Gröbner Bases in Mathematica 3.0

Daniel Lichtblau

Wolfram Research, Inc.
100 Trade Centre Dr.
Champaign IL 61820
danl@wolfram.com

In: The *Mathematica* Journal **6**(4): 81−88. 1996.

Gröbner bases are used heavily throughout computational mathematics. This article demonstrates new functionality of *Mathematica*'s **GroebnerBasis** command and describes two new related functions. As examples, we show how **GroebnerBasis** can be used to rationalize denominators, and how it can be extended from polynomial ideals to modules.

This paper is dedicated to the memory of my friend and colleague Jerry Keiper.

Introduction

Since their invention by Bruno Buchberger more than thirty years ago, Gröbner bases have become a pervasive tool in computational mathematics. This is no surprise, as a Gröbner basis generalizes to an arbitrary set of polynomials the notion of triangularization of a set of linear polynomials. (See [Adams and Loustaunau 1994; Becker and Weispfenning 1993; Buchberger 1982; Cox, Little, and O'Shea 1992] for definitions, theory, and numerous applications.)

Mathematica's **GroebnerBasis** function has been entirely rewritten for Version 3.0. In previous versions, **GroebnerBasis** took no options. Now, one can now control a host of important parameters. The function is also much faster than before. This article explains the new features of **GroebnerBasis** and introduces the related new functions **PolynomialReduce** and **MonomialList**. We elaborate on some of the finer points that one would not expect to find in [Wolfram 1996].

Basic definitions

A monomial is a product of powers of variables, such as $x^2 y^3 z$. We will also allow coefficients in the product, such as $3 x y^4$. For purposes of divisibility, coefficients do not matter (we assume they lie in some field). One monomial is said to divide another if the powers of its variables are no greater than the corresponding powers in the second monomial. For example, $x^2 y$ divides $x^3 y$, but does not divide $x y^3$. If the powers of the variables are all zero, the monomial is a constant.

A term order on a set of monomials is a total ordering such that constants are lower than any monomials involving variables, and if r, s and t are monomials with s greater in the ordering than r, then $t s$ is greater than $t r$. Another way to view this is via the exponent vectors. To each monomial there corresponds an n−tuple of exponents, n being the number of variables. We order these so that every such tuple containing a nonzero entry is greater than (0, 0, ..., 0). This enforces the requirement that variables are larger than constants. We furthermore require that if $(a_1, ..., a_n)$ is greater than $(b_1, ..., b_n)$, then $(a_1 + c_1, ..., a_n + c_n)$ is greater than $(b_1 + c_1, ..., b_n + c_n)$, enforcing the multiplicative requirement above.

A monomial m is said to reduce with respect to a polynomial if the leading monomial of that polynomial divides m. For example, $x^2 y$ reduces with respect to $2 y x + x + 3$, because $x y$ divides $x^2 y$. The result of this reduction is $x^2 y - \frac{1}{2} x (2 y x + x + 3)$, or $-\frac{x^2}{2} - \frac{3 x}{2}$. A polynomial is reduced by reducing its monomials, beginning with the greatest and proceeding downward. One reduces a polynomial with respect to a set of polynomials by reducing in turn with respect to each element in that set. A polynomial is fully reduced once none of its monomials can be reduced.

A set of polynomials generates an ideal. This is the set of all sums of products of polynomials in the original set with arbitrary polynomials. Given a set of polynomials, and a term order on monomials, a Gröbner basis for that set, with respect to the term order, can be defined formally as follows. It is a finite set of polynomials that generates the same ideal, such that for any polynomial poly in the ideal there is a polynomial in the basis whose leading monomial divides the leading monomial of poly. For example take the polynomial ideal generated by $\{y x^2 + y, x y^2 + x\}$, and order terms so that any monomial containing y is greater than all monomials in x alone (that is, order lexicographically with y greater than x). It is not a Gröbner basis because the ideal in question contains $y (y x^2 + y) - x (x y^2 + x) = y^2 - x^2$, and neither $x^2 y$ nor $x y^2$ divides y^2, the lead monomial. In fact, a bit of thought shows that this cannot be a Gröbner basis with respect to any term order, not just the one used above.

Fully reducing a polynomial by the members of a Gröbner basis produces a result that cannot be reduced by any element in the ideal (not just in the basis). The importance of this property cannot be overstated: questions regarding the polynomial ideal, an infinite set, can be reduced to problems involving a finite Gröbner basis for the ideal. One consequence is that reduction with respect to a

Gröbner basis is canonical. In other words, no matter what order one performs reductions, one gets the same result. The problems of ideal membership and canonical form were the original impetus for Buchberger's early work in this field.

Another important application of Gröbner bases is the solution of systems of polynomial equations. A lexicographic term order gives rise to a basis that is triangular in the sense that each polynomial has a leading term involving a variable at least as high as the preceding polynomial. To solve a system of polynomial equations, one finds a Gröbner basis, extracts roots of the first polynomial, back−substitutes the result into the next polynomial, solves for roots, and so on.

The GroebnerBasis Function

The interface, as before, is **GroebnerBasis**[*polys*, *vars*]. If the second argument is omitted, the variables are deduced from the first argument exactly as for **Solve** and related functions.

One can specify the coefficient domain over which to work, e.g. rationals, integers modulo a prime *p*, rational function field in some set of indeterminates. One does this with the **CoefficientDomain** and **Modulus** options. Variable ordering is determined by the order in which variables are listed. Term ordering is then specified by the **MonomialOrder** option e.g. **Lexicographic** (the default) or **DegreeReverseLexicographic**.

The time and memory required to calculate a Gröbner basis depend very much on the variable ordering, monomial ordering, and on which (if any) variables are regarded as invertible (that is, are field elements in a polynomial ring over a rational function field). For example, it is typical for degree reverse lexicographic monomial ordering to be faster and to give simpler output than pure lexico−graphic ordering, other things being the same.

To illustrate, we define a function to show the timing of Gröbner basis calculations, as well as the number of polynomials and the number of terms, the total degree, and the largest coefficient for each polynomial. Note that we store the Gröbner basis in a global variable **$GB** to avoid recomputing it in one of the examples below. The examples shown here were run on a 300 MHz Intel proces−sor, under the Linux operating system.

```
gbstats[polys_, vars_, opts___] :=
 Module[{timing, numberPolys, numberTerms, totalDegrees, w, maxcoeffs},
  timing = First[Timing[$GB = GroebnerBasis[polys, vars, opts]]];
  numberPolys = Length[$GB];
  numberTerms = Length /@ $GB;
  totalDegrees = (Exponent[#1, w] &) /@ ($GB /. Thread[vars → w]);
  maxcoeffs = (N[Max[Abs[List @@ #1 /. Thread[Variables[polys] → 1]]]] &) /@ $GB;
  Print[timing, "    ", numberPolys, " polynomials"];
  TableForm[Transpose[{numberTerms, totalDegrees, maxcoeffs}],
   TableHeadings → {None, {"terms", "total deg", "max coeffs"}}]]
```

Here is a set of polynomials from [Cox, Little, and O'Shea 1992]. We will find several Gröbner bases for this set.

```
polys = {x^5 + y^4 + z^3 - 1, x^3 + y^2 + z^2 - 1};

gbstats[polys, {x, y, z}]
```

```
0.09 Second    7 polynomials
terms    total deg    max coeffs
25       12           30.
49       13           108.
53       12           302.
9        5            2.
49       12           192.
6        4            1.
4        3            1.
```

Suppose we want to find a Gröbner basis for this set, regarded as polynomials in the variables *x* and *y*. One way is to treat *z* as a coefficient, so that our coefficients are the field of rational functions in *z*. We can instead treat *z* as a variable with a lower lexico−graphic order than *x* and *y*, and continue to work in the polynomial ring over all three variables. Strictly speaking, this latter approach is not equivalent to the former, but, under certain hypotheses, it can be shown that treating parameters as variables still gives a Gröbner basis for the ideal regarded as polynomials in *x* and *y*, though possibly with extra polynomials (that is, it is not "minimal"). It suffices, for example, to order the parameter variable last if the monomial ordering is lexicographic.

The first method will slow down the computation of the Gröbner basis because rational function arithmetic must be used, but it gives a simpler result. This trade−off is common in Gröbner computations. It is influenced by several factors, such as relative difficulty of computing integer vs. polynomial greatest common divisors. Here we show the previous example, but with *z* treated as a coefficient parameter rather than a variable.

The first method will slow down the computation of the Gröbner basis because rational function arithmetic must be used, but it gives a simpler result. This trade−off is common in Gröbner computations. It is influenced by several factors, such as relative difficulty of computing integer vs. polynomial greatest common divisors. Here we show the previous example, but with *z* treated as a coefficient parameter rather than a variable.

```
gbstats[polys, {x, y}, CoefficientDomain → RationalFunctions]
```

```
0.24 Second    2 polynomials
```

terms	total deg	max coeffs
13	12	13.
16	10	77.

The choice of term order can have a substantial effect on time of computation and complexity of the result.

```
gbstats[polys, {x, y, z}, MonomialOrder → DegreeReverseLexicographic]
```

```
0.01 Second    4 polynomials
```

terms	total deg	max coeffs
4	3	1.
6	4	1.
9	5	2.
19	6	3.

Changing y^2 to y^3 in the second polynomial makes this a much harder problem for some variable orders.

```
gbstats[{x^5 + y^4 + z^3 - 1, x^3 + y^3 + z^2 - 1}, {x, y, z}]
```

```
2.04 Second    8 polynomials
```

terms	total deg	max coeffs
27	15	30.
274	26	5533.
271	25	1.27272×10^{8}
9	6	2.
275	25	1.72226×10^{8}
282	25	1.70255×10^{8}
6	5	1.
4	3	1.

Finally, note that the even variable order can influence the speed of computation and the complexity of the result. A felicitous choice of variable order below quickly yields a simple basis.

```
gbstats[polys, {y, z, x}]
```

```
0.01 Second    2 polynomials
```

terms	total deg	max coeffs
7	6	2.
4	3	1.

One of the settings for **MonomialOrder** is **EliminationOrder**. Here, we use it to eliminate two of the variables from three polynomials, to get a single polynomial. The second variable list, as in **Solve**, specifies the variables that are to be eliminated. (Lexicographic Gröbner bases are an essential tool for algebraic equation solvers, so the similarity between the arguments of **GroebnerBasis** and **Solve** is no coincidence.)

```
GroebnerBasis[{-a d - a b c x - a b e y, a b c x - e x - 2 d x^2, 2 x^2 - c y - a b e y},
 {a, b}, {x, y}, MonomialOrder → EliminationOrder]
```

$$\{a^2 b^2 c^4 d^2 + 2 a c^2 d^4 + a^3 b^3 c^3 d e - a b c^3 d^2 e + 2 a^3 b^3 c^3 d^2 e + 4 a^2 b c d^4 e - 2 a^2 b^2 c^2 d e^2 + a^4 b^4 c^2 d e^2 - 2 a^2 b^2 c^2 d^2 e^2 + a^4 b^4 c^2 d^2 e^2 + 2 a^3 b^2 d^4 e^2 + a b c d e^3 - 2 a^3 b^3 c d e^3 - a^3 b^3 c d^2 e^3 + a^2 b^2 d e^4\}$$

Gröbner basis calculations over the rationals are notorious for the size of intermediate coefficients. This next example runs much

more slowly in the non−modular case. We will use the Gröbner bases generated later, so we save the global variable **$GB** after each computation below.

```
polys = {-3 p q + 2 a p q - 2 b p q - p^2 q + 2 a p^2 q - 3 r + 2 a r - 2 m r + 2 b q r + r^2,
   -1 + b - m + p - a p + b q - r + a r, 2 - a + 2 b - 2 a p, 2 - 2 b + b^2 + 2 m - 2 b m + 2 p - 3 a p +
    a^2 p + 2 b p - 2 a b p - a p^2 + a^2 p^2 + 2 b^2 q - 2 b r + 2 a b r, -2 + a - 2 m + 2 b q + 2 a r};

vars = {a, b, p, q, r, m};

prim = Prime[1000];

gbstats[polys, vars, Modulus→ prim]
```

```
1.91 Second    7 polynomials
```

terms	total deg	max coeffs
28	10	7795.
118	18	7852.
109	17	7898.
112	17	7821.
3	1	7915.
109	17	7901.
110	17	7879.

```
gbmod = $GB;

gbstats[polys, vars]
```

```
7.75 Second    7 polynomials
```

terms	total deg	max coeffs
28	10	3368.
118	18	3.3837×10^{10}
109	17	4.90541×10^{46}
112	17	2.49682×10^{30}
3	0	1.
110	17	4.53656×10^{44}
110	17	1.12421×10^{45}

```
gbrat = $GB;
```

For a given set of polynomials and term order, almost every prime *p* will be "lucky" in the sense that the Gröbner basis for that set, regarded as polynomials over the integers modulo *p*, will agree (up to multiplication by a unit) with the mod *p* reduction of the Gröbner basis for the same polynomial set taken over the rationals. For the example above, we will see that **Prime[1000]** is a lucky prime.

We define a routine to "normalize" so that leading coefficients are one.

```
lcoeffs[polys_, vars_] :=
  Map[First[MonomialList[#, vars]] &, polys] /. Thread[vars → 1];

modNormalize[polys_, vars_, p_] :=
 Module[{lcs = lcoeffs[polys, vars], mults}, mults = Map[PowerMod[#, -1, p] &, lcs];
  Map[PolynomialMod[#, p] &, mults * polys]]

modNormalize[gbmod, vars, prim] ==
 modNormalize[PolynomialMod[gbrat, prim], vars, prim]
```

```
True
```

Here is a standard example from the Gröbner basis literature. The Gröbner basis computation is slow for pure lexicographic ordering with the variables in the order given.

```
polys = {-36 - 165 b + 45 p + 35 s, 35 p - 27 s + 25 t + 40 z, -165 b^2 + 25 p s - 18 t + 15 w + 30 z,
   15 p t - 9 w + 20 s z, -11 b^3 + p w + 2 t z, 3 b^2 - 11 b s + 99 w};
```

```
vars = {b, t, s, w, p, z};
gbstats[polys, vars]
```

```
7.04 Second    6 polynomials
terms    total deg    max coeffs
11       10           2.89828×10^32
11       9            1.42095×10^153
11       9            8.41165×10^150
11       9            1.04655×10^153
11       9            1.1764×10^154
11       9            6.25164×10^151
```

```
gbrat = $GB;
```

Inexact arithmetic speeds up the process by avoiding huge integers in the course of the computation. Setting the option **CoefficientDomain** to **InexactNumbers** normalizes Gröbner bases over inexact numbers so that the coefficient of the leading monomial of each polynomial is one.

```
gbstats[polys, vars, CoefficientDomain→ InexactNumbers]
```

```
0.31 Second    6 polynomials
terms    total deg    max coeffs
11       10           11.9205
11       9            9.94451
11       9            5.46377
11       9            82.3984
11       9            102.913
11       9            14.7663
```

```
gbreal = $GB;
```

We find the list of corresponding coefficients for the exact case and divide to match the normalization.

```
Chop[N[Expand[gbrat / lcoeffs[gbrat, vars]], 100] - gbreal]
{0, 0, 0, 0, 0, 0}
```

The list of zeros demonstrates that corresponding polynomials in the two bases agree up to multiplication by constants. At present, we numericize to about 100 decimal places and use *Mathematica*'s default bignum arithmetic. In a future version, the initial precision may be set by the caller.

It should be noted that the realm of Gröbner bases over inexact numbers is poorly understood at present. An expected difficulty is that, due to accumulation of rounding or cancellation error, one may fail to recognize a coefficient of zero. Such failure is problematic because one relies on certain (exact) cancellations to take place during the course of the algorithm. It is quite difficult to guard against this error when one works with fixed precision arithmetic. As *Mathematica* instead uses significance arithmetic by default, it is not likely to make this mistake: we detect that a coefficient is zero to all reliable (significant) digits. Our model of arithmetic is instead vulnerable to the opposite error; a nonzero coefficient with scale smaller than its apparent accuracy may be mistaken for zero (because it has no significant digits). Roughly speaking, the expected result of such a mistake is to obtain an algebraic variety "near" to the one that is desired. In a bad case, one might obtain a variety with a smaller zero set. It might even be that one can define an intrinsic "condition number" for an ideal, as is already done in numeric linear algebra. How this might interact with variants of Buchberger's algorithm for Gröbner basis computation is not known. In any case, *Mathematica*'s inexact Gröbnerization is sufficiently robust to get a result that agrees with the exact computation in the example above. I remark that the *Mathematica* implementation of significance arithmetic is an example of the excellent work of the late Jerry Keiper.

Given an ordering of variables, we can use a matrix of weight vectors to specify a term ordering (see [Robbiano 1985]). In *Mathematica*, any matrix of integer or rational weight vectors can be used, provided it has full row rank and defines a valid term order. This capability allows for all the orders of "lexicographic type." (In a future release, we may provide even more general orderings.)

The following weight matrix is an example of a block ordering. That is, a term containing the first two variables is higher than any term in only the last two variables. Such orders can always be specified by a weight matrix that is block diagonal. They generalize lexicographic ordering, wherein every variable comprises a distinct block and we can use the identity matrix as the weight matrix.

```
wt = {{1, 1, 0, 0}, {1, 0, 0, 0}, {0, 0, 1, 1}, {0, 0, 0, 1}};

polys =
  {-(47 a) - 221 a b x - 3 a b y + 11, -x^2 + 66 a b x - 7 x + 14 a b y + 40, 22 x^2 - 138 y - 13 a b y};

GroebnerBasis[polys, {a, b, x, y}, MonomialOrder → wt]

{1452 x^3 + 520 y - 9199 x y + 295 x^2 y - 1932 y^2,
 -41 360 + 7238 x - 210 452 b x + 34 034 b x^2 + 6486 y - 5885 b y + 30 960 b x y + 414 b y^2,
 24 440 - 4277 x + 124 358 b x + 13 865 x^2 - 20 111 b x^2 + 60 839 b x^3 - 90 804 y - 399 648 b x y,
 -124 358 + 40 326 a + 20 111 x - 60 839 x^2 + 399 648 y}
```

As block orders are compatible with elimination, we can eliminate the variables of the highest block, by listing them separately as a third argument.

```
GroebnerBasis[polys, {a, b}, {x, y}, MonomialOrder → wt]

{-2 304 324 + 19 691 496 a - 42 068 196 a^2 - 301 295 676 a b + 1 245 260 112 a^2 b +
  179 856 780 a^3 b + 40 232 539 841 a^2 b^2 - 15 328 247 992 a^3 b^2 - 192 238 225 a^4 b^2 -
  76 517 933 885 a^3 b^3 + 24 572 275 625 a^4 b^3 - 7 565 816 362 a^4 b^4 + 2 453 393 514 a^5 b^4}
```

We obtain a Gröbner basis for the "elimination ideal" of the polynomials, that is, the ideal formed by intersecting the original ideal with the set of all polynomials in only the remaining variables.

PolynomialReduce

The function **PolynomialReduce** is new in Version 3.0. It finds the "normal form" of a given polynomial with respect to a list of polynomials, using a process known as generalized division.

```
PolynomialReduce[a + b + a b, {a b c - d, a b + b c + c a - e, a + b + c - f}]

{{0, 1, 1 - c}, -c + c^2 + e + f - c f}
```

PolynomialReduce[*poly*, *polylist*, *vars*] returns the list of the generalized quotients and the remainder (the normal form). These satisfy the relation

```
quotientlist.polylist + remainder = poly
```

The second argument to **PolynomialReduce** may be an individual polynomial rather than a list. If the third argument is omitted, the variables will be deduced from the polynomials and ordered according to some internal sorting convention. **PolynomialReduce** takes same options as **GroebnerBasis**.

If the given list of polynomials is a Gröbner basis with respect to the term ordering and underlying coefficient domain, the result of **PolynomialReduce** is canonical. In particular, if we have a polynomial in an ideal, its normal form with respect to a Gröbner basis of the ideal must be zero. For example:

```
polys = {x^5 + y^4 + z^3 - 1, x^3 + y^2 + z^2 - 1};

origgb = GroebnerBasis[polys, {x, y, z}, MonomialOrder → DegreeLexicographic];

poly = origgb[[5]] + y origgb[[3]];

gb = GroebnerBasis[polys, {y, z, x}];

PolynomialReduce[poly, gb, {y, z, x}][[2]]
0
```

A common special case is to write a polynomial *f* modulo a second polynomial *g*. One can use either **PolynomialMod** or **PolynomialReduce** to do the reduction. An advantage of the latter is that one can specify variable and monomial ordering explicitly. Here is a set of examples that came our way in a bug report. The gist is that one can get different results depending on the ordering of variables.

```
PolynomialMod[a^3 x^7 + x^4, x^2 + a]

a^2 - a^6 x
```

```
PolynomialMod[a^3 x^7 + x^4 + a, x^2 + a]
```

$$-x^2 + x^4 - x^{13}$$

If, say, you want the order to be {*a*, *x*} in the first example (so *a* will be replaced by *x*) we can use **PolynomialReduce**:

```
PolynomialReduce[a^3 x^7 + x^4, x^2 + a, {a, x}][[2]]
```

$$x^4 - x^{13}$$

There are two situations in which **PolynomialMod** will likely be the better function to use. The first is when we want to reduce a polynomial modulo an integer. For the second, say we have a pair of multivariate polynomials f and g over the integers and we wish to check whether g divides f with a quotient that is also a polynomial over the integers. This problem arises frequently in polynomial algebra, for example, in implementing polynomial greatest common divisor algorithms. Regardless of variable or term ordering, g divides f precisely when **PolynomialMod**[f, g] or **PolynomialReduce**[f, g][[2]] is zero. Both functions need to translate their arguments to an internal format. **PolynomialReduce** uses internally a so−called "sparse distributed" polynomial format. **PolynomialMod**, in contrast, uses a recursive format. That is, a polynomial is written in descending powers of the leading variable, with each coefficient itself a polynomial in the remaining variables (see [Stoutmeyer 1984]). It is generally faster to do both the translation and the exact division when working with the recursive form, hence **PolynomialMod** will likely be the better choice for this type of problem.

The pair of functions **GroebnerBasis** and **PolynomialReduce** effectively supercedes **AlgebraicRules** and **ReplaceAll**. The function **AlgebraicRules** is no longer documented and will not be upgraded in the future. There are several reasons why the new technology is an improvement. First, it is much more flexible. It will handle arbitrary term orders rather than just lexicographic. It will work over several coefficient rings. It is much better about not emitting annoying warning messages. It does not require that the reducing polynomial list be a Gröbner basis. One small disadvantage I am aware of is that **AlgebraicRules** encodes a Gröbner basis in an internal format. If one must reduce with respect to many polynomials, then the time needed for **PolynomialReduce** to convert to internal format can be an issue, albeit usually minor.

Here is a simple example to show how **GroebnerBasis** and **PolynomialReduce** work in place of **AlgebraicRules** and **ReplaceAll**.

```
poly = x^3 + 2 y^3 - 3 z^2 x + 7;
polylist = {x^5 + y^4 + z^3 - 1, x^3 + y^2 + z^2 - 1};
vars = {x, y, z};
Timing[gb = GroebnerBasis[polylist, vars];]
```

{0.09 Second, Null}

```
Timing[red1 = PolynomialReduce[poly, gb, vars][[2]]]
```

{0.23 Second, $8 - y^2 + 2 y^3 - z^2 - 3 x z^2$}

```
Timing[ar = AlgebraicRules[Thread[polylist == 0], vars];]
```

{3.08 Second, Null}

```
Timing[red2 = poly /. ar]
```

{0.02 Second, $8 - y^2 + 2 y^3 - z^2 - 3 x z^2$}

```
red1 == red2
```

True

Note that while the **ReplaceAll** works faster than **PolynomialReduce**, the time needed to form the **AlgebraicRulesData** object is vastly greater than that needed by **GroebnerBasis**.

MonomialList

The function **MonomialList** is also new in Version 3.0. The calling syntax is **MonomialList**[*poly*, *vars*] or **MonomialList**[{*poly1*, *poly2*, ...}, *vars*]. It takes the same options as **GroebnerBasis** and **PolynomialReduce**. If the last argument is omitted, the variables will be deduced exactly as for **GroebnerBasis**.

```
poly = 3 x y + 2 y^2 + 16 y z - z^2 - x y^3 z^2 - 7 x^3 y - 3 x y z^2 + 2 x z - 6 z^2 + 8 x z^2;
vars = {x, y, z};

MonomialList[poly, vars]
{-7 x^3 y, -x y^3 z^2, -3 x y z^2, 3 x y, 8 x z^2, 2 x z, 2 y^2, 16 y z, -7 z^2}
```

GroebnerBasis provides a **Sort** option, which sorts the variables according to a heuristic designed to improve efficiency for the lexicographic term order (this ordering is important for equation solving). The heuristic is a variation on that found in [Boege, Gebauer, and Kredel 1986]. One can see the polynomials in list form, ordered by term order, by using **MonomialList** with **Sort** set to **True**.

```
vars = {u, v, p, q, y};

polys = {-50 + u + v, -1600 + p^2 + 2 p u + u^2 + y^2,
   -10 000 + q^2 + 2 q v + v^2 + y^2, -10 p - 10 u + p y, -15 q - 15 v + q y};

MonomialList[polys, Sort → True]
{{u, v, -50}, {u^2, 2 p u, p^2, y^2, -1600},
 {v^2, 2 q v, q^2, y^2, -10 000}, {-10 u, p y, -10 p}, {-15 v, q y, -15 q}}
```

One can discern that the variable ordering has $u > v > \{q, p\} > y$. It is not clear exactly how q and p are ordered. One can obtain the full ordering by the simple trick of adjoining the polynomial $p + q + u + v + y$ to the list of polynomials. This change will not substantially affect the ordering used by the **Sort** algorithm. That is, the resulting ordering will be the same unless there is a tie broken by an internal sort that was altered due to the presence of the new polynomial.

```
MonomialList[Join[{u + v + q + y + p}, polys], vars, Sort → True][[1]]
{u, v, q, p, y}
```

The result shows a variable ordering of {*u*, *v*, *p*, *q*, *y*}. When one forms a lexicographic Gröbner basis of the original set of polynomials using this ordering, the largest coefficient has 16 digits.

```
gb = GroebnerBasis[polys, {u, v, p, q, y}];

Log[10., Max[Abs[Flatten[MonomialList[gb, vars] /. Thread[{u, v, p, q, y} → 1]]]]]
15.4866
```

The order {*y*, *p*, *q*, *v*, *u*} will give a basis with a largest coefficient of 37
digits. Not surprisingly, it takes a bit longer to compute.

Another use for **MonomialList** is as a "distributed" version of **Collect**. In Version 1 of *Mathematica*, all specified variables were treated as equals, and so each power product in the given variables was grouped individually with a coefficient (which could be a number or involve "uncollected" variables). Thus, the polynomial would be represented in distributed form relative to the list of variables. Version 2 changed **Collect** so that the variables were ordered; it would **Collect** with respect to the leading variable, and each coefficient would then be collected with respect to the next variable, and so on. This gives a "recursive" representation of the polynomial. Both representations can be useful.

Here, we show two ways to recover the distributed form. The first involves collecting on a list of pattern variables (this functionality also existed in Version 1, in slightly different form, and was lost in Version 2). This method is similar to one described in [Harris 1994], from which this example is taken

```
poly = a^3 - d^3 + 3 a^2 x + 3 a x^2 + x^3 + 3 a^2 c y + 3 a^2 d y +
    6 a c x y + 6 a d x y + 3 c x^2 y + 3 d x^2 y + 3 a c^2 y^2 + 6 a c d y^2 + 3 a d^2 y^2 +
    3 c^2 x y^2 + 6 c d x y^2 + 3 d^2 x y^2 + c^3 y^3 + 3 c^2 d y^3 + 3 c d^2 y^3 + d^3 y^3;
d1 = Collect[poly, {x^_.*y^_., x^_., y^_.}]
```

$$a^3 - d^3 + 3\,a^2\,x + 3\,a\,x^2 + x^3 + \left(3\,a^2\,c + 3\,a^2\,d\right)\,y + (6\,a\,c + 6\,a\,d)\,x\,y + (3\,c + 3\,d)\,x^2\,y + \left(3\,a\,c^2 + 6\,a\,c\,d + 3\,a\,d^2\right)\,y^2 + \left(3\,c^2 + 6\,c\,d + 3\,d^2\right)\,x\,y^2 + \left(c^3 + 3\,c^2\,d + 3\,c\,d^2 + d^3\right)\,y^3$$

This method will obviously get cumbersome for more variables. One could write code to generate all the power product patterns, as the list could get quite large. Instead, we show the second way to implement a distributed **Collect**, using **MonomialList**.

```
d2 = Plus @@ MonomialList[poly, {x, y}, CoefficientDomain → RationalFunctions]
```

$$a^3 - d^3 + 3\,a^2\,x + 3\,a\,x^2 + x^3 + \left(3\,a^2\,c + 3\,a^2\,d\right)\,y + (6\,a\,c + 6\,a\,d)\,x\,y + (3\,c + 3\,d)\,x^2\,y + \left(3\,a\,c^2 + 6\,a\,c\,d + 3\,a\,d^2\right)\,y^2 + \left(3\,c^2 + 6\,c\,d + 3\,d^2\right)\,x\,y^2 + \left(c^3 + 3\,c^2\,d + 3\,c\,d^2 + d^3\right)\,y^3$$

Collect now takes, as an optional third argument, a function to apply to each collected term. This new feature is quite useful. For example, we can do partial factorization of the coefficients in the previous result.

```
Collect[poly, {x^_.*y^_., x^_., y^_.}, FactorSquareFree]
```

$$a^3 - d^3 + 3\,a^2\,x + 3\,a\,x^2 + x^3 + 3\,a^2\,(c + d)\,y + 6\,a\,(c + d)\,x\,y + 3\,(c + d)\,x^2\,y + 3\,a\,(c + d)^2\,y^2 + 3\,(c + d)^2\,x\,y^2 + (c + d)^3\,y^3$$

We can obtain the same result using the **MonomialList** method.

```
Plus @@ FactorSquareFree /@
  MonomialList[poly, {x, y}, CoefficientDomain → RationalFunctions]
```

$$a^3 - d^3 + 3\,a^2\,x + 3\,a\,x^2 + x^3 + 3\,a^2\,(c + d)\,y + 6\,a\,(c + d)\,x\,y + 3\,(c + d)\,x^2\,y + 3\,a\,(c + d)^2\,y^2 + 3\,(c + d)^2\,x\,y^2 + (c + d)^3\,y^3$$

We could do full factorization, using **Factor**, but **FactorSquareFree** is preferable because it uses more efficient technology.

Example: polynomial inversion

We apply **GroebnerBasis** to the task of rationalizing denominators. Specifically, we define a function **PolynomialInverseMod** that finds the inverse of a polynomial f modulo another polynomial g. The arguments are polynomials in one variable (say, x). Other variables, if present, are treated as invertible parameters. For an irreducible polynomial g, we find a polynomial r such that $r\,f == 1$ over the algebraic number ring $\mathbb{Q}[x]$ modulo $g(x)$ (or, in the modular case, over the finite field $\mathbb{Z}_p[x]$ modulo $g(x)$). The method can be found in [Buchberger 1982, p. 204]. For extensions of this method to more general computations in algebraic number fields, see [Becker and Weispfenning 1993, chap. 7; Adams and Loustaunau 1994, chap. 2, sec. 6].

The idea is simple. We define a new variable r, which will be the reciprocal of f. We compute a Gröbner basis for the set $\{r\,f - 1, g\}$, using lexicographic order with r higher than x. The Gröbner basis is ordered smallest to largest by term order and only the first polynomial does not contain r. (This follows from some well−known facts about lexicographic Gröbner bases.) So, we simply use the second element to solve for r in terms of x.

```
PolynomialInverseMod[f_, g_, x_, p_Integer: 0] :=
 Module[{r, gb}, gb = GroebnerBasis[{r f - 1, g},
     {r, x}, Modulus → p, CoefficientDomain → RationalFunctions];
   First[PolynomialMod[r /. Solve[gb[[2]] == 0, r], p]]] /; PrimeQ[p] || p === 0
```

The code does not check that f is actually invertible in the ring. That would involve checking that the polynomial gcd of f and g is a unit, that is, it is free of the variable x. A variation of this code appears in the standard package **Algebra‘PolynomialMod‘**.

Here is an example over the integers modulo 7.

```
f = x^2 + 3 x + 17 - 2 x y^2 + 5 y;
g = 11 x^2 - 4 x + 3;
```

```
inv = PolynomialInverseMod[f, g, x, 7]
```

$$\frac{1 + 3\,x + 5\,y + 5\,y^2 + 2\,x\,y^2}{2 + 4\,y + 5\,y^2 + 4\,y^3 + 3\,y^4}$$

We check that this result is correct using **PolynomialReduce**.

```
Last[PolynomialReduce[inv f - 1, g, Modulus → 7]]
0
```

What we have done is tantamount to rationalizing a denominator. Given an algebraic number t we can find its minimal polynomial g with **RootReduce**. Then to rationalize a denominator in t we use the fact that the field $\mathbb{R}(t)$ is isomorphic the polynomial ring $\mathbb{R}[t]$. Hence any rational function in t is equivalent to a polynomial. But $\mathbb{R}(t)$ is also isomorphic to $\mathbb{R}[x] / g(x)$ (that is, the polynomials over $\mathbb{R}$ modulo the polynomial g), with the algebraic number t mapped to the indeterminate x. This equivalence gives us the means to use **PolynomialInverseMod**.

For example, say t is $\frac{1}{\sqrt[3]{2-\sqrt{3}}} + 2\sqrt{3}$ and we want to represent $f(t) = \frac{1}{t^2-2t+7}$ as a polynomial in t over the rationals. Then

```
RootReduce[(2 - Sqrt[3])^(-1 / 3) + 2 * Sqrt[3]]
Root[-1871 - 144 #1 + 396 #1^2 - 4 #1^3 - 36 #1^4 + #1^6 &, 2]
```

tells us that t satisfies the irreducible polynomial

$$g(t) = x^6 - 36\,x^4 - 4\,x^3 + 396\,x^2 - 144\,x - 1871$$

So $\mathbb{Q}(t)$ is isomorphic to the ring $\mathbb{Q}[x]/g(x)$. Then $f(t)$ is simply the polynomial given by

```
PolynomialInverseMod[x^2 - 2 x + 7, -1871 - 144 x + 396 x^2 - 4 x^3 - 36 x^4 + x^6, x]
```

$$\frac{1\,633\,833 + 245\,649\,x - 114\,933\,x^2 - 26\,653\,x^3 + 2605\,x^4 + 741\,x^5}{13\,537\,964}$$

Example: The Gröbner basis of a module

We show how one can use **GroebnerBasis** to find the Gröbner basis of a module. The interested reader may consult [Adams and Loustaunau 1994, chap. 3; Becker and Weispfenning 1993, chap. 10, sec. 4; Helzer 1995] for further details. Our treatment will be similar to that of [Helzer 1995]. We want to find a Gröbner basis for a submodule of $\mathbb{A}^n$, where $\mathbb{A}$ is a polynomial ring in some indeterminates. As is often done in linear algebra, we could represent elements of this module as n−tuples $\{a_1, a_2, ..., a_n\}$, where each entry lies in $\mathbb{A}$. Instead, we let $\boldsymbol{e_1}$, $\boldsymbol{e_2}$, ..., $\boldsymbol{e_n}$ be n new "coordinate variables" and we represent elements as $a_1\,e_1 + a_2\,e_2 + ... + a_n\,e_n$, with a_1, a_2, ..., a_n all elements of $\mathbb{A}$. This difference, while mathematically meaningless, is important for the task at hand. It turns out that we can treat the new variables as we do all other variables, and just use **GroebnerBasis** to find a basis for modules.

A Gröbner basis for the "position−over−term" ordering may be obtained as an ordinary Gröbner basis where we have the coordinate variables ordered above the ring variables, and we also impose the relations that the product of any (not necessarily distinct) pair of coordinate variables is zero. Their purpose is to keep out polynomials that are not linear in the coordinate variables. Once finished with the computation we remove them. This is not the most efficient way to compute a module Gröbner basis. It would be better if, in the Gröbnerization algorithm, we could to impose the condition that the least common multiplier between any pair of elements is zero if the head terms do not lie the same coordinate. This would necessitate treating coordinate variables differently from other variables, and is not implemented in *Mathematica*'s **GroebnerBasis** algorithm. A still more general approach, described in [Becker and Weispfenning 1993, chap. 10, sec. 2−4], would be to implement a "truncated" version of Buchberger's algorithm, wherein only S−polynomials that are homogeneous of degree one in the coordinate variables are utilized. It turns out that the method we will use is not too far from optimal. While we still form S−polynomials (see any of [Adams and Loustaunau 1994; Becker and Weispfenning 1993; Buchberger 1982; Cox, Little, and O'Shea, 1992] for the definition of these) for polynomials whose head terms lie in different coordinates, the conditions we impose on the coordinate variables force these extra polynomials to reduce quickly to zero.

Note that we could find a Gröbner basis for the "term−over−position" ordering of module entries, simply by ordering coordinate variables after ring variables rather than before them. More exotoc orders could be obtained by use of a weight matrix as **MonomialOrder**

Complement

MonomialOrder option. Another small subtlety is that we remove the unwanted monomials by mapping them to the empty list and flattenning the result. We do not use **Complement** because, while it is faster, it will reorder the result. We prefer a Gröbner basis that is sorted by ascending term order.

```
moduleGroebnerBasis[polys_, vars_, cvars_, opts___] :=
 Module[{newpols, rels, len= Length[cvars], gb, j, k, ruls},
  rels = Flatten[Table[cvars[[j]] * cvars[[k]], {j, len}, {k, j, len}]];
  newpols = Join[polys, rels];
  gb = GroebnerBasis[newpols, Join[cvars, vars], opts];
  rul = Map[(# :> {}) &, rels];
  gb = Flatten[gb /. rul];
  Collect[gb, cvars]]
```

The example below is taken from [Helzer 1995]. It solves a particular polynomial interpolation problem. We find the Gröbner basis of a certain submodule of the free module of $\mathbb{A}^3$, where $\mathbb{A}$ is the polynomial ring $\mathbb{Q}[x, y, z]$. The module is generated by

$$\left\{\boldsymbol{e_1} - \boldsymbol{e_2} - \boldsymbol{e_3},\ \left(x^2 + y^2 - 1\right)\boldsymbol{e_2},\ z\,\boldsymbol{e_2},\ \left((y-1)^2 + z^2 - 1\right)\boldsymbol{e_3},\ x\,\boldsymbol{e_3}\right\}$$

```
ii = {x^2 + y^2 - 1, z};
```

```
jj = {(y - 1)^2 + z^2 - 1, x};
```

```
gens = Join[{e[1] - e[2] - e[3]}, e[2] ii, e[3] jj];
```

```
gb = moduleGroebnerBasis[gens, {x, y, z}, {e[2], e[3], e[1]}]
```

$$\left\{\left(-2\,y\,z + y^2\,z + z^3\right)\,e[1],\ x\,z\,e[1],\ \left(2\,y - 2\,x^2\,y - y^2 + x^2\,y^2 - 2\,y^3 + y^4 - z^2 + 2\,y\,z^2 - z^4\right)\,e[1],\right.$$
$$\left(-x + x^3 + x\,y^2\right)\,e[1],\ \left(-3 + 3\,x^2 + 2\,y - 2\,x^2\,y + 3\,y^2 - 2\,y^3 - z^2 - 2\,y\,z^2\right)\,e[1] + 3\,e[3],$$
$$\left.\left(-3\,x^2 - 2\,y + 2\,x^2\,y - 3\,y^2 + 2\,y^3 + z^2 + 2\,y\,z^2\right)\,e[1] + 3\,e[2]\right\}$$

In [Helzer 1995] it is explained that

```
p = PolynomialReduce[e[2] + 2 e[3], gb, {e[2], e[3], e[1], x, y, z}][[2]] /. e[1] → 1
```

$$2 - x^2 - \frac{2\,y}{3} + \frac{2\,x^2\,y}{3} - y^2 + \frac{2\,y^3}{3} + \frac{z^2}{3} + \frac{2\,y\,z^2}{3}$$

produces a polynomial $p(x, y, z)$ such that p is identically 1 on the unit circle defined by the pair of equations $\{x^2 + y^2 == 1, z == 0\}$, and p is identically 2 on the unit circle (interlocking the first one) given by the pair of equations $\{(y-1)^2 + z^2 == 1, x == 0\}$. We can check this fact.

```
PolynomialReduce[p, {x^2 + y^2 - 1, z}][[2]]
```

```
1
```

```
PolynomialReduce[p, {(y - 1)^2 + z^2 - 1, x}][[2]]
```

```
2
```


Acknowledgements

I thank Bruno Buchberger, David Cox, and Troels Petersen for their several helpful comments on earlier drafts of this paper.

References

Adams, W., and P. Loustaunau. 1994. An Introduction to Gröbner Bases, Graduate Studies in Mathematics Vol. 3. American Mathematical Society.

Becker, T. and V. Weispfenning (with H. Kredel). 1993. Gröbner Bases: A Computational Approach to Computer Algebra. Springer−Verlag.

Boege, W., R. Gebauer, and H. Kredel. 1986. Some examples for solving systems of algebraic equations by calculating Gröbner bases. J. Symbolic Computation **1**:83−98.

Buchberger, B. 1982. Gröbner Bases: An algorithmic method in polynomial ideal theory. In Multidimensional Systems Theory, ch. 6. N. K. Bose, ed. Van Nostrand Reinhold.

Cox, D, J. Little, and D. O'Shea. 1992. Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Computer Algebra. Springer−Verlag.

Harris, J. 1994. Rearranging expressions by patterns. The Mathematica Journal **4**(3):82−85.

Helzer, G. 1995. Gröbner bases. The Mathematica Journal **5**(1):67−73.

Helzer, G. 1995. Gröbner bases. The Mathematica Journal **5**(1):67−73.

Robbiano, L. 1985. Term orderings on the polynomial ring. In: EUROCAL ’85 Vol. II, 513−517. Springer Lecture Notes in Computer Science **204**.

Stoutemyer, D. 1984. Which polynomial representation is best? Surprises abound! In: Proceedings of the Third Macsyma Users’ Conference, Schenectady, NY. 221−243.

Wolfram, S. 1996. The Mathematica Book, 3rd ed. Wolfram Media, Inc.

Biographical sketch

Daniel Lichtblau finished his degree in the math department at the University of Illinois in August of 1991. He found a job about a mile away at Wolfram Research, Inc, where he has remained holed up ever since, working on several math algorithms in the Mathematica kernel. He is, in the words of one math professor, "forgotten, but not gone."