Implementing Gröbner Basis Methods

Benjamin J. Keller*
Department of Computer Science, Montana Tech
Butte, Montana 59701 USA
bkeller@mtech.edu

June 21, 1999

We look at the problem of implementing the algorithms for computing Gröbner bases in noncommutative algebras. We restrict our attention to free associative algebras and path algebras [3], which are free in the sense that no two words are equal (except possibly as zero divisors). Other algebras defined as monoid or group rings [13, 16, 17] require additional computational mechanism that we will not deal with here. Other "almost" commutative algebras are often implemented in commutative systems (these algebras include Weyl algebras [7], enveloping algebras of Lie algebras [2], algebras of solvable type [10], Grassman algebras [9, 19], and Clifford algebras [9]).

Noncommutative Algebras For our purposes an algebra $K\langle A\rangle$ consists of elements which are K-linear combinations of words over some alphabet A, where K is some field. We refer to elements of the algebra as *polynomials* (although some algebraists prefer *relations* to refer to elements of noncommutative algebras).

The words of A^* can be ordered with admissible orders on strings [4]. These orders are used to provide a canonical sequencing of the terms of polynomials during a Gröbner basis computation. Given an admissible order $<$, the leading term of a polynomial p is called the *tip* of p and is denoted $tip(p)$.

Gröbner Bases We restrict our attention to two-sided ideals. We denote a two-sided ideal generated by a set F as $\langle F\rangle$. A Gröbner basis for an ideal $\langle F\rangle$ is determined by F, and which terms of each generator is the leading term. The set of tips for the polynomials in the set F is denoted $Tip(F)$. A Gröbner basis for the ideal $\langle F\rangle$ is a set G such that $\langle G\rangle = \langle F\rangle$ and $\langle Tip(G)\rangle = Tip(\langle F\rangle)$.

When computing a Gröbner basis from a generating set F, pairs of elements from F are chosen such that their tips overlap. An overlap of words w_1 and w_2 is a nonempty word s such that $w_1 = ls$ and $w_2 = sr$ for words l and r. If $w_1 = tip(p)$ and $w_2 = tip(q)$ with the overlap s then a new polynomial called an

*Partially supported by MONTS (the Montana NSF EPSCoR Program)

overlap relation (an s-polynomial in the commutative case) can be computed as $p \cdot r - l \cdot q$. The common multiple of the leading terms cancels in the overlap relation, potentially finding a tip that doesn't already occur in the working basis. For (free) noncommutative algebras, all possible overlaps of leading terms must be considered. The algorithm we use maintains a set of triples, where each triple consists of a pair of polynomials and a non-empty overlap of their leading terms. The computation adds new elements to F by forming overlap relations for each triple, stopping when no unconsidered overlaps remain. Since this only happens when a finite Gröbner basis exists for the given set and order, the computation might not terminate.

Implementing Algorithms Buchberger's algorithm, like other completion procedures, is simple, but is difficult to implement efficiently. Of course, "efficiency" is relative, since often the procedure does not terminate. Efficiency can be gained in three different ways:

1. *Avoiding the computation and reduction of overlap relations that reduce to zero.* Buchberger [5] stated several criteria that allow this prediction in commutative algebras, and similar results have been shown for the free noncommutative case [14, 15, 11]. In the noncommutative case, there is one criterion on the common multiple formed by the leading terms of the polynomials that make up a overlap relation. Simplified slightly, the criterion is that if the common multiple is divisible by the leading term of another polynomial in the Gröbner basis, then the corresponding overlap relation is redundant. This criterion is used to filter the set of triples during the computation. Efficient implementations of this approach depend on proper selection of the next overlap relation, as well as operations for filtering the triples. Currently, the most efficient scheme known is to select triples with the shortest common multiple, and to test triples before they are added to the set, and at selection [11].

2. *Improving cost of frequent operations.* There are several operations that are repeated frequently throughout the computation including reduction, and set management operations for both triples and the set of polynomials. One way to improve the costs of many of these operations is to focus on the pattern matching operations on the terms of polynomials used to find overlaps, find divisors for reduction, and find divisible elements for set reduction. These operations include finding overlaps, subwords and superwords, and can be efficiently implemented using an adaptation of the dynamic dictionary matching approach [1, 11]. (A similar approach for prefix reductions has been described by Reinert and Zeckzer [17, 18].)

   The reduction operation can be improved by observing that repeated reductions have the form $p - l_1 \cdot q_1 \cdot r_1 - \cdots - l_{n-1} \cdot q_{n-1} \cdot r_{n-1}$. If a polynomial is stored as a sorted list of terms, then this expression requires a merging of n lists. The natural approach is to do each reduction separately, but postponing the merges allows each term to be considered individually and the term products to be found as each term is merged. Yan [20] exploits the idea of postponing merging with his Geobucket data structure.

3. *Computing only the portions of the basis that are needed.* Ideally we would like to restrict our Gröbner basis computations to computing only the elements that are necessary for our task. Unfortunately, it is not clear what this means, so generally all it means is being able to bound the computation. The most reasonable bound is on Gröbner bases of weight or length homogeneous generators, for which we can set a bound on the maximum weight (length) element we want to consider. Other bounds are to simply count the number of new elements found.

References

[1] Amir, A., Farach, M., Galil, Z., Giancarlo, R., and Park, K. Dynamic dictionary matching, *Journal of Computer and Systems Sciences 49*, 2 (1994) 208–222.

[2] Apel, J., and Lassner, W. An Extension of Buchberger's algorithm and calculations in enveloping fields of Lie algebras. In *Computational Aspects of Commutative Algebra*, L. Robbiano, Ed. Academic Press, 1989, pp. 227–236.

[3] Auslander, M., Reiten, I., and Smalø, S. O. *Representation Theory of Artin Algebras.* Cambridge University Press, 1995.

[4] Book, R. V., and Otto, F. *String-Rewriting Systems.* Springer-Verlag, 1993.

[5] Buchberger, B. A criterion for detecting unnecessary reductions in the construction of Gröbner-bases. In *EUROSAM '79*, E. W. Ng, Ed., LNCS# 72. Springer-Verlag, Berlin, 3–21.

[6] Farkas, D. R., Feustel, C. D., and Green, E. L. Synergy in the theories of Gröbner bases and path algebras, *Canadian Journal of Mathematics 45*, 4 (1993), 727–739.

[7] Galligo, A. Some algorithmic questions on ideals of differential operators. In EUROCAL '85 (1985), LNCS# 204, pp. 413–421.

[8] Gebauer, R. and Möller, H. M. On an installation of Buchberger's Algorithm. In *Computational Aspects of Commutative Algebra*, L. Robbiano, Ed. Academic Press, 1989, pp. 141–152.

[9] Hartley, D., and Tuckey, P. Gröbner bases in Clifford and Grassman algebras. *Journal of Symbolic Computation 20* (1995), 197–205.

[10] Kandri-Rody, A. and Weispfenning, A. Non-commutative Gröbner bases in algebras of solvable type. *Journal of Symbolic Computation 9* (1990), 1–26.

[11] Keller, B. J. *Algorithms and Orders for Finding Noncommutative Gröbner Bases.* PhD thesis, Virginia Polytechnic Institute & State University, 1997. (`http://scholar.lib.vt.edu/theses/theses.html`).

[12] Keller, B. J. Alternatives in implementing noncommutative Gröbner basis systems. In *Symbolic Rewriting Techniques*, M. Bronstein, J. Grabmeier and V. Weispfenning, Eds. Birkhauser-Verlag, 1998.

[13] Madlener, K. and Reinert, B. Computing Gröbner bases in monoid and group rings. In *ISSAC '93* (1993), ACM, pp. 254–263.

[14] Mora T. An introduction to commutative and non-commutative Gröbner bases. *Theoretical Computer Science 134*, 1 (1994) 131–173.

[15] Pritchard, F.L. The Ideal Membership Problem in Non-Commutative Polynomial Rings. *Journal of Symbolic Computation 22*, 1 (1996), 27–48.

[16] Reinert, B. *On Gröbner Bases in Monoid and Group Rings.* PhD thesis, FB Informatik, University of Kaiserslautern, 1995.

[17] Reinert, B., and Zeckzer, D. MRC — A System for Computing Gröbner Bases in Monoid and Group Rings. *6th Rhine Workshop on Computer Algebra* (Sankt Augustin, 1998).

[18] Reinert, B., and Zeckzer, D. MRC — A System for Computing Gröbner Bases in Monoid and Group Rings. Reports on Computer Algebra 20, Centre for Computer Algebra, Universität Kaiserslautern, 1998. Available online at `www.mathematik.uni-kl.de/~zca`.

[19] Stokes, T. Gröbner bases in exterior algebras. *Journal of Automated Reasoning 6* (1990), 233–250.

[20] Yan, T. The Geobucket Data Structure for Polynomials. *Journal of Symbolic Computation 25*, 3 (1998), 285-294.