R-Basis for the ideal generated by G.

```
Newpoly := true
WHILE Newpoly DO
        Newpoly := false
        G := Interreduce(G)
        WHILE not Newpoly DO
                B := { (f1,f2,M) | f1, f2 ∈ G;  M a distinguished
                                common multiple for (LM(f1),LM(f2)) }
                WHILE not Empty(B) and not Newpoly DO
                        select m ∈ B; B := B - {m}
                        (p1,p2) := CritPair(m)
                        p1 := NForm(p1,G); p2 := NForm(p2, G)
                        f := p1 - p2
                        IF f ≠ 0 THEN G := G∪{f};
                                        Newpoly := true
```

In this version a Boolean variable Newpoly is set if a new basis element is added. The inner loops are skipped when this happens and the procedure is restarted. Since the procedure is restarted when a new basis element is added, the mechanism involving H is no longer needed. As it is now written, the inner loops exit before processing pending matches in B. When the basis is infinite, as in the work I did with linear systems [HWS], it is necessary for correct results to process all pending matches before adding new elements to the basis. If the basis is finite, short-circuiting the processing of pending matchs should not affect the outcome (it has a dramatic effect on time).

The inner loops exit when a new basis element is found. The procedure as a whole exits when a sweep through the inner loops does not produce any new polynomials.

Optimization
Interreduction seems, on the face of it, an expensive operation. Every element of the current basis has to be reduced by the others. The reduction process is an expensive operation. Some implementations of the Buchberger algorithm in the commutative case (like Singular [Sing]) return an unreduced basis. The authors presumably regard interreduction as a step which would slow down the algorithm.

However, the cost of interreduction is often balanced by the fact that the basis is smaller both in the number of elements and the size of individual basis elements. For our work, interreduction after each basis element has been a major optimization.

Example:

For the theorem that if x^3-x is central then R is commutative we start with a large set of relations. Let F(a,b) = (a^3-a)b – b(a^3-a). The starting basis has 21 elements starting with F(x,y), F(y,x), …
The older version of the algorithm yields an (unreduced) basis of 85 elements. It is estimated that the proof, if allowed to complete, would take about 100 hours. The newer version (with interreduction) produces a basis with 9 elements in 11 seconds. [Actually the proof is not complete at this point. We further reduce F(x^2-y^2,x) - F(x^2-y^2,y) which, in 123 reduction steps, reduces to xy-yx.]

An important part of the project is developing ways to study algorithms. The drastic difference in execution speed has been traced to the final step in which we check all pairs of basis elements to find if there are any new critical pairs which fail to reduce to zero (hence produce new basis elements). If the basis has 9 fairly simple elements this process is quick. If the basis has 85 fairly large polynomials, the process is quite time consuming. In fact the 85 basis elements are produced reasonably quickly (about 2 mins). They reduce to the same basis of 9 elements obtained by the new version.

A very effective optimization in the commutative case is due to Buchberger [Buch2]. Since reduction to an irreducible form is a very expensive process, Buchberger provides a criterion on two polynomials f,g which predicts (in some cases) that Spol(f,g) will reduce to zero without actually reducing it. Experiments show that the criterion eliminates a high percentage of unnecessary reductions and produces a significant gain in execution speed.

Example:

In a simple test of the Buchberger (commutative) algorithm there were 22 S-polynomials generated. Of these 6 failed the Buchberger criterion (the criterion predicted that they would reduce to zero). Of the 16 that did pass the Buchberger criterion, 6 did yield new basis elements and 10 did not (the S-poly reduced to zero). The criterion successfully screened out 6 but did not catch 10. With the Buchberger criterion the algorithm took 5.9 seconds. Without the Buchberger criterion it took 9.5 seconds.

In the non-commutative case the Buchberger criterion does not apply. An analogous criterion was proposed in a paper I refereed, but it was found not to apply very often. At the moment there is no efficient way to detect unnecessary reduction even in the case of a field (Mora algorithm).

There are some "rules of thumb" for a modest increase in speed given by Buchberger in the commutative case. These have to do with the order in which basis elements are used to test for divisibility and the order in which elements should be extracted from the B-list. I have experimented with some traditional ways to speed up the Grobner algorithm. In the literature for the commutative case it is said that the polynomials in the basis should be ordered so that the division algorithm tests for

divisors starting with polynomials with the smallest leading terms. In my algorithm I use the reverse – trying division first by basis elements which have the highest leading term. I have tested both ways. They seem close in execution speed. My choice seems marginally faster on the average.

The other optimization is that one should order the S-polynomials in the list of those waiting to be tested (in Buchberger's B-list) so that (in the commutative case) the pair with the smallest LCM is used first. I too use a B-list so that we do not introduce pairs with a newly generated polynomial before waiting pairs are processed. In this case my results confirm that you should start by processing pairs with the smallest common multiple of leading terms.

In the example given above (the case in which x^3-x is central) the 85 elements in the unreduced basis were generated fairly quickly. This is true in this case because new basis elements were produced fairly quickly (not too many pairs had to be tried). One way to optimize would be to provide some heuristics for pairs most likely to yield a new basis element and to order the basis so that these pairs are selected first. I have done some analyses of the details of the algorithm to see if certain basis elements are used more often than others in yielding new elements.

Other Applications

We have discussed the usefulness of the algorithm to study relations in rings. There are other potential applications.

Finitely Presented Groups

The algorithm provides a viable alternative to coset enumeration for presentations of finite groups.

Example:

For the dihedral group of order 10 we take generators x, y. The relations are basis elements: x^5-1, y^2-1, 1 yx-x^4y. The algorithm produces a Gröbner basis with 6 elements: yxy-x^4, y^2-1, yx^2-x^3y, xyx-y, yx^3-x^2y, yx^2y-x^3. Remainders on division give a complete set of representatives for the congruence classes of this ideal:

1 x y xx yx xy xxx yxx xxy yxy

The multiplication table can be obtained by concatenating two words and then reducing using the Gröbner basis.

1	x	y	xx	yx	xy	xxx	yxx	xxy	yxy
x	xx	xy	xxx	y	xxy	yxy	yx	yxx	1
y	yx	1	yxx	x	yxy	xxy	xx	xxx	xy
xx	xxx	xxy	yxy	xy	yxx	1	y	yx	x
yx	yxx	yxy	xxy	1	xxx	xy	x	xx	y
xy	y	x	yx	xx	1	yxx	xxx	yxy	xxy
xxx	yxy	yxx	1	xxy	yx	x	xy	y	xx
yxx	xxy	xxx	xy	yxy	xx	y	1	x	yx
xxy	xy	xx	y	xxx	x	yx	yxy	1	yxx
yxy	1	yx	x	yxx	y	xx	xxy	xy	xxx

Quotients of Free Algebra

Let $R = \mathbb{Z}\langle x_1,..,x_n\rangle/I$ where I is a finitely generated ideal. If the algorithm produces a finite Gröbner basis for I, we can do computation in R using the division theorem. The flavor is like that of finitely presented groups discussed above.

Progress since ISSAC99

The work on optimization is new. The proof of one of the commutativity theorems took two days in the earlier version. It now takes 10 seconds. Improvements have been made in methods to study the progress of the algorithm (this led to the new optimization results). A study has been pursued of strategies of choosing and ordering the initial bases. The algorithm has proved to be very sensitive to the choice of initial basis. It is hoped that this study will lead to some heuristics to guide the selection of the initial basis. A module was created for applications to finitely presented groups.

Research Goals

The method we have applied in the study of relations on rings is in two stages: choice of an initial basis (substitutions) and then application of the basis algorithm. The choice of initial substitutions has been found to have great bearing on the performance and even the success of the basis algorithm. One goal is to study the selection of the initial basis. The algorithm discussed here has been applied to problems which are equivalent to the word problem. The word problem is known to be undecidable, so it is impossible to find an algorithm which works universally. We have given a number of instances where our algorithm is successful. A second research goal is to identify the situations in which the algorithm yields a Gröbner basis. (For example, Mora's algorithm works whenever it terminates. Is this true in the integer case as well?). A third goal is to continue improving the basis algorithm.

References

[Buch] B. Buchberger. Groebner bases: an algorithmic method in polynomial ideal theory in Multidimensional Systems Theory ed. N.K. Bose D. Reidel Publishing Company, Dordrecht (1985) 184-232.

[Buch2] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner-bases. Symbolic and algebraic computation (EUROSAM '79, Internat. Sympos., Marseille, 1979), pp. 3--21, Lecture Notes in Comput. Sci., 72, Springer, Berlin-New York, 1979

[Buch3] B. Buchberger. A critical-pair/completion algorithm for finitely generated ideals in rings. *Logic and machines: decision problems and complexity (Münster, 1983)*, 137--161, Lecture Notes in Computer Science 171, Springer, Berlin (1984).

[IVA] D. Cox, J. Little, and D. O'Shea. Ideals, Varieties, and Algorithms. 2nd Ed. Springer-Verlag, New York, 1996.

[HWS] J. W. Helton, M. Stankus and J.J. Wavrik: "Computer Simplification of Formulas in Linear Systems Theory" IEEE Transactions on Automatic Control , 43, no. 3 (1998) , 302-314.

[Mora] F. Mora, "Groebner Bases for Non-commutative Polynomial Rings" *Lecture Notes in Computer Science*, number 229 (1986) 353–362.

[Sing] Singular: A Computer Algebra System for Polynomial Computations
http://www.singular.uni-kl.de/

[Wav] J. J. Wavrik: "Rewrite Rules and Simplification of Matrix Expressions"
Computer Science Journal of Moldova,
4:3 (1996), 360-398.

[ISSAC] J. J. Wavrik: "Commutativity Theorems: Examples In Search of
Algorithms", In Proceedings of 1999 International Symposium on
Symbolic and Algebraic Computation,(1999), 31-36.