Programming and Computer Software, Vol. 29, No. 2, 2003, pp. 72–74. Translated from Programmirovanie, Vol. 29, No. 2, 2003.
Original Russian Text Copyright © 2003 by Gerdt, Yanovich.

Implementation of the FGLM Algorithm and Finding Roots of Polynomial Involutive Systems

V. P. Gerdt and D. A. Yanovich
Joint Institute for Nuclear Research, Dubna, Moscow oblast, 141980 Russia
e-mail: gerdt@jinr.ru, yan@jinr.ru

Received June 3, 2002

Abstract—In this paper, an implementation of the FGLM algorithm that transforms Gröbner bases from one ordering to another is presented. Some additional optimizations that considerably expedite computations are considered. It is shown that this algorithm can be used for finding roots of polynomial systems represented in the involutive form.

1. INTRODUCTION

For the computation of Gröbner bases [1] and their particular case, involutive bases [2–4], the DegRevLex ordering is most convenient. However, the basis with such ordering is difficult to use for one of the most important applications, namely, for computing roots of systems of equations. For this purpose, a lexicographically ordered basis is most convenient. The algorithm FGLM (the abbreviation stands for the first letters of its authors' names) [5] transforms Gröbner bases from one ordering to another. Since a Gröbner basis can easily be

```
Algorithm **FGLM**
**Input:** *OldBasis* – a Gröbner basis in the original ordering
         $\prec$ – admissible ordering
**Output:** *NewBasis* – Gröbner basis in the new ordering
**Functions:** *NormalForm* returns the reduced normal form of the polynomial
           *pop* deletes the first element from the list and returns it
           *InsertNexts* places monomial extensions into *ListOfNexts* and sorts out the list
deleting duplicates
**begin**
  MBasis = []; staircase = []; NewBasis = []; ListOfNexts = [1];
  **while** ListOfNexts $\neq \varnothing$ **do**
    Monom = pop(ListOfNexts)
    **if** Monom divides none of elements of staircase
      **then** vector = NormalForm(Monom)
      **if** $\exists$ linear combination vector = $\sum_{v \in M\ Basis} \lambda_v seconds(v)$
        **then**
          poly = Monom + $\sum_{v \in M\ Basis} \lambda_v first(v)$
          NewBasis = poly $\cup$ NewBasis
          staircase = Monom $\cup$ staircase
        **else**
          MBasis = [Monom, vector] $\cup$ MBasis
          InsertNexts(Monom)
      **end if**
    **end if**
  **end while**
**end**
```

Figure.

derived from an involutive basis (it is sufficient to carry out reduction with respect to an ordinary division), this algorithm can be efficiently used for the computation of a lexicographical basis of the system represented in an involutive form and subsequent finding of its roots.

2. FGLM ALGORITHM

The FGLM algorithm is presented in the figure. The transformation of a basis from one ordering to another consists in the following. First, a sequence of monomials is constructed, starting from the least ones in the new ordering and their normal forms in the old basis (structure *M Basis* in the algorithm) until there appears a monomial whose normal form is a linear combination of normal forms of the preceding monomials. In this case, we add the element obtained to the new basis (structure *staircase* for the leading monomials of the new basis) and turn to the construction of a basis element with the next variable in this ordering.

The computation of the normal form of a monomial is simplified if we use the fact that the normal forms of its divisors are already known. Let $m = x_i \cdot m'$ and $p = NormalForm(m')$. Then,

$$NormalForm(m) \\ = NormalForm(NormalForm(x_i) \times p).$$

3. SOME OPTIMIZATIONS

The major part of the algorithm operation time is spent on checking whether the normal form of the current monomial is a linear combination of those of the preceding monomials. This checking reduces to solving systems of linear equations with very long (thousands of decimal digits) coefficients. The number of systems to be solved is pretty large (equal to the number of monomials under consideration), which results in considerable time expenditures on the corresponding arithmetic operations. On the other hand, arithmetic calculations related to the checking of linear dependence can be reduced if we first perform this operation over a ring of coefficients modulo a certain prime number. Clearly, if there is no linear dependence in the modular case, it lacks in the case of long coefficients either. Indeed, let us assume that there exists a linear dependence in the latter case. Then, dividing all coefficients of the corresponding linear combination by the integer greatest common divisor and taking them modulo the selected prime number, we obtain linear dependence for the modular case. Conversely, it is evident that the existence of modular linear dependence implies linear dependence in the case of integer coefficients. Taking this into account, we first check modular linear dependence; then, only if it exists, we turn to the arithmetic over the ring of integers for the explicit finding of the coefficients of the linear combination.

Another expedient makes it possible to reduce overheads related to the work with memory (which can add up to 80% of the algorithm operation time). We use the fact that, after the involutive basis has been computed, the number of roots of the system and, thus, the maximum dimension of the matrix to work with is known. Therefore, it is sufficient to place this matrix into the memory only once, upon the algorithm initialization, and, then, to fill it by using the pointer exchange.

Table

Example	var	$T_{DegRevLex}$	T_{FGLM}	Roots	$Basis_{lex}$ (Kb)
assur44	8	65.60	2540.06	56	2506
chemkin	10	67.62	7086.27	80	4093
chemequs	5	4.09	163.24	16	1105
cpdm5	5	4.79	452.84	213	492
cyclic6	6	0.36	2.15	156	12
cyclic7	7	193.60	38554.1	924	696
d1	12	40.98	1250.82	48	2770
dessin18_3	8	0.81	235.56	46	986
dessin22_24	10	3.14	94.87	42	512
eco8	8	1.12	119.50	64	91
extcyc5	6	4.36	184.10	350	118
fabrice24	9	649.14	4883.80	40	5616
filter9	9	38.73	1.22	192	18
jcf26	34	1317.05	30075.50	40	8373
reimer5	5	0.98	8.53	144	29
reimer6	6	50.24	2132.89	576	140
virasoro	8	25.10	120.92	256	992

These two simple techniques made it possible to considerably expedite computations. For example, the run time for the "virasoro" test when using the old and new versions of the program was more than one day and about 121 s, respectively. Such a great reduction in the computation time took place in other examples as well; the corresponding results are presented in the table. Moreover, without the optimizations discussed, some examples (e.g., "cyclic7") took too much computer time (certainly, more than several days).

4. RESULTS

The computations were carried out on PIII-700 MHz, 1 Gb RAM, under RedHat Linux 6.2. The program was compiled by Intel C/C++ v(5.0.1.). All timings are in seconds. The test examples[1] were borrowed from the databases [6, 7].

ACKNOWLEDGMENTS

This work was supported in part by INTAS, project no. 99-1222, and by the Russian Foundation for Basic Research, project nos. 00-15-96691, 01-01-00708, and 02-01-06440.

[1] These, as well as some other, examples can be found on the Internet: http://invo.jinr.ru

REFERENCES

1. Buchberger, B., Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, *Symbolic and Algebraic Computations*, Buchberger, B., Collins, G., and Loos, R., Eds., Wien: Springer, 1982. Translated under the title *Komp'yuternaya algebra. Simvol'nye i algebraicheskie vychisleniya*, Moscow: Mir, 1986.

2. Gerdt, V.P. and Blinkov, Yu.A., Involutive Bases of Polynomial Ideals, *Math. Comput. Simulation*, 1998, vol. 45, pp. 519–542.

3. Gerdt, V.P. and Blinkov, Yu.A., Minimal Involutive Bases, *Math. Comput. Simulation*, 1998, vol. 45, pp. 543–560.

4. Gerdt, V.P., Blinkov, Yu.A., and Yanovich, D.A., Construction of Janet Bases. II. Polynomial Bases, *Proc. of the Conf. "Computer Algebra in Scientific Computing" (CASC'01)* (2001), Berlin: Springer, 2001.

5. Faugere, J.C., Gianni, P., Lazard, D., and Mora, T., Efficient Computation of Zero Dimensional Gröbner Bases by Change of Ordering, *J. Symbolic Computation*, 1993, vol. 16, pp. 329–344.

6. Bini, D. and Mourrain, B., Polynomial Test Suite, 1996, http://www-sop.inria.fr/saga/POL.

7. Verschelde, J., The Database with Test Examples, http://www.math.uic.edu/~jan/demo.html.