

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

162

Computer Algebra

EUROCAL '83, European Computer Algebra Conference
London, England, March 28–30, 1983
Proceedings

Edited by J. A. van Hulzen



Springer-Verlag
Berlin Heidelberg New York Tokyo 1983

Editorial Board

D. Barstow W. Brauer P. Brinch Hansen D. Gries D. Luckham
C. Moler A. Pnueli G. Seegmüller J. Stoer N. Wirth

Editor

J. A. van Hulzen
Twente University of Technology
Department of Computer Science
P.O.Box 217, 7500 AE Enschede, The Netherlands

CR Subject Classifications (1982): I.1., J.2.

ISBN 3-540-12868-9 Springer-Verlag Berlin Heidelberg New York Tokyo
ISBN 0-387-12868-9 Springer-Verlag New York Heidelberg Berlin Tokyo

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to "Verwertungsgesellschaft Wort", Munich.

© by Springer-Verlag Berlin Heidelberg 1983
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

PREFACE

EUROCAL '83 is the second major conference which is organized by the European computer algebra community. However, there is a remarkable difference with the previous symposium, EUROCAL '82, held in Marseille in April 1982 (this Series, nr. 144). In view of the success EUROCAM '82 certainly was, it was decided during this meeting to formally establish a European organization, called SAME, Symbolic and Algebraic Manipulation in Europe. This decision naturally evolved from the increasing interest in algebraic computation and symbolic manipulation in Europe during the past decade. The main motivation behind the creation of SAME was the wish to be able to easily coordinate the activities of different smaller groups, some of which had already a "long" history, at least in terms of computer science. Hence EUROCAL '83 is the first reflection of the existing intentions to annually organize meetings in Europe.

It is fair to state that ACM's Special Interest Group on Symbolic and Algebraic Manipulation (SIGSAM), the major world-wide organization in this area of computer science, had a stimulating and constructive influence on these European developments. Its regularly held and excellent conferences also served to bring interested Europeans together. In fact SAME can be considered as a credit for SIGSAM. This in view of the willingness of SIGSAM officials to regularly organize symposia in Europe, always in cooperation with the European groups: EUROSAM '74 and EUROSAM '79 (this Series nr. 72) will be followed by EUROSAM '84, to be held in Cambridge, England, during July 9-11, 1984. This forthcoming meeting is a joint effort of SAME and SIGSAM, with the latter organization having the main responsibility. And like its predecessor, EUROCAL '83 was organized in cooperation with SIGSAM, with official approval of ACM. This cooperation serves our computer algebra community as a whole. It motivated the Europeans to decide not to organize an official conference in those years SIGSAM is having its official symposia.

A conference like EUROCAL '83 can only be successful when many individuals are willing to spend part of their time and efforts in its preparation. Sincere thanks are due to A.C. Hearn, past SIGSAM chairman, and M. Mignotte, my predecessor as chairman of SAME, who gave their support and advice, to J. Aman, SAME secretary, who spent a lot of time in distributing information and announcements, to the members of the program committee, who devoted many hours to the submissions and the referee reports before the final program was settled, and last, but certainly not least to Patricia D. Pearce, who was responsible for the local arrangements. She did a

marvellous job. The enjoyable hospitality of Kingston Polytechnic largely contributed to the success of the Conference. I owe a special word of thanks to mrs. Therese ter Heide-Noll. Without her secretarial support, given during all stages of preparing EUROCAL '83, I would have been helpless.

J.A. van Hulzen

ORGANIZING COMMITTEE

SAME Chairman : M. Mignotte, Université de Strasbourg, France

SAME Secretary : J. Åman, Universitet Stockholm, Sweden

Program Chairman : J.A. van Hulzen, Technische Hogeschool Twente, the Netherlands

Program Committee : B. Buchberger, Johannes Kepler Universität, Linz, Austria
J. Campbell, University of Exeter, England
J. Della Dora, IMAG, Grenoble, France
J.C. Lafon, Université de Strasbourg, France
M. Pohst, Universität Düsseldorf, Germany-West

Panel : R. Loos, Universität Karlsruhe, Germany-West
J.H. Davenport, IMAG, Grenoble, France and
University of Cambridge, England

Local Arrangements : P.D. Pearce, Kingston Polytechnic, Surrey, England

Proceedings Editor : J.A. van Hulzen, Technische Hogeschool Twente, the Netherlands

SAME

EUROCAL '83 was organized under responsibility of
SAME and in cooperation with SIGSAM, with official
approval of ACM

acm

REFEREES

The persons listed below have contributed their time and effort to the reviewing process for this conference. Their invaluable assistance to the Program Committee is gratefully acknowledged.

N. Anderson, Kent State University
D.S. Arnon, Purdue University
L. Bachmair, Johannes Kepler Universität, Linz
K.A. Bahr, GMD-Darmstadt
J. Calmet, IMAG, Grenoble
B.F. Caviness, University of Delaware
G. Cherry, University of Delaware
I. Cohen, Stockholm Universitet
G.E. Collins, University of Wisconsin, at Madison
J. Davenport, University of Cambridge and IMAG, Grenoble
J.P. Fitch, University of Bath
J.M. Gentleman, University of Waterloo
M. Gini, Universita di Milano
P. Gragert, THT/TW, Enschede
M.L. Griss, University of Utah
G. Havas, IAS SNW, Math., Canberra
J.A. Howell, Los Alamos Scientific Lab.
R.D. Jenks, IBM Research
E. Kaltofen, University of Toronto
P. Kersten THT/TW, Enschede
D. Lazard, Université de Poitiers
A. K. Lenstra, MC, Amsterdam
F. Lichtenberger, Johannes Kepler Universität, Linz
R. Loos, Universität Karlsruhe
M. Mignotte, Université de Strasbourg
H.M. Möller, FernUniversität, Hagen
F. Mora, Universita di Genova
J. Morgenstern, Université de Nice
E.W. Ng, JPL, Pasadena.
G. Pilz, Johannes Kepler Universität, Linz

T. Sasaki, IPCR, Saitama, Japan
B.D. Saunders, RPI
J. Siekmann, Universität Karlsruhe
J. Smit, THT/EL, Enschede
D.R. Stoutemyer, University of Hawaii
P.S. Wang, Kent State University
S. Watanabe, University of Tokyo and MIT
F. Winkler, Johannes Kepler Universität, Linz
S. Wolfram, Caltech and Princeton University
H.G. Zimmer, Universität Saarbrücken
R. Zippel, MIT

TABLE OF CONTENTS

INTRODUCTION.....1

Algorithms 1 -Miscellaneous

INTEGRATION - WHAT DO WE WANT FROM THE THEORY? (Invited Paper).....2
J.H. Davenport, IMAG, Grenoble, France

THE EUCLIDEAN ALGORITHM FOR GAUSSIAN INTEGERS.....12
H. Rolletschek, University of Delaware, U.S.A.

MULTI POLYNOMIAL REMAINDER SEQUENCE AND ITS APPLICATION TO LINEAR DIOPHANTINE
EQUATIONS.....24
A. Furukawa, Tokyo Metr. University, Japan, & T. Sasaki, IPCR, Saitama, Japan

Applications - Miscellaneous

TOWARDS MECHANICAL SOLUTION OF THE KAHAN ELLIPSE PROBLEM I.....36
D.S. Arnon & S.F. Smith, Purdue University, U.S.A.

AUTOMATICALLY DETERMINING SYMMETRIES OF ORDINARY DIFFERENTIAL EQUATIONS.....45
F. Schwarz, Universität Kaiserslautern, Germany-West

ALGEBRAIC COMPUTATION OF THE STATISTICS OF THE SOLUTION OF SOME NONLINEAR
STOCHASTIC DIFFERENTIAL EQUATIONS.....55
F. Lamnabhi-Lagarrigue, LSE, Gif sur Yvette, France & M. Lamnabhi, LEPA,
Limeil-Brevannes, France

CHARACTERIZATION OF A LINEAR DIFFERENTIAL SYSTEM WITH A REGULAR SINGULARITY....68
A. Hilali, IMAG, Grenoble, France

Systems and Language Features

THE BATH CONCURRENT LISP MACHINE.....	78
J. Marti, Rand Corporation, Santa Monica, U.S.A. & J.P. Fitch, University of Bath, England	
THE ECOLOGY OF LISP OR THE CASE FOR THE PRESERVATION OF THE ENVIRONMENT	91
J.A. Padgett, University of Bath, England	
THE DESIGN OF MAPLE : A COMPACT, PORTABLE AND POWERFUL COMPUTER ALGEBRA SYSTEM (Invited Paper).....	101
B.W. Char, K.O. Geddes, W.M. Gentleman & G.H. Gonnet, University of Waterloo, Canada	
LISP COMPIILATION VIEWED AS PROVABLE SEMANTICS PRESERVING PROGRAM TRANSFORMATION.....	116
H. Stoyan, Universität Erlangen, Germany-West	
IMPLEMENTING REDUCE ON A MICRC-COMPUTER.....	128
J.P. Fitch, University of Bath, England	

Algorithms 2 - Polynomial Ideal Bases

A NOTE ON THE COMPLEXITY OF CONSTRUCTING GRÖBNER-BASES.....	137
B. Buchberger, Johannes Kepler Universität, Linz, Austria	
GRÖBNER BASES, GAUSSIAN ELIMINATION AND RESOLUTION OF SYSTEMS OF ALGEBRAIC EQUATIONS.....	146
D. Lazard, Université de Poitiers, France	
THE COMPUTATION OF THE HILBERT FUNCTION.....	157
F. Mora, Università di Genova, Italy & H.M. Möller, FernUniversität, Hagen, Germany-West	
AN ALGORITHM FOR CONSTRUCTING DETACHING BASES IN THE RING OF POLYNOMIALS OVER A FIELD.....	168
F. Winkler, Johannes Kepler Universität, Linz, Austria	

Algorithms 3 - Computational Number Theory

ON THE PROBLEM OF BEHĀ EDDĪN 'AMŪLĪ AND THE COMPUTATION OF HEIGHT FUNCTIONS (Invited Paper).....	180
H.G. Zimmer, Universität des Saarlandes, Germany-West	
A PROCEDURE FOR DETERMINING ALGEBRAIC. INTEGERS OF GIVEN NORM.....	194
U. Fincke & M. Pohst, Universität Düsseldorf, Germany-West	
COMPUTATION OF INTEGRAL SOLUTIONS OF A SPECIAL TYPE OF SYSTEMS OF QUADRATIC EQUATIONS.....	203
M. Pohst, Universität Düsseldorf, Germany-West	

Algorithms 4 - Factorization

FACTORIZATION OF SPARSE POLYNOMIALS.....	214
J.H. Davenport, IMAG, Grenoble, France	
EARLY DETECTION OF TRUE FACTORS IN UNIVARIATE POLYNOMIAL FACTORIZATION.....	225
P.S. Wang, Kent State University, U.S.A.	
ON THE COMPLEXITY OF FINDING SHORT VECTORS IN INTEGER LATTICES.....	236
E. Kaltofen, University of Toronto, Canada	
FACTORING POLYNOMIALS OVER ALGEBRAIC NUMBER FIELDS.....	245
A.K. Lenstra, Mathematisch Centrum, Amsterdam, The Netherlands	

System Oriented Applications

THE CONSTRUCTION OF A COMPLETE MINIMAL SET OF CONTEXTUAL NORMAL FORMS.....	255
M. Rice, CRI, Nancy, France	
A KNOWLEDGE-BASED APPROACH TO USER-FRIENDLINESS IN SYMBOLIC COMPUTING.....	267
F. Gardin & J.A. Campbell, University of Exeter, England	

COMPUTER ALGEBRA AND VLSI, PROSPECTS FOR CROSS FERTILIZATION.....	275
J. Smit, Technische Hogeschool Twente, the Netherlands	
CODE OPTIMIZATION OF MULTIVARIATE POLYNOMIAL SCHEMES: A PRAGMATIC APPROACH....	286
J.A. van Hulzen, Technische Hogeschool Twente, the Netherlands	
Appendix:	
The Conference Program.....	301

AUTHOR INDEX

Arnon.....	36
Buchberger.....	137
Campbell.....	267
Char.....	101
Davenport.....	2, 214
Gardin.....	267
Geddes.....	101
Gentleman.....	101
Gonnet.....	101
Fincke.....	194
Fitch.....	78, 128
Furukama.....	24
Hilali.....	68
Kaltofen.....	236
Lamnabhi.....	55
Lamnabhi-Lagarrique.....	55
Lazard.....	146
Lenstra.....	245
Marti.....	78
Möller.....	157
Mora.....	157
Padget.....	91
Pohst.....	194, 203
Rice.....	255
Rolleitschek.....	12
Sasaki.....	24
Schwarz.....	45
Smit.....	275
Smith.....	36
Stoyan.....	116
Van Hulzen.....	286
Wang.....	225
Winkler.....	168
Zimmer.....	180

INTRODUCTION

This is the third volume, appearing in the Lecture Notes in Computer Science Series of Springer Verlag, which is dedicated to a computer algebra conference. Traditionally European Computer Algebra Symposia not only provide a platform for presenting well qualified research results, as reflected by the contents of this volume. They also serve to discuss thoughts and ideas about ongoing research and future trends. Hence the conference programs contain official sessions, as well as more informal sessions about work in progress. To show the flavour of these meetings, the complete EUROCAL '83 program is given in an appendix.

The contents of this volume again shows a rich diversity of research contributions. Supplementum 4 of Computing, published by Springer Verlag in 1982, was also dedicated to computer algebra. This volume consists of sixteen survey articles and covers most of the important theoretical results, algorithms, software methods and recent applications, together with systematic references to literature, than known. It is thus a good source for obtaining a first impression of computer algebra and consequently also for situating the material presented in these proceedings.

Computer algebra has many fascinating aspects and might have numerous applications. Although such applications can be straightforward in nature, at least in terms of computer algebra, the use of one of the existing systems might be instrumental or surprisingly in the context of another discipline. Consequently such applications ought to be presented in their own context. Here an application is intuitively understood to be a contribution to a further development of either the theory or the underlying software. These considerations serve, as usual, to enlighten discussions about future trends in computer algebra or, attempting at being more precisely, about a transparent definition of the field itself. One of the charms of computer algebra is that such a definition is hard to give. This reflects, like these proceedings, the vivid interest of both "pure" mathematicians and those attracted by its hardware or software aspects, when just mentioning two "extrema". It underlines the three recommendations, given by Paul S. Wang during his banquet address. The first is education, the second is education and the third is education too. Many of the conference participants recognized that publication of real lecture notes about fundamental aspects and use of computer algebra, for instance in this Series, might largely contribute in establishing the user community and the more general interest computer algebra deserves, at least according to its adepts. This will certainly contribute to a communis opinio, and probably also to a "definition".

Integration - What do We Want from the Theory?

J.H. Davenport¹

Equipe d'Analyse Numérique,
Laboratoire I.M.A.G., B.P.68,
38402 Saint Martin d'Hères, France

Abstract. The theory of integration has moved a long way in the last fourteen years, though not far enough to satisfy the demands placed on it by its customers. This paper outlines what problems have yet to be solved, and tries to explain why they are not trivial.

Introduction

This paper considers the theory of integration, what algorithms are known, and what the stumbling blocks are in the way of a wider range of integration algorithms. We also look at the types of functions covered by existing algorithms, and ask whether we can expect better algorithms. The author's view of the subject has been largely shaped by his discussions with B. Buchberger, J.P. Fitch and B.M. Trager; however only the author can be blamed for any difficulties or bad predictions.

Rational Functions

The integration of rational functions can almost be regarded as a solved problem. We have [Yun, 1977 and the references cited therein] algorithms for finding the rational part of such a function about as efficiently as can be expected - $O(\gcd)$. It remains to be proved that integration can not be more efficient (in O terms) than the gcd operation. but this is more of a problem of complexity theory than of integration.

For the logarithmic part, we know [Trager, 1976] how to compute the logarithmic part of the integral while working in the smallest possible extension field. This brings us, however, to the reason why the word "almost" was used above - the general

1. Permanent address: Emmanuel College, Cambridge, England.

state of the algorithms for manipulating algebraic extensions is less than perfect. Consider a simple problem, such as working in the splitting field over \mathbb{Q} of x^4+2x^3+5 . Trager [1976] presents algorithms for computing a primitive element of this field, with the implicit assumption that the primitive element is the best representation. Loos [1982] makes similar proposals. When we try to find a primitive element of this field, we find that its minimal polynomial is, as expected, of degree 24, but also that its coefficients are 15 digits long [Zejli, 1983]. The author knows of no way of determining if this is the "simplest" primitive element, or even of a rigorous definition of "simplest".

In general, any manipulation of algebraic numbers is extremely expensive, and not much is known of the inherent cost of such operations, though we know that the naive method of multiplication is asymptotically of the optimal order of growth [Winograd, 1979, 1980].

Algebraic Functions

The integration of algebraic functions is, in principle, a solved problem [Davenport, 1981], though it is clear that much remains to be done before there exist program capable of integrating (by which we include proving unintegrable, if that is the case) all types of algebraic functions. In this section, we discuss some of the problems in the way, over and above the problems we have already mentioned; those of manipulating algebraic extensions.

We have the obvious generalisation of Trager's characterisation of the minimum extension field necessary for expressing the integral of an algebraic function.

Theorem 1. If $f \in K(x,y)$, where y is algebraic over $K(x)$, and f has an elementary integral, then the integral can be expressed as

$$\int f dx = v_0 + \sum c_i \log v_i,$$

where $v_0 \in K$, $c_i \in L$ and $v_i \in L(x,y)$, L being the extension of K by all the residues of f . L is the smallest field over which the integral can be expressed.

Proof. Such an expression for the integral, but with L replaced by the algebraic closure of K , is well-known [Risch, 1970; Davenport, 1981]. That L is the smallest such field follows from the fact that the residues of f can be expressed as linear combinations of the c_i , with integer coefficients.

Given any such expression, we can choose the c_i to be linearly independant over \mathbb{Q} , and so that c_1, \dots, c_m belong to L . Then the residues of f at the poles and zeroes of v_{m+1} (if it exists) contain a term of c_{m+1} , and hence do not belong to L - a contradiction. Thus all the c_i belong to L . Suppose now that the v_i are expressed, not over L , but over a larger field M , and write G for $\text{Gal}(M^*/L)$, the group of all

automorphisms of M leaving L fixed, and n for the number of elements of G . Then

$$\begin{aligned} f &= \frac{1}{n} \sum_{\sigma \in G} \sigma(f) \\ &= \frac{1}{n} \sum_{\sigma \in G} \sigma(v_0 + \sum_i c_i \log v_i)' \\ &= \frac{1}{n} \left(\sum_{\sigma \in G} \sigma(v_0) \right)' + \sum_i \frac{c_i}{n} \left(\sum_{\sigma \in G} \log v_i \right)' \\ &= \frac{1}{n} \left(\sum_{\sigma \in G} v_0 \right)' + \sum_i \frac{c_i}{n} (\log \prod_{\sigma \in G} v_i)', \end{aligned}$$

and, by the fundamental theorem of symmetric functions all terms in this last equation can be expressed over L .

A similar argument based on $\text{Gal}(L^*/K)$ shows that v_0 must belong to K , and also that the coefficients c_i of the logarithms do, in fact, generate a normal extension of K , as one might have expected. Q.E.D.

The obvious problem is that this theorem suggests no way of computing the integral within L , and in fact, the author knows of no way¹ of computing L without completely factorising the denominator of f , or, more generally, without working in the algebraic closure of K . The last remark is important - in the case of rational functions, the field generated by the residues is a subfield of the splitting field of the denominator, whereas for algebraic functions this is no longer the case.

$$\int \frac{\sqrt{x+2}}{x} dx = \sqrt{2} \log \left(\frac{\sqrt{x+2} - \sqrt{2}}{\sqrt{x+2} + \sqrt{2}} \right) + 2\sqrt{x+2}.$$

One might think that it was sufficient to extend the splitting field of the denominator by the values of the algebraic functions at all the roots of the denominator, but it is actually necessary to consider infinity as a potential pole, as well, as the following example shows:

$$\int \sqrt{2x^2+1} dx = \frac{x \sqrt{2x^2+1}}{2} - \frac{\log(\sqrt{2x^2+1} + x\sqrt{2})}{2\sqrt{2}}.$$

This field, $M=K(\text{poles of } f, \text{ values of algebraic functions at poles})$, does in fact contain all the residues, and, hence is large enough to express the integral in. However, current algorithms for the integration of algebraic functions are not capable of restricting themselves even to this field, and may require additional algebraic extensions in order to investigate the torsion nature of the divisors. Hence we can pose the following sub-goals for reducing the number of algebraic extensions required for the integration of algebraic functions:

1. However, Trager's thesis will probably shed light on this question.

- 1) Determine the residues, and hence L , the minimum field for the integral, directly;
- 2) Investigate the torsion nature of the divisors over their fields of definition;
- 3) Determine the divisors without leaving L .
- 4) Find the non-logarithmic part without leaving K .

In the case of rational functions, (2) does not exist, and (1) and (3) are resolved by Trager [1976], while (4) is classical [Hermite, 1872]. For the case of algebraic functions, if the logarithmic part has been found, then (4) can be solved, since the integral must satisfy certain degree constraints (be larger than a particular divisor), which can in turn be bounded by a method that does not involve leaving K . However, it would probably be easier (judging from the rational function case) to solve the other problems if the integral were known to be purely logarithmic.

We note that, while the residues were merely an optimization in the case of rational functions, though an important one, they are necessary for algebraic functions (at least as far as we know).

We mention here some more general questions on the integration of algebraic functions. They are admittedly somewhat technical, but, as has been obvious since at least the days of Abel, the integration of algebraic functions is at the same time extremely technical and connected to some of the deepest ideas in mathematics.

- a) Is the generalisation by Zolotarev [1874] of Chebyshev's method equivalent to the use of a generalised Lutz-Nagell Theorem?
- b) How, in practice, can one bound the torsion of a divisor on a curve of higher genus, since the methods outlined by Davenport [1981] seem completely impractical?
- c) Is there any systematization of the various methods known [Hardy, 1916, pp. 50-51] for reducing particular classes of problems (e.g.

$$\int R(x, \sqrt{x^5+ax^4+bx^2+c}) dx ,$$

where R is a rational function, to elliptic problems?

Transcendental Functions

In this section, we consider purely transcendental elementary functions in the sense of Risch [1969], for which he gave a complete decision procedure. From our present point of view, it contained one weakness - that it was necessary to factor the denominators completely in order to find the logarithmic part of the integral. Rothstein [1976 Theorem 1, p.56; 1977], see also Norman [1982] generalised the theory of residues to cover this case. We can see that the generalisation is more complicated (in terms of a definition: it is efficient as a computational procedure)

than just taking power series coefficients with respect to the most significant variable, by the following two examples, for which the power series with respect to the main variable are the same:

$$\int \frac{1}{x+2} dx = \log(x+2);$$

$$\int \frac{1}{e^x + 2} dx = \frac{x}{2} - \frac{\log(e^x + 2)}{2}.$$

A different type of flaw is that the algorithm is extremely complicated. In this respect the parallel method [Davenport, 1982 and the references cited therein] is much simpler - regrettably it is not always justified. The justification of this method is a major open problem, and the only progress I have to report is a deepening conviction that it is hard, bolstered by examples like:

$$\int \frac{(x+1) e^{x/e^{1/x}}}{e^{2/x}} dx = \frac{(x-e^{1/x}) e^{x/e^{1/x}}}{e^{1/x}},$$

where the derivative of each term in the integral has a denominator of x , but their sum does not (it is interesting to note that the version of MACSYMA currently available to me (version 296) does not solve this integral).

We should note that this method would benefit greatly from an exact determination of the logarithmic parts, since the residues (c_j in the notation of Davenport [1982]) are carried symbolically throughout the calculation. The theorem of Rothstein quoted above does not do this, though, since it only applies to logarithmic terms containing the most significant variable. Furthermore, Rothstein's theorem only applies to integrands from which the rational part has already been removed. Nevertheless, its application would certainly help.

Mixed Functions

This title covers functions that are transcendental, but not purely transcendental, i.e; those that are a mixture of the elementary transcendental functions of the previous section and the algebraic functions, such as $\sin \sqrt{x-1}$, $\sqrt{1+x} + \sin x$. Obviously, the difficulties of the two are combined. Having remarked that a knowledge of the residues was necessary for the integration of algebraic functions, we see that Rothstein's theorem needs to be generalised to the computation of residues of mixed functions, which probably awaits a better method for the calculation of residues of algebraic functions. The currently known method [Davenport, 1981] is based on power series, and it is not at all clear how to generalise it to transcendental integrands, such as

$$\int \frac{\log x}{\text{Tog}(x+1) - 1} dx,$$

which has a fairly bizarre singularity at $x=0$.

The two special cases for which the author knows of solutions are:

- a) An n -th root extension of an elementarily transcendental field of functions [Trager, 1979], with the proviso that the radical extension should not give rise to any logarithms (which implies that it is not necessary to compute the residues);
- b) An elementarily transcendental extension of an algebraic function field [Davenport, 1983], where the residues need only be calculated over the function field, which we know how to do, even if the methods are not as efficient as we would like.

It should be emphasized that a general theory of residues appears to be a necessary condition for an integration method that handles the mixed case: it would not, at the moment be sufficient in general, nor would it permit us to lift the restriction in (a) above. Even for the logarithmic part, it would be necessary to find a theory of divisors of finite order on algebraic surfaces, as well as the appropriate generalisation of Coates' algorithm. However, it is not necessary to have all this machinery working before anything useful can be said. For example, without a theory of divisors of finite order, it is possible to assert "Either this function is unintegrable, or it is integrable and its integral is truly horrible".

For the non-logarithmic part, one reason for the persistence of difficulties is that Trager's [1979] treatment of integrals relies heavily on the fact that the algebraic variable is an n -th root, and it is not currently clear how to remove this restriction.

Also, Risch's integration method reduces ordinary differential equations of the form $y(\theta)' + f(\theta)y(\theta) = \sum c_i g_i(\theta)$ (θ being elementarily transcendental over the underlying field) to a series of similar equations in the underlying field which can be solved sequentially (in other words, the system is triangular). If there are algebraic extension involved, it would appear that we can no longer do this, and that some treatment of systems of equations of the above form will be necessary.

Higher Functions

This section discusses the prospects of integrating functions which are not elementary, i.e. are defined by some process other than the taking of exponentials and logarithms and the solution of algebraic equations. As Moses and Zippel [1979] point out, one can not allow an arbitrary form for the process. They considered in particular the dilogarithm, also known as the Spence function, defined by

$$\text{Sp}(z)' = -z' \frac{\log z}{z-1}.$$

The limited their attention to higher functions satisfying the rule that their

derivative should lie in the field containing the original integrand. Regrettably, the only special function (up to equivalences) which always satisfies this is the logarithm (proof: for this to be true, $f(z)'$ must belong to $K(z)$, which implies that $f(z)$ is the integral of a rational function, and hence can be expressed in terms of rational functions and logarithms). As an example of the weakness of their approach, consider:

$$\int \frac{\log(x-1)}{x} dx = \log x \log(x-1) + Sp(x).$$

Here we have $Sp(x)$ despite the fact that $\log x$ does not occur in the integrand, and $\log x$ in fact appears with a non-constant coefficient. This does not mean, of course, that their theorems are useless, merely that one must be careful about expressing the results, which will be of the form: "your integral can not be expressed in terms of logarithms, exponentials and the following Spence functions ...", rather than "your integral can not be expressed in terms of logarithms, exponentials and Spence functions".

Singer et al. [1981] limit their attention, not unreasonably, to special functions of two kinds: integrals of rational functions of logarithms of rational functions (subject to the caveat of the next paragraph) and integrals of rational functions of exponentials of rational functions. In this context they prove a theorem that says (see their paper for a precise statement) that such functions appear linearly with constant coefficients, thus generalising Liouville's Principle on which all the work described in the preceding sections has been based.

The caveat that Singer et al. impose is that, for a special function of the form $f(z) = H(\log(S(z)))$, where H and S are rational functions of one variable, $\text{degree}(\text{numerator}(S)) < \text{degree}(\text{denominator}(S))+1$. Their paper contains a clear example that the generalized Liouville principle is false without this caveat, but I must admit that I can not find any intuitive explanation for it.

As it stands, their theorem applies to the logarithmic integral

$$li(z) = \int \frac{z'}{\log z} dx ,$$

but not to the exponential integral

$$Ei(u) = \int \frac{e^u}{u} dx .$$

However, we have the equation $li(e^u) = Ei(u)$, so that in fact we can circumvent the problem. Another way of looking at this is less re-assuring, though: it means that the exponential integral, which contains no logarithm, can be integrated by means of li , which one might have thought was only useful for integrands containing logarithms. When added to the algorithmic problems Singer et al. mention, it is clear that a great many problems lie in the way of a more general algorithm.

We should note that their theorem is far from providing an algorithm, since, for example, they are only able to assert that certain integrals have n -th powers in the field of functions which defines the integral, where even n is unknown. rather

than belonging to the field itself.

One further point that needs to be considered is that the theory of integration is intimately connected with that of structure theorems. Structure theorems for non-elementary functions are substantially more complicated [Rothstein & Caviness, 1979], but, which is worse, they also seem to involve new transcendental constants more readily. The above is not a statement that I am in a position to make precise, but nevertheless I feel that it requires attention. We can express, and simplify, expression of the form $P(e^x) e^{x^2}$ easily, where P is the polynomial $\sum a_i x^i$, and they integrate to expressions of the form

$$\sum_{i=0}^{\deg P} c_i \operatorname{Erf}(x-i),$$

$(\operatorname{Erf}(x) = \int e^{-x^2} dx$, where the usual constant is suppressed for clarity). Here the c_i are constants, which are not completely determined by the algebraic formulation (since e^x is not distinguishable from x^e algebraically). Regrettably, if we are to have a consistent analytic interpretation of Erf , the c_i are not necessarily algebraic over $\mathbb{Q}(a_i)$, being, in fact, $a_i e^{-i^2/4}$. Of course, in this case we know that e is transcendental over \mathbb{Q} , but this illustrates the additional problems that arise. It is also linked to the final problem area that we mention: that of analytic problems.

Analytic Problems

The present theory of integration is purely algebraic (as contrasted with Ritt's [1948] development of the original Liouville theory). In most cases, this is a great advantage, and it allows us to pose questions about extensions of the fields of constants, and parameterized integrals, in a way that an analytic theory would not. But it is not the case that all analytic difficulties are avoided. The problem of finding $\int f(x) dx = g(x)$ can indeed be posed purely algebraically, but the problem of evaluating

$$\int_a^b f(x) dx = [g(x)]_a^b$$

has an inherent analytic content.

Also, it is with this kind of problem that our customers are mostly concerned. The problem, of course, is to find an expression for $g(x)$ that can be continued analytically from a to b . While this may appear easy, I should point out that, when B.M. Trager and I were recently engaged on a "large" multiple integral, this part of the problem turned out to be fully as difficult as the task of determining the indefinite integral.

Bibliography

Davenport, 1981

Davenport,J.H., On the Integration of Algebraic Functions. Springer Lecture Notes in Computer Science 102, Springer-Verlag, Berlin-Heidelberg-New York, 1981. Zbl. 471.14009.

Davenport, 1982

Davenport,J.H., On the Parallel Risch Algorithm (I). Proc. EUROCAM 82 (Springer Lecture Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg-New York, 144, 1982) pp. 144-157.

Davenport,1983

Davenport,J.H., Les Equations Différentielles Ordinaires de Risch sur une Courbe Algébrique. To appear (and I.M.A.G. Research Report, April 1983).

Hardy,1916

Hardy,G.H., The Integration of Functions of a Single Variable (2nd. ed.). Cambridge Tract 2, C.U.P., 1916.

Hermite. 1972

Hermite,E., Sur l'intégration des fractions rationnelles. Nouvelles Annales de Mathématiques, 2 Sér., 11(1872) pp. 145-148. Annales Scientifiques de l'Ecole Normale Supérieure, 2 Sér., 1(1872) pp. 215-218.

Loos, 1982,

Loos,R., Computing in Algebraic Extensions. Computing Supplementum 4 (ed. B. Buchberger et al.), Springer-Verlag, Wien-New York, 1982, pp. 173-187.

Moses & Zippel,1979

Moses,J. & Zippel,R.E., An Extension of Liouville's Theorem. Proc. EUROSAM 79 (Springer Lecture Notes in Computer Science 72, Springer-Verlag, Berlin-Heidelberg-New York, 1979) pp. 426-430.

Norman,1982

Norman,A.C., Integration in Finite Terms. Computing Supplementum 4 (ed. B. Buchberger et al.), Springer-Verlag, Wien-New York, 1982, pp. 57-69.

Risch, 1969

Risch,R.H., The Problem of Integration in Finite Terms. Trans. A.M.S. 139(1969) pp. 167-189. MR 38(1969) #5759. Zbl. 184,67.

Risch,1970

Risch,R.H., The Solution of the Problem of Integration in Finite Terms. Bull. A.M.S. 76(1970) pp. 605-608. MR 42(1971) #4530.

Rothstein, 1976

Rothstein,M., Aspects of Symbolic Integration and Simplification of Exponential and Primitive Functions. Ph.D. Thesis, University of Wisconsin, Madison, 1976 (Xerox Univeristy Microfilms 77-8809).

Rothstein. 1977

Rothstein,M., A New Algorithm for the Integration of Exponential and

- Logarithmic Functions. Proc. 1977 MACSYMA Users' Conference (NASA Publication CP-2012, National Technical Information Service, Springfield, Virginia, 1977) pp.263-274.
- Rothstein & Caviness, 1979
 Rothstein,M. & Caviness,B.F., A Structure Theorem for Exponential and Primitive Functions. SIAM J. Comp 8(1979) pp. 357-367. MR 80m:12028.
- Singer et al., 1981
 Singer,M.F., Saunders,B.D. & Caviness,B.F., An Extension of Liouville's Theorem on Integration in Finite Terms. Proc. SYMSAC 81 (ACM, New York, 1981), pp. 23-24. Zbl. 482.12008.
- Trager, 1976
 Trager,B.M., Algebraic Factoring and Rational Function Integration. Proc. SYMSAC 76 (ed. R.D. Jenks), ACM. New York, 1976, pp. 219-226.
- Trager, 1979
 Trager,B.M., Integration of Simple Radical Extensions. Proc. EUROSAM 79 (Springer Lecture Notes in Computer Science 72, Springer-Verlag, Berlin-Heidelberg-New York, 1979) pp. 408-414. MR 82j:12032.
- Winograd, 1979
 Winograd,S., On Multiplication in Algebraic Extension Fields. Theor. Comp. Sci. 8(1979) pp. 359-377.
- Winograd, 1980
 Winograd,S., On the Multiplication of Polynomials Modulo a Polynomial. SIAM J. Comp. 9(1980) pp. 225-229. Zbl. 446.68032.
- Yun, 1977
 Yun,D.Y.Y., Fast Algorithm for Rational Function Integration. Proc. IFIP 77 (ed. B. Gilchrist), North-Holland, Amsterdam-New York-Oxford. 1977. pp. 493-498 [IBM Research Report RC 6563 6th. January 1977].
- Zejli, 1983
 Zejli,H., Private Communication.
- Zolotarev, 1874
 Zolotarev,E., Sur la méthode d'intégration de M. Tchebycheff. Math. Annalen Band V Sér 560. Journal de Mathématiques 2 Sér. 19(1874).

The Euclidean Algorithm for Gaussian Integers

Heinrich Rolletschek
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19711/USA

Abstract

A theorem by Lame' (1845) answers the following questions: given N , what is the maximum number of divisions, if the Euclidean algorithm is applied to integers u, v with $N \geq u > n \geq 0$? In this paper we give an analogous result for the Euclidean algorithm applied to Gaussian integers, that is, complex numbers $a+bi$, where a and b are integers.

1. The problem. As is well-known, there exist a number of algebraic extension fields of \mathbb{Q} , the field of rational numbers, in which the algebraic integers form a Euclidean domain, for instance, the quadratic number fields $\mathbb{Q}(\sqrt{D})$ for $D = -3, -2, -1, 2, 3, 5$ and several other values of D , see, for instance, Hasse [2], section 16.6. In $\mathbb{Q}(i)$ the algebraic integers are the Gaussian integers, the complex numbers of the form $a+bi$, $a, b \in \mathbb{Z}$. (We denote by \mathbb{Z} the set of rational integers, by $\mathbb{Z}[i]$ the set of Gaussian integers; of course, $i = \sqrt{-1}$.) An efficient variant of the Euclidean algorithm in $\mathbb{Z}[i]$ is given and investigated in Caviness/Collins[1]. The following question is left open.

Problem (C): Given N , what is the maximal number of divisions, when the Euclidean algorithm is applied to Gaussian integers u, v , $N \geq |u| \geq |v|$?

The analogous problem for the integral domain \mathbb{Z} was solved by Lame' (1845), see, for instance, Knuth [3], section 4.5.3. More precisely, Lame's result gives the solution for the dual problem, which can be stated as follows:

Problem (D): Given n , find u, v such that

- i) $|u| \geq |v|$,
- ii) The Euclidean algorithm, applied to u, v , takes exactly n steps,
- iii) $|u|$ is minimal satisfying i), ii).

The answer for the domain Z is $u = F_{n+2}$, $v = F_{n+1}$ for $n \geq 2$, where F_0, F_1, \dots are the Fibonacci numbers, defined by

$$F_0 = 0, F_1 = 1,$$

$$F_r = F_{r-2} + F_{r-1} \text{ for } r \geq 2.$$

Of course, each of the problems (C) and (D) can be stated in exactly the same way for the integral domains Z and $Z[i]$; for both domains, a solution for (C) can easily be derived from a solution for (D).

In $Z[i]$ the Euclidean algorithm is determined as soon as division with remainder is defined. This division with remainder is given by $u = qv+r$, where the integer quotient q is chosen such that $|r|$ becomes minimal. If several values of q satisfy this condition, then any effective selection will do. Caviness/Collins[1] put $q = \lfloor a+\frac{b}{2} \rfloor + \lfloor b+\frac{a}{2} \rfloor$, where $a+bi = u/v$ is the (exact) quotient in the field of complex numbers. In this paper, however, we will only require that r be minimal, q need not be determined according to [1]. Thus the Euclidean algorithm won't be completely specified.

Next we point out an important difference between the solution of problem (D) in Z on the one hand, in $Z[i]$ on the other. Let R be either Z or $Z[i]$. Then consider sequences $\Sigma_n^R = \langle w_1, \dots, w_{n+1} \rangle$, which are built as follows: let (u, v) be a solution of the dual problem in integral domain R for the given value of n . Then let $w_{n+1} = u$, $w_n = v$, and let $\langle w_{n-1}, \dots, w_1 \rangle$ be non-zero remainders of the divisions, when the Euclidean algorithm for R is applied to u, v .

The sequences Σ_n^Z are uniquely determined, and it follows immediately from Lame's result that Σ_n^Z is an initial segment of Σ_{n+1}^Z for $n \geq 2$. The case $n=1$ is an exception, since $\Sigma_1^Z = \langle 1, 1 \rangle$.

These remarks don't carry over to $R = Z[i]$. The sequences $\Sigma_n^{Z[i]}$ are not uniquely determined, though sample computations make it likely that for $n \geq 2$ they are uniquely determined up to trivial modifications. But no matter how these sequences are chosen, $\Sigma_n^{Z[i]}$ won't be an initial segment of $\Sigma_{n+1}^{Z[i]}$. For instance, a brute-force algorithm shows that the following are possible choices of $\Sigma_1^{Z[i]}, \dots, \Sigma_4^{Z[i]}$:

$$\begin{aligned}\Sigma_1^{\mathbb{Z}[i]} &= \langle 1, 1 \rangle, \\ \Sigma_2^{\mathbb{Z}[i]} &= \langle i, 1+i, 2+i \rangle, \\ \Sigma_3^{\mathbb{Z}[i]} &= \langle 1, 1-i, 3-2i, 3+2i \rangle, \\ \Sigma_4^{\mathbb{Z}[i]} &= \langle 1, 1-i, 3i, 7-i, 7+2i \rangle.\end{aligned}$$

One reason why $\Sigma_{n+1}^{\mathbb{Z}[i]}$ cannot be formed from $\Sigma_n^{\mathbb{Z}[i]}$ by attaching an element at the end is that in general u, v cannot be the remainders of two consecutive divisions, when the Euclidean algorithm is applied to larger Gaussian integers u', v' , as we will now show.

Consider a division with remainder of Gaussian integers s, t , given by $s = qt + r$. As stated, q is a Gaussian integer such that $|r| = |s - qt|$ is minimal. Equivalently, 0 is one of the Gaussian integers closest to $\alpha = q - s/t = -r/t$. Hence

$$\alpha = x + yi \text{ with } |x| \leq \frac{1}{2}, |y| \leq \frac{1}{2}. \quad [1]$$

We define $S_1 = \{x + iy \mid |x| \leq \frac{1}{2}, |y| \leq \frac{1}{2}\}$. Now consider a sequence $\Sigma = \langle w_1, \dots, w_{n+1} \rangle \in (\mathbb{Z}[i])^{n+1}$, where w_{n-1}, \dots, w_1 is a possible sequence of remainders, when the Euclidean algorithm is applied to w_{n+1}, w_n . Applying [1] to $s = w_{k+1}, t = w_k$, we get: $w_k/w_{k+1} \in S_1$ for $k=1, \dots, n-1$. Σ can be an initial segment of a longer sequence built according to the same scheme, if and only if this condition holds for $k=n$, too. This leads to the following.

Definition 1: A sequence $\Sigma = \langle w_1, \dots, w_{n+1} \rangle \in (\mathbb{Z}[i])^{n+1}$ is legal, iff

- i) w_{n-1}, \dots, w_1 is one of the possible sequences of non-zero remainders, when the Euclidean algorithm is applied to w_{n+1}, w_n and
- ii) $w_n/w_{n+1} \in S_1$.

If in addition w_{n+1} is minimal under conditions i), ii), then Σ is n-minimal.

Instead of the dual problem, we will solve

Problem (M): Given n , find an n -minimal sequence.

2. The construction of an n -minimal sequence. It was conjectured by B. Caviness (private communication) that an n -minimal sequence is given by $\tilde{\Sigma}_n = \langle \tilde{w}_1, \dots, \tilde{w}_{n+1} \rangle$, where

$$w_k = \begin{cases} E_{k-1} + i E_{k-1} & \text{for even } k \geq 2, \\ E_{k-1} & \text{for odd } k, \end{cases}$$

with

$$\begin{aligned} E_0 &= E_1 = 1, \\ E_{2j} &= 2E_{2j-1} + E_{2j-2}, \\ E_{2j+1} &= E_{2j} + E_{2j-1} \end{aligned} \quad \left. \right\} \text{for } j \geq 1$$

In this section we will give a proof for this conjecture (corollary 1).

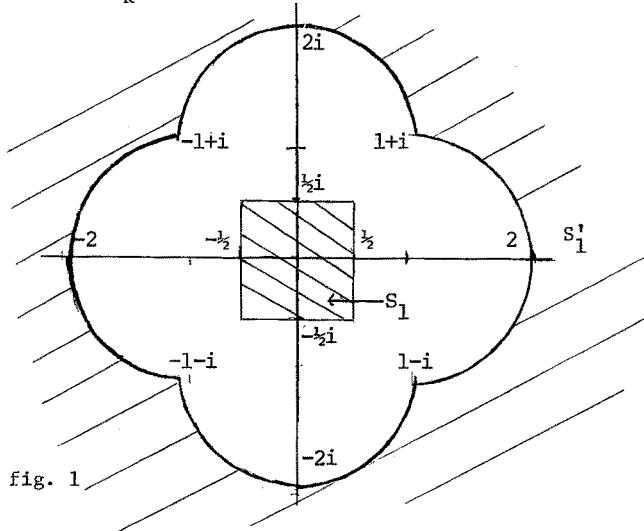
Consider a sequence w_1, \dots, w_{n+1} of complex numbers $w_i \neq 0$; it will be a legal sequence in most cases. Formally we put $w_0 = 0$. Then we introduce the following notation:

$$\begin{aligned} \alpha_k &= w_k / w_{k+1}, \quad k=0, \dots, n, \\ w_k &= w_{k+1} / w_k, \\ \text{If } <w_1, \dots, w_{n+1}> \text{ is legal, then} \\ q_k &\text{ is defined by } w_{k+1} = q_k w_k + w_{k-1} \end{aligned} \quad \left. \right\} \quad \text{k=1, \dots, n.} \quad [2]$$

For sequences $<w'_1, \dots, w'_{n+1}>$ we will similarly use α'_k etc., without repeating the definitions. Dividing the last equality (definitions of q_k) by w_k , we get

$$\omega_k = q_k + \alpha_{k-1} \quad \text{for } k=1, \dots, n. \quad [3]$$

If $<w_1, \dots, w_{n+1}>$ is legal, then the condition $\alpha_k \in S_1$ induces a condition for ω_1 . A simple calculation shows that ω_k must not lie inside any of the four circles with radius 1 and centers $1, i, -1, -i$, and the value 0 must be excluded, too, of course. All this has already been shown in Caviness/Collins [1]. The region S , and the region S'_1 of all possible values of ω_k are shown in fig. 1.



We summarize the facts about legal sequences in the following

Lemma 1: Let $\langle w_1, \dots, w_{n+1} \rangle$ be a legal sequence. Then $\alpha_k \in S_1$ for $k=0, \dots, n$, and $w_k \in S'_1$ for $k=1, \dots, n$. Also, $w_k = q_k + \alpha_{k-1}$. Conversely, let $\Sigma = \langle w_1, \dots, w_{n+1} \rangle$ be any sequence of Gaussian integers $\neq 0$, such that $w_{k+1} = q_k w_k + w_{k-1}$ for $k=1, \dots, n$ for Gaussian integers q_1, \dots, q_n (putting $w_0=0$), and such that $\alpha_k \in S_1$, or, equivalently, $w_k \in S'_1$ for $k=1, \dots, n$. Then Σ is legal.

It is natural to construct a legal sequence from bottom up: after w_1, \dots, w_k have been defined, it suffices to choose q_k such that $q_k + \alpha_{k-1} \in S'_1$, which determines w_{k+1} . Lemma 1 implies that the resulting sequence will indeed be legal. The next, fairly obvious lemma shows that we can start with $w_1=1$:

Lemma 2: For every $n \geq 1$, there exists an n -minimal sequence $\langle w_1, \dots, w_{n+1} \rangle$ with $w_1=1$. In every n -minimal sequence $\langle w_1, \dots, w_{n+1} \rangle$, w_1 is one of the units $1, i, -1, -i$.

Proof: Let $\Sigma' = \langle w'_1, \dots, w'_{n+1} \rangle$ be n -minimal. The value w'_1 is the result of the Euclidean algorithm, the greatest common divisor of w'_{n+1} and w'_n , and also of all w'_k . Define $w_k = w'_k/w'_1$, $\Sigma = \langle w_1, \dots, w_{n+1} \rangle$. Then Σ is legal, too, $w_1 = 1$ and $|w_{n+1}| \leq |w'_{n+1}|$. Since Σ' was n -minimal, it follows that in fact $|w_{n+1}| = |w'_{n+1}|$, w'_1 is a unit and Σ is also n -minimal. \square

It is clear that certain values of w_k like $1+i + (0.2 + 0.3i)$ on the one hand, $1-i + (0.3 - 0.2i)$ on the other are not essentially different. In order to reduce number of cases to be considered to a minimum, it is mandatory not to consider such cases separately. This leads to

Definition 2: Let $z = x + iy$. A complex number z' is equivalent to z ($z \sim z'$), iff z' is one of the numbers $\pm x \pm iy$, $\mp y \pm ix$, that is, z' is an associate of z or the conjugate of an associate of z .

Assume we are given a legal sequence $\langle w_1, \dots, w_{n+1} \rangle$, and we have constructed an initial segment $\langle w'_1, \dots, w'_k \rangle$ ($k \leq n$) of another legal sequence, such that $\alpha'_{k-1} \sim \alpha_{k-1}$. Then we can choose q'_k, \dots, q'_n such that $w'_\ell \sim w_\ell$ for $\ell = k, \dots, n$.

The crucial properties of \sim are given in the following

Lemma 3:

- i) \sim is an equivalence relation, compatible with taking the absolute and the

reciprocal value and with membership in S_1 and S'_1 .

- ii) If $p, p' \sim p$ and q are given, then there exists $q' \sim q$ such that $p'+q' \sim p+q$.
 Similarly, if p, q and $r' \sim p+q$ are given, then there exist $p' \sim p$ and $q' \sim q$
 such that $p' + q' = r'$.

Proof: Immediate. \square

It will be convenient to consider the following more general variant of the notion of legal sequence:

Definition 3: A sequence $\Sigma = \langle w_1, \dots, w_{n+1} \rangle$ of complex numbers $w_k \neq 0$ is pseudolegal, iff $\alpha_k \in S_1$, and $w_k = q_k + \alpha$ for some Gaussian integer q_k and for some $\alpha \sim \alpha_{k-1}$ ($k=1, \dots, n$).

If Σ is in fact legal, then we can choose q_k according to [2], (p. 4), and $\alpha = \alpha_{k-1}$. Hence our notation is consistent.

Definition 4: Two sequences $\Sigma = \langle w_1, \dots, w_{n+1} \rangle$ and $\Sigma' = \langle w'_1, \dots, w'_{n+1} \rangle$ are equivalent ($\Sigma \sim \Sigma'$), if $w_1 \sim w'_1$ and $w_k \sim w'_k$ for $k=1, \dots, n$.

Lemma 4:

- a) If $\langle w_1, \dots, w_{n+1} \rangle \sim \langle w'_1, \dots, w'_{n+1} \rangle$, then $|w_j| = |w'_j|$ for all $j \leq n+1$.
- b) Let $\Sigma = \langle w_1, \dots, w_{n+1} \rangle$ with $w_1 \in Z[i]$. Then there exists a legal sequence $\Sigma' = \langle w'_1, \dots, w'_{n+1} \rangle \sim \Sigma$.

Proof:

a) Immediate, since $w_k \sim w'_k$ implies $|w_k| = |w'_k|$.

b) The elements w'_k can be defined in a straightforward way by induction on k , using lemma 3ii). \square

We use the following additional notation:

$Q_k = \{r(\sin \phi + i \cos \phi) \mid r \geq 0, (k-1)\pi/4 \leq \phi \leq k\pi/4\}$ ($k=1, \dots, 8$),
 (k) is the (uniquely determined) element $z' \sim z$ such that $z' \in Q_k$. For instance, if $z = 2+i$, then $z^{(1)} = z$, $z^{(2)} = 1+2i$, $z^{(3)} = -1+2i$ etc. Note that the numbers $z^{(1)}, \dots, z^{(8)}$ are not necessarily different.

We now return to the problem of constructing a legal sequence $\langle w_1, \dots, w_{n+1} \rangle$, attempting to make $|w_{n+1}|$ as small as possible. The next question we answer is the following: assume we have already defined w_1, \dots, w_k , how do we have to choose q_k in order to minimize $|w_{k+1}|$, or, equivalently, $|w_k| = |q_k + \alpha_{k-1}|$? An obvious try would be to make $|q_k|$ as small as possible. However, it is obvious from fig. 1 that if we

choose for q_k one of the smallest values 1, i , -1, $-i$, then we would not meet the condition $q_k + \alpha_{k-1} \in S'_1$. On the other hand, we can always choose one of the values $\pm i$ for q_k , depending on α_{k-1} . For instance, if $\alpha_{k-1} \in Q_1 \cup Q_2$, then $q_k = 1+i$ works. The remaining cases can be reduced to this one by applying lemma 3 ii).

The sequences $\tilde{\Sigma}_n$ defined at the beginning of this section are actually constructed that way. More precisely, the following holds:

Lemma 5: Let $\tilde{\Sigma}_n$ be the sequence $\langle \tilde{w}_1, \dots, \tilde{w}_{n+1} \rangle$, defined at the beginning of this section. Then $\tilde{\Sigma}_n$ is legal, and

$$\tilde{q}_k = \begin{cases} 1+i & \text{for } k \text{ odd,} \\ 1-i & \text{for } k \text{ even.} \end{cases}$$

Proof: Straightforward, by induction on k . \square

For some values of α_{k-1} , $1+i+\alpha_{k-1}^{(8)}$, $2+\alpha_{k-1}^{(3)}$ and equivalent values are also legal values of ω_k , that is, elements of S'_1 . The set of all these values α_{k-1} forms a region S_2 , which can be determined readily. Equivalently, ω_{k-1} must lie in a region S'_2 , which can be determined similarly as S'_1 ; the result is $S'_2 = S_2 \cap \{x+yi \mid 1 \leq |x| \leq 2 \text{ or } |y| \leq 2\}$. S_2 and S'_2 are shown in fig. 2.

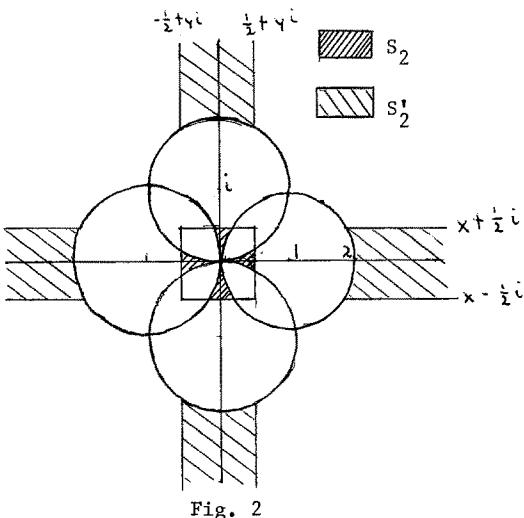


Fig. 2

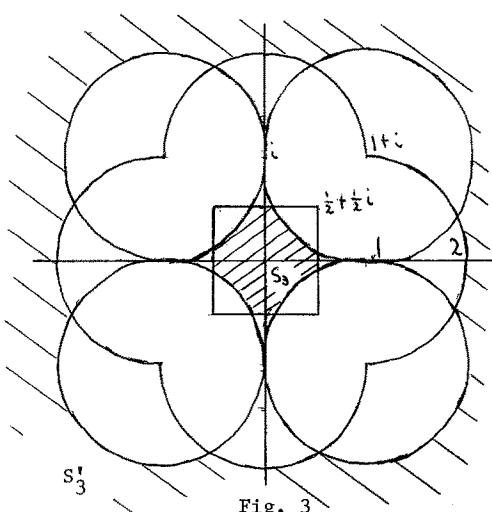


Fig. 3

A weaker condition $\alpha_{k-1} \in S_3$ ($\omega_{k-1} \in S'_3$) is necessary and sufficient in order that $2+i+\alpha_{k-1}^{(5)}$, $2+i+\alpha_{k-1}^{(6)}$ are legal values for ω_k . Curiously, both α_{k-1} and ω_{k-1} must lie outside or on the boundary of the circles with radius 1 and centers $1+i$, $-1+i$, $-1-i$, $1-i$. S_3 and S'_3 are given in Fig. 3. Note that all the regions S_i , S'_i are closed.

The question stated at the bottom of page 6 can now be answered easily by all possible values of ω_k of the form $q + \alpha_{k-1}^{(j)}$ ($q \in \mathbb{Z}[i]$, $1 \leq j \leq 8$) which are not "obviously too large".

Lemma 6: Let $\langle w_1, \dots, w_{n+1} \rangle$ be a legal sequence, $1 \leq k \leq n$. If $|\omega_k|$ has the smallest possible value for the given value of α_{k-1} , then

$$\omega_k \sim \begin{cases} 1+i+\alpha_{k-1}^{(8)} & \text{for } \alpha_{k-1} \in S_2, \\ 1+i+\alpha_{k-1}^{(1)} \text{ or } 2+i+\alpha_{k-1}^{(5)} & \text{for } \alpha_{k-1} \in S_3 - S_2 \\ 1+i+\alpha_{k-1}^{(1)} & \text{for } \alpha_{k-1} \in S_3. \end{cases} \quad [4]$$

Definition 5: Let $\langle w_1, \dots, w_{n+1} \rangle$ be a legal sequence. Then ω_k is constructed according to strategy (0), iff ω_k satisfies [4].

In order to prove the main theorem, some additional preparation is needed.

Lemma 7:

- i) Assume that in a legal sequence $\langle w_1, \dots, w_{n+1} \rangle$ ω_k has been constructed according to strategy (0). Then $\alpha_k \notin S_3$. Hence, if ω_{k+1} has also been constructed according to strategy (0), then $\omega_{k+1} \sim 1+i+\alpha_k^{(1)}$.
- ii) $\alpha_k \in S_2$ if and only if $\omega_k \sim q + \alpha_{k-1}^{(j)}$, where q is a (rational) integer ≥ 2 .

Lemma 8: Assume that t_0, t_1 are given and that for $j \geq 2$, t_j is defined by one of the following recursion formulas.

- i) Let

$$t_j = \begin{cases} (1-i) t_{j-1} + t_{j-2} & \text{for } j \geq 2, \text{ even,} \\ (1+i) t_{j-1} + t_{j-2} & \text{for } j \geq 3, \text{ odd.} \end{cases} \quad [5]$$

Then

$$t_{2j} = \frac{((\sqrt{3}-1)t_0 + (1-i)t_1)(2+\sqrt{3})^j - ((-\sqrt{3}-1)t_0 + (1-i)t_1)(2-\sqrt{3})^j}{2\sqrt{3}},$$

$$t_{2j+1} = \frac{((\sqrt{3}+1)t_1 + (1+i)t_0)(2+\sqrt{3})^j - ((-\sqrt{3}+1)t_1 + (1+i)t_0)(2-\sqrt{3})^j}{2\sqrt{3}}.$$

- ii) Let

$$t_j = \begin{cases} -it_{j-2} + 2t_{j-1} & \text{for } j \geq 2, \text{ even,} \\ it_{j-2} + 2t_{j-1} & \text{for } j \geq 3, \text{ odd.} \end{cases} \quad [6]$$

Then

$$t_{2j} = \frac{((-2+\sqrt{3}-i)t_0+2t_1)(2+\sqrt{3})^j - ((-2-\sqrt{3}-i)t_0+2t_1)(2-\sqrt{3})^j}{2\sqrt{3}},$$

$$t_{2j+1} = \frac{((2+\sqrt{3}+i)t_1-2it_0)(2+\sqrt{3})^j - ((2-\sqrt{3}+i)t_1-2it_0)(2-\sqrt{3})^j}{2\sqrt{3}}.$$

Proof: Straightforward, by induction on j . These formulas can also be derived by applying the well-known technique of generating functions.

Lemma 8 will be applied to sequences whose elements are denoted differently, for instance, t'_0, t'_1, \dots , where [5] or [6] holds with t_m being replaced by t'_m for all indices m . In this case we say that the numbers t'_j are defined by [5] or [6] without pointing out the deviation of notation.

The following statement follows readily from lemma 8 i), but it can also be proved directly by induction on j :

Lemma 9: Let t_0, t_1, \dots be defined by [5], as in lemma 8 i). Then for all j , $t_j = [c_1 t_0 + c_2 (1-i)t_1]x$, where c_1, c_2 are real numbers 0 and $x = \begin{cases} 1 & \text{for } j \text{ even}, \\ 1+i & \text{for } j \text{ odd}. \end{cases}$. Furthermore, $c_1 \leq c_2 \leq 2c_1$ for $j \geq 2$.

For we are ready to prove the main theorem.

Theorem 1: Let $\langle w_1, \dots, w_{n+1} \rangle$ be a legal sequence, where some w_j has not been constructed according to strategy (0). Then there exists a legal sequence $\langle w'_1, \dots, w'_{n+1} \rangle$ such that $|w'_{n+1}| < |w_{n+1}|$ and $|w'_j| \leq |w_j|$ for all $j \leq n+1$.

Proof: Let k be the largest j such that w_j has not been constructed according to strategy (0). If $k=n$, then define $w'_j = w_j$ for $j=1, \dots, n$, and construct w'_n according to strategy (0). Then by lemma 6, $|w'_n| < |w_n|$, hence $|w'_{n+1}| < |w_{n+1}|$ as required.

Now assume $k < n$, so that w_{k+1} has been constructed according to strategy (0). Then w_{k+1} may be equivalent to $1+i+\alpha_k^{(1)}$, $1+i+\alpha_k^{(8)}$ or $2+i+\alpha_k^{(5)}$.
Case 1: $w_{k+1} \sim 1+i+\alpha_k^{(1)}$. We can assume w.l.o.g. that $\alpha_k \in Q_1$, so that $\alpha_k \in Q_8$, $q_{k+1} = 1-i$, $q_{k+2} = 1+i$ etc. Define $t_j = w_{k+j}/w_k$. Then $t_0 = 1$, $t_1 = w_k$, and the numbers t_j are defined by [5] for $j \geq 2$. Hence, t_j has a representation $[c_1 + c_2 (1-i)w_k]x$ according to lemma 9.

In the legal sequence $\langle w'_1, \dots, w'_{n+1} \rangle$ to be defined, $w'_j = w_j$ for $j=1, \dots, k$, and w'_k will be constructed according to strategy (0), as are w'_j and w'_j for $j > k$. First we define a number w , which will be equivalent to w'_k , by

$$\omega = \begin{cases} 1+i+\alpha_{k-1}^{(8)} & , \text{ if } \alpha_{k-1} \in S_2, \\ 2+i+\alpha_{k-1}^{(5)} & , \text{ if } \alpha_{k-1} \in S_3 - S_2 \text{ and } \omega_k = 2+i+\alpha_{k-1}^{(6)}, \\ 1+i+\alpha_{k-1}^{(1)} & \text{otherwise.} \end{cases}$$

Let $t'_0 = 1$, $t'_1 = \omega$, and let t'_j be defined by [5] for $j \geq 2$.

Also, let $\Delta\omega = \omega_k - \omega$.

Then $\Delta\omega \in Q_1 \cup Q_2 \cup Q_8$, as can readily be seen by considering the possible values $1+i+\alpha_{k-1}^{(1)}$, $2+\alpha_{k-1}^{(3)}$, $2+\alpha_{k-1}^{(2)}$, $2+i+\alpha_{k-1}^{(5)}$, $2+i+\alpha_{k-1}^{(6)}$ for ω_k ; for other, "worse" values of ω_k , the statement follows immediately. Furthermore, $\Delta\omega \neq 0$, since ω_k was not constructed according to strategy (0). Now, for even j , $t'_j \in Q_8$, and

$\Delta t'_j = t'_j - t'_j = c_2(1-i) \Delta\omega \in Q_1 \cup Q_7 \cup Q_8$. Hence the angle between t'_j and $\Delta t'_j$ is $\leq \pi/2$, therefore $|t'_j| = |t'_j + \Delta t'_j| \geq |t'_j|$. This inequality also holds for odd j , since the additional factor x has no influence on it.

Define

$$w''_j = \begin{cases} w_j & \text{for } j \leq k, \\ w_k \cdot t'_{j-k} & \text{for } j \geq k. \end{cases}$$

(Both cases of the definition apply for $j=k$.)

Then $\Sigma'' = \langle w''_1, \dots, w''_{n+1} \rangle$ is pseudolegal. Also, $|w''_j| = |w_k| \cdot |t'_{j-k}| < |w_k| \cdot |t'_{j-1}| = |w_j|$ for $j > k$. By lemma 4, there exists a legal sequence $\Sigma' = \langle w'_1, \dots, w'_{n+1} \rangle \sim \Sigma''$ with $w'_j = w''_j$. Then Σ' satisfies all requirements.

Case 2: $\omega_{k+1} \sim 2+i+\alpha_k^{(5)}$. Define $t'_j = \frac{w_{k+j+1}}{w_k}$ ($j=-1, 0, \dots$). Again t'_j can be expressed in terms of t'_{-1} , t'_0 , using lemma 8. In the sequence $\langle w'_1, \dots, w'_{n+1} \rangle$, w'_k, \dots, w'_{n+1} are all constructed according to strategy (0), which corresponds to a sequence t'_{-1}, t'_0, \dots , as in case 1. An easy comparison shows that $|t'_j| < |t'_j|$ for $j \geq 1$, using the fact that $t'_0 \in S_3$, hence $|w'_j| < |w_j|$ for $j \geq k+1$. For a detailed proof, see Rolletschek [5].

Case 3: $\omega_{k+1} \sim 1+i+\alpha_k^{(8)}$. This is the most difficult case. We have to define m to be the largest $m \leq k$ such that $\omega_m \not\sim 2+\alpha_{m-1}^{(3)}$. w'_j can be expressed in terms of w_{m-1}, w_m , since we may assume $\omega_m \sim 2+\alpha_{m-1}^{(j)}$, $j \in \{0, 1\}$. We have to use both parts of lemma 8 and the fact that w_{m+1}, \dots, w_k are defined from w_{m-1}, w_m by [6] up to equivalence. In the sequence $\Sigma' = \langle w'_1, \dots, w'_{n+1}, w'_m, \dots, w'_n \rangle$ are all constructed according to strategy (0). A lengthy, but fairly straight forward calculation shows again that

Σ satisfies the requirements fo the theorem.

Corollary 1: The sequence $\tilde{\Sigma}_n$ defined on p. 2 is n-minimal.

Proof: Assume $\Sigma = \langle w_1, \dots, w_{n+1} \rangle$ is an n-minimal sequence with $w_1=1$. In this sequence, w_1, \dots, w_n must be constructed according to strategy (0) by theorem 1. Hence $q_1 \sim 1+i$ by definition of strategy (0), since $\alpha_0=0$, and $q_i \sim 1+i$ for $i=2, \dots, n$ by lemma 7 i). We may choose $q_1 = 1+i$; other choices lead to equivalent sequences.

Then

$$q_k = \begin{cases} 1+i & \text{for } k \text{ odd,} \\ 1-i & \text{for } k \text{ even, } \geq 2. \end{cases}$$

(A more general statement than this was pointed out after the proof of lemma 7.) By lemma 5, the sequence Σ constructed that way is just the sequence $\tilde{\Sigma}$. This argument shows that every n-minimal sequence Σ with $w_1=1$ satisfies $\Sigma \sim \tilde{\Sigma}$. However, an n-minimal sequence $\langle w_1, \dots, w_{n+1} \rangle$ with $w_1=1$ does exist by lemma 2. Hence Σ must be one.

3. Conclusion. While problem (M) is solved by corollary 1, problem (D) remains open. We recall the definition of $\Sigma_n^{Z[i]} = \langle w_1, \dots, w_{n+1} \rangle$ on p. 1. This sequence satisfies part ii) of definition 1, and $|w_{n+1}| \leq |w_n|$. Generally we have

Definition 6: A sequence $\Sigma = \langle w_1, \dots, w_{n+1} \rangle \in (Z[i])^{n+1}$ is near-legal, if $|w_{n+1}| \geq |w_n| > 0$ and $\langle w_{n-1}, \dots, w_1 \rangle$ is a possible sequence of remainders, when the Euclidean algorithm is applied to w_{n+1}, w_n . If in addition $|w_{n+1}|$ is minimal under these conditions, then is n-minimal - near-legal.

Like legal sequences, near-legal sequences may be constructed from bottom up. The significance of the sets S_2, S'_2, S_3, S'_3 carries over to the last step: w_n may be chosen equivalent to $1+\alpha_{n-1}^{(3)}$, if $\alpha_{n-1} \in S_2$, equivalent to $1+i+\alpha_{n-1}^{(5)}$, if $\alpha_{n-1} \in S_3$.

Sample computations indicate that an n-minimal-near-legal sequence

$\Sigma = \langle w_1, \dots, w_{n+1} \rangle$ is given by

$$w_1 = 1$$

$$\omega_k = \begin{cases} 1-i+\alpha_{k-1} & \text{for } k \text{ even,} \\ 1+i+\alpha_{k-1} & \text{for } k \text{ odd} \end{cases} \quad \left\{ \begin{array}{l} 1 \leq k \leq n-2, \\ \end{array} \right.$$

$$\omega_{n-1} = 2+\alpha_{n-2},$$

$$\omega_n = \begin{cases} i+\alpha_{n-1} & \text{for } n \text{ even,} \\ -i+\alpha_{n-1} & \text{for } n \text{ odd.} \end{cases}$$

However, this assertion has not been proved.

On the other hand, the answer to problem (M) suffices for an almost complete solution of (C). Again let $\langle \tilde{w}_1, \dots, \tilde{w}_{n+1} \rangle$ be the n -minimal sequence defined at the beginning of section 2, and let $\langle w_1, \dots, w_{n+1} \rangle$ be n -minimal - near-legal. Then $|w_{n+1}| \leq |\tilde{w}_{n+1}|$, since every legal sequence is near-legal, and $|w_{n+1}| \geq \tilde{w}_n$, since $\langle w_1, \dots, w_n \rangle$ is legal and $\langle \tilde{w}_1, \dots, \tilde{w}_n \rangle$ is $n-1$ -minimal (except for the case $n=1$, which can be considered separately).

The numbers w_2, w_3, \dots are defined from $w_0=0, w_1=1$ by a recursion formula analogous to [5], p. 8, except that $l+i$ and $l-i$ must be interchanged. Using lemma 7 i), we get after simplification:

$|w_k|^2 = \frac{1}{6}[(2+\sqrt{3})^k + 2(-1)^k + (2-\sqrt{3})^k]$ for every k . For given N let k_N be the largest k such that $|w_k| \leq N$. Then

$$k_N = \lfloor \log_{2+\sqrt{3}} (6N^2+3) \rfloor.$$

For an almost complete solution we have to combine this formula with the inequality $|\tilde{w}_n| \leq |w_{n+1}| \leq |\tilde{w}_{n+1}|$. The result is

Theorem 2: Let $N > 0$ be given. The maximum number n of deviations, when the Euclidean algorithm is applied to Gaussian integers $u, v, |N| \geq |u| \geq |v|$, is either $\lfloor \log_{2+\sqrt{3}} (6N^2+3) \rfloor$ or $\lfloor \log_{2+\sqrt{3}} (6N^2+3) \rfloor - 1$; hence $n = \log_{2+\sqrt{3}} N + M$ $= 1.0526 \log_2 N + M$ with $-1 \leq M \leq 2$.

Of course it can be shown easily that n grows logarithmically as a function of N , but the constant factor $1.0526 = \log_{2+\sqrt{3}} 2$ from theorem 2 can only be derived from corollary 1.

References:

- [1] B. F. Caviness, G. E. Collins: Algorithms for Gaussian Integer Arithmetic. In: Proceedings of the 1976 Symposium on Symbolic and Algebraic Computation.
- [2] H. Hasse: Vorlesungen ueber Zahlentheorie. Springer-Verlag, Berlin. 1964.
- [3] D. E. Knuth: The Art of Computer Programming. Vol. 2: Seminumerical Algorithms Addison-Wesley, Reading, Massachusetts 1981.
- [4] H. Rolletschek: The Euclidean Algorithm for Gaussian Integers. Technical report, University of Delaware, to appear.

MULTI POLYNOMIAL REMAINDER SEQUENCE AND
ITS APPLICATION TO LINEAR DIOPHANTINE EQUATIONS

Akio Furukawa^{*)} and Tateaki Sasaki^{**)}

*) Department of Mathematics, Tokyo Metropolitan University
Setagaya-ku, Tokyo 158, Japan

**) The Institute of Physical and Chemical Research
Wako-shi, Saitama 351, Japan

§1. Motivation and introduction

The polynomial remainder sequence (PRS in short) is one of the most useful concepts in computer algebra, and it was thoroughly investigated by Habicht[1], Collins[2,3], Brown and Traub[4], and Brown[5]. (See, also [6].) We can regard the calculation of PRS as a reduction of a set of two polynomials, that is, $(P_1(x), P_2(x)) \rightarrow (P_2(x), P_3(x)) \rightarrow \dots \rightarrow (P_k(x), P_{k+1}(x))$, where the reduction is made by eliminating high degree terms of a polynomial in each set. In many cases of algebraic computation, we encounter the necessity of reducing not only a set of two polynomials but also a set of $m(\geq 3)$ polynomials. A typical example is the calculation of elementary divisors of a matrix with polynomial elements. As is well known, the main step of this calculation is to transform the matrix into a diagonal form by applying row/column eliminations repeatedly. The row/column elimination is nothing but the calculation of remainders or pseudo-remainders of a set of polynomials by choosing a suitable element of the matrix as the divisor.

The notion of reduction of a set of more than two polynomials leads us to a concept of multi-polynomial remainder sequence (multi-PRS in short) as a natural generalization of PRS. As for the conventional PRS, an elegant theory of subresultant has been developed [1,2,3,4,5] making the calculation of PRS quite efficient. We reasonably expect that the concept of subresultant can be generalized to the case of multi-PRS, and a generalization was performed in our recent paper[7]. However, it turned out that the generalization necessitated us to introduce several new concepts concerning the subresultant.

In section 2, we define multi-PRS and explain how the subresultant is generalized to the case of multi-PRS. The main results of our study of multi-PRS are surveyed in section 3.

Section 3 also explains a concept of "secondary-PRS" which is obtained by a special choice of divisor polynomials in multi-PRS. In section 4, we show an application of multi-PRS to solving a system of linear Diophantine equations with polynomial coefficients. Section 5 presents an example of solving a Diophantine equation.

§2. Multi-PRS and PRS-matrix

Given a set of starting polynomials $\{P_0^{(1)}(x), \dots, P_0^{(m)}(x)\}$ with coefficients in an integral domain I , we generate a sequence of sets of remainders $\{P_i^{(1)}(x), \dots, P_i^{(m)}(x)\}$, $i=1, 2, \dots$, successively by the following formulas:

$$\left. \begin{array}{l} \nu_i \in \{1, 2, \dots, m\}, \\ \beta_i^{(\mu)} P_{i+1}^{(\mu)} = \alpha_i^{(\mu)} P_i^{(\mu)} - Q_i^{(\mu)} P_i^{(\nu_i)}, \quad \deg(P_{i+1}^{(\mu)}) < \deg(P_i^{(\nu_i)}), \\ \text{for } \mu \text{ such that } \deg(P_i^{(\mu)}) \geq \deg(P_i^{(\nu_i)}), \quad \mu \neq \nu_i, \\ \beta_i^{(\mu)} P_{i+1}^{(\mu)} = \alpha_i^{(\mu)} P_i^{(\mu)} \\ \text{for } \mu \text{ such that } \deg(P_i^{(\mu)}) < \deg(P_i^{(\nu_i)}), \\ \alpha_i^{(\mu)}, \beta_i^{(\mu)} \in I, \\ P_{i+1}^{(\nu_i)} = P_i^{(\nu_i)} \text{ or } \alpha_i^{(\nu_i)} = \beta_i^{(\nu_i)} = 1. \end{array} \right\} \quad (1)$$

We call the sequence $(\{P_0^{(1)}, \dots, P_0^{(m)}\}, \{P_1^{(1)}, \dots, P_1^{(m)}\}, \dots)$ multi-PRS. Note that, in formulas (1), only one polynomial $P_i^{(\nu_i)}$ is used as a divisor to generate the set of $(i+1)$ st remainders $\{P_{i+1}^{(1)}, \dots, P_{i+1}^{(m)}\}$. We can define a more general multi-PRS in which the $(i+1)$ st remainders are generated by more than one divisor polynomial. Therefore, we had better call the sequence defined by (1) the multi-PRS in a narrow sense. Although the multi-PRS in a wide sense is also important in practice, this paper considers only the sequence defined by (1) and we simply call it multi-PRS.

Let F, G, H be polynomials of degrees ℓ, m, n , respectively, with coefficients in I :

$$\left. \begin{array}{l} F(x) = f_\ell x^\ell + f_{\ell-1} x^{\ell-1} + \dots + f_0, \quad f_i \in I, \\ G(x) = g_m x^m + g_{m-1} x^{m-1} + \dots + g_0, \quad g_i \in I, \\ H(x) = h_n x^n + h_{n-1} x^{n-1} + \dots + h_0, \quad h_i \in I. \end{array} \right\} \quad (2)$$

Let $\ell \geq m$ and the sequence $(P_1=F, P_2=G, P_3, P_4, \dots)$ be a PRS. The subresultant theory asserts that, for each polynomial P_i in the PRS, there exists a matrix M_i such that

$$P_i(x) \sim |M_i|, \quad (3)$$

where \sim denotes the similarity, i.e., $A(x) \sim B(x)$ if $aA(x)=bB(x)$ for some nonzero a and b in

I, and every nonzero element of the matrix M_i is either $x^k F$, $x^k G$, $k=0,1,\dots$, or a coefficient of F or G .

Let $\{P_i^{(1)}, \dots, P_i^{(m)}\}$, $i=0,1,2,\dots$, be a multi-PRS, and let $M_{i,j}^{(\mu)}$, $1 \leq \mu \leq m$, $0 \leq j \leq i-1$, denote a square matrix, where nonzero elements in the first column of $M_{i,j}^{(\mu)}$ are $x^k P_j^{(\mu)}$, $k=0,1,\dots$, $1 \leq \mu \leq m$, and other nonzero elements of $M_{i,j}^{(\mu)}$ are coefficients of $P_j^{(\mu)}$. Main problems in the theory of multi-PRS are (1) to find an $M_{i,j}^{(\mu)}$ such that $P_j^{(\mu)} \sim M_{i,j}^{(\mu)}$, (2) to determine the proportional factor $\lambda_{i,j}^{(\mu)}$ such that $P_j^{(\mu)} = \lambda_{i,j}^{(\mu)} |M_{i,j}^{(\mu)}|$, and (3) to find efficient algorithms for calculating multi-PRS over I. In order to explain the matrix $M_{i,j}^{(\mu)}$ explicitly, we present a simple example of multi-PRS by setting

$$\{P_0^{(1)}, P_0^{(2)}, P_0^{(3)}\} \equiv \{F_0 = F, G_0 = G, H_0 = H\},$$

$$\deg(F_0) = \ell = n+1, \quad \deg(G_0) = m = n.$$

Below, prem denotes the pseudo-remainder, and we divide each polynomial by its largest constant factor. Choosing G_0 as the first divisor, we obtain

$$F_1 \equiv \text{prem}(F_0, G_0) = \begin{vmatrix} F & f_n & f_{n+1} \\ G & g_n & \\ xG & g_{n-1} & g_n \end{vmatrix}, \quad \deg(F_1) = n-1,$$

$$G_1 \equiv G_0, \quad \deg(G_1) = n,$$

$$H_1 \equiv \text{prem}(H_0, G_0) = \begin{vmatrix} H & h_n \\ G & g_n \end{vmatrix}, \quad \deg(H_1) = n-1.$$

Choosing F_1 as the second divisor, we obtain

$$F_2 \equiv F_1, \quad \deg(F_2) = n-1,$$

$$G_2 \equiv \text{prem}(G_1, F_1) / \text{lc}(G_0)^2 = \begin{vmatrix} F & f_{n-1} & f_n & f_{n+1} \\ xF & f_{n-2} & f_{n-1} & f_n & f_{n+1} \\ G & g_{n-1} & g_n & \\ xG & g_{n-2} & g_{n-1} & g_n \\ x^2G & g_{n-3} & g_{n-2} & g_{n-1} & g_n \end{vmatrix},$$

$$H_2 \equiv \text{prem}(H_1, F_1) / \text{lc}(G_0) = \begin{vmatrix} F & f_{n-1} & f_n & f_{n+1} \\ G & g_{n-1} & g_n & \\ xG & g_{n-2} & g_{n-1} & g_n \\ H & h_{n-1} & h_n & \end{vmatrix}.$$

Choosing H_2 as the next divisor, we can calculate G_3 easily as

$$G_3 = \text{prem}(G_2, H_2) / \text{lc}(F_1)$$

$$= \left| \begin{array}{ccccc} F & f_{n-2} & f_{n-1} & f_n & f_{n+1} \\ xF & f_{n-3} & f_{n-2} & f_{n-1} & f_n & f_{n+1} \\ G & g_{n-2} & g_{n-1} & g_n & & \\ xG & g_{n-3} & g_{n-2} & g_{n-1} & g_n & \\ x^2G & g_{n-4} & g_{n-3} & g_{n-2} & g_{n-1} & g_n \\ H & h_{n-2} & h_{n-1} & h_n & & \end{array} \right|.$$

Contrary to the apparent simplicity of above matrices, the determinant representation for F_3 is complicated as shown below:

$$F_3 = \text{prem}(F_2, H_2) / \text{lc}(F_1)$$

$$= \left| \begin{array}{ccccc|cc} F & f_{n-2} & f_{n-1} & f_n & f_{n+1} & f_n & f_{n+1} \\ xF & f_{n-3} & f_{n-2} & f_{n-1} & f_n & f_{n+1} & - - - - - \\ \hline G & g_{n-2} & g_{n-1} & g_n & & g_n & \\ xG & g_{n-3} & g_{n-2} & g_{n-1} & g_n & & \\ x^2G & g_{n-4} & g_{n-3} & g_{n-2} & g_{n-1} & g_n & - - - - - \\ \hline 0 & & & & & g_{n-1} & g_n \\ \hline H & h_{n-2} & h_{n-1} & h_n & & h_n & \\ xH & h_{n-3} & h_{n-2} & h_{n-1} & h_n & & \end{array} \right|. \quad (4)$$

For convenience, we call the matrix $M_{i,j}^{(\mu)}$ PRS-matrix. As the matrix in (4) indicates, all the nonzero elements in any row of the PRS-matrix are specified by only one polynomial, say $P_j^{(\mu')}$, hence the row is called of type $P_j^{(\mu')}$, or of type μ' in short. It is convenient to divide the rows and the second to last columns of the PRS-matrix into blocks, row-blocks and column-blocks. For example, the matrix in (4) contains four row-blocks and two column-blocks. Furthermore, it is necessary to introduce the concepts of normal and abnormal rows. The abnormal rows may appear in $M_{i,j}^{(\mu)}$ when all PRSs contained in $M_{i,j}^{(\mu)}$ are abnormal. For the details of PRS-matrices, see [7].

§3. Main theorems and secondary-PRS

The main theorems we have obtained on multi-PRS are as follows (lengthy proofs are found in [7]).

Theorem 1: For each polynomial $P_i^{(\mu)}(x)$ in the multi-PRS $\{P_i^{(1)}, \dots, P_i^{(m)}\}$, $i=1, 2, \dots$, generated by (1), there exist PRS-matrices $M_{i,j}^{(\mu)}$, $j=0, 1, \dots, i-1$, such that

$$P_i^{(\mu)}(x) \sim \left| M_{i,j}^{(\mu)} \right|. \quad (5)$$

In [7], we described a method of constructing PRS-matrices explicitly. In order to specify the PRS-matrix we define the following quantities.

$L(\mu, i, j)$: The number of column-blocks in $M_{i,j}^{(\mu)}$.

$N[\ell]_{i,j}^{(\mu)}(k)$: The number of normal rows of type k in the ℓ -th column-block in $M_{i,j}^{(\mu)}$.

$A[\ell]_{i,j}^{(\mu)}(k)$: The number of abnormal rows of type k in the ℓ -th column-block in $M_{i,j}^{(\mu)}$.

In terms of these quantities, the order of matrix $M_{i,j}^{(\mu)}$ is given by

$$\sum_{k=1}^m \sum_{\ell=1}^{L(\mu, i, j)} (N[\ell]_{i,j}^{(\mu)}(k) + A[\ell]_{i,j}^{(\mu)}(k)). \quad (6)$$

Theorem 2: Let $n_i^{(\mu)}$ and $c_i^{(\mu)}$ denote, respectively, the degree and the leading coefficient of $P_i^{(\mu)}$, and let $d_i^{(\mu)}$ denote the degree difference between $P_i^{(\mu)}$ and $P_i^{(\nu_i)}$: $n_i^{(\mu)} = \deg(P_i^{(\mu)})$, $c_i^{(\mu)} = \text{lc}(P_i^{(\mu)})$, $d_i^{(\mu)} = n_i^{(\mu)} - n_i^{(\nu_i)}$. Let $n_{i+1}^{(\mu)} = n_{i'+1}^{(\mu)}$ and $n_{i'}^{(\mu)} \geq n_{i'}^{(\nu_i)}$ for $i' \leq i$, i.e., $P_{i+1}^{(\mu)} \sim \dots \sim P_{i'+1}^{(\mu)} \not\sim P_i^{(\mu)}$. Let $P_i^{(\mu)}(x) = \lambda_{i,j}^{(\mu)} \cdot \left| M_{i,j}^{(\mu)} \right|$, with the matrix $M_{i,j}^{(\mu)}$ constructed in the way described in [7]. Then, we have

$$\begin{aligned} \lambda_{i+1,j}^{(\mu)} &= \sigma \cdot \left\{ \prod_{j=i'+1}^i (\alpha_{j'}^{(\mu)} / \beta_{j'}^{(\mu)}) \cdot (\beta_{i'}^{(\mu)})^{-1} \cdot (\alpha_{i'}^{(\mu)} / (c_{i'}^{(\nu_i)})^{d_{i'}^{(\mu)}+1}) \right. \\ &\quad \times \prod_{j'=i'-1}^j \left\{ (1/c_{j'}^{(\nu_i)})^{\sum_{\ell=1}^{L(\mu, i'+1, j')} \{ \delta N[\ell]_{i'+1, j'}^{(\mu)} + \delta A[\ell]_{i'+1, j'}^{(\mu)} \}} \right. \\ &\quad \times \left. \prod_{k=1}^m (\alpha_j^{(k)} / \beta_j^{(k)})^{\sum_{\ell=1}^{L(\mu, i'+1, j'+1)} \{ N[\ell]_{i'+1, j'+1}^{(\mu)}(k) + A[\ell]_{i'+1, j'+1}^{(\mu)}(k) \}} \right\}, \end{aligned} \quad (7)$$

where σ is the sign factor, i.e. $\sigma=1$ or -1 , and $\delta N[\ell]_{i+1,j}^{(\mu)}$ and $\delta A[\ell]_{i+1,j}^{(\mu)}$ are defined as

$$\begin{aligned} \delta N[\ell]_{i+1,j}^{(\mu)} &= N[\ell]_{i+1,j}^{(\mu)}(\nu_j) - N[\ell]_{i+1,j+1}^{(\mu)}(\nu_j), \\ \delta A[\ell]_{i+1,j}^{(\mu)} &= A[\ell]_{i+1,j}^{(\mu)}(\nu_j) - A[\ell]_{i+1,j+1}^{(\mu)}(\nu_j). \end{aligned} \quad (8)$$

In [7], we derived the recurrence formulas for calculating $N[\ell]_{i+1,j}^{(\mu)}$ and $A[\ell]_{i+1,j}^{(\mu)}$. On the basis of (7) and the recurrence formulas, we obtained two algorithms for calculating multi-PRS over I in such a way that $P_1^{(\mu)}(x) = \sigma \left| M_{1,0}^{(\mu)} \right|$. An important notice is that, in the case of multi-PRS, even the Collins' simplest choice of $\alpha_i^{(\mu)}$ and $\beta_i^{(\mu)}$, i.e. $\beta_{i+1}^{(\mu)} = \alpha_i^{(\mu)} = [c_i^{(\nu_i)}]^{d_i^{(\mu)}+1}$, does not guarantee that $\lambda_{i,0}^{(\mu)} \in I$ in general. We convince ourselves of this by the example presented in the previous section.

So far, the choice of divisor polynomial $P_i^{(\nu_i)}$ is arbitrary. Accordingly, many types of rows may be included in a single PRS-matrix, making the order of PRS-matrix large (see

(6)). If, however, we specify the choice of the divisor polynomials suitably, we can make the PRS-matrices simpler. The simplest choice of the divisor polynomials will be that $P_i^{(\nu_i)} \in \{P_i^{(1)}, P_i^{(2)}\}$, i.e., to calculate $\{P_i^{(1)}, P_i^{(2)}\}$, $i=1, 2, \dots$, as the conventional PRS. In this case, it is convenient to divide the set $\{P_i^{(1)}, P_i^{(2)}, \dots, P_i^{(m)}\}$ into $m-2$ sets of three polynomials as $\{P_i^{(1)}, P_i^{(2)}, P_i^{(3)}\}$, $\{P_i^{(1)}, P_i^{(2)}, P_i^{(4)}\}$, \dots , $\{P_i^{(1)}, P_i^{(2)}, P_i^{(m)}\}$. Then, each of these sets is generated by the following same formulas:

$$\begin{aligned} P_1 &= P_0^{(1)}, \quad P_2 = P_0^{(2)}, \quad \tilde{P}_1 \in \{P_0^{(3)}, \dots, P_0^{(m)}\}, \\ \beta_i P_{i+1} &= \alpha_i P_{i-1} - Q_i P_i, \quad \deg(P_{i+1}) < \deg(P_i), \quad i=2, 3, \dots, \\ \tilde{\beta}_i \tilde{P}_i &= \tilde{\alpha}_i \tilde{P}_{i-1} - \tilde{Q}_i P_i, \quad \deg(\tilde{P}_i) < \deg(P_i), \quad i=2, 3, \dots, \end{aligned} \quad \left. \right\} \quad (9)$$

We may call the sequence (P_1, P_2, \dots, P_k) , which is a conventional PRS, main-PRS and the subsidiary sequence $(\tilde{P}_1, \tilde{P}_2, \dots, \tilde{P}_k)$ secondary-PRS. (Comment: We conceived the concept of secondary-PRS first, and then generalized it to multi-PRS.)

In [7], we have showed that the PRS-matrix for the secondary-PRS contains only one column-block and no abnormal row. Furthermore, we have

$$N[1]_{i,j}^{(\mu \geq 3)}(k) = \begin{cases} 1 & \text{if } k=\mu, \\ 0 & \text{if } k \neq 1, 2, \mu. \end{cases}$$

Therefore, the PRS-matrix for the secondary-PRS has very simple form. We show the PRS-matrix explicitly in the case $\{P_1(x)=F(x), P_2(x)=G(x), \tilde{P}_1(x)=H(x)\}$, where F, G, and H are defined by (2) with $\ell \geq n$:

$$\tilde{P}_i \sim \left| \begin{array}{cccccc} x^0 F & f_j & \dots & f_{\ell-1} & f_\ell \\ x^1 F & f_{j-1} & \dots & f_{\ell-1} & f_\ell \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ x^{m-j-1} F & f_{2j+1-m} & \dots & \dots & f_{\ell-1} & f_\ell \\ x^0 G & g_j & \dots & g_{m-1} & g_m \\ x^1 G & g_{j-1} & \dots & g_{m-1} & g_m \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ x^{\ell-j-1} G & g_{2j+1-\ell} & \dots & \dots & g_{m-1} & g_m \\ x^0 H & h_j & \dots & h_{n-1} & h_n \end{array} \right|, \quad j=\deg(P_i). \quad (10)$$

The proportional factor $\lambda_{i,j}^{(\mu)}$ for the secondary-PRS also becomes very simple, and we can refine multi-PRS algorithms for calculating secondary-PRS over I. In fact, after analyzing the PRS-matrix in details, [8] presents two algorithms which calculate \tilde{P}_i such that $\tilde{P}_i =$

$|M_{i,0}^{(\mu \geq 3)}|$. Choosing $\tilde{P}_i = |M_{i,0}^{(\mu \geq 3)}|$, (10) shows immediately that

$$\left. \begin{aligned} \tilde{P}_i &= A_i P_1 + B_i P_2 + C_i \tilde{P}_1, \\ \deg(A_i) < \deg(P_2) - \deg(P_1) &\leq \deg(P_2) - \deg(\tilde{P}_i) - 1, \\ \deg(B_i) < \deg(P_1) - \deg(P_i) &\leq \deg(P_1) - \deg(\tilde{P}_i) - 1, \\ \deg(C_i) &= 0, \end{aligned} \right\} \quad (11)$$

where

$$A_i = \left| \begin{array}{cccccc} x^0 & f_j & \dots & f_{\ell-1} & f_\ell \\ x^1 & f_{j-1} & \dots & f_{\ell-1} & f_\ell \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ x^{m-j-1} & f_{2j+1-m} & \dots & \dots & f_{\ell-1} & f_\ell \\ 0 & g_j & \dots & g_{m-1} & g_m \\ 0 & g_{j-1} & \dots & g_{m-1} & g_m \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & g_{2j+1-\ell} & \dots & \dots & g_{m-1} & g_m \\ 0 & h_j & \dots & h_{n-1} & h_n \end{array} \right|, \quad j = \deg(P_i), \quad (12)$$

and B_i and C_i are obtained by replacing the first column of A_i by $(0 \dots 0 \ x^0 \ x^1 \ \dots \ x^{\ell-j-1} \ 0)$ and $(0 \dots 0 \ 0 \ \dots \ 0 \ 1)$, respectively. The A_i , B_i , C_i , $i=3,4,\dots$ are generalizations of those for the cofactor sequences which are calculated by the extended Euclidean algorithm. For the determinant representations for cofactor sequences, see [6] and [9]; the latter presents two other determinant representations for each of the cofactor sequences.

§4. Application to linear Diophantine equations

The multi-PRS, in particular the secondary-PRS, is nicely applicable to solving a system of linear Diophantine equations with polynomial coefficients. Let the ring R be Z (the ring of rational integers) or $Z[x_1, \dots, x_s]$, and let the quotient field of R be S . We consider the following system of linear Diophantine equations:

$$\left. \begin{aligned} a_{11}y_1 + \dots + a_{1m}y_m &= b_1, \\ \vdots & \quad \dots \quad \vdots \quad \dots \\ a_{n1}y_1 + \dots + a_{nm}y_m &= b_n, \end{aligned} \right\} \quad (13)$$

where $m > n$, $a_{ij} \in R[x]$, $b_i \in R[x]$, and we want to obtain the solution

$$\bar{y} = (y_1, \dots, y_m) \quad (14)$$

such that $y_i \in S[x]$, $i=1, \dots, m$, if any. That is, we search for the solution which is polynomial in x with rational coefficients. Note that the Cramer's formula gives the solutions which are rational in x .

It is well known that (13) has not always a solution. Furthermore, since the number of equations, n , is less than the number of unknowns, m , the possible solutions of (13) are not unique. The general solution of (13), if it exists, is represented as

$$\bar{y} = \bar{y}_0 + c_1 \bar{y}_1 + \dots + c_r \bar{y}_r, \quad (15)$$

where $m-1 \geq r \geq n-n$, \bar{y}_0 is a particular solution of (13), $\{\bar{y}_1, \dots, \bar{y}_r\}$ is a basis of the space of the solutions of homogeneous equations

$$\left. \begin{array}{l} a_{11}y_1 + \dots + a_{1m}y_m = 0, \\ \vdots \quad \dots \quad \cdot \quad \cdot \quad \cdot \\ a_{n1}y_1 + \dots + a_{nm}y_m = 0, \end{array} \right\} \quad (16)$$

and c_1, \dots, c_r are arbitrary elements in $S[x]$.

We solve (13) in the following way [10], where the calculation is performed in $R[x]$ as far as possible. We first solve the equation

$$a_{11}y_1 + \dots + a_{1m}y_m = b_1$$

(an actual method is given later), and obtain the general solution

$$\bar{y} = \bar{y}_0^{(1)} + c_1^{(1)} \bar{y}_1^{(1)} + \dots + c_{m-1}^{(1)} \bar{y}_{m-1}^{(1)}, \quad (17)$$

if it exists. Here, $c_1^{(1)}, \dots, c_{m-1}^{(1)}$ are any elements in $S[x]$, hence we represent them by indeterminates y_{m+1}, \dots, y_{2m-1} :

$$\bar{y} = \bar{y}_0^{(1)} + y_{m+1} \bar{y}_1^{(1)} + \dots + y_{2m-1} \bar{y}_{m-1}^{(1)}. \quad (17')$$

Substituting (17') for y_1, \dots, y_m in the rest $n-1$ equations of (13), we obtain a reduced system of $n-1$ equations in $m-1$ unknowns y_{m+1}, \dots, y_{2m-1} :

$$\left. \begin{array}{l} a'_{11}y_{m+1} + \dots + a'_{1,m-1}y_{2m-1} = b'_1, \\ \vdots \quad \dots \quad \cdot \quad \cdot \quad \cdot \\ a'_{n-1,1}y_{m+1} + \dots + a'_{n-1,m-1}y_{2m-1} = b'_{n-1}, \end{array} \right\} \quad (13')$$

where we reduce $\bar{y}_i^{(1)}$, $i=0, \dots, m-1$, to a common denominator, hence $a'_{ij} \in R[x]$ and $b'_i \in R[x]$.

Note that some equation in (13') may be nil. If this is the case, the dimension of the solution space increases, i.e., $r > m-n$.

Continuing the above reduction, we finally obtain a Diophantine equation in $r+1$ unknowns

$y_{\mu+1}, \dots, y_{\mu+r+1}$:

$$a''_1 y_{\mu+1} + \dots + a''_{r+1} y_{\mu+r+1} = b'', \quad (13'')$$

where $a''_i, b'' \in R[x]$ and each of y_1, \dots, y_m is linearly related to $y_{\mu+1}, \dots, y_{\mu+r+1}$. We solve (13'') and, if the solution exists, obtain the general solution

$$\bar{y}'' = \bar{y}_0'' + c_1 \bar{y}_1'' + \dots + c_r \bar{y}_r'', \quad (17'')$$

where $\bar{y}'' = (y_{\mu+1}, \dots, y_{\mu+r+1})$, \bar{y}_0'' is a particular solution of (13''), $\{\bar{y}_1'', \dots, \bar{y}_r''\}$ is a basis of the space of the solutions of homogeneous equation, and c_1, \dots, c_r are arbitrary elements in $S[x]$. Substituting \bar{y}'' into y_1, \dots, y_m , we obtain the general solution of (13) in the form (15).

Our problem is, therefore, reduced to solving the following linear Diophantine equation:

$$P_0^{(1)}(x) y_1 + \dots + P_0^{(m)}(x) y_m = P_0^{(m+1)}(x), \quad (18)$$

where $P_0^{(i)}(x) \in R[x]$, $i=1, \dots, m+1$, and we want to obtain the solution $y_i \in S[x]$, $i=1, \dots, m$, if any. Equation (18) can be solved by successively eliminating higher degree terms of $P_0^{(1)}, \dots, P_0^{(m)}$ as follows. Let $P_0^{(\nu)} \in \{P_0^{(1)}, \dots, P_0^{(m)}\}$, where $\deg_x(P_0^{(\nu)}) \leq \deg_x(P_0^{(\mu)})$ for at least one $\mu \neq \nu, m+1$. Performing pseudo-divisions of $P_0^{(\mu)}$, $\mu \neq \nu$, by $P_0^{(\nu)}$, we have

$$\begin{aligned} \alpha_0 P_0^{(\mu)} &\equiv Q_0^{(\mu)} P_0^{(\nu)} + P_1^{(\mu)}, \quad \mu=1, \dots, \nu-1, \nu+1, \dots, m, \\ P_0^{(\nu)} &\equiv P_1^{(\nu)}, \end{aligned} \quad \left. \right\} \quad (19)$$

where α_0 is chosen so that $P_1^{(\mu)} \in R[x]$, $\mu=1, \dots, m$. Substituting $P_0^{(i)}$ in (18) by the r.h.s. expressions in (19), we have

$$\begin{aligned} P_0^{(\nu)}(Q_0^{(1)} y_1 + \dots + \alpha_0 y_\nu + \dots + Q_0^{(m)} y_m) \\ + P_1^{(1)} y_1 + \dots + P_1^{(\nu-1)} y_{\nu-1} + P_1^{(\nu+1)} y_{\nu+1} + \dots + P_1^{(m)} y_m = \alpha_0 P_0^{(m+1)}. \end{aligned}$$

Hence, introducing a new unknown y'_ν defined by

$$y'_\nu = Q_0^{(1)} y_1 + \dots + \alpha_0 y_\nu + \dots + Q_0^{(m)} y_m, \quad (20)$$

we can rewrite (18) as

$$P_1^{(1)} y_1 + \dots + P_1^{(\nu)} y'_\nu + \dots + P_1^{(m)} y_m = \alpha_0 P_0^{(m+1)} \equiv P_1^{(m+1)}. \quad (18')$$

Equation (18') is simpler than (18) in the sense that the degrees, in x , of coefficient polynomials are reduced.

Continuing the above reduction, we finally obtain

$$P_k^{(1)}(x) y''_1 + \dots + P_k^{(m)}(x) y''_m = P_k^{(m+1)}(x), \quad (18'')$$

where either $\deg_x(P_k^{(i)})=0$ for every $i \leq m$ and $P_k^{(\nu)} \neq 0$ for some $\nu \leq m$, or $\deg_x(P_k^{(\nu)}) > 0$ and $P_k^{(i)}=0$ for every $i=1, \dots, \nu-1, \nu+1, \dots, m$. Note that some of $P_k^{(i)}$, $i=1, \dots, m$, may be zero.

Case 1: $\deg_x(P_k^{(i)})=0$ for $i=1,\dots,m$ and $P_k^{(\nu)} \neq 0$ for some $\nu \leq m$. In this case, we can rewrite (18'') as

$$P_k^{(\nu)} y_{\nu}'' = P_k^{(m+1)} - \sum_{i=1, i \neq \nu}^m P_k^{(1)} y_i'' . \quad (21)$$

Hence, $\{y_{\nu}'' = P_k^{(m+1)} / P_k^{(\nu)}, y_{j \neq \nu}'' = 0\}$ is a particular solution of (18''), and $\{y_i'' = 1, y_{\nu}'' = -P_k^{(1)} / P_k^{(\nu)}, y_{j \neq i, \nu}'' = 0\}$, $i=1,\dots,\nu-1,\nu+1,\dots,m$, constitute a basis of the space of the solutions of homogeneous equation. Since y_1, \dots, y_m are linearly related to y_1'', \dots, y_m'' , backward substitution of this solution gives the solution of (18) in the form (15).

Case 2: $\deg_x(P_k^{(\nu)}) > 0$ and $P_k^{(1)} = 0$ for $i=1,\dots,\nu-1,\nu+1,\dots,m$. In this case, (18'') turns out to be $P_k^{(\nu)} y_{\nu}'' = P_k^{(m+1)}$ which has a solution only if

$$P_k^{(\nu)}(x) \mid P_k^{(m+1)}(x). \quad (22)$$

If the condition (22) is satisfied, then $\{y_{\nu}'' = P_k^{(m+1)} / P_k^{(\nu)}, y_{j \neq \nu}'' = 0\}$ is a particular solution of (18'') and $\{y_i'' = 1, y_{j \neq i}'' = 0\}$, $i=1,\dots,\nu-1,\nu+1,\dots,m$, constitute a basis of the space of the solutions of homogeneous equation.

Although the method described above is quite simple in principle, actual computation requires a large amount of time ([10] gives a time complexity analysis). In fact, the situation is much worse than the calculation of PRS, because the above method is a repetition of multi-PRS calculation and fraction reduction to a common denominator. Using the theorem 2, however, we can improve the situation considerably.

When calculating the $(i+1)$ st remainders $P_{i+1}^{(\mu)}$, $\mu=1,\dots,m$, from the i -th remainders $P_i^{(\mu)}$, formula (19) should be read as

$$\left. \begin{aligned} \alpha_i P_i^{(\mu)} &= Q_i^{(\mu)} P_i^{(\nu_i)} + P_{i+1}^{(\mu)}, \quad \mu=1,\dots,\nu-1,\nu+1,\dots,m, \\ P_i^{(\nu_i)} &= P_{i+1}^{(\nu_i)}, \quad \alpha_i P_i^{(m+1)} = P_{i+1}^{(m+1)}, \end{aligned} \right\} \quad (19')$$

where

$$\alpha_i = [lc(P_i^{(\nu_i)})]^{d_i+1}, \quad d_i = \max[\deg(P_i^{(\mu)}) - \deg(P_i^{(\nu_i)})], \quad \mu=1,\dots,m. \quad (23)$$

Our improvement is to use, instead of (19'), the following formulas:

$$\left. \begin{aligned} \alpha_i P_i^{(\mu)} &= Q_i^{(\mu)} P_i^{(\nu_i)} + \beta_i P_{i+1}^{(\mu)}, \quad \mu=1,\dots,\nu-1,\nu+1,\dots,m, \\ P_i^{(\nu_i)} &= P_{i+1}^{(\nu_i)}, \quad \alpha_i P_i^{(m+1)} = \beta_i P_{i+1}^{(m+1)}, \end{aligned} \right\} \quad (24)$$

where β_i is determined by the multi-PRS theory so that $P_{i+1}^{(\mu)} \in R[x]$, $\mu=1,\dots,m+1$.

Correspondingly, (20) should be replaced by

$$\beta_i y_{\nu}'' = Q_i^{(1)} y_1 + \dots + \alpha_i y_{\nu} + \dots + Q_i^{(m)} y_m. \quad (25)$$

Noting that the factors independent of x are irrelative to the essence of the multi-PRS

calculation, we can improve the calculation method further by putting

$$P_i^{(\mu)} = r_i^{(\mu)} \tilde{P}_i^{(\mu)}, \quad \mu=1, \dots, m+1, \quad (26)$$

where we calculate $\tilde{P}_i^{(\mu)}$ by a multi-PRS algorithm with

$$\alpha_i^{(\mu)} = [\text{lcf}(P_i^{(\nu_i)})]^{d_i^{(\mu)}+1}, \quad d_i^{(\mu)} = \max[-1, \deg(P_i^{(\mu)}) - \deg(P_i^{(\nu_i)})]. \quad (27)$$

Since $\alpha_i^{(\mu)} | \alpha_i$ and $\tilde{P}_i = \pm \left| M_{i,0}^{(\mu)} \right|$ for every μ , we find

$$r_i^{(\mu)} = \prod_{j=0}^{i-1} [\text{lcf}(P_j^{(\nu_j)})]^{e(\mu,i,j)}, \quad e(\mu,i,j) \text{ is an integer } \geq 0. \quad (28)$$

and it is easy to obtain $r_i^{(\mu)}$ in this factored form. Hence, calculating $P_i^{(\mu)}$ in the factored form (26), we can improve the reduction step for solving (18) drastically. Note that, in the above formulas, we had better calculate $\tilde{P}_i^{(\mu)}$, $\mu=3,4,\dots,m$, as secondary-PRSs with the divisor polynomial $\tilde{P}_i^{(1)}$ or $\tilde{P}_i^{(2)}$, because the expressions $\tilde{P}_i^{(\mu)}$ become almost smallest in this case.

§5. Example

Let us solve, for example, the following Diophantine equation:

$$F_1 y_1 + F_2 y_2 + \tilde{F}_1 y_3 = 1, \quad (29)$$

where

$$F_1 = x^4 + x^3 - 3x^2 + 1,$$

$$F_2 = 3x^4 + 5x^3 - 9x^2 + 21,$$

$$\tilde{F}_1 = 2x^4 - x^3 - 2x^2 + 1.$$

We first apply the formulas (19*) and (23) to solve (29). Denoting the general solution of (29) as $\bar{y} = \bar{y}_0 + c_3 \bar{y}_3 + c_7 \bar{y}_7$, we obtain

$$\begin{aligned} y_1 &= \{ (281715378192x^3 + 757904884992x^2 + 363625526592x - 1751359292784) \\ &\quad + c_3(-1383865015680x^3 + 1062147867840x^2 - 2317918368960x - 11423735363520) \\ &\quad + c_7(3x^4 + 5x^3 - 9x + 21) \} / 10102066528320, \\ y_2 &= \{ (-93905126064x^3 - 190031544288x^2 + 224591148144x + 564448848624) \\ &\quad + c_3(461288338560x^3 - 661574848320x^2 + 137349898560x + 62936611200) \\ &\quad + c_7(-x^4 - x^3 + 3x^2 - 1) \} / 10102066528320, \\ y_3 &= c_3. \end{aligned} \quad (30)$$

In deriving (30), we generated PRS (F_1, F_2, \dots, F_6) and secondary-PRS $(\tilde{F}_1, \tilde{F}_2, \dots, \tilde{F}_5)$ with the starting polynomials F_1 , F_2 , and \tilde{F}_1 . The secondary-PRS generated is

$$\tilde{F}_2 = -13x^3 - 6x^2 + 18x - 39.$$

$$\tilde{F}_3 = 210x^2 - 162x + 312,$$

$$\tilde{F}_4 = 5969052x - 3584412,$$

$$\tilde{F}_5 = 84244166694535680.$$

We next apply the formulas (24) and (23) with $\beta_{i+1} = \alpha_i$. Then, the large coefficients in the PRS and secondary-PRS are reduced and we obtain the following solution:

$$\left. \begin{aligned} y_1 &= \{ (15219x^8 + 40944x^7 + 19644x^6 - 94613) \\ &\quad + c_3(-74760x^3 + 57380x^2 - 125220x - 617140) \\ &\quad + c_7(3x^4 + 5x^3 - 9x + 21) \} / 545740, \\ y_2 &= \{ (-5073x^8 - 10266x^7 + 2133x^6 + 30493) \\ &\quad + c_3(24920x^3 - 35740x^2 + 7420x + 3400) \\ &\quad + c_7(-x^4 - x^3 + 3x^2 - 1) \} / 545740, \\ y_3 &= c_3. \end{aligned} \right\} \quad (30')$$

We see that the calculation was improved remarkably. Note that (30) and (30') are the same solution because c_3 and c_7 are arbitrary rational numbers.

Acknowledgement

One of the authors (T.S.) acknowledges The Kurata Foundation for partially supporting this work.

References

- [1] W. Habicht, Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens, Comm. Math. Helvetica 21, pp.99-116 (1948).
- [2] G. E. Collins, Polynomial remainder sequences and determinants, Amer. Math. Mon. 73, No.7, pp.708-712 (1966).
- [3] G. E. Collins, Subresultants and reduced polynomial remainder sequences, J. ACM 14, No.1, pp.128-142 (1967).
- [4] W. S. Brown and J. F. Traub, On Euclid's algorithm and the theory of subresultants, J. ACM 18, No.4, pp.505-514 (1971).
- [5] W. S. Brown, The subresultant PRS algorithm, ACM Trans. Math. Soft. 4, No.3, pp.237-249 (1978).
- [6] R. Loos, Generalized polynomial remainder sequences, Computing Suppl. 4, "Computer Algebra," eds. B. Buchberger, G. E. Collins and R. Loos, pp.115-137, Springer-Verlag, 1982.
- [7] T. Sasaki and A. Furukawa, Theory of multi-polynomial remainder sequence, preprint of IPCR, November 1982 (submitted for publication).
- [8] T. Sasaki and A. Furukawa, Secondary-polynomial remainder sequence and an extension of subresultant theory, preprint of IPCR, May 1982 (submitted for publication).
- [9] T. Sasaki, Extended Euclidean algorithm and determinants, preprint of IPCR, April 1982 (submitted for publication).
- [10] A. Furukawa, Algebraic methods for linear Diophantine equations and theory of secondary-polynomial remainder sequence (in Japanese), Master Thesis, Tokyo Metropolitan Univ., March 1982.

TOWARDS MECHANICAL SOLUTION OF
THE KAHAN ELLIPSE PROBLEM I

by

Dennis S. Arnon
Computer Science Department
Purdue University
West Lafayette, Indiana, USA 47907

Scott F. Smith
Computer Science Department
Purdue University
West Lafayette, Indiana, USA 47907

ABSTRACT

Tarski (1948) gave a quantifier elimination procedure for elementary algebra and geometry which, when implemented, was found to be impractical. Collins (1975) gave a different and far more efficient algorithm, which has revived interest in the feasibility of quantifier elimination as a means of solving nontrivial problems in elementary algebra and geometry. Collins' algorithm has recently been implemented (1981). Experience so far obtained with the software indicates that, while some worthwhile problems can now be solved, it is desirable to find methods for improving performance. In this paper we report on several such methods, that we have developed in the course of applying the Collins algorithm to the Kahan ellipse problem. These methods have yielded substantial performance improvements; in fact, they have made feasible computations relating to the Kahan problem that initially could not be carried out at all in our computing environment. We are pursuing the further development of these techniques, with the goal of eventually producing a solution to the Kahan problem using the Collins algorithm.

1. Introduction. Tarski [Tar51a] gave a quantifier elimination algorithm (and hence a decision method) for elementary algebra and geometry. Its computing time was found to be so great, however, that it had little practical value. Collins [Col75a] gave a different and far more efficient method, which has revived interest in the feasibility of quantifier elimination as a means of solving nontrivial problems in elementary algebra and geometry. Recently [Arn81a] an implementation of Collins' method was completed (in the SAC-2 Computer Algebra system [Col80a] on a VAXTM computer). However, experience so far obtained with the software for Collins' algorithm indicates that using it is not a simple matter. Nontrivial problems have indeed been solved by just applying the program to an appropriate formula, but so naive an approach has not been successful for any problem generally considered interesting.

The Kahan ellipse problem [Kah75a], which was SIGSAM problem No. 9, illustrates the situation. The problem is: in the equation

$$\frac{(x - c)^2}{a^2} + \frac{(y - d)^2}{b^2} - 1 = 0,$$

let a and b be interpreted as the principal semi-axes of an ellipse and $\langle c, d \rangle$ as its center. Find the conditions on a , b , c , d so that the ellipse be inside the unit circle $y^2 + x^2 - 1 = 0$. M. Lauer [Lau77a] solved the problem by reducing it to certain standard calculations on certain polynomials, which he then carried out using a computer algebra system. His solution involved a mathematical analysis of the problem which was far from mechanical. Quantifier elimination can be brought to bear by observing that one can write a formula in the theory of elementary algebra and geometry asserting that "for all points $\langle x, y \rangle$ in the plane, if $\langle x, y \rangle$ is on the ellipse, then $\langle x, y \rangle$ is inside the unit circle". For example:

$$(\forall x)(\forall y)[\frac{(x - c)^2}{a^2} + \frac{(y - d)^2}{b^2} - 1 = 0 \Rightarrow y^2 + x^2 - 1 < 0].$$

Note that a , b , c , and d are free variables in this formula. An equivalent quantifier-free formula will be a boolean combination of polynomial equations and inequalities in a , b , c , and d , with the property that the 4-tuples $\langle a, b, c, d \rangle$ satisfying it are precisely the 4-tuples satisfying the original formula. Hence such an equivalent quantifier-free formula is a solution to the Kahan problem. When Collins' algorithm was applied to the formula above, our time and space resources (24 hours maximum on the VAX computer with 2 Megabytes memory) proved insufficient. Indeed it seems unlikely that any existing installation could carry out such a computation.

Experience of this kind leads one to seek improvements to the original Collins algorithm. In the present paper we give several such enhancements. The first pertains not to the algorithm itself, but to the form of the input formula. For formulas having a particular structure, we show how to reduce the original quantifier elimination problem to several smaller q.e. problems. Second, we use known restrictions on free variables to reduce the work done, and hence time required, by the quantifier elimination algorithm. Third, we show how one may be able to "customize" a certain phase of the quantifier elimination process to the particular problem being considered, and thereby save time. We will illustrate our enhancements for a special case of the Kahan problem where they have been particularly useful. This special case is defined by constraining the center of the ellipse to lie on the x -axis.

Section 2 below reviews quantifier elimination for elementary algebra and geometry, and gives an example of Collins' algorithm. Sections 3, 4, and 5 describe our enhancements. Sec-

tion 6 summarizes our experience. Our goal is to continue improving the algorithm's performance, in order to eventually construct a solution to the full Kahan problem.

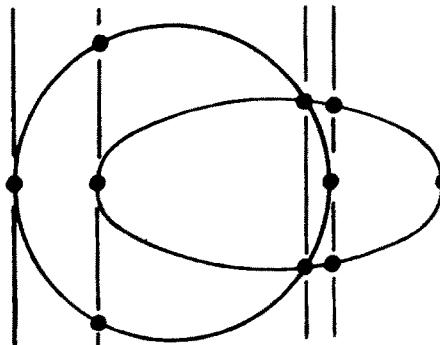
2. Elementary algebra and geometry. The theory of "elementary algebra and geometry" is more formally known as "the first order theory of real closed fields" (see [Kre67a]). By the term *formula* in this paper we mean a well-formed formula of this theory. Such formulas involve polynomials in several variables with integer coefficients. We assume without further comment that all polynomials considered have this form, or possibly have rational number coefficients and so can be easily brought to this form by clearing denominators.

The core of Collins' quantifier elimination method is the construction of a *cylindrical algebraic decomposition (cad)* of r -dimensional euclidean space E^r , where r is the total number of variables in the input formula. Such a decomposition partitions E^r into connected subsets, called *cells*, such that the truth value of the unquantified matrix of the formula is invariant on each cell. Further information on cad's may be found in [Arn82a, Arn82b], or [Col75a]. Experience to date indicates that the time required for cad construction in the Collins' algorithm far exceeds the time needed to subsequently complete the quantifier elimination.

We now give a brief example of the quantifier elimination method. Suppose we wish to decide whether some *particular* ellipse is inside the unit circle. For example, consider the ellipse with center $\langle 1/2, 0 \rangle$, major semi-axis $a = 3/4$ and minor semi-axis $b = 1/4$. It will suffice to eliminate the quantifiers from the following formula:

$$(\forall x)(\forall y)[y^2 + x^2 - 1 = 0 \Rightarrow 144 y^2 + 16 x^2 - 16 x - 5 > 0].$$

The following picture shows the cad that would be constructed. Each "dot", "arc", and "patch of white space" is a cell.



One sees that the plane has been partitioned into six two-dimensional "strips", separated by five "lines". Each strip and each line is itself partitioned into cells "stacked on top of each other". The truth value of the unquantified matrix $[y^2 + x^2 - 1 = 0 \Rightarrow 144y^2 + 16x^2 - 16x - 5 > 0]$ is invariant on each cell, i.e. for each cell, either the matrix is true at every point of the cell, or it is false at every point of the cell. Note that the standard projection of each strip and of each line is an interval or point in the x -axis; we call this interval or point the *base* of the strip or line. Furthermore, the bases of the different strips and lines are all disjoint, and their union is the entire x -axis. In fact, the set of all the bases is a cad of the x -axis. In general, the cells of a cad of E^r are arranged into "stacks" over the cells of a cad of E^{r-1} , so that a cad of E^r "induces" cad's of $E^{r-1}, E^{r-2}, \dots, E^1$.

The innermost, or $(\forall y)$, quantifier is eliminated first. Since it is a universal quantifier, we proceed as follows. We examine each strip and line in turn. For each one, if the matrix is true on every cell in that strip or line, we mark its base "true". If the matrix is false on at least one cell, we mark the base "false". When we are done, the x -axis has been partitioned into points and intervals, each labelled "true" or "false". For the example above, we will have:



We now eliminate the outer, or $(\forall x)$, quantifier. If all points and intervals in the x -axis are labelled "true", our quantifier-free formula equivalent to the original formula will be " $0 = 0$ ", i.e. true. If at least one point or interval in the x -axis is labelled "false", the equivalent quantifier-free formula will be " $1 = 0$ ", i.e. false. In our example we get " $1 = 0$ ", indicating that the ellipse is not inside the circle.

If the inner quantifier above had been existential, the rule used to eliminate it would have been: if the matrix is true on at least one cell of the strip or line, mark the base "true"; otherwise mark the base "false". Similarly, if the outer quantifier had been existential, we would produce " $0 = 0$ " if at least one point or interval in the x -axis had been marked "true".

As a further illustration of Collins' method, suppose that there had been no outer quantifier, i.e. suppose that the quantifier elimination problem had been

$$(\forall y)[y^2 + x^2 - 1 = 0 \Rightarrow 144y^2 + 16x^2 - 16x - 5 > 0].$$

Then it would be the task of the algorithm to construct an equivalent quantifier-free formula in x . Collins' method contains a subalgorithm which can construct, for each cell of a cad, a

quantifier-free formula defining that cell. The final quantifier-free formula, equivalent to the input formula, that will be constructed is the disjunction of the defining formulas for all the cells in the x -axis marked true by the elimination of the inner quantifier. In the example we are considering this is:

$$(x < -1) | (x = -1) | (-1 < x < -1/4) | (x = -1/4) | \\ (-1/4 < x < 63/64 \& 128x^2 + 16x - 139 < 0) | \\ (1 < x < 5/4) | (x = 5/4) | (x > 5/4)$$

(Vertical bar ("|") denotes disjunction.)

3. Simplification of quantified formulas. The theorem below shows that for prenex formulas with a certain structure, quantifiers can be "distributed" into the formula in a certain fashion. The effect of this transformation is to reduce a given quantifier elimination problem to two new quantifier elimination problems. In general, it might or might not be faster to solve these two problems separately. For the special case of the Kahan problem that we consider, however, the transformation is advantageous.

In the following theorem, u denotes a vector of variables $\langle u_1, \dots, u_s \rangle$, and v denotes a vector of variables $\langle v_1, \dots, v_t \rangle$. $F(u)$ is any polynomial in u_1, \dots, u_s , in which no u_i occurs. Similarly, $G(v)$ is any polynomial in v_1, \dots, v_t in which no v_j occurs. We follow the convention that $\psi(x_1, \dots, x_k)$ denotes a formula ψ in which all occurrences of x_1, \dots, x_k are free, and in which each x_i may or may not occur. $\varphi(u, G(v))$ denotes an arbitrary formula which has free occurrences of (some or all of) u_1, \dots, u_s , which has free occurrences of (some or all of) v_1, \dots, v_t , and which can be obtained from some formula $\Psi(u, z)$ by replacing every occurrence of the (scalar) variable z by $G(v)$.

THEOREM 3.1. *The formula*

$$(\forall u)(\forall v)[\{ F(u) = G(v) \} \Rightarrow \varphi(u, G(v))]$$

is equivalent to

$$(\forall u)[\{ (\exists v)(F(u) = G(v)) \} \Rightarrow \varphi(u, F(u))].$$

Proof.

$$(\forall u)(\forall v)[F(u) = G(v) \Rightarrow \varphi(u, G(v))]$$

implies

$$(\forall u)(\forall v)[F(u) = G(v) \Rightarrow \{F(u) = G(v) \& \varphi(u, G(v))\}],$$

which in turn implies

$$(\forall u)(\forall v)[F(u) = G(v) \Rightarrow \varphi(u, F(u))].$$

A similar argument establishes the opposite implication, so the first and last of these three formulas are equivalent. It is a standard (and easily proven) fact that if $\lambda(z)$ is a formula containing the (scalar) free variable z , and ρ is formula in which z is not free, then $(\forall z)[\lambda(z) \Rightarrow \rho]$ is equivalent to

$[(\exists z)\lambda(z)] \Rightarrow \rho$. One easily sees that the vector version of this proposition is valid. Hence the last formula above is equivalent to

$$(\forall u)[\{(\exists v)(F(u) = G(v))\} \Rightarrow \varphi(u, F(u))].$$

Consider the special case of the Kahan ellipse problem in which the center $\langle c, d \rangle$ of the ellipse lies on the x -axis, i.e. $d = 0$. It will suffice to work with a formula asserting that "for all points $\langle x, y \rangle$ in the plane, if $\langle x, y \rangle$ is on the unit circle, then $\langle x, y \rangle$ is outside the ellipse." Note that the variables a and b in the equation for an ellipse occur only in the monomials a^2 and b^2 . Hence we may replace these monomials by a and b respectively, provided that we restrict the new a and b to be nonnegative, and eventually restore them to a^2 and b^2 . Hence a suitable formula is:

$$\begin{aligned} & (\forall x)(\forall y)[y^2 + x^2 - 1 = 0 \Rightarrow \\ & a y^2 + b x^2 - 2 b c x + b c^2 - a b > 0]. \end{aligned} \quad (*)$$

By the Theorem, this is equivalent to

$$\begin{aligned} & (\forall x)[\{(\exists y)y^2 + x^2 - 1 = 0\} \Rightarrow \\ & b x^2 - a x^2 - 2 b c x + b c^2 - a b + a > 0]. \end{aligned}$$

But it is straightforward to eliminate the quantifier from $(\exists y)y^2 + x^2 - 1 = 0$, since this defines the projection of the unit circle on the x -axis, which is just $-1 \leq x \leq 1$, i.e. $x^2 \leq 1$. Hence the formula above is equivalent to

$$(\forall x)[x^2 \leq 1 \Rightarrow b x^2 - a x^2 - 2 b c x + b c^2 - a b + a > 0]. \quad (**)$$

Hence we have eliminated one quantifier "by inspection". The computing time of the Collins algorithm may be doubly exponential in the number of variables (see [Col75a]). Thus it is worthwhile to reduce the number of variables.

4. A priori restrictions on free variables. In the Kahan problem, obviously the center of the ellipse must be inside the circle, and each of the semi-axes must be less than one. Hence

$$0 < a < 1$$

$$0 < b < 1$$

$$-1 < c < 1.$$

$$-1 < d < 1.$$

One's first thought might be to simply conjunct such restrictions to the formula to which one applies the quantifier elimination algorithm. Thus in the specialized Kahan problem we consider, one might envision an input formula of

$$(\forall x)[x^2 \leq 1 \Rightarrow b x^2 - 2 b c x + b c^2 a x^2 + a - a b > 0] \ \&$$

$$0 < a < 1 \ \& \ 0 < b < 1 \ \& \ -1 < c < 1.$$

However, this is likely to increase computing time. The cad of E^3 to be constructed must be "compatible" with the zeros of the polynomials in the matrix of the input formula, in order that the truth value of the matrix be invariant on each cell. By adding new conjuncts to the input formula we increase the number of polynomials to be taken into account, hence the cad of E^3 will almost certainly have more cells and take longer to construct.

As seen in our sketch of Collins' quantifier elimination method in Section 2, for a (quantified) input formula with k free variables, the equivalent quantifier-free formula constructed by the Collins algorithm is the disjunction of the defining formulas for the cells in the induced cad of E^k marked "true" ($k = 1$ in Section 2). One way of interpreting *a priori* restrictions on free variables is that they restrict the cells in the induced cad of E^k which can potentially be marked "true". Hence, using the original input formula, we proceed as follows. As we construct cad's of E^1, E^2, \dots, E^k during the quantifier elimination algorithm, we test each cell of each such cad to see if it satisfies the applicable restrictions. If it does, we retain it. If it does not, we throw it out. We end up not with a cad of E^k , but a subset of a cad of E^k , consisting of cells which can potentially be marked true. We extend over those cells, and only those cells, to obtain a subset of a cad of E^r . We then eliminate quantifiers in the usual fashion with the cells we have in E^r, E^{r-1}, \dots, E^k , and end up marking certain cells in E^k "true". Finally, we produce an equivalent quantifier-free formula by forming the disjunction of the defining formulas for the cells in E^k marked "true." Since fewer cells are constructed, the

running time of the cad algorithm will surely be reduced.

For formula (**) of Section 3, we use the first three of the four restrictions above. Without the use of these restrictions, the induced cad of E^1 has 147 cells. When we make use of the restrictions, we retain 35 of these cells. With the restrictions, we get 593 cells in E^2 , and approximately 6000 cells in E^3 (we are in the process of construcing the E^3 cells; the computation pushes our resources to their limits). We do not know how many cells would be obtained in E^2 or E^3 without the use of the restrictions, nor how much time would be needed to construct them, but doing so is most likely well beyond our resources.

5. Customized quantifier elimination. Suppose, for (**) with the first three restrictions of Section 4 imposed, that we have constructed the cells which can potentially be marked "true" in the induced cad of E^3 . Let g be one of these cells. As described in [Arn82a] and [Col75a], we will have constructed a particular point $\langle A, B, C \rangle$ of g that we can compute with. The general Collins algorithm would extend g to a collection of cells in E^4 , and use the truth values of the matrix of the input formula to determine if g should be marked "true". However, by examining (**), we can proceed as follows. In order for g to be marked true, it must be the case that for every point $\langle A, B, C, x \rangle$ of E^4 , either $x^2 > 1$, or $p(x) = Bx^2 - 2Bx + B - A > 0$. Hence we may proceed as follows. If $p(0) = B - A \leq 0$, then g is marked false. Assume $p(0) > 0$. Then if the discriminant of $p(x)$ is negative, p has no real roots, hence p is positive everywhere, hence g can be marked true. If the discriminant of p is nonpositive, then p has a root in $[-1, 1]$ if and only if g is to be marked false. since $p(0) > 0$, we can determine the presence of a root of p in $[-1, 1]$ as follows. Evaluate $p(1)$ and $p(-1)$; if either is nonpositive, then we are done. If both are positive, then determine if the linear polynomial $p'(x)$ has a root α in $[-1, 1]$. If not, we are done. If so, then find the sign of $p(\alpha)$ and complete the analysis. (These calculations are more efficient than using the quadratic formula to find the roots of p).

Preliminary experiments suggest that on average, 7.5 seconds will be required by this method to determine whether a particular cell in E^3 should be marked true or false. Assuming 6000 cells in E^3 , this means that between nine and eighteen hours will be required to carry out the quantifier elimination. In all likelihood, it would be impossible to do the quantifier elimination with the general algorithm.

6. Conclusions. Experience with a computer program for the Collins quantifier elimination algorithm suggests that one will do well to consider procedures such as converting the given input formula into an equivalent formula likely to run faster; reducing a quantifier elimination problem to several smaller q.e. problems; and applying all available knowledge about the problem to avoid unnecessary computation. One can envision an environment in which users state a q.e. problem by means of a formula convenient for them, and supply auxiliary information about the problem. The system then applies an array of simplifying techniques in an effort to find an equivalent version of the problem that can be solved efficiently. In the present paper we have provided three such techniques.

Acknowledgments We are indebted to Prof. L. Lipshitz for bringing to our attention the equivalence of $(\forall z)[\lambda(z) \Rightarrow \rho]$ and $[(\exists z)\lambda(z)] \Rightarrow \rho$, and to Prof. G. Collins for suggesting the use of restrictions on free variables.

References

- Arn81a. DS Arnon, "Algorithms for the geometry of semi-algebraic sets," Technical Report #436, Computer Science Dept., University of Wisconsin, Madison, Wisconsin(1981). (Ph.D. thesis)
- Arn82a. DS Arnon, GE Collins, and S McCallum, "Cylindrical algebraic decomposition I: the basic algorithm," Technical Report CSD TR-427, Computer Science Dept., Purdue University(December, 1982).
- Arn82b. DS Arnon and S McCallum, "Cylindrical algebraic decomposition by quantifier elimination," pp. 215-222 in *Proceedings of the European Computer Algebra Conference (EURO-CAM '82)*, ed. J Calmet ,Lecture Notes in Computer Science, 144, Springer-Verlag, Berlin(1982).
- Col75a. GE Collins, "Quantifier elimination for real closed fields by cylindrical algebraic decomposition," pp. 134-163 in *Proceedings of the Second GI Conference on Automata and Formal Languages*, Lecture notes in Computer Science, 33, Springer-Verlag, Berlin(1975).
- Col80a. GE Collins, "SAC-2 and ALDES now available," *SIGSAM Bulletin (of the Association for Computing Machinery)* 14, p. 19 (1980).
- Kah75a. W Kahan, "Problem #9: An ellipse problem," *SIGSAM Bulletin of the Assoc. Comp. Mach.* 9, p. 11 (1975).
- Kre67a. G Kreisel and JL Krivine, *Elements of mathematical logic (model theory)*, North-Holland, Amsterdam(1967).
- Lau77a. M Lauer, "A solution to Kahan's problem (SIGSAM problem no. 9)," *SIGSAM Bulletin of the Assoc. Comp. Mach.* 11, pp. 16-20 (1977).
- Tar51a. A Tarski, *A decision method for elementary algebra and geometry*, University of California Press(1951). (second revised edition)

AUTOMATICALLY DETERMINING SYMMETRIES
OF ORDINARY DIFFERENTIAL EQUATIONS

Fritz Schwarz
FB Physik
Universitaet
6750 Kaiserslautern
Germany

Abstract.

A REDUCE program is presented which allows to determine the symmetries of ordinary differential equations by using it interactively on a computer. The program does the "easy part" of the work which essentially means anything but genuinely solving differential equations. This is in general rather time consuming and error prone if it is done by hand. If the system arrives at a deadlock the user is asked to supply some additional information. After it is provided the system continues with the solution procedure by itself.

1. Introduction.

Considering any differential equation it is of utmost importance to know its symmetries under point transformations. From a mathematical point of view, the symmetries are the foundation of the integration theory of Sophus Lie. To know a symmetry usually makes it easier to find a solution. For the physical problem which is described by a certain differential equation, the symmetries are closely related to the conserved quantities and therefore lead to additional insight. All aspects which are related to this subject may be found in references [1] to [9]. It turns out that the computations which are necessary for determining all symmetries of a given differential equation are extremely extensive. So it seems rather natural to apply the methods of computer algebra to this problem. To find all symmetries of an ordinary differential equation means to find the most general functions $\xi(x,y)$ and $\eta(x,y)$ in the infinitesimal generator U , eq.(4), such that the solution set is transformed into itself by the corresponding one parameter group. This procedure falls into two major steps.

- i.) Find the determining system of linear partial differential equations for the functions $\xi(x,y)$ and $\eta(x,y)$.
- ii.) Find the general solution of this system.

To perform the first step, there is already a set of REDUCE programs available [10]

which returns the determining system if the differential equation is given as input. This program package covers the most general case, i.e. systems of partial differential equations with an arbitrary number of dependent and independent variables. It turns out that also the second step may be rather time consuming. This is especially burdensome if many differential equations are to be investigated. It is the purpose of this paper to describe a REDUCE program which may be used interactively to determine all symmetry generators of a given ordinary differential equation. In many cases the generators are returned without additional information. In some other cases there occur certain differential equations which the system cannot handle at present and the system asks for a hint to solve them. After this hint has been provided the system continues its own search for the solution. The program together with the theoretical background is described in Section 2. In Section 3 it is shown how it is applied and examples are given. In the final Section 4 extensions of the present work are suggested.

2. Theoretical Background. Description of the Program.

Let the differential equation be

$$(1) \quad \omega(x, y, y', \dots, y^{(n)}) = 0$$

where $y(x)$ is the unknown function. A one parameter group of point transformations is given by

$$(2) \quad \tilde{x} = f(x, y, \varepsilon), \quad \tilde{y} = g(x, y, \varepsilon)$$

with the initial conditions $f(x, y, \varepsilon=0)=x$, $g(x, y, \varepsilon=0)=y$. Expanding (2) around $\varepsilon=0$ defines the infinitesimals $\xi(x, y)$ and $\eta(x, y)$ by

$$(3.a) \quad \tilde{x} = x + \varepsilon \xi(x, y) + O(\varepsilon^2)$$

$$(3.b) \quad \tilde{y} = y + \varepsilon \eta(x, y) + O(\varepsilon^2)$$

The symbol U of the infinitesimal transformation (3) is

$$(4) \quad U = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y}$$

The differential equation (1) is said to be invariant with respect to the one parameter group (2), (3) if

$$(5) \quad U^{(n)} \omega = 0$$

on the manifold $\omega=0$ in the space of the variables $x, y, y', \dots, y^{(n)}$. The n -th extension $U^{(n)}$ of U is defined by

$$(6) \quad U^{(n)} = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y} + \sum_{\ell=1}^n \zeta_\ell \frac{\partial}{\partial y^{(\ell)}}$$

with

$$(7) \quad \zeta_1 = D(\eta) - y' D(\xi), \quad \zeta_\ell = D(\zeta_{\ell-1}) - y^{(\ell)} D(\xi), \quad \ell = 2, 3, \dots$$

and

$$(8) \quad D = \frac{\partial}{\partial x} + y' \frac{\partial}{\partial y}$$

To obtain the symmetries of a differential equation (1) one has to proceed as follows. At first the n-th extension of U is computed by using (6) and (7). It is applied to the differential equation according to (5). In the resulting polynomial in y' , y'' , $y^{(n)}$ one of these derivatives is eliminated by using the differential equation itself. The coefficients of this polynomial depend on x , y , $\xi(x,y)$, $\eta(x,y)$ and derivatives thereof. To annihilate it, the coefficient of each term has to vanish. This leads to a system of linear partial differential equations for the functions $\xi(x,y)$ and $\eta(x,y)$ which may be obtained by applying the REDUCE package described in ref.[10].

An arbitrary system of this kind is rather difficult to solve and it may be impossible to obtain the general solution explicitly. It is the special structure which originates from the problem under consideration which makes it possible to simplify or even to completely solve it by applying the following set of rules.

i.) Single term relations involving $\xi(x,y)$ and $\eta(x,y)$. Any equation of the form

$$(9) \quad \frac{\partial^{m+n} \xi(x,y)}{\partial x^m \partial y^n} = 0 \quad m,n = 0, 1, \dots$$

leads to a substitution

$$(10) \quad \xi(x,y) = \sum_{\mu=0}^{m-1} c_{\mu+1}(y)x^\mu + \sum_{\nu=0}^{n-1} c_{m+\nu+1}(x)y^\nu$$

and similarly for derivatives of $\eta(x,y)$. An equation of the form (9) may also be obtained by differentiating the original system an appropriate number of times. By a substitution like (10) a function of two variables is replaced by functions depending on a single variable which generally means an important step towards the final solution of the system. The search for a relation of the form (9) is optimized with respect to the order of the derivative.

ii.) Separation with respect to specific variable. If the dependence on either variable has become completely explicit in a certain equation, it may be separated with respect to this variable. In this way the number of equations is enlarged while the new equations are usually simpler than the original one.

iii.) Single term relations involving functions $c_i(x)$, $c_i(y)$ or constants c_i . Any equation of the form

$$(11) \quad c_i^{(n)}(x)=0, \quad n=0,1,\dots$$

leads to a substitution

$$(12) \quad c_i(x)=\sum_{v=0}^{n-1} c_{i+v+1} x^v$$

and similarly for functions depending on y . As in the previous case, equations of the form (11) may also be obtained by appropriate differentiation.

iv.) Integration of sums of derivatives. Relations of the form

$$(13) \quad \sum_i \alpha_i(x,y) c_i^{(k_i)}(x) + \sum_j \beta_j(x,y) c_j^{(k_j)}(y) + \sum_\ell \gamma_\ell(x,y) c_\ell = 0$$

where α_i , β_i and γ_ℓ are explicit functions of their arguments and $k_i, k_j > 0$ may be integrated once or several times until at least a single term is free of derivatives.

v.) Arbitrary sums containing functions $c_i(x)$, $c_j(y)$ or constants c_ℓ . A relation of the form (13) where at least a single index $k_i=0$ or $k_j=0$ and at the same time the index i or j respectively does not occur in any other term of the equation, may be resolved with respect to this function. The corresponding substitution in general leads to a simplification of the system.

To apply the rules i.) to v.) the system must be able to identify the respective cases. This is most effectively performed by applying the operator PART which is available in the new REDUCE system [11]. It is essentially a short hand notation for the LISP functions CAR, CDR etc. The basic steps which are necessary for this purpose will be described now. Let the equations of the system be $GL(I)$, $I=1\dots NGL$, the value of NGL being the number of equations at a certain step. To identify a single term relation in i.) or iii.), at first numerical factors have to be removed. This is achieved by

```
ZZ:=GL(I);
IF PART(ZZ,0)=MINUS THEN ZZ:=-ZZ;
WHILE PART(ZZ,0)=TIMES DO
  IF ATOM(PART(ZZ,1))=T THEN ZZ:=ZZ/PART(ZZ,1)
  ELSE ZZ:=ZZ/PART(ZZ,-1);
```

After getting rid of these irrelevant factors the program continues in part i.) to find a derivative of $\xi(x,y)$ or $\eta(x,y)$ by

```

IF PART(ZZ,0)=DF THEN
BEGIN
ZZ:=DF(ZZ,X,2,Y,2);
PX:=PART(ZZ,3)-2; PY:=PART(ZZ,-1)-2;
IF PART(ZZ,1,0)=XI THEN <substitution command>;
IF PART(ZZ,1,0)=ETA THEN <substitution command>;
END;

```

Single term relations in part iii.) are taken into account by

```

IF PART(ZZ,0)=C THEN <substitution command>;
IF PART(ZZ,0)=DF THEN
BEGIN
IF LENGTH(ZZ)=3 THEN POT:=1;
IF LENGTH(ZZ)=4 THEN POT:=PART(ZZ,3);
VAR:=PART(ZZ,2);
N:=PART(ZZ,1,1);
<substitution command>;
END;

```

Here POT, VAR and N are the order of the derivative, the variable which occurs as argument and the index of the function respectively.

The separation with respect to explicitly occurring variables in ii.) is most easily performed by the COEFF command which is available in the algebraic mode of REDUCE.

To avoid infinite recursions while applying iv.), dependency on x or y respectively has to be assured as well as the actual integration of all terms. This is achieved by

```

IF NOT FREEOF(ZZ,X) THEN WHILE FREEOF(INT(ZZ,X),INT) DO
<<NFUN:=NFUN+1; ZZ:=INT(ZZ,X)+C(NFUN);>>;

```

and similarly for integrations with respect to y.

To apply the rule v.), the system searches for an equation of minimal length. The corresponding substitution leads to a reduction of the number of functions or constants involved. To avoid infinite recursion, the function at the left hand side of the substitution rule cannot appear at the right hand side.

The rules i.) to v.) are applied iteratively beginning with rule i.). If a hit is found all equations are evaluated anew taking into account the respective

simplification. Then the system goes back to i.). If all equations are identically zero this procedure terminates and the general solution for $\xi(x,y)$ and $\eta(x,y)$ is returned. If this is not true and the system cannot identify any more one of the cases i.) to v.) the user is asked for help. In general this means that the solution of a differential equation is needed which the system cannot find. With this additional information the system continues the solution procedure by itself.

3. How to use the system.

As it has already been mentioned above, the system is used interactively to determine the symmetries of a given differential equation. It is mostly written in the algebraic mode of REDUCE with the exception of a few procedures which are considerably faster in the symbolic mode. It will be shown now by several examples how it is used in practice. Let us consider first the differential equation $y''=0$ the symmetry group of which is known to be generated by eight independent infinitesimal operators. After the program has been called the system asks

DIFFERENTIAL EQUATION DEQ:=?

DERIVATIVE WHICH IS SUBSTITUTED SDER:=?

OUTPUT OF INTERMEDIATE STEPS PR:=YES/NO?

The user should reply first N and then answer the questions which the system has asked. For the example under consideration the appropriate answers are

DEQ:=Y(2)\$

SDER:=2\$

PR:=NO\$

CONT\$

With the final statement the system has been told to begin with the solution procedure as described in the previous section. We have chosen not to require the intermediate steps to be printed out by choosing PR=NO because the output is rather lengthy. For the present example the system finds the final answer without additional information. After a while it returns

THE GENERAL SOLUTION HAS BEEN FOUND

XI(X,Y):= $x^2*c(6)+x*y*c(10)+x*c(12)+y*c(14)+c(11)$

ETA(X,Y):= $x*y*c(6)+x*c(5)+y^2*c(10)+y*c(9)+c(8)$

Next consider the differential equation $y''+y=0$ which is known to describe a free harmonic oscillator without damping. In this case the system is not able to

complete the solution process without help. Instead the response is

SOLUTION CANNOT BE FOUND

THE REMAINING EQUATIONS ARE

GL(1):=C(3,X)+DF(C(3,X),X,2)

GL(2):=C(8)+C(2,X)+DF(C(2,X),X,2)

GL(3):=2*C(7)+4*C(4,X)+DF(C(4,X),X,2)

GL(4):=C(2,X)+DF(C(2,X),X,2)

DO YOU HAVE A HINT?

This example shows what is generally meant by the "easy part" of the problem. It is everything which does not involve genuinely solving differential equations. By applying rules i.) to iv.) the system picks out those differential equations which may obviously be solved by integration. Either this process terminates with the final solution or the system asks for help to overcome the deadlock. In our present example the general solution to the remaining differential equations is known. It has to be told to the system as the substitution command

```
C(8):=0$  
C(2,X):=C(9)*SIN(X)+C(10)*COS(X)$  
C(3,X):=C(11)*SIN(X)+C(12)*COS(X)$  
C(4,X):=C(13)*SIN(2*X)+C(14)*COS(2*X)-C(7)/2$
```

With this information the final solution is returned

THE GENERAL SOLUTION HAS BEEN FOUND

```
XI(X,Y):=SIN(X)*C(9)*Y+COS(X)*C(10)*Y  
+SIN(2*X)*C(14)-COS(2*X)*C(13)-1/2*C(6)  
ETA(X,Y):=SIN(X)*(C(11)-C(10)*Y2)+COS(X)*(C(12)+C(9)*Y2)  
+Y*(SIN(2*X)*C(13)+COS(2*X)*C(14)-1/2*C(7))
```

It is easily seen to agree with the results given in ref.[11]. These examples are representative for the two possibilities which may occur.

The standard test basis for any results related to differential equations is the collection given in the book by Kamke [13]. We have applied our system to about 100 differential equations which are mostly taken from this reference. The experience which we have gained in this way may be described as follows. In about one third of our test runs the system has been able to find the general solution without additional help. This category comprises equations like $y^{(n)}=0$, $y^{(n)}+y^m=0$ and $y^{(m)}+y^n+x^\ell=0$. Although the general solution may look rather simple or even trivial, e.g. $\xi(x,y)= n(x,y)=0$ for the latter class of

equations, the computational work at intermediate steps may be tremendous. This is especially true if the order of the differential equation is larger than two. Examples of this kind are given in Ch. 7 of ref.[13]. For most of them the symmetries have been determined with our program. To give the reader an idea of the size of these calculations, some of the results will be described in more detail. The equation describing the set of all circles in the plane is

$$(14) \quad (y'^2+1)y'''-3y'y''^2=0$$

The initial separation of the polynomial (5) with respect to derivatives of y leads to a system of 18 partial differential equations for ξ and η . After about 30 pages of output in the PR=YES version and about 200 seconds computing time on the VAX 11/750 the following result is returned

$$(15a) \quad XI(X,Y)=C(11)+C(12)*X+C(15)*Y-2*C(7)*X*Y+C(13)*(X^2-Y^2)$$

$$(15b) \quad ETA(X,Y)=C(23)-C(15)*X+C(12)*Y+2*C(13)*X*Y+C(7)*(X^2-Y^2)$$

i.e. a finite Lie group containing 6 parameters. The differential equation describing certain conic sections is

$$(16) \quad 9y''^2y^{(5)}-45y''y'''y^{(4)}+40y''^3=0$$

The initial separation leads to a set of about 50 partial differential equations for ξ and η . After a similar amount of computational work as in the previous example the general solution for XI and ETA obtained

$$(17a) \quad XI(X,Y)=C(5)+C(6)*X+C(10)*Y+C(11)*X*Y+C(7)*X^2$$

$$(17b) \quad ETA(X,Y)=C(19)+C(15)*X+C(20)*Y+C(7)*X*Y+C(11)*Y^2$$

i.e. the symmetry group of (16) is a 8 parameter Lie group. Both the examples (14) and (16) are already beyond the scope which may reasonably be performed by hand.

Physically interesting cases which have been considered are, among others, the Blasius equation (ref.[3]) $y'''+kyy''=0$ with the generators $\xi(x,y)=c_1x+c_2$, $\eta(x,y)=-c_1y$ and the equation $y'''+(y-v)y'=0$ which describes a certain class of similarity solutions of the Korteweg deVries equation; its symmetry generators are $\xi(x,y)=c_1x+c_2$ and $\eta(x,y)=2c_1(v-y)$. Both cases are completed by the system without help. Other cases of physical interest are the equations $y''+2ky^n+ay=0$ and $y''+\sin(y)=0$ which describe certain types of oscillations. In these cases the user has to provide some additional information at intermediate steps.

4. Concluding remarks.

The basic importance of a system like the one described here is obvious. In addition to that it may be taken for granted that it is only the first little step of a developement which is going to occur in the years to come. The tremendous amount of computations which occurs in connection with differential equations makes this field predetermined for the application of the methods of computer algebra. Concerning the symmetries of differential equations, the present work should be generalized to all cases which have been considered in ref [11]. This is the prerequisite for applying it to nonlinear evolution equations which is one of the most promising and most rapidly developing fields in mathematical physics.

To take advantage of the symmetries of a given differential equation, the corresponding group invariants have to be determined. They must be used as new variables. This procedure lowers the order of an ordinary differential equation and diminishes the number of independent variables in a partial differential equation. Both steps are completely algorithmic but involve considerable amounts of computations. It is highly desirable to have similar programs available to perform these steps as for determining the symmetries.

Acknowledgment.

The continuous support of Dr. A.C. Hearn in applying the new REDUCE system and the hospitality at the Rand Corporation, where part of this work has been performed, are gratefully acknowledged. The advice of Ben Hulshof in writing the REDUCE programs was extremely useful.

References.

- [1]. Sophus Lie, Differentialgleichungen, Leipzig, 1891; reprinted by Chelsea Publishing Company, New York, 1967.
- [2]. L.V.Ovsjannikov, Group Properties of Differential equations Novosibirsk, 1962, translated by G.W. Bluman; Group Analysis of Differential Equations, Nauka, 1979, English translations edited by W.F.Ames, Academic Press, New York, 1982.
- [3]. G.W.Bluman and J.D.Cole, Similarity Methods for Differential Equations, Applied Mathematics Series 13, Springer New York, 1974.
- [4]. R.L.Anderson and N.H.Ibragimov, Lie Baecklund Transformations in Applications, SIAM, Philadelphia, 1979.
- [5]. D.H.Sattinger, Les symetries des equations et leurs applications dans la mecanique et la physique, Publications Mathematiques d'Orsay 80.08, 1980.
- [6]. P. Olver, J. Differential Geometry 14, 497(1979).
- [7]. J.E.Campbell, Introductory Treatise on Lie's Theory of Finite Continuous Transformations Groups, Chelsea, New York, 1966.
- [8]. A.G.Hansen, Similarity Analysis of Boundary Value Problems in Engineering, Prentice Hall Inc., Englewood Cliffs, N.J., 1964.
- [9]. W.F.Ames, Nonlinear Partial Differential Equations in Engineering, vol.II, Academic Press, 1972.
- [10] A.C.Hearn, REDUCE User's Manual, Third Edition, Santa Monica, 1983
- [11].F.Schwarz, Computer Physics Communications 27, 179(1982).
- [12].C.E.Wulfman, B.G.Wybourne, J.Phys. A9, 507(1976).
- [13].E.Kamke, Differentialgleichungen I, Akademische Verlagsgesellschaft, Leipzig, 1944.

ALGEBRAIC COMPUTATION OF THE STATISTICS OF THE SOLUTION
OF SOME NONLINEAR STOCHASTIC DIFFERENTIAL EQUATIONS

F. LAMNABHI-LAGARRIGUE
Laboratoire des Signaux et Systèmes
E.S.E., Plateau du Moulon
91190 Gif sur Yvette

M. LAMNABHI
Laboratoire d'Electronique et de Physique Appliquée
3, Avenue Descartes
94450 Limeil-Brevannes

This paper presents an algebraic method for computing the statistics of the solution of some stochastic non-linear differential equations by mean of the Volterra functional expansion. The symbolic calculus introduced, based on noncommutative variables and iterated integrals has the advantage of allowing easily the use of symbolic computation systems, like REDUCE or MACSYMA, to perform the manipulations. This becomes necessary as soon as one tries to get high order terms.

I. INTRODUCTION

Volterra or Wiener functional series has been widely used in the study of the statistical properties of the output of nonlinear systems with a random input. They have been introduced by Wiener in 1942 [10] for nonlinear circuits analysis. Since Wiener's early work, many authors have dealt with the subject. Among them, the papers by Barrett [2] and Bedrosian and Rice [3] are of a particular importance. But the difficulties involved in obtaining the terms of the series and in performing the required integrations reduce the interest of the method.

Recently, a new approach of causal functionals was proposed, using non-commutative variables and iterated integrals [4]. In this approach, the input/output behaviour of a nonlinear system is represented in termes of a formal power series, called the generating power series. There is, in fact, a strong relationship between Volterra series and noncommutative generating power series. The present paper shows how to use this series

to determine the statistics of the output of some nonlinear systems driven by a white Gaussian noise. It is a sequel to an earlier paper [7] where we described an algebraic algorithm for computing the response of a nonlinear system to deterministic inputs. The symbolic calculus introduced has the advantage of allowing easily the use of symbolic computation systems, like REDUCE or MACSYMA, to perform the manipulations. This becomes necessary as soon as one tries to get high order terms.

II. VOLTERRA SERIES

Let us consider a system described by the Volterra series [8],[9]

$$y(t) = h_0(t) + \int_0^t h_1(t, \tau_1) u(\tau_1) d\tau_1 + \dots + \int_0^t \int_0^{\tau_n} \dots \int_0^{\tau_2} h_n(t, \tau_n, \dots, \tau_1) u(\tau_1) \dots u(\tau_n) d\tau_1 \dots d\tau_n + \dots \quad (1)$$

where $y(t)$ is the system output and $u(t)$ is the system input. The kernels h_i are assumed to be analytic. Their Taylor expansion may be written

$$h_n(t, \tau_n, \dots, \tau_1) = \sum_{i_0, i_1, \dots, i_n \geq 0}^{(i_0, i_1, \dots, i_n)} h_n^{(i_0, i_1, \dots, i_n)} \\ \times \frac{(t - \tau_n)^{i_n} (\tau_n - \tau_{n-1})^{i_{n-1}} \dots (\tau_2 - \tau_1)^{i_1} \tau_1^{i_0}}{i_n! \dots i_1! \dots i_0!}$$

with respect to the new variables $\tau_1, \tau_2 - \tau_1, \dots, t - \tau_n$.

Each n-dimensional integral

$$\int_0^t \int_0^{\tau_n} \dots \int_0^{\tau_2} \frac{(t - \tau_n)^{i_n} (\tau_n - \tau_{n-1})^{i_{n-1}} \dots (\tau_2 - \tau_1)^{i_1} \tau_1^{i_0}}{i_n! i_{n-1}! \dots i_0!} u(\tau_1) \dots u(\tau_n) d\tau_1 \dots d\tau_n$$

can be shown to be equal to the iterated integral

$$\underbrace{x_0 \dots x_0}_{i_n} \underbrace{x_1 \dots x_0}_{i_{n-1}} \dots \underbrace{x_1 \dots x_1}_{i_1} \underbrace{x_0 \dots x_0}_{i_0}$$

where the letter x_0 denotes the integration with respect to time and the letter x_1 the integration with respect to time after multiplying by the input u . This is a generalization of the well known formula

$$\int_0^t \frac{(t-\tau)^n}{n!} u(\tau) d\tau = \int_0^t d\tau_n \int_0^{\tau_n} \dots \int_0^{\tau_2} d\tau_1 \int_0^{\tau_1} u(\tau_o) d\tau_o$$

This allow to write (1) symbolically in the form

$$g = \sum_{n \geq 0} \sum_{i_0, i_1, \dots, i_n \geq 0} h_n^{(i_0, i_1, \dots, i_n)} x_0^{i_n} x_1^{i_{n-1}} \dots x_o^{i_1} x_1^{i_0}$$

g is called the *noncommutative generating power series* associated with the system. This power series can, as we shall see in a next section, be derived directly from the nonlinear differential equations governing the dynamic of a system.

Of course this is a noncommutative series because

$$\int_0^t d\tau_1 \int_0^{\tau_1} u(\tau_2) d\tau_2 \neq \int_0^t u(\tau_1) d\tau_1 \int_0^{\tau_2} d\tau_2$$

that is $x_0 x_1 \neq x_1 x_0$.

III. NONCOMMUTATIVE GENERATING POWER SERIES

Let $X = \{x_0, x_1\}$ be a finite alphabet and X^* the monoid generated by X . An element of X^* is a word, i.e. a sequence $x_{j_v} \dots x_{j_0}$ of letters of the alphabet. The product of two words $x_{j_v} \dots x_{j_0}$ and $x_{k_u} \dots x_{k_0}$ is the concatenation $x_{j_v} \dots x_{j_0} x_{k_u} \dots x_{k_0}$. The neutral element is called the empty word and denoted by l . A formal power series with real or complex coefficients is written as a formal sum

$$g = \sum_{w \in X^*} (g, w) w, \quad (g, w) \in \mathbb{R} \text{ or } \mathbb{C}.$$

Let g_1 and g_2 be two formal power series, the following operations are defined :

$$\underline{\text{Addition}} \quad g_1 + g_2 = \sum_{w \in X^*} [(g_1, w) + (g_2, w)] w$$

$$\underline{\text{Cauchy product}} \quad g_1 \cdot g_2 = \sum_{w \in X^*} \left[\sum_{w_1 w_2 = w} (g_1, w_1) (g_2, w_2) \right] w$$

$$\underline{\text{Shuffle product}} \quad g_1 \bowtie g_2 = \sum_{w_1, w_2 \in X^*} (g_1, w_1) (g_2, w_2) w_1 \omega w_2$$

The shuffle product of two words consists of mixing the letters of the two words keeping the order of each one. For example

$$\begin{aligned} x_0 x_1^{\omega} x_1 x_p &= \overbrace{x_0 x_1}^{\omega} \underbrace{x_1 x_0}_{\omega} + \overbrace{x_0 x_1}^{\omega} \underbrace{x_1 x_0}_{\omega} + \overbrace{x_0 x_1}^{\omega} \underbrace{x_0 x_1}_{\omega} + \overbrace{x_1 x_0}^{\omega} \underbrace{x_1 x_0}_{\omega} \\ &\quad + \underbrace{x_1 x_0}_{\omega} x_1 + \underbrace{x_1 x_0}_{\omega} x_1 \\ &= 2x_0 x_1^2 x_0 + x_0 x_1 x_0 x_1 + x_1 x_0 x_1 x_0 + x_1 x_0^2 x_1 \end{aligned}$$

Let us now consider two generating power series g_1 and g_2 associated respectively with the output systems $y_1\{t, u(t)\}$ and $y_2\{t, u(t)\}$. It can be shown [4] that the generating power series associated with the product

$$y_1\{t, u(t)\} \times y_2\{t, u(t)\}$$

is the shuffle product of the generating power series g_1 and g_2 .

IV. DERIVATION OF GENERATING POWER SERIES

In this section, we describe an algorithm for finding algebraically the generating power series associated with the solution of a nonlinear forced differential equation. The equation we are going to consider is

$$\begin{aligned} Ly(t) + \sum_{i=2}^m a_i y^i(t) &= u(t) \\ L = \sum_{i=0}^n \ell_i \frac{d^i}{dt^i}, \quad (\ell_n &= 1) \end{aligned}$$

or, in its integral form

$$\begin{aligned} y(t) + \ell_{n-1} \int_0^t y(\tau_1) d\tau_1 + \ell_{n-2} \int_0^t d\tau_2 \int_0^{\tau_2} y(\tau_1) d\tau_1 \\ + \dots + \ell_0 \int_0^t d\tau_n \int_0^{\tau_n} d\tau_{n-1} \dots d\tau_2 \int_0^{\tau_2} y(\tau_1) d\tau_1 \\ + \sum_{i=2}^m a_i \int_0^t d\tau_n \int_0^{\tau_n} d\tau_{n-1} \dots d\tau_2 \int_0^{\tau_2} y^i(\tau_1) d\tau_1 = \\ = \int_0^t d\tau_n \int_0^{\tau_n} d\tau_{n-1} \dots d\tau_2 \int_0^{\tau_2} u(\tau_1) d\tau_1 \end{aligned} \tag{2}$$

Here we assume, for simplicity's sake, zero initial conditions.

Let g denotes the generating power series associated with $y(t)$, then (2) can be written symbolically

$$g + \sum_{j=0}^{n-1} \ell_j x_o^{n-i} g + x_o^n \sum_{i=1}^m a_i \underbrace{gw\dotswg}_{i \text{ times}} = x_o^{n-1} x_1$$

where $\underbrace{gw\dotswg}_{i \text{ times}}$ corresponds, according to the previous theorem, to the nonlinear functional $y^i(t)$.

This algebraic equation can be solved iteratively, following the recursive scheme

$$g = g_1 + g_2 + \dots + g_n + \dots$$

with

$$g_1 = \left(1 + \sum_{i=0}^{n-1} \ell_i x_o^{n-i} \right)^{-1} x_o^{n-1} x_1$$

and

$$g_n = - \left(1 + \sum_{i=0}^{n-1} \ell_i x_o^{n-i} \right)^{-1} x_o^n \sum_{i=2}^m a_i \sum_{v_1+v_2+\dots+v_i=n} g_{v_1} \omega g_{v_2} \dots \omega g_{v_i}$$

To have the closed form expression of g_i , one only need to compute the shuffle product of noncommutative power series of the form

$$\left(1 - a_o x_o \right)^{-1} x_{i_1} \left(1 - a_1 x_o \right)^{-1} x_{i_2} \dots x_{i_p} \left(1 - a_p x_o \right)^{-1} ; \quad i_1, i_2, \dots, i_p \in \{0, 1\}.$$

This results from the following proposition [6] :

Proposition 1 : Given two formal power series

$$g_1^p = \left(1 - a_o x_o \right)^{-1} x_{i_1} \left(1 - a_1 x_o \right)^{-1} x_{i_2} \dots x_{i_p} \left(1 - a_p x_o \right)^{-1} = g_1^{p-1} x_{i_p} \left(1 - a_p x_o \right)^{-1}$$

and

$$g_2^q = \left(1 - b_o x_o \right)^{-1} x_{j_1} \left(1 - b_1 x_o \right)^{-1} x_{j_2} \dots x_{j_q} \left(1 - b_q x_o \right)^{-1} = g_2^{q-1} x_{j_q} \left(1 - b_q x_o \right)^{-1}$$

where p and q belongs to \mathbb{N} , the subscripts $i_1, \dots, i_p, j_1, \dots, j_q$ to $\{0, 1\}$ and a_i, b_j to \mathbb{C} ; the shuffle product is given by induction on the length by

$$\begin{aligned} g_1^p \omega g_2^q &= \left(g_1^{p-1} \omega g_2^{q-1} \right) x_{j_q} \left[1 - (a_p + b_q) x_o \right]^{-1} \\ &\quad + \left(g_1^{p-1} \omega g_2^q \right) x_{i_p} \left[1 - (a_p + b_q) x_o \right]^{-1} \end{aligned}$$

$$\text{with } \left(1 - ax_o \right)^{-1} \omega \left(1 - bx_o \right)^{-1} = \left[1 - (a+b)x_o \right]^{-1}.$$

Using this proposition, g_i is obtained as a finite sum of expressions of the form :

$$\left(\frac{1-a_0 x_0}{x_0} \right)^{-1} x_{i_1} \left(\frac{1-a_1 x_0}{x_0} \right)^{-1} x_{i_2} \dots x_{i_n} \left(\frac{1-a_n x_0}{x_0} \right)^{-1} ; \quad i_1, \dots, i_n \in \{0, 1\} \quad (3)$$

Example 1 :

The technique presented above is now used to compute the generating power series associated with the nonlinear differential equation

$$\dot{y}(t) + k_1 y(t) + k_2 y^2(t) = u(t) \quad (4)$$

Its integral form is :

$$y(t) + k_1 \int_0^t y(\tau) d\tau + k_2 \int_0^t y^2(\tau) d\tau = \int_0^t u(\tau) d\tau$$

where we assume a zero initial condition.

Thus, the generating power series g is simply the solution of

$$g + k_1 x_0 g + k_2 x_0 g \otimes g = x_1$$

This equation is solved iteratively by a computer program (table 1).

$$\begin{aligned}
 g &= 1 \\
 &\quad k_1 x_1 0 \\
 -2 &\quad k_2 x_0 2k_1 x_1 k_1 x_1 0 \\
 +4 &\quad k_2^2 x_0 2k_1 x_1 k_1 x_0 2k_1 x_1 k_1 x_1 0 \\
 +12 &\quad k_2^2 x_0 2k_1 x_0 3k_1 x_1 2k_1 x_1 k_1 x_1 0 \\
 -8 &\quad k_2^3 x_0 2k_1 x_1 k_1 x_0 2k_1 x_1 k_1 x_0 2k_1 x_1 k_1 x_1 0 \\
 -24 &\quad k_2^3 x_0 2k_1 x_0 3k_1 x_1 2k_1 x_1 k_1 x_0 2k_1 x_1 k_1 x_1 0 \\
 -72 &\quad k_2^3 x_0 2k_1 x_0 3k_1 x_1 2k_1 x_0 3k_1 x_1 2k_1 x_1 k_1 x_1 0 \\
 -24 &\quad k_2^3 x_0 2k_1 x_1 k_1 x_0 2k_1 x_0 3k_1 x_1 2k_1 x_1 k_1 x_1 0 \\
 -144 &\quad k_2^3 x_0 2k_1 x_0 3k_1 x_0 4k_1 x_1 3k_1 x_1 2k_1 x_1 k_1 x_1 0
 \end{aligned}$$

table 1

where the symbolic notation

$$\frac{x_{i_1}}{a_0} \quad \frac{x_{i_2}}{a_1} \quad \dots \quad \frac{x_{i_n}}{a_{n-1}} \quad ; \quad i_1, \dots, i_n \in \{0, 1\}$$

stands for $\left(1 + a_0 x_o\right)^{-1} x_{i_1} \left(1 + a_1 x_o\right)^{-1} x_{i_2} \dots \left(1 + a_{n-1} x_o\right)^{-1} x_{i_n} \left(1 + a_n x_o\right)^{-1}$

Remark : The expansion, in table 1, is equivalent to the Volterra series expansion of the solution up to order 5. The algebraic closed form expression of triangular Volterra kernels can be easily deduced from it [6], since the expression (3) is the symbolic representation of the n-dimensional integral

$$\int_0^t \int_0^{\tau_n} \dots \int_0^{\tau_2} e^{a_0(\tau - \tau_n)} e^{a_1(\tau_n - \tau_{n-1})} \dots e^{a_{n-1}(\tau_2 - \tau_1)} e^{a_n \tau_1} u_{i_n}(\tau_1) \dots u_{i_2}(\tau_{n-1}) u_{i_1}(\tau_n) d\tau_1 \dots d\tau_n$$

where

$$\{i_1, \dots, i_n\} \in \{0, 1\}, \quad u_o(\tau) = 1 \quad \text{and} \quad u_1(\tau) = u(\tau),$$

which we shall denote later by

$$\left[\left(1 - a_0 x_o\right)^{-1} x_{i_1} \left(1 - a_1 x_o\right)^{-1} x_{i_2} \dots x_{i_n} \left(1 - a_n x_o\right)^{-1} \right]_o^t$$

V. A SYMBOLIC CALCULUS FOR THE OUTPUT STATISTICS

In the previous section, we used noncommutative generating power series to derive, by simple algebraic manipulations, a functional expansion (i.e. the Volterra series) of the solution of some nonlinear differential equations. In the following the derived series is used to obtain closed form expressions for the *moments* and *correlations* of the output of a Volterra system driven by Gaussian white noise.

Output moments

Let us consider a zero-mean Gaussian process $u(t)$ with correlation function

$$E[u(t)u(\tau)] = \sigma^2 \delta(t-\tau)$$

where δ is the Dirac delta function (a process of this kind is usually called a white Gaussian noise) [1]. The first-order moment of the output $E[y(t)]$ is then obtained by a simple rule on each term (3) of the functional expansion of $y(t)$. This rule results [5] from the classical rules of stochastic differential calculus and is given by induction on the length by

$$\begin{aligned}
& E \left[\int_0^t \int_0^{\tau_n} \cdots \int_0^{\tau_2} e^{a_o(t-\tau_n)} u_{i_1}(\tau_n) e^{a_1(\tau_n-\tau_{n-1})} u_{i_2}(\tau_{n-1}) \cdots u_{i_n}(\tau_1) e^{a_n \tau_1} d\tau_1 \cdots d\tau_n \right] \\
& = \begin{cases} \int_0^t e^{a_o(t-\tau_n)} d\tau_n E \left[\int_0^{\tau_n} e^{a_1(\tau_n-\tau_{n-1})} u_{i_2}(\tau_{n-1}) \cdots u_{i_n}(\tau_1) e^{a_n \tau_1} d\tau_1 \cdots d\tau_{n-1} \right] & \text{if } i_1 = 0 \\ \left(\frac{\sigma^2}{2} \right) \int_0^t e^{a_o(t-\tau_n)} d\tau_n E \left[\int_0^{\tau_n} e^{a_2(\tau_n-\tau_{n-2})} u_{i_3}(\tau_{n-2}) \cdots u_{i_n}(\tau_1) e^{a_n \tau_1} d\tau_1 \cdots d\tau_{n-2} \right] & \text{if } i_1 = i_2 = 1 \\ 0 \quad \underline{\text{otherwise}} \end{cases}
\end{aligned}$$

In fact, the moment of the output function $y(t)$ can be obtained directly from its associated generating power series by the following algebraic rule which is a symbolic representation of the previous one :

$$\begin{aligned}
& < \left(1 - a_o x_o \right)^{-1} x_{i_1} \left(1 - a_1 x_o \right)^{-1} x_{i_2} \cdots x_{i_n} \left(1 - a_n x_o \right)^{-1} > \quad (5) \\
& = \begin{cases} \left(1 - a_o x_o \right)^{-1} x_o < \left(1 - a_1 x_o \right)^{-1} x_{i_2} \cdots x_{i_n} \left(1 - a_n x_o \right)^{-1} > & \text{if } i_1 = 0 \\ \left(\frac{\sigma^2}{2} \right) \left(1 - a_o x_o \right)^{-1} x_o < \left(1 - a_2 x_o \right)^{-1} x_{i_3} \cdots x_{i_n} \left(1 - a_n x_o \right)^{-1} > & \text{if } i_1 = i_2 = 1 \\ 0 \quad \underline{\text{otherwise}} \end{cases}
\end{aligned}$$

This results in a rational fraction in the only variable x_o . Its decomposition into partial fractions and the following lemma give its corresponding expression in time.

Lemma : The rational fraction

$$\left(1 - ax_o \right)^{-p}$$

corresponds to the exponential polynomial

$$\sum_{j=0}^{p-1} \binom{j}{p-1} \frac{a^j t^j}{j!} e^{at} .$$

In order to illustrate the use of this rule, consider again the nonlinear differential equation (4). For the first-order moment of $y(t)$ we have then

$$\begin{aligned} \langle g \rangle = & -2\left(\frac{\sigma^2}{2}\right)k_2 \begin{matrix} x_0 \\ k_1 \end{matrix} \begin{matrix} x_0 \\ 2k_1 \end{matrix} \begin{matrix} x_0 \\ 0 \end{matrix} \\ & -24\left(\frac{\sigma^2}{2}\right)^2 k_2^3 \begin{matrix} x_0 \\ k_1 \end{matrix} \begin{matrix} x_0 \\ 2k_1 \end{matrix} \begin{matrix} x_0 \\ 3k_1 \end{matrix} \begin{matrix} x_0 \\ k_1 \end{matrix} \begin{matrix} x_0 \\ 2k_1 \end{matrix} \begin{matrix} x_0 \\ 0 \end{matrix} \\ & -144\left(\frac{\sigma^2}{2}\right)^2 k_2^3 \begin{matrix} x_0 \\ k_1 \end{matrix} \begin{matrix} x_0 \\ 2k_1 \end{matrix} \begin{matrix} x_0 \\ 3k_1 \end{matrix} \begin{matrix} x_0 \\ 4k_1 \end{matrix} \begin{matrix} x_0 \\ 2k_1 \end{matrix} \begin{matrix} x_0 \\ 0 \end{matrix} \\ & \vdots \end{aligned}$$

By decomposing into partial fractions, we get the corresponding time function :

$$\begin{aligned} E[y(t)] = & 2\left(\frac{\sigma^2}{2}\right)\frac{1}{k_1^2} \left(e^{-k_1 t} - \frac{3}{4} - \frac{1}{4}k_1 t e^{-2k_1 t} \right) \\ & + 24\left(\frac{\sigma^2}{2}\right)^2 \frac{1}{k_1^5} \left[\left(\frac{1}{4} + \frac{1}{2}k_1 t \right) e^{-k_1 t} - \left(\frac{3}{4} - \frac{1}{4}k_1 t \right) e^{-2k_1 t} + \frac{1}{12}e^{-3k_1 t} + \frac{1}{12} \right] \\ & + 144\left(\frac{\sigma^2}{2}\right)^2 \frac{1}{k_1^5} \left[\frac{1}{6}e^{-k_1 t} - \frac{1}{4}k_1 t e^{-2k_1 t} - \frac{1}{6}e^{-3k_1 t} + \frac{1}{48}e^{-4k_1 t} - \frac{1}{48} \right] \\ & + \dots \end{aligned}$$

High order moments $E[y^n(t)]$ result in the same way from the series $\underbrace{wg \dots wg}_{n \text{ times}}$.

Output autocorrelation

In this section, we are interested in computing the output autocorrelation function defined by

$$R_{yy}(t_1, t_2) = E[y(t_1)y(t_2)]$$

where $y(t)$ is the functional expansion derived previously. As $y(t)$ is a sum of expressions of the form (3), we only need to compute the partial autocorrelation functions :

$$\begin{aligned} E\left[\left(\int_0^{t_1} \dots \int_0^{t_p} e^{a_0(t_1-\tau_p)} u_{i_1}(\tau_p) \dots u_{i_p}(\tau_1) e^{a_p \tau_1} d\tau_1 \dots d\tau_p\right) \right. \\ \left. \left(\int_0^{t_2} \dots \int_0^{t_q} e^{b_0(t_2-\tau_q)} u_{j_1}(\tau_q) \dots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \dots d\tau_q\right)\right] \quad (6) \end{aligned}$$

Let us assume $t_2 > t_1$; then the second integral may be decomposed as follows :

$$\begin{aligned}
& \int_0^{t_2} \int_0^{\tau_q} \cdots \int_0^{\tau_2} e^{b_o(t_2 - \tau_q)} u_{j_1}(\tau_q) \cdots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \cdots d\tau_q = \\
& \left(\int_0^{t_1} \int_0^{\tau_q} \cdots \int_0^{\tau_2} e^{b_o(t_2 - \tau_q)} u_{j_1}(\tau_q) \cdots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \cdots d\tau_q \right) \left(\int_{t_1}^{t_2} e^{b_o \tau_1} d\tau_1 \right) \\
& + \left(\int_0^{t_1} \int_0^{\tau_{q-1}} \cdots \int_0^{\tau_2} e^{b_1(t_2 - \tau_{q-1})} u_{j_2}(\tau_{q-1}) \cdots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \cdots d\tau_{q-1} \right) \\
& \quad \times \left(\int_{t_1}^{t_2} \int_{t_1}^{\tau_2} e^{b_o(t_2 - \tau_2)} u_{j_1}(\tau_1) e^{b_1 \tau_1} d\tau_1 d\tau_2 \right) \\
& + \dots \\
& + \left(\int_0^{t_1} e^{b_q \tau_1} d\tau_1 \right) \left(\int_{t_1}^{t_2} \int_{t_1}^{\tau_q} \cdots \int_{t_1}^{\tau_2} e^{b_o(t_2 - \tau_q)} u_{j_1}(\tau_q) \cdots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \cdots d\tau_q \right)
\end{aligned}$$

and this leads for the expression (6) to

$$\begin{aligned}
& E \left[\left(\int_0^{t_1} \cdots \int_0^{\tau_2} e^{a_o(t_1 - \tau_p)} u_{i_1}(\tau_p) \cdots u_{i_p}(\tau_1) e^{a_p \tau_1} d\tau_1 \cdots d\tau_p \right) \right. \\
& \left. \left(\int_{t_1}^{t_2} \cdots \int_{t_1}^{\tau_2} e^{b_o(t_2 - \tau_q)} u_{j_1}(\tau_q) \cdots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \cdots d\tau_q \right) \right] E \left[\int_{t_1}^{t_2} e^{b_o \tau_1} d\tau_1 \right] \\
& + \dots \\
& + E \left[\left(\int_0^{t_1} \cdots \int_0^{\tau_2} e^{a_o(t_1 - \tau_p)} u_{i_1}(\tau_p) \cdots u_{i_p}(\tau_1) e^{a_p \tau_1} d\tau_1 \cdots d\tau_p \right) \left(\int_0^{t_1} e^{b_q \tau_1} d\tau_1 \right) \right] \\
& \quad \times E \left[\int_{t_1}^{t_2} \cdots \int_{t_1}^{\tau_2} e^{b_o(t_2 - \tau_q)} u_{j_1}(\tau_q) \cdots u_{j_q}(\tau_1) e^{b_q \tau_1} d\tau_1 \cdots d\tau_q \right]
\end{aligned}$$

where we used statistical independance between integrals over $[0, t_1]$ and $[t_1, t_2]$.

Now recalling that the product of two iterated integrals over the same interval corresponds to the shuffle product of their generating power series, we derive the following symbolic rule :

$$\begin{aligned}
& \left[\left(1 - a_o x_o \right)^{-1} x_{i_1} \cdots x_{i_p} \left(1 - a_p x_o \right)^{-1} \right]_o^{t_1} \times \left[\left(1 - b_o x_o \right)^{-1} x_{j_1} \cdots x_{j_q} \left(1 - b_q x_o \right)^{-1} \right]_o^{t_2} \\
& = \left[\left\langle \left(1 - a_o x_o \right)^{-1} x_{i_1} \cdots x_{i_p} \left(1 - a_p x_o \right)^{-1} \right\rangle_w \left\{ \left(1 - b_o x_o \right)^{-1} x_{j_1} \cdots x_{j_q} \left(1 - b_q x_o \right)^{-1} \right\} \right]_o^{t_1} \times \\
& \quad \left[\left\langle \left(1 - b_o x_o \right)^{-1} \right\rangle \right]_{t_1}^{t_2} \quad (7) \\
& + \left[\left\langle \left(1 - a_o x_o \right)^{-1} x_{i_1} \cdots x_{i_p} \left(1 - a_p x_o \right)^{-1} \right\rangle_w \left\{ \left(1 - b_1 x_o \right)^{-1} x_{j_2} \cdots x_{j_q} \left(1 - b_q x_o \right)^{-1} \right\} \right]_o^{t_1} \times \\
& \quad \left[\left\langle \left(1 - b_o x_o \right)^{-1} x_{j_1} \left(1 - b_1 x_o \right)^{-1} \right\rangle \right]_{t_1}^{t_2} \\
& + \dots \\
& + \left[\left\langle \left(1 - a_o x_o \right)^{-1} x_{i_1} \cdots x_{i_p} \left(1 - a_p x_o \right)^{-1} \right\rangle_w \left(1 - b_{q-1} x_o \right)^{-1} \right]_o^{t_1} \times \\
& \quad \left[\left\langle \left(1 - b_o x_o \right)^{-1} x_{j_1} \cdots x_{j_q} \left(1 - b_q x_o \right)^{-1} \right\rangle \right]_{t_1}^{t_2}
\end{aligned}$$

Example 2 : Let us consider the linear stochastic differential equation

$$\dot{v} = -\alpha v + u(t), \quad v(0) = c$$

where $u(t)$ is a white Gaussian process and c is assumed independent of $u(t)$. The generating power series associated with $v(t)$ is given by

$$\langle g \rangle = \left(1 + \alpha x_o \right)^{-1} x_1 + c \left(1 + \alpha x_o \right)^{-1}.$$

In accordance with (5) and (7), $v(t)$ has mean value

$$\langle g \rangle = E(c) \left(1 + \alpha x_o \right)^{-1}$$

that is

$$E(v(t)) = E(c)e^{-\alpha t}$$

and covariance

$$\begin{aligned}
& \left\langle [g]_o^{t_1} [g]_o^{t_2} \right\rangle = \left\langle \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_2} \right\rangle \\
& + \left\{ \left\langle c \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_2} \right\rangle = 0 \right\} \\
& + \left\{ \left\langle c \left[\left(1 + \alpha x_o \right)^{-1} \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_2} \right\rangle = 0 \right\} \\
& + \left\langle c^2 \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_2} \right\rangle = \\
& = \left[\left\langle \left(1 + \alpha x_o \right)^{-1} x_1 \omega \left(1 + \alpha x_o \right)^{-1} x_1 \right\rangle \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} \right]_{t_1}^{t_2} + \\
& \left\{ \left\langle \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} x_1 \right]_o^{t_2} \right\rangle = 0 \right\} + \left\langle c^2 \left[\left(1 + \alpha x_o \right)^{-1} \right]_o^{t_1} \left[\left(1 + \alpha x_o \right)^{-1} \right]_o^{t_2} \right\rangle \\
& \text{that is } E(v(t_1)v(t_2)) = \frac{\sigma^2}{2\alpha} (1 - e^{-2\alpha t_1}) e^{-\alpha(t_2-t_1)} + E(c^2) e^{-\alpha(t_1+t_2)}
\end{aligned}$$

If we begin with an $N(0, \frac{\sigma^2}{2\alpha})$ -distributed c then $v(t)$ is a stationary Gaussian process (sometimes called a colored noise) such that

$$E(v(t)) = 0 \quad \text{and} \quad E(v(t_1)v(t_2)) = \frac{\sigma^2}{2\alpha} e^{-\alpha|t_2-t_1|}$$

Remark : The techniques presented in this paper still apply for a rationally correlated noise input ; for example considering example 1 in conjunction with example 2 allows to deal with the nonlinear system

$$\dot{y}(t) + k_1 y(t) + k_2 y^2(t) = v(t)$$

where $v(t)$ is a Gaussian colored noise. The following equivalent nonlinear system with a white Gaussian noise input $u(t)$ is then considered :

$$\begin{cases} \dot{v} = -v + u(t) \\ \dot{y} = -k_1 y - k_2 y^2 + v(t). \end{cases}$$

BIBLIOGRAPHIE

- [1] L.ARNOLD, Stochastic differential equations. Wiley, New York, 1974.
- [2] J.F.BARRETT. The use of functionals in the analysis of nonlinear physical systems, J.Electron. & Contr. 15, 1963, pp. 567-615.
- [3] E.BEDROSIAN and S.O.RICE. The output properties of Volterra systems (nonlinear systems with memory) driven by harmonic and Gaussian inputs. Proc.IEEE, 59, 1971, pp. 1688-1708.
- [4] M.FLIESS. Fonctionnelles causales non linéaires et indéterminées non commutatives. Bull.Soc.Math.France, 109, 1981, pp. 3-40.
- [5] M.FLIESS and F.LAMNABHI-LAGARRIGUE. Application of a new functional expansion to the cubic anharmonic oscillator. J.Math.Phys. 23, 1982, pp. 495-502.
- [6] F.LAMNABHI-LAGARRIGUE and M.LAMNABHI. Détermination algébrique des noyaux de Volterra associés à certains systèmes non linéaires. Ricerche di Automatica, 1979, 10, pp. 17-26.
- [7] F.LAMNABHI-LAGARRIGUE and M.LAMNABHI. Algebraic computation of the solution of some nonlinear differential equations. In "Computer algebra" (J.Calmet, éd.), Lect.Notes Comput.Sc. 144, Springer Verlag, Berlin, 1982, pp. 204-211.
- [8] W.J.RUGH. Nonlinear system.Theor. John Hopkins, Baltimore 1981.
- [9] M.SCHETZEN. The Volterra and Wiener theories of nonlinear systems. John Wiley, New York, 1980.
- [10] N.WIENER. Response of a nonlinear device to noise. M.I.T. Radiation Laboratory, Cambridge, Mass. Report 129, 1942.

* *
*

CHARACTERIZATION OF A LINEAR DIFFERENTIAL SYSTEM WITH A REGULAR SINGULARITY

Aziz HILALI
Laboratoire IMAG
B.P. N° 68
38402 SAINT MARTIN D'HERES (FRANCE)

SUMMARY :

Two methods are known for determining if a singularity of a linear differential system is regular, and finding the regularizing transformation.

The cyclic vector method is complete in principle, but very expensive. We present another method applicable in most cases (including all with $n \leq 3$), which finds much simpler transforms.

I. INTRODUCTION

This paper is concerned with systems of ordinary differential equations :

$$Y'(x) = A(x) Y(x) \quad ((1))$$

where $Y(x)$ is a vector with n components and $A(x)$ is $n \times n$ matrix.

The elements of $A(x)$ are assumed to be formal series of the type $\sum_{k=-p}^{+\infty} a_k \cdot x^k$ with a pole at $x=0$.

$$A(x) = \frac{1}{x^p} \sum_{k=0}^{+\infty} A_k \cdot x^k \quad (A_0 \neq 0) \quad ((1'))$$

Where p is an integer and the A_k are constant matrices. The aim of Fuchs's theory is to study the nature of the solution vector $Y(x)$ near the singularity $x=0$.

For $p=1$, the origin is a regular singular point. But this may be true also for $p > 1$. In fact any differential system of type ((1)) with a regular singularity can be transformed by :

$$Y(x) = T(x) Z(x) \quad ((2))$$

into a system :

$$\begin{aligned} Z'(x) &= B(x)Z(x), \\ B &= T^{-1}AT - T^{-1}T' \end{aligned} \quad ((3))$$

Where $B(x)$ has a pole of first order only. Here $T(x)$ is matrix whose elements are formal series having at most a pole at $x=0$ and which satisfies $\det(T(x)) \neq 0$ (i.e. $T \in GL(n, \mathbb{C}((x)))$).

It is well known that the differential system of the type ((1)) has a regular singularity at $x=0$ if and only if every fundamental system of solutions Y is of the form below :

$$Y(x) = P(x) \cdot x^R$$

where $P(x)$ is matrix whose elements are formal series having at most a pole at $x=0$ and R constant matrix.

EXAMPLE 1

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{x^2} \begin{bmatrix} -1 & x^2-x+1 \\ +1 & x+1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

with the transformed variables :

$$\begin{aligned} z_1 &= xy_1 + xy_2 \\ z_2 &= x^2y_2 \end{aligned}$$

We obtain the new system :

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \frac{1}{x} \begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

for which a fundamental system of solutions is

$$Z(x) = x \begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix}$$

Then a fundamental system of solutions for the initial system is :

$$Y(x) = \frac{1}{x^2} \begin{bmatrix} x & -1 \\ 0 & 1 \end{bmatrix} x \begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix}$$

Thus the singularity $x=0$ is regular. though the order of the pole equals 2. One can notice that matrix A_0 in this system is nilpotent.

From this example, it appears that the main problem is to recognize whether the singularity is regular and to calculate the transformation $T(x)$ which reduces this system ((1)) to a system of type ((3)) we have given in (5) two algorithms for the computation of $T(x)$.

FIRST METHOD : cyclic vectors (1,2)

In outline, we write :

$$Y_0 = (\lambda_1(x), \lambda_2(x), \dots, \lambda_n(x))$$

where the λ_i are random polynomials of degree $\leq n-1$. The recurrence $Y_i = xY_{i-1} + Y_{i-1}(xA(x)-(i-1)I)$ defines a matrix S which rows are Y_1, Y_2, \dots, Y_{n-1} . If S is non-singular, we say that Y_0 is a **cyclic vector**. In this case, which happens "almost always", we write $T = S^{-1}$, and the transformation $Y(x) = T(x)Z(x)$ gives ((3)) This gives the following characterization (6) : " $x=0$ is a regular singularity of ((1)) if and only if the matrix B has pole of order exactly one there". Unfortunately as we see in the following example, T can be extremely complicated :

MATRICE DU SYSTEME DIFFERENTIEL :

```
-----  
a(1,1) := 1  
a(1,2) := (- 2*x + 1)/x3  
a(1,3) := 4/x  
a(2,1) := x*(x + 1)2  
a(2,2) := (- x + 2)/x  
a(2,3) := 4/x  
a(3,1) := 3*x2  
a(3,2) := 0  
a(3,3) := (x + 2)/x2
```

COEFFICIENTS DU VECTEUR CYCLIQUE Z0

```
-----  
z0(1) = 1  
z0(2) = x  
z0(3) = 1
```

IMPRESSION DE LA TRANSFORMATION T

$$T(1,1) = (40x^{11} - 79x^{10} - 74x^9 - 59x^8 + 209x^7 - 69x^6 + 224x^5 + 70x^4 -$$

$$28x^3 + 28x^2 + 169x - 4)/(40x^{15} + 9x^{14} + 35x^{13} + 28x^{12} + 31x^{11}$$

$$x^{10} - 114x^9 - 68x^8 - 153x^7 - 41x^6 - 85x^5 + 39x^4 + 72x^3$$

$$x^2 - 26x^1 - 28x^0 + 169x - 4)$$

$$T(1,2) = (x * (40x^{12} + 59x^{11} + 18x^{10} + 59x^9 + 35x^8 - 39x^7 - 89x^6 + 48x^5 + 1))$$

$$(40x^{15} + 9x^{14} + 35x^{13} + 28x^{12} + 31x^{11} - 114x^9 - 68x^8 -$$

$$153x^7 - 41x^6 - 85x^5 + 39x^4 + 72x^3 - 26x^2 - 28x^1 + 169x$$

- 4)

$$T(1,3) = (x * (-x^2 - 5x^1 - 3x^0 - 28x + 1))/(40x^{13} + 9x^{12} + 35x^{11} + 28$$

$$x^{10} + 31x^9 + 114x^8 - 68x^7 - 153x^6 - 41x^5 - 85x^4 + 39x^3$$

$$x^2 + 72x^1 - 26x^0 - 28x + 169x - 4)$$

$$T(2,1) = (x * (40x^{13} + 9x^{12} + 33x^{11} + 26x^{10} + 209x^9 - 114x^8 - 1289x^7 -$$

$$114x^6 + 129x^5 - 89x^4 + 4))/(40x^{15} + 9x^{14} + 35x^{13} + 28x^{12}$$

$$+ 31x^{11} - 114x^9 - 68x^8 - 153x^7 - 41x^6 - 85x^5 + 39x^4$$

$$+ 72x^3 - 26x^2 - 28x^1 + 169x - 4)$$

$$T(2,2) = (x * (40x^{12} + 16x^{11} + 6x^{10} - 59x^9 + 23x^8 - 64x^7 + 89x^6 + 89x^5 +$$

$$x^4 + 9x^3 + 35x^2 + 28x^1 + 31x^0 - 114x^9 - 68x^8 - 153x^7 - 41x^6 - 85x^5 -$$

$$89x^4 - 41x^3 - 85x^2 + 39x^1 + 72x^0 - 26x^9 + 28x^8 + 169x - 4$$

)

$$T(2,3) = (x * (-x^2 - x^1 - x^0 - 3x^9 + x^8 + 3x^7 + 3x^6 + 13x^5 + 14x^4 + 13x^3 +$$

$$x^2 + 28x^1 + 31x^0 - 114x^9 - 68x^8 - 153x^7 - 41x^6 - 85x^5 + 39x^4 + 72x^3 + 26x^2 - 28x^1 + 169x - 4)$$

$$T(3,1) = (x * (20x^{13} + 29x^{12} + 79x^{11} + 116x^{10} + 67x^9 + 199x^8 - 210x^7 - 289x^6 + 239x^5 + 29x^4 + 23))/$$

$$(40x^{15} + 9x^{14} + 35x^{13} + 28x^{12} + 31x^{11} - 114x^9 - 68x^8 - 153x^7 - 41x^6 - 85x^5 + 39x^4 + 72x^3 + 26x^2 - 28x^1 + 169x - 4)$$

$$x^2 + 26x^1 - 28x^0 - 28x + 169x - 4)$$

$$T(3,2) = (x * (-10x^{14} - 21x^{13} - 24x^{12} - 12x^{11} + 9x^{10} - 11)/$$

$$(40x^{15} + 9x^{14} + 35x^{13} + 28x^{12} + 31x^{11} - 114x^9 - 68x^8 - 153x^7 - 41$$

$$x^6 - 89x^5 + 39x^4 + 72x^3 - 26x^2 - 28x^1 + 169x - 4)$$

$$T(3,3) = (x * (x^2 + x^1 + 3x^0 - 28x^9 + 29x^8 + 29x^7 - 29x^6 + 29x^5 + 13x^4 + 14x^3 +$$

$$(40x^{15} + 9x^{14} + 35x^{13} + 28x^{12} + 31x^{11} - 114x^9 - 68x^8 - 153x^7 - 41x^6 - 85$$

$$x^5 + 39x^4 + 72x^3 - 26x^2 - 28x^1 + 169x - 4)$$

We see that this method presents several practical difficulties because of :

- i) The choice of Y_0 , which affects the complexity of T ;
- ii) In general the coefficients of T , as found by this method, are for larger than necessary, since :

$$\tilde{T} = \begin{bmatrix} 0 & 1/x & 0 \\ 0 & 0 & x \\ 1 & 0 & 0 \end{bmatrix}$$

is also a regularizing transformation.

NEW METHOD :

If our system appears to have an irregular singularity, then $p > 1$. Write $r = \text{rank}(A_0)$ Moser (7) defines :

- $m(A) = p-1 + r/n$
- $\mu(A) = \inf \{m(T^{-1} \cdot A \cdot T - T^{-1} \cdot T')\}$

where the \inf is taken over all T in $GL(n, \mathbb{C}((x)))$ we say that A is **reducible** if $m(A) > \mu(A)$.

THEOREM : (Moser (7))

If $m(A) > 1$ the matrix $A(x)$ is reducible if and only if the polynomial

$$\theta(\lambda) = x^r \det(\lambda I + x^{p-1} A(x))|_{x=0}$$

vanishes identically in λ .

We remark that, in fact, this condition depends only on A_0 and A_1 , since :

$$\theta(\lambda) = x^r \det(\lambda I + \frac{A_0}{x} + A_1)|_{x=0}$$

we can always suppose that A_0 is nilpotent (see Wasow (8) for details). In this case, its Jordan form J can be written as

$$J = \text{diag}(H_1, H_2, \dots, H_m) \quad ((4))$$

where H_1 is a $n_1 \times n_1$ zero matrix and H_2, \dots, H_m are $n_i \times n_i$

shifting matrices. If P is the constant matrix which transforms A_0 into J . $J = P^{-1}A_0P$ we have :

$$\theta(\lambda) = x^r \det(\lambda I + \frac{J}{x} + G)|_{x=0}$$

where $G = P^{-1} \cdot A_1 \cdot P$

We can decompose G according to the decomposition of J :

$$G = (G_{ij}) \quad 1 \leq i, j \leq n$$

G_{ij} is an $n_i \times n_j$ block.

The special form of J and G allow us to show (5) that $\theta(\lambda) = \det(\hat{G}(\lambda))$, where

n_1	$m-1$
$\lambda I_{n_1} + G_{11}$	$G_{12}(1,1) \ G_{13}(1,1) \ \dots \ G_{1m}(1,1)$ $G_{12}(2,1) \ G_{13}(2,1) \ \dots \ G_{1m}(2,1)$
$G_{21}(n_2,1) \ \dots \ G_{21}(n_2, n_1)$ $G_{31}(n_2,1) \ \dots \ G_{31}(n_3, n_1)$	$G_{22}(n_2,1) \ G_{23}(n_2,1) \ \dots \ G_{2m}(n_2,1)$ $G_{32}(n_3,1) \ G_{33}(n_3,1) \ \dots \ G_{3m}(n_3,1)$
.
$G_{m1}(n_m,1) \ \dots \ G_{m1}(n_m, n_1)$	$G_{m2}(n_m,1) \ G_{m3}(n_m,1) \ \dots \ G_{mm}(n_m,1)$

Hence we consider whether $\hat{G}(\lambda)$ is singular (identically in λ) The following conditions are obviously sufficient for this

- I That the last $m-1$ columns of \hat{G} are linearly dependant or
- II That the last $m-1$ rows of \hat{G} are linearly dependant.

In general these conditions are not necessary but an important case is $n < 3$, when they are both necessary and sufficient.

As the following theorem shows, these special cases are linked to simple forms of the transformation matrix.

THEOREM (5) :

If $A(x)$ is reducible then the reducibility can be given with the transformation

$$T(x) = P \cdot S(x)$$

where

- P is a constant matrix which transforms A_0 into a Jordan form that is supposed of the form ((4))
 - $S(x) = \text{diag}(S_1, S_2, \dots, S_m)$
- with
- $S_1 = I_{n_1}$
 - S_i ($i=2, \dots, m$) are $n_i \times n_i$ diagonal matrices and defined by :
 - (i) $S_i(x) = \text{diag}(1, x, \dots, x)$ if the Condition I is satisfied,
 - ii) $S_i(x) = \text{diag}(1/x, 1/x, \dots, 1/x, 1)$ if the Condition II is satisfied.

Then we obtain an algorithm which can carry out the Computation of successive transformations

$$T_0, T_1, \dots, T_s$$

according to conditions I and II.

In the case $n < 3$, it permits us to transform the system ((1)) into a system of the following form :

$$z'(x) = \frac{1}{x^{p^*}} \left(\sum_{k=0}^{+\infty} B_k x^k \right) z(x)$$

with the transformation $T = T_0 \times T_1 \times \dots \times T_s$ where p^* is the minimum value of p when we apply in ((1)) all the transformation of type ((2)). The regular singularity is characterized by $p^* = 1$. The algorithm stops after $(n-1)(p-1)$ steps.

We shall compare these methods on example 2, to which we have already applied the method of cyclic vector :

$$Y'(x) = \frac{1}{x^3} \begin{bmatrix} x^3 & 1-2x & 4x^2 \\ x^5+x^6 & 2x^2-x^3 & 4x^2 \\ 3x^5 & 0 & 2x^2+x^4 \end{bmatrix} Y(x) \quad ((6))$$

We find that :

$$T_0(x) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & x \\ 1 & 0 & 0 \end{bmatrix}, \quad T_1(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/x & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and the transformation :

$$T(x) = T_0(x) \cdot T_1(x) = \begin{bmatrix} 0 & 1/x & 0 \\ 0 & 0 & x \\ 1 & 0 & 0 \end{bmatrix}$$

transform the system ((6)) into :

$$Z'(x) = \frac{1}{x} \begin{bmatrix} 2+x^2 & 3x^2 & 0 \\ 4x & 1+x & 1-2x \\ x & x+x^2 & 1-x \end{bmatrix} Z(x)$$

CONCLUSION

The problem of finding solutions to systems of linear differential equations with polynomial (or series) coefficients is a difficult one. The difficulties are of two types :

algebraic the study of the singularities and associated transformation,

analytic questions of convergence etc. of the formal solutions obtained.

In this paper we have attacked an algebraic problem, the search for regularizing transformations, and have found an algorithmic solution for nearly all cases (except $\det(G(\lambda)) \equiv 0$ without I or II being satisfied). This method is not trivial, relying on Jordan

forms, and hence on algebraic extensions, but it is the only one known which leads to a classification of singularities of systems of differential equations with formal series coefficients.

REFERENCES

- (1) **G.D. BIRKHOFF**
 "Formal theory of irregular linear difference equations".
Acta Math. Vol. 54 (1930), p. 207-209.
- (2) **F.T. COPE**
 "Formal solutions of irregular linear differential equations". *Am. J. Math.* Vol. 58 (1936), p. 130-140.
- (3) **A. HILALI**
 "Réduction d'un système différentiel linéaire homogène à une équation différentielle scalaire linéaire homogène". R.R. N° 240, Labo. IMAG, USMG, Grenoble (1981).
- (4) **A. HILALI**
 "Réductibilité d'un système différentiel linéaire homogène".
 (To appear, *Num. Math.*).
- (5) **A. HILALI**
 "Contribution à l'étude des points singuliers des systèmes différentiels linéaires". Thèse de Troisième Cycle, Labo. IMAG, USMG, Grenoble (1982).
- (6) **B. MALGRANGE**
 "Sur les points singuliers des équations différentielles".
L'ens. Math., t.xx, 1-2 (1974), p. 147-176.
- (7) **J. MOSER**
 "The order of singularity in Fuchs's theory". *Math. Zeitshrift*, 72 (1960), p. 379-398.
- (8) **W. WASON**
 "Asymptotic expansions for ordinary differential equations".
 Wileley (1973).

THE BATH CONCURRENT LISP MACHINE*

Jed Marti+

John Fitch

School of Mathematics

University of Bath

England

*Funded in part by the Science and Engineering Research Council.

+Present address: RAND Corporation, Santa Monica, California, USA.

Abstract

The Bath Concurrent LISP machine is a general purpose multiprocessor designed for research in symbolic computation and expert systems. The system is particularly suited to tree structured computations with dynamic data requirements and complex interactions. It accepts conventional LISP programs, performs a data flow analysis on them, and creates a compiled module in which available concurrency is exploited. This paper presents an overview of the system hardware and software and the directions in which research is proceeding.

Introduction

The Bath Multiprocessor was built by the Electrical Engineering and Mathematics Schools at the University of Bath starting in the spring of 1982. It consists of six Motorola 68000 microprocessors connected in a topology to facilitate dynamic task scheduling and minimize communication overhead. Its intended tasks are processing of quadtrees for a computer graphics system, and general highspeed computations in symbolic computation and expert systems. This report overviews the progress we have made towards building an operational concurrent processing environment.

Hardware

The initial hardware configuration is six individual Motorola 68000 computer systems, each with 256K of 500 nanosecond memory, up to 64K bytes of ROM, and a 16 bit data path. For machines with no peripherals there is room for up to 1.5 megabytes of high speed store, soon to be expanded to 3 megabytes for a total possible configuration of

over 16 megabytes. One machine has a DMA floppy disk controller with access to 2.4 megabytes of transportable storage. Through high speed parallel and serial interfaces, the other 5 machines are connected to a 26 megabyte winchester disk operated by an intelligent disk controller. The department also possesses a standalone 68000 system with 512k bytes of RAM, 15 megabytes of disk storage, network connections, floppy disks and so on. By connection through the intelligent disk server we were able to bring up the first versions of the operating system and LISP. The intelligent disk controller also runs LISP and served as a test bed for much of the language processor and data flow analysis software. The machine is also connected to a bit mapped color graphics display [4], a 68000 based quad tree display [13], and soon to the departmental token ring network [14]. The current configuration is the four processor network shown in Figure 1.

Interprocessor communication is conducted through memory windows; 2K byte blocks of dual ported memory shared by adjacent processors. There are three such windows per processor. The effective throughput under program control is somewhat under 1 megabyte per second per processor for a total machine capacity of 6 megabytes per second.

The numbering scheme for processors places the storage for each set of three windows in even numbered processors and an empty interface in odd. The four processor configuration of Figure 1 uses only two memory windows per processor while the six processor configuration in Figure 2 utilizes all three [16].

The properties of this network have been investigated by simulation [1]. The results indicate is that the valency of a node (the number of outward edges) influences the scheduling and duration of a computation. Too low a valence inhibits potential concurrency and to high causes more overhead in task scheduling and storage. The simulation indicates that a valence of 3 is optimal for many tasks.

Supporting System

A number of programs support the operation of the concurrent machine. Many of these were developed and debugged on the standalone processor before the multiprocessor was completed. Each machine runs a complete copy of the TRIPoS operating system [11] which occupies about 90k bytes of available storage. This supports a hierarchical filing system, a multitasking supervisor, a resident disassembler, and debugging program. These have proven invaluable during the initial implementation effort. As more of the system becomes operational, subsidiary processors will no longer need large parts of the operating system to the point where well over 200K bytes of storage should be available for user programs.

TRIPOS maintains the BCPL environment in which Cambridge LISP is implemented [3]. The original version of this system was implemented on an IBM 370/165 and with reimplementations of a few small parts in assembler, it has been ported to the TRIPOS world with relative ease. We currently operate two versions: a large version for software development and a very small system for the subsidiary processors.

A small set of assembly routines provides two levels of access to the memory windows. The simplest form treats the window as a byte stream and is used for transferring relocatable programs between machines. The second form treats the window as a block of memory and does very fast transfers to LISP vectors by taking advantage of the 68000's operation prefetch. This form is used for transferring structured information (control blocks and the like) between processors.

Above this lowest level are a number of software abstraction levels to provide high level interprocessor communication primitives. For example:

```
(WREAD n)      - Read an S-expression from window n.  
(WPRINT x n)  - Send the S-expression x to window n.  
(WLOAD n)     - Load a precompiled file from window n.  
(WSEND file n) - Send a file to window n.  
(BOOTALL file) - Send file to all processors in network.
```

Precompiled program modules are stored in an absolutely relocatable form. In a multiprocessor environment this takes on added significance as the modules must exist in different places in different machines. Initially we have adopted the practice of having every processor in the network have a complete copy of the program to be executed. This will be changed to a load on demand strategy at a later date.

Above the communication primitives lies the LISP task monitor, a simple coroutine handler. The machine which has the operator's terminal serves as the master processor. It sends tasks to other processors to perform which may in turn send tasks back to the master if it is idle. The initial implementation has this task scheduling done on an imperative basis by the spawning process. We plan to experiment with the opposite scheme in which idle processors actively seek tasks to perform.

The task monitor is implemented in Cambridge LISP. The concurrent LISP system is also implemented in this "systems LISP" and provides the concurrent evaluation environment seen by the user. It is designed in such a way that multiple users may be operating on the same LISP system without interfering with each other. It provides separate data areas by prefixing a task code to variables in the symbol table and by not

permitting redefinition of the basic functions of systems LISP. At this level programs are implemented with critical areas, sentinel processes, monitors, and the like.

Input and Output in the concurrent environment is treated as a side effect. Synchronization is accomplished by permitting any processor to modify the I/O streams by accessing the buffers, but only the base processor may actually perform the I/O.

Interprocedural Data Flow Analysis

Starting at the top level of the system, we have expended considerable effort in developing a compiler which is capable of converting a sequential program into an equivalent concurrent one which runs faster. The motivation for this approach is our belief that implementing programs at the monitor, critical area, or sentinel level is a complex and error prone task, one that is likely to distract attention from the already complex task of programming symbolic systems. These constructs can be avoided if one is willing to operate in a purely functional environment but it is our belief that this too is an unnecessary restriction. It is the business of an intelligent compiler, not the programmer, to produce a program with proper synchronization of access to data. The Bath concurrent compiler accepts conventional LISP programs and compiles them into a program capable of executing in a multiprocessing environment of any configuration with synchronization of data access controlled to make the user see only one machine.

Only in very special circumstances is the programmer justified in implementing communicating concurrent processes himself. In these cases it is permissible to use the task scheduling and communication primitives provided by the system. The data flow analyzer will ignore exclusive data access conflicts in these circumstances.

To create concurrently executable code, the compiler must be able to determine mutual exclusion between functional forms. The interprocedural data flow analysis program [10] scans an entire LISP module and produces a closure for each function in it. Mutual exclusion can be determined by examination of this information. The data flow analysis occurs in two steps:

1. The first step computes an extended path expression [12] which contains closure information and function names whose closures have not yet been computed. This information is built into a regular expression which summarizes the order of sequential execution.
2. For each entry point in the module, a depth first search fills in all closure information for each function name occurring in an extended path expression. The

resulting expression is a simple closure. Functions which can not be found in the module are either part of the base system (the closures of these functions are known) or are part of another module. A special IMPORTS declaration locates these closures in a file rather than recomputing them. If no entry points are specified, each function in the module is treated as an entry.

At the conclusion of the data flow analysis phase, the closure of each function is known. This represents a set of accesses to the LISP environment which includes the global value cell of an identifier and its property list. The set of accesses is subdivided into three sets according to use:

- r: a set of "read only" locations.
- w: a set of "write then read" locations whose values are modified when the function is first entered. The previous value of the location is ignored.
- rw: a set of "read then write" locations which are read before they are changed.

If the set of locations accessed cannot be properly determined, a special "universe" closure is created. The data flow analysis algorithm is sophisticated enough to recognize large classes of locations rather than simple sets or single locations. For example, it understands the set of locations which represent the value associated with a single indicator on the property list of all identifiers. The program also analyzes ill-structured PROG forms which involves a considerable refinement over the previous effort [8]. In the appendix is the data flow analysis for a small program for the manipulation of multivariate polynomials copied from the NLARGE system [2].

Complexity Analysis

During the compilation of the data flow information, a measure of program complexity is computed [15]. This measure is a number which approximates the running time of the function given an 'average' set of data. The information is used by the compiler to decide whether or not a particular function is worth evaluating in another processor or not. Each of the primitive LISP functions was assigned a number based on its execution time. Composite functions have their complexity computed by summing up the measures of their individual pieces with products being formed for loops and recursive calls. An extended complexity measure is a regular expression built from the path expression computed by the data flow analysis procedure. It contains references to as yet uncomputed measures. The second pass of the data flow analysis procedure completes the computation for all functions. The complexity measures of the algebra program are given in the appendix. Any procedure which has runtime complexity above a certain

threshold is given a *** value, meaning "very large".

The Concurrent Compiler

The compiler is an amalgamation of the Portable LISP Compiler and one produced as part of a previous effort [5, 8]. It uses the data flow analysis and complexity information to create code with FORK and JOIN operations in the appropriate places. A discussion of the methods appears in [8]. The code produced is incompatible with the base system LISP for two reasons:

1. Local variables and return addresses are stored in a "Cactus Stack" [7]. Consequently more than one stack is present and holes in the stack are removed by the base system garbage collector. Stack frames are stored as system LISP vectors.
2. The complete environment of a task must be preserved and restored. Multiple environments may be present with each having its own global variables.

These are supported in systems LISP by vector data types. A combination of binary vectors and LISP vectors provides stack frames and task control blocks. The compacting garbage collector in the base system relocates these as well as dealing with removal of redundant frames. Efforts are being made to add a coroutine mechanism to the base LISP system enabling tasks to spawn subprocesses in all neighbors, even if they are already active. Viewed in this sense the network can be thought of as a tertiary tree with as many nodes as there are possible co-routines in the system.

Processes will communicate by passing messages [6]. Our system differs from other systems of this type by moving access exclusivity from run time to compile time, and by hiding synchronization from the programmer.

Project Status

The concurrent hardware first became operational in the beginning of November 1982 and as of the middle of December, the 4 processor network has been running 4 copies of system LISP and transferring compiled LISP files. The polynomial program in the appendix in an interpreted form with a very simple task monitor has the following running times:

Task: $(x+1)^i$

Processors:	1	2	3	4
i = 1	660	920	640	660
i = 5	14160	14200	12080	12140
i = 10	64480	47620	44420	47000
i = 15	163700	109540	89980	94660
i = 20	328340	204460	163380	157500

All times are in milliseconds. These results are displayed graphically in Figure 3.

We are actively pursuing the development of the concurrent compiler and the concurrent task monitor. We have completed the data flow analysis procedure though it must be made faster. We have completed most of the systems LISP development and the concurrent LISP environment. The initial concurrent bootstrap process and memory window software has been completed and tested.

Conclusions

Speedup of functional programs is important to symbolic computation for two reasons:

1. Larger problems can be handled if smaller computations can be completed in less time.
2. Space can be traded for speed, that is, recomputation can be used to save the space taken by intermediate results.

The Bath Concurrent LISP Machine provides a powerful computational device at a fraction of the cost of equivalent mainframes.

List of References

1. Bowyer, A., Willis, P., Woodwark, J., 'A multiprocessor architecture for solving spatial problems'. *The Computer Journal*, 24, No. 4, 1981, pp353-357.
2. Fitch, J. P., Marti, J., 'NLARGEing a Z80 microprocessor'. *Proceedings of EUROCAM 1982* in *Lecture Notes in Computer Science* 144, 'Computer Algebra'. J. Calmet editor. Springer-Verlag, pp249-255, 1982.
3. Fitch, J. P., Norman, A. C., 'Implementing LISP in a high-level language'. *Software-Practice and Experience*, 7, pp713-725, 1977.
4. Hanson, J. B., Willis, P. J., 'A graphics art computer system'. *Proceedings of Electronics Displays*, 1982.
5. Hearn, A. C., Griss, M. L., 'The portable LISP compiler'. *Software-Practice and Experience*, 11, pp541-605, 1981.
6. Hewitt, C., 'The Apiary network architecture for knowledgeable systems'. *Proceedings of the 1980 LISP Conference*, pp107-117.
7. Lindstrom, G., Soffa, M. L., 'Referencing and retention in block-structured coroutines'. *TOPLAS*, 3, No. 3, July 1981.
8. Marti, J., 'A concurrent processing system for LISP'. Thesis, Department of Computer Science, University of Utah, 1980.
9. Marti, J., 'Compilation techniques for a control-flow concurrent LISP system'. *Proceedings of the 1980 LISP Conference*, pp203-207.
10. Marti, J., 'An interprocedural data flow analysis for LISP', in preparation.
11. Richards, M., Aylward, A., Bond, P., Evans, R., Knight, B., 'TRIPOS - A Portable Operating System for Mini-computers'. *Software Practice and Experience*, 9, pp513-526, 1979.
12. Tarjan, R. E., 'Fast algorithms for solving path expressions'. *Journal of the ACM*, 28, No. 3, pp594-614, July 1981.
13. Willis, P., Milford, D., Woodwark, J., 'Exploiting area coherence in raster scan displays'. *Proc. Elec. Displays 1981*, 3, pp34-46, 1981.
14. Willis, P. J., 'An implementation of a token ring'. *Proceedings of the 6th International Conference on Computer Communications*, pp149-153, 1982.
15. Woodward, M., et al, 'A measure of control flow complexity in program text'. *IEEE Transactions on Software Engineering*, SE-5-1, pp45-50, January 1979.
16. Bakti, Z., in preparation.

APPENDIX

The following is a set of simple multivariate polynomial manipulation routines in REDUCE syntax. The BLOCK and ENTRY statements are for the global data flow analysis procedure and the compiler.

```
BLOCK();
 $\$$ 
 $\$$  Structures associated with polynomials.
MACRO PROCEDURE pcreate x; LIST('NCONS, CADR x);
MACRO PROCEDURE term x; LIST('CAR, CADR x);
MACRO PROCEDURE nextterm x; LIST('CDR, CADR x);
MACRO PROCEDURE coefficient x; LIST('CDR, CADR x);
MACRO PROCEDURE variable x; LIST('CAAR, CADR x);
MACRO PROCEDURE exponent x; LIST('CDAR, CADR x);
MACRO PROCEDURE factor x; LIST('CAR, CADR x);
MACRO PROCEDURE numeric x; LIST('NUMBERP, LIST('CAR, CADR x));

ENTRY 'P!+;
EXPR PROCEDURE P!*(A, B);
 $\$$  Multiply the two polynomials A and B.
Left!-Associate(P!+, LAMBDA (x) PTTIMES(x, B), A);

EXPR PROCEDURE PTTIMES(TT, A);
 $\$$  TT is a single polynomial term to multiply times the polynomial A.
Left!-Associate(P!+, LAMBDA (x) TTTIMES(x, TT), A);

EXPR PROCEDURE TTTIMES(TA, TB);
 $\$$  TA and TB are two polynomial terms. Return their product as a
 $\$$  polynomial.
IF NUMBERP TA THEN
    IF NUMBERP TB THEN pcreate(TA * TB)
        ELSE PNTIMES(TB, TA)
    ELSE IF NUMBERP TB THEN PNTIMES(TA, TB)
        ELSE PPTIMES(TA, TB);

EXPR PROCEDURE PNTIMES(P, N);
 $\$$  Multiply the polynomial term P by the number N.
IF N = 0 THEN pcreate 0
ELSE pcreate(factor P . P!*(coefficient P, pcreate N));

EXPR PROCEDURE PPTIMES(TA, TB);
 $\$$  Multiply two polynomial terms by each other. If they have the same
 $\$$  variable, add their coefficients together. If not, scan down one
 $\$$  coefficient or the other to preserve the canonical ordering.
BEGIN SCALAR ORD;
    ORD := PORDERP(variable TA, variable TB);
    RETURN
    IF ORD = 0 THEN
        NCONS((variable TA .
            (exponent TA + exponent TB)) .
            P!*(coefficient TA, coefficient TB))
    ELSE IF ORD = 1 THEN
        NCONS(factor TA . P!*(NCONS TB, coefficient TA))
    ELSE
        NCONS(factor TB . P!*(NCONS TA, coefficient TB))
END;
```

```

ENTRY 'P!+;

EXPR PROCEDURE P!+(A, B);
% Form the sum of the two polynomials A and B.
IF NULL A THEN B ELSE IF NULL B THEN A
ELSE IF numeric A AND numeric B THEN pcreate(term A + term B)
ELSE IF numeric A THEN term B . P!+(A, nextterm B)
ELSE IF numeric B THEN term A . P!+(B, nextterm A)
ELSE BEGIN SCALAR ORD;
    ORD := PORDERP(variable term A, variable term B);
    IF ORD = 1 THEN RETURN term A . P!+(nextterm A, B);
    IF ORD = -1 THEN RETURN term B . P!+(nextterm B, A);
    IF exponent term A = exponent term B THEN RETURN
        BEGIN SCALAR AA, BB;
            AA := P!+(coefficient term A, coefficient term B);
            IF AA = 0 THEN RETURN P!+(nextterm A, nextterm B);
            AA := NCONS(factor term A . AA);
            BB := P!+(nextterm A, nextterm B);
            RETURN P!+(AA, BB)
        END;
    IF exponent term A > exponent term B THEN RETURN
        term A . P!+(nextterm A, B);
    RETURN term B . P!+(nextterm B, A)
END;

```

```

ENTRY 'PPRINT;

EXPR PROCEDURE PPRINT X;
% PPRINT -
%     Print a polynomial in infix notation with implied multiplication.
<< WHILE nextterm X DO
    << PTPRINT term X;
    PRIN2 " + ";
    X := nextterm X >>;
    PTPRINT term X >>;

```

```

EXPR PROCEDURE PTPRINT X;
% Display a polynomial term.
IF NUMBERP X THEN PRIN2 X
ELSE <<
    IF numeric coefficient X THEN PRIN2 coefficient X
    ELSE << PRIN2 "("; PPRINT coefficient X; PRIN2 ")" >>;,
    PRIN2 variable X;
    IF exponent X > 1 THEN
        << PRIN2 "^";
        PRIN2 exponent X >> >>;

```

```

EXPR PROCEDURE PORDERP(A, B);
% Return:
%     0 - A EQ B
%     1 - A is ordered before B.
%     -1 - A is ordered after B.
IF A EQ B THEN 0
ELSE IF ORDERP(A, B) THEN 1 ELSE -1;
BLOCKEND;

```

The data flow analysis information follows. Each function name is immediately followed by its complexity measure in square brackets (note: *** means very complex) and its read, write, read/write sets follow. The set Ro means that the closure is empty. The rather large sets are accesses to the I/O system global variables.

```
-- Analysis:
PORDERP[2] :: Ro

PTPRINT[53] :: 
[read:{!-fout:*, !-f1:!-maxvert, !-f2:!-maxvert,
  !-f1:!-maxhorz, !-f2:!-maxhorz}
 write:{}
 read/write:{!-f1:!-horzposn, !-f2:!-horzposn,
  !-f1:!-vertposn, !-f2:!-vertposn}]

PPRINT[165] :: 
[read:{!-fout:*, !-f1:!-maxvert, !-f2:!-maxvert,
  !-f1:!-maxhorz, !-f2:!-maxhorz}
 write:{}
 read/write:{!-f1:!-horzposn, !-f2:!-horzposn,
  !-f1:!-vertposn, !-f2:!-vertposn}]

P!+[1688] :: Ro

PPTIMES[***] :: Ro

PNTIMES[3396] :: Ro

TTTIMES[***] :: Ro

PTTIMES[***] :: Ro

P!*[***] :: Ro
```

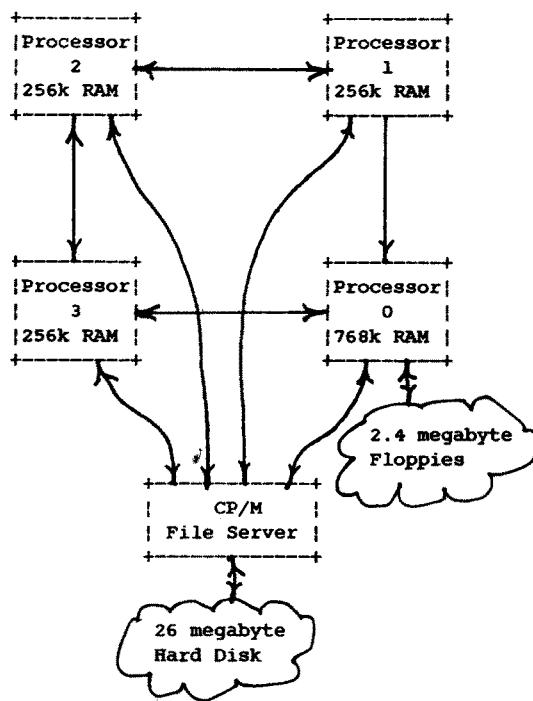


Figure 1. Working configuration, valence = 2.

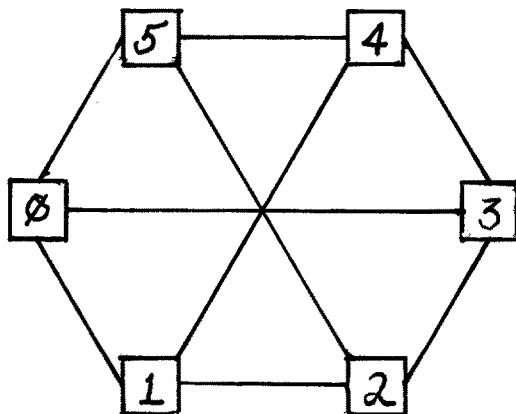


Figure 2. 6 processor configuration, valence = 3.

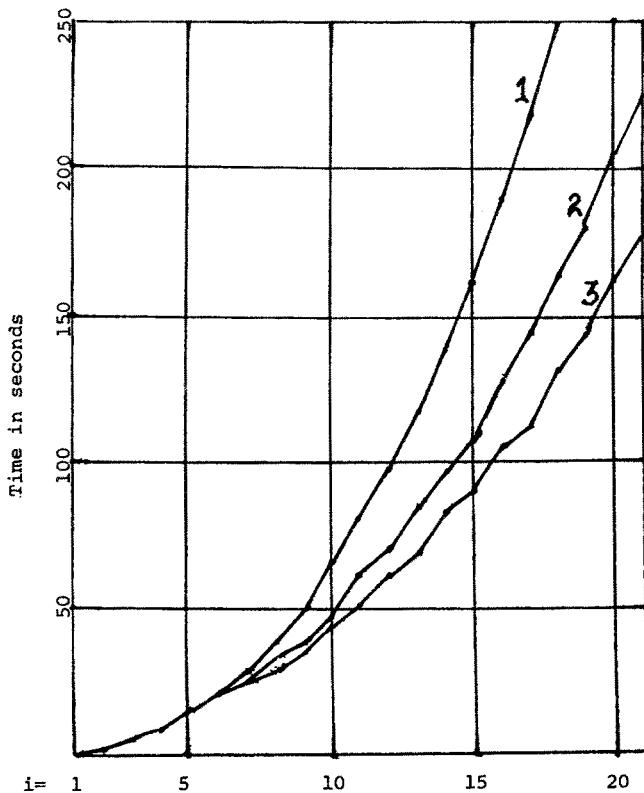


Figure 3. $(x + 1)^i$ for 1, 2, and 3 processors.

The Ecology of LISP
or
The case for the preservation of the environment

Julian A. Padgett.

School of Mathematics.

University of Bath.

Bath, England.

Abstract

A new binding model for dynamically scoped functional languages (such as LISP) is presented. The scheme attempts to combine the fast variable access of shallow binding with the functional advantages of deep binding without placing a high cost on context-switching. Tags are used to label the nodes of the environment tree, and this is used to determine whether a given node is an ancestor of the current node, and thus which value of a variable is valid in the current environment. The efficiency of the new method is compared with the classical models (shallow and deep), and then viewed in the light of more recent developments such as the cache-cell system of MDL and Baker's rerooting scheme.

Details of the current implementation are described briefly, followed by a discussion of the implications of the model for garbage collection - in particular the need to regenerate tags on the environment tree - and how the scheme will affect the compilation process.

Introduction

The advantages and disadvantages of deep and shallow binding are well known - a trade off between speed and functionality. In fact it is worth noting that they appear to be duals: this is not so strange however on consideration: if the shallow binder is based on a perfect hashing function, with one slot for each name, then the deep binder has a hashing function with very poor distribution; that is it always hashes to the same slot. In between these two extremes lies a continuum of binding schemes which trade speed for functionality in varying degrees. It is important to understand the relevance of the above remarks, since the scheme described below does not lie in this interval, and hence is in no way a derivative of previous methods.

Aims and Requirements

An ideal binding model would offer both name interrogation and context switching in small constant time. Shallow binding permits fast name-value lookup but poor environment control. Deep binding on the other hand cannot put an upper bound on the time required to find the current value of a variable, but is good at changing between environments. The scheme described later tries to reconcile these two needs.

Dynamic scoping imposes peculiar constraints on the representation of a function's environment. Since scope cannot be determined at declaration, some sort of relationship must be established between name, value and environment at run-time. In addition because environments are heavily nested, there must be some ordering function on them.

Deep binding satisfies these conditions by collecting the name-value pairs into objects which are generally referred to as frames. These frames are organised into a tree structure which reflects the current state of the computation. In this way a name-value combination is explicitly associated with the new environment, by being placed in the frame which expresses the extension of the previous environment. From a given frame (say the current frame), the links between the frames mark the path to the root of the tree from the current frame. The name-value pairs along this route constitute the current environment, and the ordering of the frames (or environment extensions) ensures that the most recent binding of a name is found first. Hence the interrogation algorithm for deep binding: to search from the current frame towards the root until the desired name is found - in which case the paired value is returned - or until the root is encountered. In this latter case, the name may yet be found amongst the globals, but if not the name is said to be unbound.

Shallow binding creates an implicit link between value and environment, by ensuring that the correct value for the current environment is always in the same location in the global symbol table. Here symbol table is taken to mean that structure used to maintain all the relevant information about an identifier. It is also referred to as the oblist. Shallow binding still employs an hierarchy of frames organised into a tree, but the information they contain now mirrors the changes that have taken place in the symbol table as each environment is extended. When a new name is bound, a new entry is made in the oblist. When a name is rebound however, the value currently in the value cell of the identifier is paired with the name and inserted into the new frame being built. The new value then replaces the old value in the symbol table.

The tag binding scheme, which is the subject of this paper, forms an indirect link by marking a value as belonging to a particular environment. These tag-value pairs are

kept in the symbol table as part of each atom's entry, in a slot called the value chain. The ordering of the chain works to the same end as the ordering of the frames in deep binding. Frames still exist, but now only contain overhead information, and the tag calculated for that environment. Binding interrogation now entails searching down the given identifier's value chain for the first 'visible' value in a sense determined by the mark associated with the value at binding time. (Note that this search is proportional to the frequency of name clashes rather than the depth of function call, and that the required value will most frequently be at the front of the chain). Merely marking a value as belonging to a particular environment is not sufficient, it must also be a member of any extension of that environment, therefore it is desirable that the tag should contain information to determine whether one environment is an ancestor of another.

Context switching has approximately the same cost as in deep binding, involving the modification of the current frame pointer. Since binding lookup will then be constrained by the tag in the new frame the properties of the marking scheme will ensure the correct value is found. This is the fundamental requirement for this method to succeed, this problem, and a solution to it, is described in the next section.

Development of an heredity function

This resolves itself into a graph problem, the question being: how to mark an n-ary tree such that it is possible to determine directly from the marking of a node whether it is the ancestor of another node. An investigation of graph theory literature proved unhelpful in this respect. The following examples are restricted to binary trees for simplicity, the extension to n-ary is immediate and straightforward.

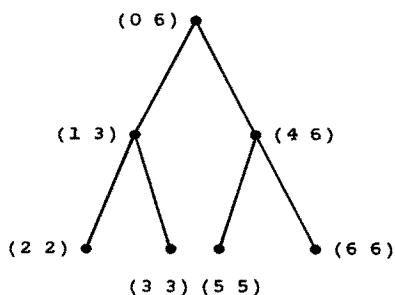


Figure 1.

The first practical solution developed, is to carry out an in-order traversal of the tree, labelling each node with a pair of numbers corresponding to the node count when

a given node was first visited and last visited, the count being incremented as each node is encountered (see Figure 1).

For two arbitrary nodes labelled $(i \ j)$ and $(m \ n)$ the heredity test is true iff $i \leq m \leq j$, which implies that node $(m \ n)$ is a descendant of node $(i \ j)$. In this form the marking algorithm has the drawback that it is inextensible, so although it is possible to close an environment, it cannot be continued from in such a way that the labelling scheme remains consistent.

A solution is to make the tag into a triple, whose elements shall be called sequence, span and generation. Now, when a closed node is continued from the generation count is increased, hence:

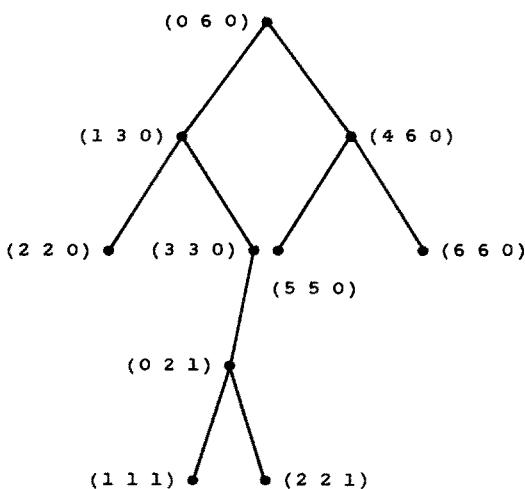


Figure 2.

Tag comparison is a little more complicated; given two nodes labelled $(i \ j \ k)$ and $(l \ m \ n)$, the heredity test fails if $k > n$. If $k = n$, then the result rests on $i \leq l \leq j$ as before, since both tags are in the same subtree, in a sense defined by the generation number. Otherwise $k < n$, and so, by induction, if the current environment is a descendant of the environment being checked, then so is the root of the current generation tree, however that node will be a member of an earlier generation tree (in the example above $(3 \ 3 \ 0)$ is the root of the tree of generation 1). Hence, when testing, the root tag can be substituted for the current environment tag and the algorithm starts over. This is best described in a piece of code:

```
(DE ANCESTORP (ENV1 ENV2)
  (;; T iff env1 is an ancestor of env2)
  (COND ((EQ (GENERATION ENV1) (GENERATION ENV2))
         (CHK (SEQUENCE ENV2) (SEQUENCE ENV1) (SPAN ENV2)))
        ((GREATERP (GENERATION ENV1) (GENERATION ENV2)) NIL)
        (T (ANCESTORP ENV1 (ROOT ENV2)))))
```

where (CHK I J K) returns I<J<=K
and ROOT, SEQUENCE, SPAN and GENERATION are selectors.

Comparison to Baker's rerooting algorithm

Baker's rerooting algorithm [Baker78] suggests a method whereby the environment tree is restructured in such a way that the current environment can become the root of the tree. Thus it can be asserted that all the values in the symbol table are correct for the current environment. However, the frames of the environment tree, rather than reflecting the entropy of the environment define the environments in the same way as a deep bound system. Thus it is possible to interrogate a binding by looking in the symbol table or searching the tree. On a context switch there is the option of rerooting the tree to the new context - and subsequently using the symbol table - or leaving the root where it is and finding values by searching, the decision is left to the user.

Rerooting the tree could be a fairly long and expensive process [Baker78] (which will have to be undone later), on the other hand, running a function deep bound could similarly be inefficient. In some cases the choice may be obvious, but for the rest the difference will be marginal, and offer no gains. It is important to note that under the rerooting scheme the provision of functional objects does not incur unnecessary expense on the user who does not avail himself of the facility.

Under the tag binding regime, the value is always found by examining the symbol table. The current value of a name is discovered by searching down the value chain of the identifier. Where the use of functional objects is avoided, there is little penalty since the correct value will be at the front of the chain - that is three indirections instead of one. Performing a context switch is very simple and thereafter the correct value is found using the heredity function described above. The search is likely to be short for most reasonable cases, since order of magnitude is related to the number of times a name has been bound (ie the number of name clashes).

Implementation and Efficiency

This binding scheme has been implemented by modifying the version of Cambridge LISP [Fitch&Norman77], which runs on the Motorola 68000. The initial coding, which only concerns itself with the interpreter, is rather naive, the intention being to show the validity of the approach.

The original Cambridge interpreter (written for the IBM370) has no genuine facility for FUNARGs, hence frames can be stack allocated. Under the experimental system, frames have to be allocated from the heap, incurring a garbage collection overhead. (In addition the retention of frames creates some problems for the collector since a FUNARG object is a pointer to a collection of frames, but those frames are also referenced from the oblist). Access to a value now requires several indirections and a few comparisons, rather than a single indirection; this can be improved upon using the methods outlined later.

As a consequence of having heap allocated frames, the tracing side of the garbage collector now becomes considerably longer, perhaps indicative of a need to pursue allocation schemes such as that outlined in [Bobrow&Wegbreit73]. Since garbage collection demands that all the currently retained frames will have to be traced in order to mark them, it is worthwhile including here the subsidiary task of retagging the environment tree in such a way that as many nodes as possible (if not all) are members of the same generation, because this will improve the performance of `ancestorp`, particularly when many closures exist. The tracing algorithm is as follows:

- (i) Trace and mark all frames reachable from the current environment
- (ii) Scan the oblist tracing and marking all values reachable from the current environment. Any functional objects encountered are appended to a list for processing later.
- (iii) If the above list of funargs is not empty, then take the first item on the list and work from stage (i) using the first item as the current environment.
- (iv) Scan oblist examining all the values on each value chain and cutting out any references to values bound in unreachable environments. The mark phase is now complete.

As a by-product of the process described above a complete list of the reachable leaves of the environment tree has been generated, from which it is possible to relabel the tree. The algorithm to achieve this is somewhat tedious and inelegant to explain here without causing confusion. The interested reader is referred to the author for further details.

Work on compilation is in hand. It is expected that this will show significantly less overhead than the interpreter, since only a relatively small number of variables in a program generally require treatment as fluids, and the optimisations described below will have been included. However a particular feature of Cambridge LISP is that functions are also handled as fluids, a notion entirely consistent with the consideration of LISP as a functional language. This might give some cause for concern as to the effect on speed, but in fact it is quite small. Assuming the introduction of the optimisations given below, there will be a small cost (three indirections) on the first occasion the value is accessed in an environment but thereafter the cost is just one comparison.

The compilation scheme will fit in well with the existing method. At present a function called fluidbind is given a list of names paired with stack frame locations, all dotted with the stack frame offset in which the list itself should be stored eg.

```
((foo . 3) (bar . 5) (baz . 6) . 9)
```

It is obvious how this is used to implement shallow binding and is more fully described elsewhere [Fitch83]. In the tag binding model there is no need to store information in the frame, hence a simpler form suffices:

```
(foo bar baz . 6)
```

The fluid binding process now consists of running down the above list installing new tag-value pairs on the value chains of the specified identifiers. The fluid restore process is simply the removal of those tag-value pairs.

In order to improve the efficiency of the interpreter and ensure consistency of semantics between interpreted and compiled code, it has been considered that it might be worthwhile adopting of one of the specifications of the Standard LISP Report[Marti78], that variables to be regarded as fluids in Interpreted code, should be flagged as such and cause an error otherwise. As well as providing greater consistency between development and production code, this extra information can be used to good effect in the interpreter, to permit earlier frame deallocation and more efficient storage of variables which are known not to be fluid.

An Example

As a simple example of how function can be used the following program describes the well known algorithm for generating a list of primes known as the Sieve of

Eratosthenes. This dynamically creates a chain of communicating filters, fed by a number generator, and driven by a need to print at the other.

```
(DE FROM (N) (CONS N (FUNCTION (FROM (ADD1 N)))))

(DE FILTER (P Y)
  (COND ((ZEROP (REMAINDER (CAR Y) P)) (FILTER P (EVAL (CDR Y))))
        (T (CONS (CAR Y) (FUNCTION (FILTER P (EVAL (CDR Y)))))))

(DE SEIVE (X)
  (CONS (CAR X) (FUNCTION (SEIVE (FILTER (CAR X) (EVAL (CDR X)))))))
```

Optimisation

The first and obvious improvement is to include a cache-cell in the atom structure as proposed in MDL [Galley&Pfister75]. The MDL scheme required the storage of a value and an environment pointer, similarly, for tag binding a value and its tag must be stored. Now the lookup process would investigate the cached value first and only searche when this fails.

Analysis of program to determine whether an environment has been saved for access or control (ie whether control will pass back through the environment), indicates which frames have to be retained in a FUNARG. If an environment is purely for access, then all but the frames containing fluid bindings may be discarded, otherwise everything is still needed.

Conclusions and Further Work

The provision of functional objects at reasonable cost at last allows a fuller investigation of their properties and utility. Small experiments have been carried out using this implementation on their application to generators, coroutines [Padgett82] and lazy evaluation [Henderson&Morris76]. Such control constructs enhance the feasibility of a functional operating system and programming environment. However it is not yet clear how comfortably eager and lazy evaluation coexist.

Rather than modify the existing (mark and sweep) garbage collector to account for the structures created by the retention of environments, new garbage collection techniques

are being investigated with particular reference to the problems of large address spaces.

A parallel LISP machine is under development running extended Cambridge LISP[Fitch&Marti83], and a compiler with comprehensive data-flow analysis [Marti-unpublished]. At present fluids are not fully supported on this system. Because of the way in which the tag binding method groups all the values of an identifier together, such a block could be transmitted as an interprocessor message. Similarly FUNARGs await implementation on this machine, but again the characteristics of tag binding will make this much easier for the same reasons as above.

A new approach to binding for LISP and LISP-type languages has been described. Context switching is provided in small bounded time. Variable access is unbounded, but will generally be small for the first access with subsequent access in the same environment being in small bounded time.

Acknowledgements

Thanks are due to Robin Sibson, Ian Holyer and Bernard Silbermann for discussions on graph marking algorithms, and to John Fitch for much advice and help.

References

[Baker78]

Baker H G
 Shallow binding in LISP 1.5
 CACM Jul 1978, Vol 21, pp 565-569

[Bobrow&Wegbreit73]

Bobrow D G, Wegbreit B
 A Model and Stack Implementation of Multiple Environments
 CACM Oct 1973, Vol 16, pp 591-603

[Fitch&Norman77]

Fitch J P, Norman A C
 Implementing LISP in a high level language
 Software Practice and Experience, Vol 7 (1977), pp 713-725

[Fitch83]

Fitch J P
 Implementing REDUCE on a microcomputer
 These Proceedings

[Galley&Pfister75]

Galley S, Pfister G
 The MDL Programming Language Manual
 Available from MIT Laboratory for Computer Science

[Henderson&Morris76]

Henderson P, Morris J H
 A Lazy Evaluator
 Proceedings 3rd Symposium on Principles of Programming Languages

[Marti78]

Marti J B et al.
 Standard LISP report
 SIGPLAN Vol 14, No 10, 1978

[Fitch&Marti83]

Fitch J P, Marti J B
 The Bath Concurrent LISP machine
 These Proceedings

[Marti-unpublished]

Marti J B
 Interprocedural Data Flow analysis of LISP programs
 In preparation

[Padgett82]

Padgett J A
 Escaping from Intermediate expression swell: a continuing saga
 Proceedings of EUROCAM-82, Marseille, 1982
 Published in Springer Verlag Lecture Notes in Computer Science 144.

**THE DESIGN OF MAPLE:
A COMPACT, PORTABLE, AND POWERFUL
COMPUTER ALGEBRA SYSTEM***

Bruce W. Char
Keith O. Geddes
W. Morven Gentleman
Gaston H. Gonnet

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

The Maple system has been under development at the University of Waterloo since December 1980. The kernel of the system is written in a BCPL-like language. A macro-processor is used to generate code for several implementation languages in the BCPL family (in particular, C). Maple provides interactive usage through an interpreter for the user-oriented, higher-level, Maple programming language.

This paper discusses Maple's current solution to several design issues. Maple attempts to provide a natural syntax and semantics for symbolic mathematical computation in a calculator mode. The syntax of the Maple programming language borrows heavily from the Algol family. Full "recursive evaluation" is uniformly applied to all expressions and to all parameters in function calls (with exceptions for only four basic system functions).

Internally, Maple supports many types of objects: integers, lists, sets, procedures, equations, and power series, among others. Each internal type has its own tagged data structure. "Dynamic vectors" are used as the fundamental memory allocation scheme. Maple maintains a unique copy of every expression and subexpression computed, employing hashing for efficient access. Another feature relying upon hashing is the "remembering" facility, which allows system and user-defined functions to store results in internal tables to be quickly accessed in later retrieval, thus avoiding expensive re-computation of functions.

The compiled kernel of the Maple system is relatively compact (about 100K bytes on a VAX under Berkeley Unix). This kernel includes the interpreter for the Maple language, basic arithmetic (including polynomial arithmetic), facilities for tables and arrays, print routines (including two-dimensional display), basic simplification, and basic functions (such as *coeff*, *degree*, *map*, and *divide*). Some functions (such as *expand*, *diff* (differentiation), and *taylor*) have a "core" in the kernel, and automatically load external user-language library routines for extensions. The higher-level mathematical operations (such as *gcd*, *int* (integrate), and *solve*, are entirely in the user-language library and are loaded only when called.

* This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, and by the Academic Development Fund of the University of Waterloo.

The approach to portability of the Maple system is also discussed. Maple currently runs in C under Berkeley Vax/Unix, and B under a Honeywell GCOS operating system. Maple is currently being ported to Motorola 68000 microprocessor systems on "Unix-like" operating systems.

1. Motivation for Designing a New System

Maple is a language and system for symbolic mathematical computation, under development at the University of Waterloo since December, 1980. (The name "Maple" is not an acronym but rather it is simply a name with a Canadian identity.) The type of computation provided by Maple is known by various other names such as "algebraic manipulation" or "computer algebra". The Maple system can be used interactively as a mathematical calculator, and computational procedures can be written using the high-level Maple programming language.

With so many languages and systems already developed and being developed, the question arises: "Why develop yet another system?". We will explain our motivation for developing the Maple system and the goals we are trying to achieve with Maple.

The primary motivation can be described as *user accessibility*. This concept has several aspects. The state of the art in 1980 was such that in order to have access to a powerful system such as MACSYMA (or Vaxima)[Mos74a,Fod81a] it was necessary to have a large, relatively costly mainframe computer and then to dedicate it to a small number of simultaneous users. In the university setting, this meant it was not feasible to offer symbolic computation to large classes for student computing. In a broader context, this meant that a large community of potential users of symbolic mathematical computation remained non-users. The development of the MUMATH[Ric79a] and PICOMATH[Sto80a] systems showed that a significant symbolic computation capability could be provided on low-cost, small-address-space microcomputers. It seemed clear that it should be possible to design a symbolic system with a full range of capabilities for symbolic mathematical computation which was neither restricted by the small address space of the early microcomputers nor "inaccessible to the masses" because of unreasonable demands on computing resources. In particular, it seemed possible to design a modular system whose demands on memory would grow gracefully with the needs of the application program.

Portability was another of our earliest concerns, partly because we found ourselves users of a computing environment in transition, and partly because it was clear that a wide variety of computer systems would be coming onto the market in the decade of the 1980's. It was also recognized that "user accessibility" is greatly affected by the quality of user interface which a system provides.

Thus the primary design goals of the Maple system are: *compactness*, a *powerful set of facilities* for symbolic mathematical computation, *portability*, and a *good user interface*. These issues are discussed in more detail in the following sections.

2. Syntax and Semantics

Part of our attempt to provide a good user interface has been to try to design a syntax which is mathematically natural. This goal is conditioned by our current assumption that most users will be accessing Maple from "ordinary" terminals using one-dimensional ASCII

input. (An interesting direction for the future would be to address the design of a good user interface based upon more sophisticated peripherals, building upon previous work such as [Hof79a].) Under the current assumption, many mathematical operations are specified by the traditional function-call syntax common to many programming languages. However, Maple's syntax is enriched with mathematical constructs such as *equations*, and *ranges* (e.g. 1..3).

2.1. Sample Maple Statements

The following sample statements serve to illustrate some of Maple's syntax. (Note that the double-quote operator " is used as a "ditto" symbol to specify the latest expression.)

```
taylor( exp(3*x**2 + x), x=0, 4 );
```

$$1 + x + \frac{7}{2}x^2 + \frac{19}{6}x^3 + \frac{145}{24}x^4 + O(x^5)$$

```
sum( (5*i-3)*(2*i+9), i = 1..n );
```

$$\frac{10}{3}(n+1)^3 + \frac{29}{2}(n+1)^2 - \frac{269}{6}n - \frac{107}{6}$$

```
expand(");
```

$$\frac{10}{3}n^3 + \frac{49}{2}n^2 - \frac{35}{6}n$$

```
eqn1 := 3*x + 5*y = 13; eqn2 := 4*x - 7*y = 30; solve( {eqn.(1..2)}, {x, y} );
```

$$eqn1 := 3x + 5y = 13$$

$$eqn2 := 4x - 7y = 30$$

$$\{y = -\frac{38}{41}, x = \frac{241}{41}\}$$

```
limit( (tan(x)-x)/x**3, x=0 );
```

$$1/3$$

```

fibonacci := proc (n)
    option remember;
    if not type(n,integer) or n<0 then
        ERROR(`invalid argument to procedure fibonacci`)
    else
        if n<2 then n else fibonacci(n-1) + fibonacci(n-2) fi
    fi
end;

fibonacci(101);

```

573147844013817084101

2.2. Control Structures

Many of the control structures in the Maple language have been borrowed from other languages. Specifically, from Algol 68 we borrowed the repetition statement:

```

for <name> from <expr> by <expr> to <expr> while <expr>
    do <statement sequence> od

```

and the selection statement:

```

if <expr> then <statement sequence>
elif <expr> then <statement sequence>
...
else <statement sequence>
fi

```

From C we borrowed the break statement for breaking out of a loop, and RETURN(expr) for returning from a procedure. The ERROR(string) construct, similar to a feature in MACSYMA, is a special function which causes an immediate return to the top level of Maple with "ERROR: string" printed out as a message. However, a procedure may be given the "errortrap" option to allow it to "catch" an ERROR condition in it or in one of the subprocedures it calls -- this is useful for error-checking in library functions, for example.

2.3. Some Semantic Features

An important semantic feature is that Maple applies *full, recursive* evaluation of expressions as the standard evaluation rule. For example, the sequence of statements

```

a := x;
x := 3;
a;

```

yields the value 3, not x. The quoting facility for preventing the evaluation of an expression is to surround the expression with single-quotes, as in 'a+b'.

Another semantic feature in Maple is the general rule that all parameters to all functions (system-supplied or user-defined) are fully evaluated from left to right before being

passed. (Again, the quoting facility can be used to explicitly prevent evaluation). We have allowed precisely four exceptions to this general rule, for four specific system functions: *assigned* (which returns true or false depending on whether the name passed as its argument is assigned or not), *evaln* (which evaluates its argument to a name), *evalb* (which evaluates its argument as a Boolean expression), and *remember* (which is a function used to place the result of a computation in an internal table for later retrieval). Another important feature of Maple is the set of powerful primitive functions that are available when writing procedures in the user-level Maple language. Some examples of such primitive functions are *degree*, *coeff*, *lcoeff* (to extract the leading coefficient), *op* (to pick operands from an expression), and *map* (to apply a procedure onto each of the operands of an expression, separately).

2.4. Types in Maple

Maple provides a *type* function for run-time type-checking. For example, if a procedure *f* has a parameter *x* then a common construct in the procedure body is a statement such as:

```
if not type(x, algebraic) then
    ERROR('invalid argument to procedure f') fi
```

The Maple language has been designed to avoid obligatory type declarations, a principle that we think is important if we are to have a convenient interactive system. Furthermore, we think that the syntax and semantics which applies when writing Maple procedures should be identical with the syntax and semantics of Maple's interactive mode. Consequently, no type declarations are required in Maple and writing type-independent Maple code comes naturally.

On the other hand the Maple language is not type-less. Every object has a precise type and the type information is coded in the data structure. Our concept of "objects" and "types" applies not only to the conventional objects such as integers and lists, but also to mathematical objects such as sums and products, and to objects such as procedures and tables (arrays). As an illustration of the concept of a procedure as an object, a definition of the function *abs* in Maple could take the form:

```
abs := proc (x)
    if not type(x,rational) and not type(x,real) then
        ERROR('invalid argument to procedure abs')
    else
        if x < 0 then -x else x fi
    fi
end;
```

This is an ordinary assignment statement, where the procedure definition (the *proc...end* construct) on the right-hand-side is a valid Maple expression (i.e., an *object* with its own data structure of type *procedure*). The name *abs* could later be re-assigned any other value (of any type). It is also possible to have a procedure definition which has not been assigned to any name, as in the following expression to reverse the left and right hand sides of a list of equations:

```
map( proc (x) op(2,x)=op(1,x) end, [ a=b, c=d, e=f ] ) .
```

3. Data Structures

Maple has a rich set of data structures designed into it, currently about 36 different structures. Approximately one-quarter of these data structures correspond to programming language statements: assignment, if, read, etc. The remaining data structures correspond to the various types of expressions, including expressions formed using standard arithmetic and logical operators, and structures for numbers, lists, sets, tables, (unevaluated) functions, procedure definitions, equations, ranges, and series. All of these structures are represented internally as dynamic arrays (vectors), similar to the approach taken by[Nor82a].

3.1. Advantages of Dynamic Vectors

This approach using dynamic vectors at the machine level and a rich set of data structures at the abstract level has significant advantages in improved compactness and efficiency of the resulting system code. Firstly, in Maple there is only one level of abstraction above the system-level objects. It is clear that in symbolic mathematics there are many data types. The fewer and more direct the mappings between the abstract objects and the system-level objects, the simpler and more efficient will be the code that manipulates these objects. Secondly, we believe that the design of data structures should be related, if possible, to the language that describes the data objects. In our case we have a simple BNF language with the LALR(1) property, and it is natural to relate the data structures to the productions in the language. This immediately suggests the need for many data structures since there are many productions in the language. Thirdly, dynamic vectors allow us, in many cases, to have direct access to each of the components of the structure at about the same cost. This is highly desirable in some circumstances over the sequential access required when all objects are represented as lists. Fourthly, dynamic vectors are more compact than structures linked by pointers. In summary, an important part of the compactness and efficiency of Maple is due to the use of proper data structures.

3.2. Examples of Maple's Data Representation

All of the internal data structures in Maple have the same general format:

Header	data 1	data 2	...	data n
--------	--------	--------	-----	--------

The *header* field encodes the length ($n+1$) of the structure, the type, one bit to indicate simplification status, and two bits to indicate garbage collection status. Every data structure is created with its own length and this length will not change during its entire existence. Data structures are typically not changed after creation since it is not predictable how many other data structures are pointing to a given structure. The normal procedure to modify structures is to create a copy and modify the copy, hence returning a new data structure.

The following are some specific examples of data structures in Maple. The notation $\dagger<\text{xxx}>$ will be used to indicate a pointer to a structure of type *xxx*.

Negative integer

INTNEG	integer	integer	...
--------	---------	---------	-----

Here the INTNEG header includes both the tag for INTNEG and the length of the data structure, which depends upon the size of the negative number being represented. Each integer field of an INTNEG contains one base BASE digit. BASE=10000 for 32-bit machines and BASE=100000 for 36-bit machines; that is, BASE is the largest power of 10 that will fit into a half word on the host machine.

Rational number

RATIONAL	$\uparrow <\text{INTPOS or INTNEG}>$	$\uparrow <\text{INTPOS}>$	
----------	--------------------------------------	----------------------------	--

The second integer is always positive and different from 0 or 1. The two integers are relatively prime.

Sum of several terms

SUM	$\uparrow <\text{exp-1}>$	$\uparrow <\text{factor-1}>$
-----	---------------------------	------------------------------	-----	-----

This structure should be interpreted as pairs of expressions and their constant factors. The simplifier lifts all explicit constant factors from each expression and places them in the $<\text{factor-1}>$ entries. A term consisting only of a rational constant is represented with factor 1.

Product/quotient/power

PROD	$\uparrow <\text{exp-1}>$	$\uparrow <\text{expon-1}>$	$\uparrow <\text{exp-2}>$	$\uparrow <\text{expon-2}>$
------	---------------------------	-----------------------------	---------------------------	-----------------------------	-----	-----

This structure should be interpreted as a product of $<\text{exp-i}>^{<\text{expon-i}>}$. Rational number or integer expressions to an integer power are expanded. If there is a rational constant in the product, this constant will be moved to the first entry by the simplifier.

Series

SERIES	$\uparrow <\text{exp}>$	$\uparrow <\text{exp-1}>$	integer-1
--------	-------------------------	---------------------------	-----------	-----	-----

The first expression is the "taylor" variable of the series, the variable used to do the series expansion. The remaining entries have to be interpreted as pairs of coefficient and exponent. The exponents are integers (not pointers to integers) and appear in increasing order. A coefficient O(1) (function call to the function "O" with parameter 1) is interpreted specially by Maple as an "order" term.

4. The Use of Hashing in Maple

Maple handles all table searching in a uniform way. All of the searching is done by an algorithm which is a slight modification of direct-chaining hashing. Although it is not obvious, the internal tables play a crucial role; they are used for: locating variable names, keeping track of simplified expressions, keeping track of partial computations, mapping expression trees into sequential files for internal input/output, and for storing arrays and tables. It is immediately obvious that the searching in these tables has to be fast enough to

guarantee overall efficiency.

The algorithm used for these tables can be understood as an implementation of direct-chaining where instead of storing a linked list for each table entry, we store a variable-length array. This requires a versatile and efficient storage manager, but without one, symbolic computation would not be feasible regardless.

The two data structures used to implement tables are:

Table entry

HASHTAB	↑<HASH>	↑<HASH>	...	↑<HASH>
---------	---------	---------	-----	---------

Each entry points to a HASH entry or it is 0 if no entry was created. The size of HASHTAB is constant for the implementation. For best efficiency, the number of entries should be prime.

Hash-chain entry

HASH	key	value	...
------	-----	-------	-----

Each entry in the table consists of a consecutive pair, the first one being the hashing key and the second the stored value. A key cannot have the value 0 as this is the indicator for the end of a chain. For efficiency reasons, the HASH entries are incremented by 5 entries at a time and consequently some entries may not be filled. Keys may be any integer or pointer which is representable in one word. In many cases the key is itself a hashing value (two step hashing).

4.1. The Simplification Table

All simplified expressions and subexpressions are stored in the simplification table. The main purpose of this table is to ensure that expressions appear internally only once. Every expression which is entered into Maple or which is internally generated is checked against this table, and if found, the new expression is discarded and the old one is used. This task is done by the simplifier which recursively simplifies (applies all the basic simplification rules) and checks against the table.

The task of checking for equivalent expressions within thousands of subexpressions would not be possible if it was not done with the aid of a "hashing" concept. Every expression is entered in the simplification table using its *signature* as a key. The signature of an expression is a hashing function itself, with one very important attribute: it is order independent. For example, the signatures of the expressions $a+b+c$ and $c+a+b$ are identical; the signatures of $a**b$ and $b**a$ are also identical. Searching for an expression in the simplification table is done by:

- Simplifying recursively all of its components;
- Applying the basic simplification rules.

- Computing its signature and searching this signature in the table. If the signature is found then we perform a full comparison (taking into account that additions and products are commutative, etc.) to verify that it is the same expression. If the expression is found, the one in the table is used and the searched one is discarded.

The number of times that we have to do a full comparison on expressions is minimal;

it is only when we have a "collision" of signatures. Some experiments have indicated that signatures coincide once every 50000 comparisons for 32-bit signatures. (Notice that the signatures are still far from uniform random numbers). The resulting expected time spent doing full comparisons is negligible. Of course, if the signatures disagree then the expressions cannot be equal at the basic level of simplification.

4.2. The Partial Computation Table

The partial computation table is responsible for handling the option remember in function definitions in its explicit and implicit forms. Basically, the table stores function calls as keys and their results as values. Since both these objects are data structures already created, the only cost (in terms of storage) to place them in the table is a pair of entries (pointers). Searching these hashing tables is extremely efficient and even for simple functions it is orders of magnitude faster than the actual computation of the function.

The change in efficiency due to the use of the remembering facility may be dramatic. For example, the Fibonacci numbers computed with

```
f := proc(n)
    if n<2 then n else f(n-1)+ f(n-2) fi end;
```

take exponential time to compute, while

```
f := proc(n) option remember;
    if n<2 then n else f(n-1)+ f(n-2) fi end;
```

requires linear time.

Besides the facility provided to users, the internal system uses the partial computation table for diff, taylor, expand, and evalr. The internal handling of expand is straightforward. There are some exceptions with the others, namely:

- diff will store not only its result but also its inverse; in other words, if you integrate the result of a differentiation the result will be "table-looked up" rather than computed. In this sense, integration "learns" from differentiation.

- taylor and evalr need to store some additional, environment, information (Degree for taylor and Digits for evalr). Consequently the entries in these cases are extended with the precision information. If a result is requested with less precision than what is stored in the table, it is retrieved anyway and "rounded". If a result is produced with more precision than what is stored, the table entry is replaced by the new result.

- evalr only remembers function calls; it does not remember the results of arithmetic operations.

Arrays are implemented using internal tables, with the address of the (simplified) expression sequence of indices used as the hashing key. (Note that since simplified expressions appear only once, we can use their addresses as keys.) Since arrays are treated just like tables at the internal level, dense and sparse arrays are handled equally efficiently.

5. Compact Size as a Design Goal

The kernel of the Maple system (i.e., the part of the system which is written in the systems implementation language) is kept intentionally small -- for example, it occupies about 100K bytes on a VAX. The kernel system includes only the most basic facilities: the user programming language interpreter, numerical, polynomial and series arithmetic, basic simplification, facilities for handling tables and arrays, print routines, and some fundamental functions such as *coeff*, *degree*, *subs* (substitute), *map*, *igcd* (integer gcd computation), *lcoeff* (leading coefficient of an expression), *op*, *divide*, *imodp/imods* (integer modular operations using positive/symmetric representation), and a few others. Some of the fundamental functions have a small "core" coded in the kernel and an interface to the Maple library for extensions. The interface is general enough so that additional power, such as the ability to deal with new mathematical functions of interest to a particular user, can be obtained by user-defined Maple code. Some examples of functions which have such a "core" and a user interface are *diff*, *expand*, *taylor*, *type*, and *evalr* (for evaluation to a real number). Other functions supplied with the system are entirely in the Maple library, including *gcd*, *factor*, *normal* (for normalization of rational expressions), *int*, and *solve*.

The compactness of a system is affected by many different design decisions. The following points outline some of the design decisions which have contributed to the compactness of the Maple system.

1. *The use of appropriate data structures.* As we have pointed out in section 3, an important factor in compactness is the design of a rich set of data structures appropriate to the mathematical objects being manipulated, with a direct mapping between these abstract structures and the machine-level "dynamic arrays". This data structure design avoids the introduction of an intermediate "artificial" level of structure such as lists. One level of compactness is thus achieved because the number of pointers is reduced compared with a linked-list representation. Significantly, another level of compactness is achieved because the code required to manipulate these data structures is generally shorter than the code which must deal with a list representation.
2. *The use of a viable file system.* By having an efficient interpreter and by placing much of the code for system functions into the user-level library, Maple has the property that "you only pay for what you use". Writing functions in the user-level Maple language has the additional advantages of readability, maintainability, and portability. This necessarily depends upon having a file system that (at least through efficient simulation) has some desirable properties such as a tree-structured directory system and variable-length records. It may have been unreasonable a decade ago to make such assumptions about the file system, but these assumptions are (or will be) satisfied by many current and future mainframe and micro computer systems.
3. *Avoiding a large run-time support system.* Providing an "integrated programming environment" or a large run-time support system can lead to non-trivial memory requirements. For example, Franz Lisp on Berkeley Unix starts off at almost 500K bytes. We view Maple as just one of many software tools that a user may employ to solve problems, regardless of which system it may be used on. We see no need to provide all of these tools within Maple itself, not only because they greatly increase the problems of porting without providing any greater algebraic computation power, but also because many computing environments will allow their native software tools to be easily connected to Maple (say, as communicating processes) once Maple has been

ported to that environment. For example, Unix EMACS[Gos81a] can invoke Maple as a subprocess on Berkeley Unix, providing some screen managing and editing facilities for Maple. Thus we do not view the basic Maple system, which provides minimal programming support (e.g., only a simple trace package and no editor), as lacking a programming environment. Rather, we see Maple as being easy to integrate into an environment chosen by the user. We certainly think that having a good user/programming interface to Maple is important. Indeed, we look forward toward developing a "personal algebra machine" in the near future. However, we envision this kind of work as building upon the basic Maple system rather than building more into it.

4. *A policy of treating main memory as a scarce resource.* We believe that this point of view is important if we are to achieve the goal of providing a symbolic computation system to "the masses". Because we have adopted such a point of view, we are constantly concerned about which functions belong in the Maple kernel and which functions can be supplied as user-level code in the Maple library. Since we have an efficient mechanism to retrieve Maple functions from the library, and an efficient interpreter, we are not forced to abandon computational power for the sake of compactness.
5. *The choice of the BCPL family of systems implementation languages.* Implementing Maple in systems languages from the BCPL family has helped us to achieve the compactness goals outlined in the above points. These languages typically produce relatively compact and efficient object code, thus contributing directly to the goal of treating main memory as a scarce resource. The support of "dynamic arrays" in the implementation language allows the creation of compact data structures for the higher-level objects. Furthermore, an implementation language in the BCPL family typically has a run-time library that is small, selectively included, and yet provides the desired functionality.

Although the availability of inexpensive memory and hardware support for large address spaces makes it possible to design a programming system which has all of its routines contained within a large (virtual) main memory, we consider such a design to be inefficient both on mainframe timesharing systems and on the arriving generation of inexpensive but powerful microprocessor systems. It will continue to be true, in our view, that a more efficient design can be achieved by treating main memory as a scarce resource. Maple's design with a relatively small kernel interfacing to an external library takes the latter point of view.

6. Computational Power through Libraries of Functions

Another goal of the Maple system is to provide a powerful set of facilities for symbolic mathematical computation. In other words, we are not willing to achieve compactness by sacrificing the computational power of the system. Thus while the number of functions provided in the kernel system is kept to a minimum, many more functions for symbolic mathematics are provided in the system library, to be loaded as required. The functions in the system library are written in the high-level Maple programming language and are therefore readily accessible to all users of the Maple system. A load module for each library procedure is stored in "Maple internal format" which is a quick-loading expression-tree representation of the procedure definition. When a library function is invoked, its load

module is read into the Maple environment (if not already loaded) and the expression tree is interpreted by the Maple interpreter.

Since run-time loading of compiled code is not (yet) a portable feature for BCPL-family languages on most systems, the execution speed of the system is seen to depend on the interpreter for the Maple language. Maple's interpreter is relatively efficient; for example, an experiment performed by running the tak function[Gri82a] shows Maple's interpreter to be about four times faster than Vaxima's interpreter on that particular benchmark. Consequently, the tradeoff between "user-level" and "system-level" code is not as great in Maple as in other systems. When a critical function has been identified as causing a serious degradation in execution time, it has been moved into the compiled kernel system*. Undoubtedly, there would be some gain in execution speed if all of the Maple functions were coded entirely in the compiled kernel but the resulting loss of compactness, and hence of user accessibility, outweighs such gains in execution speed.

7. Portability

As part of the general goal of "user accessibility", the Maple system is not tied to one operating system, nor to one programming language. Maple is intended to be portable across several languages, descendants of BCPL. To achieve this level of portability and to have a *single* source code (multiple copies are viewed as a disastrous scenario) we use a general purpose macro-processor called Margay. Our current Margay macros define a language very similar to B or C except for the places where the languages differ, where we do one of the following:

- (i) Write a new macro which can be easily mapped onto every language. (Most of the time the macro will have some additional information which may be redundant for some languages but used by others). This is possible since the whole internal maple is relatively small (5500 lines) and we are willing to modify the code to improve portability.
- (ii) Avoid using a particular feature if it is too peculiar to a single language.
- (iii) Avoid, whenever possible, constructs that may be ambiguous across different languages.

The macro-processor is used not only as a way of providing a higher level of readability of the source code, as M6 was used with Altran [Hal71a], but also as a way to make Maple portable across several languages.

Maple is currently running under the GCOS operating system on a Honeywell 66/80 (110K words maximum address space) and under Berkeley Unix on VAX 11/780's. We have begun experiments porting to C on various operating systems on MC68000-based microcomputers, such as Xenix, Unisoft Unix, and the WICAT operating system. We have plans to port Maple into other BCPL-derivative languages in the near future, such as the locally-developed languages WSL[Bos80a] and PORT[Mal82a].

* This was done, for example, with the function for polynomial division which was first placed in the system library and then later moved into the kernel. On the other hand, some functions such as solve and int have been moved from the kernel out to the system library without causing a significant degradation in performance.

8. Notes on Software Development

Maple development started on a Honeywell system in B when the project began in 1980. When Waterloo acquired a VAX in 1981, we ported Maple to C. At that time, we were forced to demonstrate portability between languages and operating systems out of necessity, since Maple had to continue to work on the Honeywell for student use.

8.1. Choice of BCPL-derivatives as implementation language

While Maple's behaviour is based as much on our coding of algorithms and data structures as on our choice of implementation language, it seems clear to us that a general-purpose system based on a BCPL-family language can be compact, yet have reasonable performance on interesting problems. The software tools available (parser-generators, execution profilers, etc.) have made the implementation process proceed in a timely fashion with a small staff. While we don't think any final conclusions should or can yet be drawn about the relative merits of Lisp or BCPL-family languages as vehicles for symbolic systems, we do suggest that the choice of system implementation languages now seems less limited than in the early '70s when the last generation of algebraic systems were being designed. Our approach towards portability is of course tied to the health and propagation of BCPL-family languages; but we feel this is assured, at least for the next few years, given the interest of the larger computer science community in such languages. We feel that our approach frees us to concentrate on providing algebraic computation power, as opposed to worrying about machine code generators, portable subsets, or porting programming environments.

8.2. Breadboarding

Our mode of operation up to this point is akin to "breadboarding" an electrical design, in that we can observe real, not merely theoretical, performance over a long period of time, and yet be in a position to make possibly incompatible changes in a timely fashion. The compactness of the Maple kernel is what makes breadboarding feasible for us, in that someone modifying the kernel must deal with only 5500 lines of code. As a consequence of this approach, version 1 of the Maple system is almost unrecognizable as a predecessor of version 2, although version 3 (currently under development) is characterized mainly by added facilities rather than by fundamental design changes compared with version 2.

We do not claim to have worked out all of the design and implementation issues facing us. Maple has changed since the inception of the project, not only through the introduction of additional features, but also through incompatible changes made because we changed our minds. Nevertheless, we have willingly subjected the system to significant usage at every stage of its development. The first version of the Maple system was running within a week of the first discussions on its design, with significant "real-world" problems solved using it within a month. Hundreds of students at Waterloo have already used Maple in undergraduate and graduate classes *. We are continuing to operate in a mode where there is a short time period between ideas and their implementation, with the result that the practical, real, applications of "great ideas" are soon found, and the "great ideas that are not-so-

* Maple is used in an undergraduate data base class (its support of sets and tuples was used for a relational data base package), as well as courses in algebraic manipulation. It has also been used for "real" formula manipulation by some of our departmental colleagues, and as an algebraic calculator by students on a casual basis.

great" are modified or discarded. In this, we are grateful for the flexibility of our academic environment (and students!), and for the vigour of workers in algebraic manipulation of the past decade who have provided us with a wealth of implemented algorithms and applications problems that are obvious tests for Maple.

To some extent, the breadboarding approach means that we have had to proceed slowly on the design of "large features" such as user-directed simplification, but we think that by tying the design of Maple closely to its implementation and usage we have gained invaluable experience and feedback. Furthermore, we think that doing so has kept us from designing beyond our immediate capacity to remain faithful to maintaining efficiency and portability.

9. Conclusions

We expect several more cycles of building, using, and learning for Maple. Nevertheless, we believe that our accomplishments so far affirm the validity of our approach towards data representation and manipulation, towards portability, and towards making algebraic manipulation generally available. David Stoutemyer once said that one way to make computer symbolic math economically feasible for the masses would be to encourage the University of Waterloo to develop a compact "WATALG" system [Sto79a]. With the Maple system, we have taken up the spirit of that challenge.

Acknowledgments

We wish to acknowledge the assistance of: Robert Bell, Greg Fee, Brian Finch, Marta Gonnet, Barry Joe, Howard Johnson, Patrick McGeer, Michael Monagan, Mark Mutrie, Sophie Quigley, Carolyn Smith, and Stephen Watt for their various contributions to the Maple project.

References

- a.
- Bos80a. F. David Boswell, *A Secure Implementation of the Programming Language Pascal*, Dept. of Computer Science, University of Waterloo (1980). (M.Math thesis)
 - Fod81a. John Foderaro and Richard Fateman, "Characterization of VAX Macsyma," *Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation*, pp. 14-19 Association for Computing Machinery, (1981).
 - Gos81a. James Gosling, *Unix Emacs Reference Manual*. 1981.
 - Grí82a. Martin Griss, Eric Benson, and Gerald Maguire, Jr, "PSL: A Portable LISP System," *Proceedings of the 1982 ACM Symposium on Lisp and Functional Programming*, pp. 88-97 (1982).
 - Hal71a. Andrew D. Hall, Jr., "The ALTRAN System for Rational Function Manipulation - A Survey," in *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation*, ed. S.R. Petrick, Special Interest Group on Symbolic and Algebraic Manipulation, Association for Computing Machinery (1971).
 - Hof79a. Carl Hoffman and Richard Zippel, "An Interactive Display Editor for MACSYMA," *Proceedings of the 1979 MACSYMA User's Conference*, p. 344 (1979).
 - Mal82a. Michael Malcolm, Bert Bonkowski, Gary Stafford, and Phyllis Didur, *The Waterloo Port Programming System*, Dept. of Computer Science, University of Waterloo (1982).
 - Mos74a. Joel Moses, "MACSYMA - The Fifth Year," in *Proceedings of the Eurosam 74 Conference*, , Stockholm (August 1974).
 - Nor82a. Arthur Norman, "The Development of a Vector-based Algebra System," *Proceedings EUROCAM '82*, pp. 237-248 Springer-Verlag, (1982). Lecture Notes in Computer Science #144.
 - Ric79a. Art Rich and David Stoutemyer, "Capabilities of the muMATH-79 Computer Algebra System for the INTEL-8080 Microprocessor," *EUROSAM 1979*, pp. 241-248 Springer-Verlag, (1979).
 - Sto79a. David Stoutemyer, "Computer Symbolic Math and Education: a Radical Proposal," *Proceedings of the 1979 Macsyma User's Conference*, pp. 142-158 MIT Laboratory for Computer Science, (1979).
 - Sto80a. David Stoutemyer, "PICOMATH-80, an Even Smaller Computer Algebra Package," *SIGSAM Bulletin* **14**(3) pp. 5-7 (1980).

LISP COMPILEMENT VIEWED AS PROVABLE SEMANTICS PRESERVING PROGRAM TRANSFORMATION

by

Herbert Stoyan

University of Erlangen

Erlangen,Germany

Abstract. This paper is based on the view that the main stage of compilation, the code generation phase, has very much in common with the simplification of algebraic expressions. The compiler is seen as symbolic program manipulator, the so-called intermediary language as data-structure (a combination of tree structures and symbol table) to store programs which is convenient for executing simplification, decomposition and optimization steps. The main aspect of code generation is decomposition leading to a program which is still using constructs of the source language but without any composed parts. It is based on a few operations on the internal data-structure-representation of the source program.

This is demonstrated using the language LISP as an example. The decomposed program uses only seven basic language elements of LISP. To prove the preservation of semantics of the decomposing operations (described as transformation rules) we invented new axioms for LISP. Some of the axioms, transformation rules and correctness proofs are given. An example illustrates the steps of decomposition.

Introduction. Compilation is usually understood as translation process leading from strings in a source language into strings in an object language. The traditional approach concentrates on the process of accepting character strings and building up syntax trees which reflect a hypothetic generating process. By this way most work on compilation was done in the fields of lexical and syntactical analysis. The more important part of code generation was neglected. Only recently the researchers in the field of compilation have invented fairly sophisticated schemata to generate code from the syntax tree for constructing code generator generators. The possibility of such a schema and its correctness was generally taken for granted. We believe that we need a theoretical basis which enables correctness proofs for further progress. We propose that this should be found in the concept of program transformation. We want to draw a parallel between compiling and algebraic symbolic computation. Therefore we want to argue for a change in the emphasis: Only the very first and very last steps in compilation are seen to be connected with formal languages, namely language acceptance (as conversion of the external program representation into an internal program representing data-structure) and language generation (as conversion of the internal program representation into an external one). In contrast, the main steps are transformation steps similar to the simplification of algebraic expressions (BL). In the past only optimization - an important part of compilation -

was understood in this way. Now we want to extend this model of understanding even to code generation. Our approach enables us to understand and describe theoretically the code generation process. Using it we can develop models of code generators and prove them correct. Because of their formal nature one can use these models as building parts for machine independent code generators.

To do this in practice, one has first to design an abstract machine describing the "resources" needed to implement the programming language and second to develop the decomposing transformation rules. The third step is then to write Macros for implementing the abstract machine on a concrete computer. Because of the schematic nature of such a code generation process an additional peephole optimization step could increase the efficiency of the resulting code. (The same could have been achieved by developing much more detailed and adapted transformation rules.)

We give now an outline how these parts could have been designed in the case of LISP. Because of space problems we cannot give all of them complete. To cover the side of correctness proofs we give some of the axioms used as basis of the proofs and one example proof.

Abstract machines. LISP is a language enabling a high degree of composition. The computational effect of composition is the implicit saving and handling of intermediary results. If we decompose we have to make the implicit processes explicit. This means, we have to name computational resources in order to save and reactivate intermediary values.

Two other basic processes must be specified: The argument transport and the value delivery.

If we face these problems there are 5 principal variants of abstract machines (each of them must have a stack as basic means to implement the possible recursion):

1. *pure stack machine, simple variant*

The main resource is a stack with only the top being accessible. Arguments of a function to be called are simply stacked. Argument and value delivery is via the stack. To enable random access to arguments, we use the variables (LISP literal atoms) themselves.

2. *pure stack machine, frame variant*

This is in most parts the same as the "simple variant". The difference is that arguments cannot simply be stacked but the construction of a new frame is to be prepared. It is assumed that this is costly and the arguments are therefore saved on the stack until frame construction is done. After this, they are moved into the frame.

3. *frame stack machine, simple variant*

The main resource is a stack which is randomly accessible with respect to all sub-fields of a frame. The new frame is created by simply stacking arguments. Argument and value delivery is via the stack. To enable random access to arguments we use sub-fields of the frame (this realizes lexical scoping).

4. frame stack machine, frame variant

Again this is in most aspects the same as the simple variant. The modification proceeds in the same direction as in the case of the pure stack machine. (The reason for this variant is that the frame costs are assumed to be very high.)

5. APH machine

There are two different main resources. One is again a stack which must be accessible with respect to all subfields of a frame. The other is a set of resources to transport the arguments and the value. We use "argument-place-holders" to avoid the word "registers" which seems to be too close to real hardware elements (G-H).

To use one of the abstract machines for the purpose of program decomposition we introduce artificial variables which represent the stack, the argument-place-holders etc. As machine language we use LISP itself.

In the rest of the paper will base our discussion on the pure stack machine in its simple variant. The reason is that the reader may compare the result of our program transformation process with the code of the ByteLISP machine (D).

The machine instructions (compare (D), p.224) :

(SETIQ \$Si constant)	Push a constant onto the stack.
(SETIQ \$Si symbol)	Access a variable and push its value onto the stack.
(SETIQ symbol \$Si)	Move the top of stack into a variable and pop.
(SETIQ \$Si(fn \$Si ... \$Si+n-1))	Call a function of n arguments - which are already stacked (after the call the n arguments are popped and the value is on top of the stack).
(fn \$Si ... \$Si+n-1)	Call a function of n arguments - which are already stacked (after the call the arguments are popped; no value is pushed).
(PROG((var ₁ \$Si)...(var _k \$Si+k-1)var _{k+1} ... var _n) ...)	on entry: bind variables var ₁ ...var _k to their successive values on the stack; the other variables (up to var _n) to NIL. on exit: redo the binding operations done at entry.
(LAMBDA(var ₁ ... var _k) ...)	on entry: bind variables var ₁ ...var _k to their successive values on the stack. In addition do implementation dependent bookkeeping and error checking on function entry. on exit: redo the binding operations done at entry.

	try.Do implementation dependent book-keeping.
(GO tag)	Unconditional jump to tag.
(AND \$Si(GO tag))	Conditional jump (if top of stack different from NIL).Always popping.
(OR \$Si(GO tag))	Conditional jump (if top of stack equal to NIL).Always popping.
(RETURN \$RESULT)	Enabling exit from the outer (LAMBDA ...) operation.The function value is on top of the stack.

The numbers inside the variables \$Si are not very important and could be ignored.We introduce them for error checking only.(For example:The first argument and the SETQ-destination must have the same number.)They could be used for expressing multiple POP-operations:If after a (SETQ \$S5 ...) follows a (SETQ \$S3 ...),we have to pop 3 times and to push then a value.We use \$RESULT and \$S1 interchangeably:An instruction (SETQ \$RESULT \$S1) is really a no-op.

The transformation rules.We present parts of 7 rules for transforming LISP programs into equivalent decomposed programs.Some of them come along as a set of cases.The rules are to be applied one after another in the following way:a program is inspected from outside towards inside to see whether there are application cases of a given rule.If we arrive at constants or variables then we cannot find further possibilities for rule application.Then we take the next rule.

All rules share the property that the semantic equivalence between original and transformed program holds even if not all application cases are really covered.

(R1) Rewrite any form (PROG ...) by (\$PROG k (PROG ...))

(The rule is applied by matching the application pattern first to outer forms and then stepwise to inner ones.It is never applied two times on the same form.The number k is assumed to be different for any application case.This can be done easily by a counter.)

(R2) Rewrite the whole function (LAMBDA(var₁ ... var_n)form₁ ... form_{m-1} form_m) by (LAMBDA(var₁ ... var_n)(PROG(\$RESULT)(\$STAT 1 form₁)...(\$STAT 1 form_{m-1}) (SETQ \$RESULT form_m)(RETURN \$RESULT)))

(R3) Rewrite the statement-forms of the result of rule (R2) (stepwise to inner parts) by the following rewrite rules:

(R3-A1) (SETQ \$RESULT(fn a₁ ... a_r))

⇒

(PROGN(SETQ \$S1 a₁)...(SETQ \$Sr a_r)(SETQ \$RESULT(fn \$S1 ... \$Sr)))

(R3-A2) (SETQ \$Si(fn a₁ ... a_r))

⇒

(PROGN(SETQ \$Si a₁)...(SETQ \$Si+r-1 a_r)(SETQ \$Si(fn \$Si ... \$Si+r-1)))

- (R3-A3) (\$STAT i (fn a₁ ... a_r))
 ⇒
 (PROGN(SETQ \$Si a₁)...(SETQ \$Si+r-1 a_r)(fn \$Si ... \$Si+r-1))
- (R3-A4) (\$PSTAT i j (fn a₁ ... a_r))
 ⇒
 (PROGN(SETQ \$Sj a₁)...(SETQ \$Sj+r-1 a_r)(fn \$Sj ... \$Sj+r-1))
- (R3-B1) (SETQ \$RESULT(SETQ var form))
 ⇒
 (PROGN(SETQ \$RESULT form)(SETQ var \$RESULT))
 - We drop here 3 rules concerning SETQ (for space reasons) -
- (R3-C1) (SETQ \$RESULT(COND(p₁ e₁₁ ... e_{1n₁}) ... (p_r e_{r1} ... e_{rn_r})))
 ⇒
 (COND((#FCOND 1 1(SETQ \$S1 p₁))(\$STAT 1 e₁₁) ... (\$STAT 1 e_{1n₁-1}
 (SETQ \$RESULT e_{1n₁}))
 ...
 ((#FCOND r 1(SETQ \$S1 p_r))(\$STAT 1 e_{r1}) ... (\$STAT 1 e_{rn_r-1}
 (SETQ \$RESULT e_{rn_r}))
 (T(SETQ \$RESULT NIL)))
 (If there is a clause with n_j = 0 then it is transformed into:
 ((#FCOND j 1(SETQ \$S1 p_j))(SETQ \$RESULT \$S1))
 - We drop here 3 rules concerning COND (for space reasons) -
- (R3-D1) (SETQ \$RESULT(AND a₁ ... a_{r-1} a_r))
 ⇒
 (COND((NOT(#FCOND 1 1(SETQ \$S1 a₁)))(SETQ \$RESULT NIL))
 ...
 ((NOT(#FCOND r-1 1(SETQ \$S1 a_{r-1})))(SETQ \$RESULT NIL))
 (T(SETQ \$RESULT a_r)))
 - We drop here 3 rules concerning AND and 4 similar rules concerning
 OR (for space reasons) -
- (R3-F1) (SETQ \$RESULT(PROGN a₁ ... a_{r-1} a_r))
 ⇒
 (PROGN(\$STAT 1 a₁) ... (\$STAT 1 a_{r-1})(SETQ \$RESULT a_r))
 - We drop here 3 rules concerning PROGN. For PROG1 and PROG2 are simi-
 lar rules possible, too. We drop them here for space reasons.
- (R3-I1) (SETQ \$RESULT(\$PROG j (PROG((var₁ form₁)...(var_k form_k)var_{k+1} ... var_r
 st₁ ... st_s)))
 ⇒
 (PROGN(SETQ \$S1 form₁)...(SETQ \$Sk form_k)(\$SETQ \$S1(PROG((var₁ \$S1)...
 ... (var_k \$Sk)var_{k+1} ... var_r)(#PSTAT j 1 st₁)...(#PSTAT j 1 st_s)
 (SETQ \$S1 NIL)\$PLj(RETURN \$S1)))(SETQ \$RESULT \$S1))

(R3-I2) ($\text{SETQ } \$Si(\text{PROG } j(\text{PROG}((\text{var}_1 \text{ form}_1) \dots (\text{var}_k \text{ form}_k) \text{var}_{k+1} \dots \text{var}_r)$
 $\quad \quad \quad \text{st}_1 \dots \text{st}_s))$

⇒

($\text{PROGN}(\text{SETQ } \$Si \text{ form}_1) \dots (\text{SETQ } \$Si+k-1 \text{ form}_k)(\$SETQ \$Si(\text{PROG}((\text{var}_1 \Si
 $\quad \quad \quad) \dots (\text{var}_k \$Si+k-1) \text{var}_{k+1} \dots \text{var}_r)(\$PSTAT j i \text{ st}_1) \dots$
 $\quad \quad \quad (\$PSTAT j i \text{ st}_s)(\text{SETQ } \$Si \text{ NIL})\$PLj(\text{RETURN } \$Si)))\$Si$)

- We drop here 2 rules concerning PROG (for space reasons) -
 (For all rules concerning PROG: Any st_j ($1 \leq j \leq s$) which is a symbol
 (a label) remains as it was, i.e. is not put into a ($\$PSTAT \dots$).)

(R3-J1) ($\text{SETQ } \$RESULT((\text{LAMBDA}(\text{var}_1 \dots \text{var}_r) \text{form}_1 \dots \text{form}_s) a_1 \dots a_r))$

⇒

($\text{PROGN}(\text{SETQ } \$S1 a_1) \dots (\text{SETQ } \$Sr a_r)(\text{PROG}((\text{var}_1 \$S1) \dots (\text{var}_r \$Sr))$
 $\quad \quad \quad (\$STAT 1 \text{ form}_1) \dots (\$STAT 1 \text{ form}_{s-1})(\text{SETQ } \$S1 \text{ form}_s))(\text{SETQ } \$RESULT \$S1$
 $\quad \quad \quad))$)

- We drop here 3 rules concerning LAMBDA-expressions -

(R3-K) ($\$PSTAT i j(\text{GO } \text{label})$)

⇒

($\text{GO } \text{label}$)

(R3-L) ($\$PSTAT i j(\text{RETURN } \text{form})$)

⇒

($\text{PROGN}(\text{SETQ } \$Sj \text{ form})(\text{GO } \$PLi)$)

- (R4) Rewrite any form ($\$SETQ \$Si(\text{PROG}(\dots) \dots (\text{RETURN } \$Si))$) by ($\text{PROG}(\dots) \dots$) .
- (R5) Rewrite any form ($\text{PROGN } a_1 \dots a_n$) by $a_1 \dots a_n$. Drop any variable $\$Si$, which, during this rewriting process, becomes either an a_j ($1 \leq j \leq n$) of some PROGN , a statement-form of a PROG (a label) or a middle effect-expression in a COND.
- (R6) Rewrite the result of (R5), the whole function, by replacing any specialform (COND ...) with ($\$COND i(\text{COND} \dots)$), where i is an unique number (not two conditional expressions will get the same number). Apply the rule recursively to all subforms of conditional expressions and to all PROG-statement-forms.
- (R7) Rewrite any form ($\$COND i(\text{COND} \dots)$) according to their structure by the following rules:

{R7-1} If the conditional expression follows the pattern:

($\text{COND}((\$FCOND } 1 j \text{ form}_{11} \dots \text{form}_{1k_1}) e_{11} \dots e_{1n_1})$
 $\quad \quad \quad \dots$
 $\quad \quad \quad ((\$FCOND } r j \text{ form}_{r1} \dots \text{form}_{rk_r}) e_{r1} \dots e_{rn_r})$
 $\quad \quad \quad (\text{T } e_1 \dots e_s))$

(with or without the last clause) replace the whole form by

$\text{form}_{11} \dots \text{form}_{1k_1}$ (OR $\$Sj(\text{GO } \$CIL1))e_{11} \dots e_{1n_1}$ (GO \$Ci) \$CIL1
 $\quad \quad \quad \dots$
 $\quad \quad \quad \text{form}_{r1} \dots \text{form}_{rk_r}$ (OR $\$Sj(\text{GO } \$CILr))e_{r1} \dots e_{rn_r}$ (GO \$Ci) \$CILr
 $\quad \quad \quad e_1 \dots e_s$ \$Ci

If there is no clause beginning with T, drop $e_1 \dots e_s$. If there is a form form_{k1} which is a variable $\$S_m$, drop it. If there is any clause (number 1) which has a single form form_{11} and this is $(\text{SETQ } \$S_j \text{ constant})$ ($k_1 = 1$, constant $\neq \text{NIL}$) then drop all clauses with index greater than 1 (including the $(T \dots)$ -clause) and transform the 1.th clause into:

$e_{11} \dots e_{1n_1} \$C_i$

If there is a clause (number m) with $n_m = 0$ then transform it into:

$\text{form}_{m1} \dots \text{form}_{mk_m} (\text{AND } \$S_j(\text{GO } \$C_i)) .$

- We drop 3 other similar rules (for place reasons) -

An Example. The following function:

```
(DE MEMBER(EL S)(COND((NULL S)NIL)
                      ((EQ(CAR S)EL)S)
                      (T(MEMBER EL(CDR S))))))
```

is transformed via rule R1,rule R2,rules R3-C1,R3-A2,R3-A2,R3-A1,R3-A2,R3-A2,R4,R5,R6,R7-1 into:

(DE MEMBER(EL S)	(BIND(EL S)())
(PROG(\$S1 \$S2 \$RESULT)	
(SETQ \$S1 S)	(VAR S)
(SETQ \$S1(NULL \$S1))	(FN NULL 1)
(OR \$S1(GO \$C1L1))	(FJUMP C1L1)
(SETQ \$RESULT NIL)	(CONST NIL)
(GO \$C1)	(JUMP C1)
\$C1L1	C1L1
(SETQ \$S1 S)	(VAR S)
(SETQ \$S1(CAR \$S1))	(FN CAR 1)
(SETQ \$S2 EL)	(VAR EL)
(SETQ \$S1(EQ \$S1 \$S2))	(FN EQ 2)
(OR \$S1(GO \$C1L2))	(FJUMP C1L2)
(SETQ \$RESULT S)	(VAR S)
(GO \$C1)	(JUMP C1)
\$C1L2	C1L2
(SETQ \$S1 EL)	(VAR EL)
(SETQ \$S2 S)	(VAR S)
(SETQ \$S2(CDR \$S2))	(FN CDR 1)
(SETQ \$RESULT(MEMBER \$S1 \$S2))	(FN MEMBER 2)
\$C1	C1
(RETURN \$RESULT)	(RETURN)
))	

For comparison we present code in the right column which could be used for the

ByteLISP machine (D).

We turn now to the topic of correctness proofs. First we give the basic facts of the language as a set of axioms. Then we prove the correctness of one transformation rule.

Axioms.

a) Axioms for forms:

(a1) LAMBDA-introduction: form = (LAMBDA() form))

(a2) PROGN-introduction : form = (PROGN form)

(a3) COND-introduction : form = (COND(T form))

(newvar must not occur free in form and is strict local.)

(a5) restricted-SETQ-introduction: form = (SETQ newvar form)

(newvar is strict local and not used outside.)

(a6) form-decomposition :
$$(fn\ var_1\ \dots\ var_{i-1}\ a_i\ a_{i+1}\ \dots\ a_n) =$$

(PROGN(SETQ locvar a.)

(fn var₁ ... var_{i-1} locvar a_{i+1} ... a_n))

($1 \leq i \leq n$.locvar must not be identical with one of the var_j ($1 \leq j < i$), is strict local and not used outside. The computation of a_i must not involve any assignment toward one of the variables var_j nor change the definition of the function fn.)

(a7) PROGN-introduction-in-body: $\text{LAMBDA}(\text{var}_1 \dots \text{var}_n) \text{form}_1 \dots \text{form}_{m-2} \text{form}_{m-1} \text{form}_m$
 $=$

(1 \leq n, 1 \leq m)

(a8) SETQ-introduction: $\text{var} = (\text{SETQ } \text{var } \text{var})$

b) Axioms for SETQ's:

(b1) intermediary-variable-introduction: (SETQ var form)

11

(PROGN(SETQ locvar form)(SETQ var locvar))

(locvar must be strict local and not used outside.)

(b2) variable-introduction: $(\text{SETQ } \text{var } \text{form}) = (\text{PROGN}(\text{SETQ } \text{var } \text{form}) \text{var})$

(b3) *SETQ-decomposition*: $(\text{SETQ } \text{var}_1 (\text{SETQ } \text{var}_2 \text{ form}))$

11

(PROGN(SETQ var₁ form)(SETQ var₂ var₁))

c) Axioms for PROGN's:

(c1) Side-effectless-form-introduction: $(\text{PROGN } a) = (\text{PROGN form } a)$
 (form is a side-effectless form.)

(c2) PROGN-composition: $(\text{PROGN } a_1 \dots a_{i-1} a_i a_{i+1} a_{i+2} \dots a_n)$
 $=$

$(\text{PROGN } a_1 \dots a_{i-1} (\text{PROGN } a_i a_{i+1}) a_{i+2} \dots a_n)$
 $(1 \leq i < n)$

(c3) PROG-into-PROGN-composition: $(\text{PROGN } a_1 \dots a_{n-1} a_n)$
 $=$

$(\text{PROGN}(\text{PROG}()) a_1 \dots a_{n-1} a_n)$

$(1 < n. a_n$ may be subsumed into the PROG only if the whole PROGN is itself an $a_j (j < n)$ of another PROGN or PROG. None of the a_j should be a symbol.)

(c4) PROG-result-dropping: $(\text{PROGN}(\text{PROG}(\dots) \dots (\text{RETURN form}) \dots) a)$
 $=$

$(\text{PROGN}(\text{PROG}(\dots) \dots (\text{RETURN NIL}) \dots) a)$

(The RETURN occurs as direct statement-form of the PROG or as part of one and is related to the visible PROG. form must not have any side-effects.)

(c5) last-RETURN-removal: $(\text{PROGN}(\text{PROG}(\dots) st_1 \dots st_{n-1} (\text{RETURN form})) a)$
 $=$

$(\text{PROGN}(\text{PROG}(\dots) st_1 \dots st_{n-1} form) a)$

$(1 \leq n. form$ must not be a symbol.)

(c6) COND-to-OR-reduction: $(\text{PROGN}(\text{COND}((\text{NOT } p)e)) a) = (\text{PROGN}(\text{OR } p e) a)$

(c7) redundant-assignment-removal: $(\text{PROGN}(\text{SETQ var form}_1)(\text{SETQ var form}_2))$
 $=$

(SETQ var form_2)

$(form_2$ may be a constant, a variable or any combination or specialform in which var itself does not occur free. form₁ must not involve any side-effect. var must be strict local.)

(c8) exchange-with-one-argument-forms: $(\text{PROGN } a_1 (\text{fn } a_2)) = (\text{fn}(\text{PROGN } a_1 a_2))$
 $(a_1$ must not have a side-effect on the definition of fn.)

(c9) PROGN-value-propagation: $(\text{SETQ var}(\text{PROGN } a_1 a_2)) = (\text{PROGN } a_1 (\text{SETQ var } a_2))$

d) Axioms for PROG's :

Axioms (d1) - (d7) are very similar to (c1) - (c7) respectively with only little changes in conditions.

(d8) unused-label-removal: $(\text{PROG}(\dots) st_1 \dots st_{i-1} \text{label } st_{i+1} \dots st_n)$
 $=$

$(\text{PROG}(\dots) st_1 \dots st_{i-1} st_{i+1} \dots st_n)$

(d9) GO-redirection : $(\text{PROG}(\dots)\text{st}_1 \dots \text{st}_{i-1} \text{label}_1 \text{label}_2 \text{st}_{i+2} \dots \text{st}_{k-1} \dots (\text{GO } \text{label}_1) \dots \text{st}_{k+1} \dots \text{st}_n)$

=

$(\text{PROG}(\dots)\text{st}_1 \dots \text{st}_{i-1} \text{label}_1 \text{label}_2 \text{st}_{i+2} \dots \text{st}_{k-1} \dots (\text{GO } \text{label}_2) \dots \text{st}_{k+1} \dots \text{st}_n)$

$(1 \leq i < k \leq n)$. The label_1 is not necessarily the st_k ($i+1 < k \leq n$) but may occur in it. Similarly, the label_2 may be a st_j ($1 \leq j < i$) or may occur in it.)

(d10) COND-decomposition: $(\text{PROG}(\dots)\text{st}_1 \dots \text{st}_{i-1} (\text{COND}(\text{p}_1 \text{ e}_1)(\text{p}_2 \text{ e}_2))\text{st}_{i+1} \dots \text{st}_n)$

=

$(\text{PROG}(\dots)\text{st}_1 \dots \text{st}_{i-1} (\text{COND}(\text{p}_1 \text{ e}_1 (\text{GO newlab}))$

$(\text{COND}(\text{p}_2 \text{ e}_2))\text{newlab st}_{i+1} \dots \text{st}_n)$

$(1 \leq i \leq n)$. newlab must not occur among the st_j ($1 \leq j \leq n$). The e_k may be absent ($k = 1, 2$).

(d11) PROG-LAMBDA-equivalence:

$(\text{PROG}((\text{var}_1 \text{ a}_1) \dots (\text{var}_n \text{ a}_n))\text{form}_1 \dots \text{form}_{m-1} (\text{RETURN form}_m))$

=

$((\text{LAMBDA}(\text{var}_1 \dots \text{var}_n)\text{form}_1 \dots \text{form}_{m-1} \text{form}_m) \text{a}_1 \dots \text{a}_n)$

$(1 \leq m, 1 \leq n)$. None of the forms form_i ($1 \leq i < m$) is permitted to be or to contain a GO or a RETURN which is related to the outer PROG.)

We drop here 9 further axioms.

e) Axioms for conditional expressions:

(e1) PROGN-introduction in consequence: $(\text{COND}(\text{p}_1 \text{ e}_{11} \dots \text{e}_{1i-1} \text{e}_{1i} \dots \text{e}_{1k} \text{e}_{1k+1} \dots \text{e}_{1n_1})$

$(\text{T e}_{21} \dots \text{e}_{2n_2}))$

=

$(\text{COND}(\text{p}_1 \text{ e}_{11} \dots \text{e}_{1i-1} (\text{PROGN } \text{e}_{1i} \dots \text{e}_{1k}) \text{e}_{1k+1} \dots \text{e}_{1n_1})$

$(\text{T e}_{21} \dots \text{e}_{2n_2}))$

$(1 \leq i \leq k \leq n_1, 1 \leq n_2)$

(e2) COND-Composition: $(\text{COND}(\text{p}_1 \text{ e}_{11} \dots \text{e}_{1n_1})(\text{p}_2 \text{ e}_{21} \dots \text{e}_{2n_2}) \dots$

$(\text{p}_m \text{ e}_{m1} \dots \text{e}_{mn_m}))$

=

$(\text{COND}(\text{p}_1 \text{ e}_{11} \dots \text{e}_{1n_1})(\text{T}(\text{COND}(\text{p}_2 \text{ e}_{21} \dots \text{e}_{2n_2}) \dots$

$(\text{p}_m \text{ e}_{m1} \dots \text{e}_{mn_m})))$

$(1 \leq m, 0 \leq n_1, \dots, 0 \leq n_m)$

(e3) clause-inversion: $(\text{COND}(\text{p } \text{e}_1)(\text{T } \text{e}_2)) = (\text{COND}((\text{NOT } \text{p}) \text{e}_2)(\text{T } \text{e}_1))$

(e4) PROGN-removal-from-first-condition: $(\text{COND}((\text{PROGN } \text{a}_1 \text{ a}_2) \text{e}_1)(\text{T } \text{e}_2))$

=

$(\text{PROGN } \text{a}_1 (\text{COND}(\text{a}_2 \text{ e}_1)(\text{T } \text{e}_2)))$

(The case without e_1 is permitted, too.)

(e5) COND-to-AND-reduction: $(\text{COND}(\text{p } \text{e})) = (\text{AND } \text{p } \text{e})$

We drop the remainder of axioms for conditional expressions (6 of them) and 3 additional axioms related to PROG1 resp. PROG2.

An example correctness proof. We give now a proof of rule (R7-1). We assume to have proven that an expression of this kind after transformation rule (R6) may occur only as PROG-statement-form. Moreover, for every clause: $\text{form}_{ik_1} = \$Sj$. We prove by induction on the number of clauses.

Let $r=1$:

$(\$COND i (\$COND ((\$FCOND 1 j \text{form}_1 \dots \text{form}_k \$Sj) e_{11} \dots e_{1n}) (T e_1 \dots e_s)))$

this is equivalent (by renaming, axioms (c1), (c1), (d2), (c1)) to:

$(\text{COND} ((\text{PROGN } \text{form}_1 \dots \text{form}_k \$Sj) e_{11} \dots e_{1n}) (T e_1 \dots e_s))$

this is equivalent (by axioms (c2), (e4), (d2), (d2)) to:

$\text{form}_1 \dots \text{form}_k (\text{COND} (\$Sj e_{11} \dots e_{1n}) (T e_1 \dots e_s))$

this is equivalent (by axioms (e1), (e1), (d10), (e1)) to:

$\text{form}_1 \dots \text{form}_k (\text{COND} (\$Sj (\text{PROGN} (\text{PROGN} e_{11} \dots e_{1n}) (\text{GO} \$Ci))) (\text{COND} (T (\text{PROGN} e_1 \dots e_s)) \$Ci))$

this is equivalent (by axioms (c2), (e3), (d10), (a3)) to:

$\text{form}_1 \dots \text{form}_k (\text{COND} ((\text{NOT} \$Si) \text{NIL} (\text{GO} \$CiL1))) e_{11} \dots e_{1n} (\text{GO} \$Ci) \$CiL1 (\text{COND} (T (\text{PROGN} e_1 \dots e_s))) \Ci

this is equivalent (by axioms (d2), (a3), (d2), (e1)) to:

$\text{form}_1 \dots \text{form}_k (\text{COND} ((\text{NOT} \$Sj) (\text{PROGN} \text{NIL} (\text{GO} \$CiL1))) e_{11} \dots e_{1n} (\text{GO} \$Ci) \$CiL1 e_1 \dots e_s \Ci

this is equivalent (by axioms (c1), (a2), (d6)) to:

$\text{form}_1 \dots \text{form}_k (\text{OR} \$Sj (\text{GO} \$CiL1)) e_{11} \dots e_{1n} (\text{GO} \$Ci) \$CiL1 e_1 \dots e_s \Ci

and this is what we need.

We assume now the correctness is proven for $r=m$. We show it for $r=m+1$:

$(\$COND i (\$COND ((\$FCOND 1 j \text{form}_{11} \dots \text{form}_{1k_1} \$Sj) e_{11} \dots e_{1n_1}))$

$((\$FCOND 2 j \text{form}_{21} \dots \text{form}_{2k_2} \$Sj) e_{21} \dots e_{2n_2})$

...

$((\$FCOND m j \text{form}_{m1} \dots \text{form}_{mk_m} \$Sj) e_{m1} \dots e_{mn_m})$

$((\$FCOND m+1 j \text{form}_{m+11} \dots \text{form}_{m+1k_{m+1}} \$Sj) e_{m+11} \dots e_{m+1n_{m+1}})$

$(T e_1 \dots e_s))$

After applying axiom (e2) we may use the induction hypothesis to get:

$\text{form}_{11} \dots \text{form}_{1k_1} (\text{OR} \$Sj (\text{GO} \$CiL1)) e_{11} \dots e_{1n_1} (\text{GO} \$Ci) \$CiL1$

$(\text{COND} ((\$FCOND 2 j \text{form}_{21} \dots \text{form}_{2k_2} \$Sj) e_{21} \dots e_{2n_2})$

...

$((\$FCOND m j \text{form}_{m1} \dots \text{form}_{mk_m} \$Sj) e_{m1} \dots e_{mn_m})$

$((\$FCOND m+1 j \text{form}_{m+11} \dots \text{form}_{m+1k_{m+1}} \$Sj) e_{m+11} \dots e_{m+1n_{m+1}})$

$(T e_1 \dots e_s)) \$Ci$

Now we apply axiom (d2) and axiom (c1) - to reintroduce the $(\$COND i \dots)$ cover, re-

name the label \$CiL1 to \$CiL0 (by axioms (d8) and (d9)), change the numeric constants in each clause (2m-times application of axiom (c1)) and are able to apply the induction hypothesis again. We get as equivalent:

$\text{form}_{11} \dots \text{form}_{1k_1} (\text{OR } \$Sj(\text{GO } \$CiL0)) e_{11} \dots e_{1n_1} (\text{GO } \$Ci) \$CiL0$

$\text{form}_{21} \dots \text{form}_{2k_2} (\text{OR } \$Sj(\text{GO } \$CiL1)) e_{21} \dots e_{2n_2} (\text{GO } \$Ci) \$CiL1$

...

$\text{form}_{m1} \dots \text{form}_{mk_m} (\text{OR } \$Sj(\text{GO } \$CiLm)) e_{m1} \dots e_{mn_m} (\text{GO } \$Ci) \$CiLm$

$\text{form}_{m+11} \dots \text{form}_{m+1k_{m+1}} (\text{OR } \$Sj(\text{GO } \$CiLm+1)) e_{m+11} \dots e_{m+1n_{m+1}} (\text{GO } \$Ci) \$CiLm+1$

$e_1 \dots e_s \$Ci \Ci

This is very near to the required result. To clean up one has to drop the last label (axioms (d8),(d9)) and to rename m+1 labels (+ redirection of GO's) (again axioms (d8), (d9)). This completes the proof.

Notational remarks. Up to now we used almost only LISP that the notation could assumed to be evident. It should have been clear that strings as locvar or form_{ik} were used as syntactical variables for program parts describing the concrete program entity by its name. The lowercase letters in \$-symbols are not constituting parts of the symbol names but indicate only that in this position are digits which might be computed via evaluation of an expression. We have now to extend this by the definition that \$PROG,\$COND,\$FCOND,\$STAT and \$PSTAT are all different names for one functional object: for PROGN. \$SETQ is another name for SETQ.

Conclusion. We have demonstrated that for the language LISP the compilation process might be understood and performed by program transformations. This turned out to be a straightforward symbol manipulation process where deeper semantic or language dependend concepts are not needed. (A complete version can be found in (TB)). It can be shown that all local optimizations which are possible in assembly language (D) can be applied to programs resulting from the described transformation process (P0). A similar demonstration was successfully performed using FORTRAN.

Literature.

(BL) B.Buchberger,R.Loos : Algebraic Simplification.in:Computer Algebra - Symbolic and Algebraic Computation,Wien,New York 1982

(D) L.H.Masinter,L.P.Deutsch : Local optimizations in a compiler for stack-based LISP-machines. 1980 LISP conference.

(G-H) M.L.Griss,A.C.Hearn : A portable LISP compiler.Software - practice and experience,11,1981,p.541-605

(Jue) H.Jürgensen : Zur Übersetzbarkeit von Programmiersprachen.in:3.Fachtagung über Programmiersprachen,LNCS 7,Berlin etc. 1974

(PO) H.Stoyan : Peephole optimizations in the LISP-LAP.Manuscript.

Erlangen 1982

(TB) H.Stoyan : An approach to a theoretical basis for code generation in compilation of higher level programming languages.Manuscript.

Erlangen 1983

Implementing REDUCE on a Microcomputer

John Fitch

School of Mathematics

University of Bath

England

Summary

An implementation of Cambridge LISP for a Motorola 68000 based computer is described, and the implementation of the algebra system REDUCE hosted by it is considered. The resulting systems are compared against mainframes and a minicomputer running LISP and REDUCE. The assertion of this work is that a new style of personal algebra computer is now possible, which is both fast and reasonably priced, and as such provides a cheaper base system for algebra research than mainframes or LISP hardware.

Introduction

For many years REDUCE [1] has been the de facto standard for computer algebra, with a very wide range of implementations ranging from large mainframes such as Cray and IBM to the 'megaminis' such as the DEC VAX series. In a previous paper Fitch and Marti [2] described a REDUCE-like subset of rational polynomial algebra that was run on a Z80 CP/M computer. This paper is concerned with a full implementation of REDUCE on one of the new generation of microprocessors, the Motorola 68000. With this computer it is possible to have a personal algebra computer with all the power and functionality of the last decade's central machines.

The prerequisite for REDUCE is LISP, and in particular a system that conforms to the Standard LISP Report [3]. For this reason this paper precedes the description and measurement of REDUCE with a summary of the main features of the particular M68000 computer and its operating system on which the LISP is built, and a discussion of the LISP system (dubbed Cambridge M68000 Lisp). The paper concludes by comparing the overall LISP/REDUCE implementation with some current mainframe systems.

M68000 Darkstar Computer and TRIPOS Operating System

The work of this paper has all been done on a prototype of a commercial M68000 computer, codenamed Darkstar [4]. The hardware was designed by the School of Electrical Engineering at the University of Bath, and features 512Kbytes of error detecting and correcting memory, diagnostic front panel, processor running at effectively 5MHz, disk controllers and floating point hardware. The prototype had 10 Mbytes of Winchester disk, but the system also runs with 8" floppy disks. The salient points here are the provision of an adequate quantity of main memory, and the availability of disks. While the LISP system makes use of the floating point hardware, it is of only passing interest for computer algebra. There is no significant dependence on any particular features of the Darkstar.

We run the operating system TRIPOS, which is a real time, single user, multi tasking operating system that was developed at the University of Cambridge [5]. This provides many facilities, but the two features on which the LISP depends are the provision of a secure filing system and the language BCPL [6]. In fact the operating system is written almost entirely in BCPL and the general computing environment that TRIPOS provides is that of BCPL. This point has been of great importance in the design of the LISP.

The TRIPOS operating system treats the Motorola 68000 as a 32 bit computer throughout, and this has meant that it has not been necessary to find ways round the restrictions of 16 bit addressing. In fact the 24 bit addressing of the hardware was one of the main attractions of this processor, as will be seen later.

Cambridge M68000 Lisp

Fitch and Norman [7] described an implementation of LISP for the IBM370 architecture that was based on the 24 bit addressing of the machine and used the remaining 8 bits of a word as a tag indicating type. By careful choice of tags it was possible to make the commonest predicates of LISP into simple tests, which opened the way to efficient inline coding in compiled code. This LISP was written totally in BCPL, but with no intention that it should be portable. The great similarities between the IBM and Motorola address structures suggested that it was worth an attempt to implement LISP with a common code base.

In the event nearly all the code has been kept the same. In particular the basic functions and the infinite precision rational arithmetic of Cambridge LISP has been unaltered, with great savings in programmer time for the new LISP. However there were

notable areas where the systems diverged.

The first stage in implementing Cambridge LISP was to produce a modified BCPL compiler that knew about CAR and CDR operators, as well as the logical comparison operators. The full reasons for these problems can be found in [7]. In the IBM Cambridge LISP it was possible to modify the BCPL code generator to free a register to hold the value of NIL which is of great significance in obtaining the speed of that system. In the Motorola system the register allocation of the operating system, together with the non regularity between address registers and data registers, has meant that it has not been possible to make such a satisfactory arrangement. Although there is still scope for improvement in the treatment of registers, it has not yet seemed appropriate to make wholesale changes in the operating system for the sake of an improvement of a few percent in run time.

The major changes in the base system have been as a consequence of the different I/O structure of the two computers. The first and most obvious difference is that the Cambridge system was designed for the batch-like environment of OS-MVT, and interaction with the program was not considered. The TRIPOS world is one where direct user interaction is assumed. The first and simplest change which was inspired by this direct access was to make all identifiers lowercase to ease typing. This simple change has proved to be one of the most controversial, and indeed a departure from the Standard LISP report [3]. The provision of a *lower flag goes some way towards alleviating this.

In the IBM system the use of DDnames for access to files greatly simplifies the naming of external data. In TRIPOS there is no equivalent, so the functions open and close use file names. This has given rise to a class of problems that are reminiscent of those of using TOPS-10 software under TOPS-20. The system has the assumption built in at a deep level that eight characters are always sufficient to name a file. In TRIPOS file names can be much longer, and the multi level directory structure further compounds the problem. At present the length has been relaxed to eleven characters, but a major redesign will be necessary before the file handling can be considered satisfactory. A similar problem is that the IBM system made special provision for partitioned data sets, and access to members of them. These have been mapped into directories, but again this is not totally satisfactory.

Although it is not apparent to the user, important changes have had to be made in the reading of data from the disk. The IBM operating system is firmly based on records, and as a result there was a great deal of use of eighty character records internally. TRIPOS uses a stream of characters in the same places, and this has led to significant time improvements when larger buffers were read or written. One place where fixed length

records have been retained is in the file of error messages, where direct access methods have been used so the user will get the error messages reasonably fast.

Cambridge LISP on the IBM uses a partitioned dataset to hold a core image that is loaded with the initial system, and also holds modules for load-on-call. This dataset has been translated into a directory called the image. One of the restrictions in the use of partitioned datasets is that one cannot be both reading and writing the same one, so if the core is to be preserved at the end of a run, or new modules created, they must be placed in a separate PDS (the dump). For extra safety this method of working has been preserved, but a more direct method of using the same directory for both new and old has become the norm. In this case the previous core image is retained after renaming. In an interactive system the delay in copying image to dump is unacceptable.

The only other areas of change in the base system have been in the garbage collector and error recovery. The garbage collector has to be able to read the BCPL stack, and unfortunately there are a few differences in the structure between the IBM and the M68000. Also for speed parts of the garbage collector have been coded in assembler. The method of garbage collection remains that described by Fitch and Norman [8]. Error recovery is still not fully integrated into the interactive mode of work, but the largest problem here has been the need to handle user breaks. As TRIPPOS is a real time system it is possible to interrupt LISP by a higher priority task, but there is no simple way to trap user break signals. A check has been placed in the eval loop, and a trap for the TRIPPOS task kill signal, but there are still occasions when these are not sufficient. It is hoped that this can be improved.

The LISP Compiler

The one part of the original Cambridge LISP that was inappropriate for the Motorola 68000 system was the compiler and its support. At the very least it was necessary to change the orders that it produced. The IBM system uses a variation of the Hearn and Griss portable LISP compiler [9], and it was that that was adapted for the system described here. The portable compiler divides into two sections, compilation and expansion of macros into binary for the target computer. In the case of the Cambridge family of LISPs the code generated must be compatible with the host BCPL code. This extends to register allocation and stack format. It is essential that the binary programs produced by LISP should look like BCPL in almost all respects. As the BCPL implementations are rather different, there was little opportunity to re-use code.

The M68000 BCPL uses the data registers and the stack to transfer arguments to functions, and the result is always returned in D1, while D0 is used to give the size of the stackframe to be saved during a function call. All the address registers are used for some purpose, but in particular function entry and exit is achieved via fixed code pointed at by registers. The requirements of LISP function entry are somewhat more complex than those for BCPL. As is explained in Fitch and Norman [7], the system uses a single integrated heap, and garbage collection occurs whenever the top of the stack gets too close to the fringe of the heap (coming down from high memory). It is the responsibility of the function linkage code to check for space exhaustion and to call the garbage collector. Also it is basic to the design of Cambridge LISP that there are no 'fast links', and all functions not treated specially by the compiler can be redefined with immediate effect and with recovery of the binary program or list space. For this reason when a function (foo say) is called with one argument from compiled code, it must be checked that foo expects only one argument, and is compiled. If there is a mis-match then the standard interpreter is invoked. As there are many different types of function in Cambridge LISP there have to be versions of this entry code for each one. For that reason whereas BCPL uses one simple entry sequence, LISP has an area of fixed code, which is addressed relative to the entry register, to handle the entry sequences. A sample of the Motorola entry is given in figure 1. It has not been possible to get the high efficiency of the IBM entry sequence, but time measurements have indicated that the M68000 is running about a twentyfifth of the speed of the IBM system on a 3081.

The Cambridge system also needs other fixed code, for example to allow CAR and CDR legality checking in compiled code, and to provide fluid binding and other utilities.

REDUCE

As the system here described is so similar to the Cambridge IBM system it was sufficient to use the IBM sources of REDUCE (1979 version) and their package for adapting the dialect of LISP. Apart from minor difficulties involving a simple compiler bug, the omission of a few functions from the base system and the handling of floating point, the implementation of REDUCE was the simplest sort of computing task, consisting mainly of waiting for rather slow disks and watching the output. The implementation of REDUCE on the Motorola 68000 has taken a little under six months of my part time effort.

REDUCE is divided into a number of modules which are loaded on demand. This increases execution time but allows the system to run in 400 Kbytes. Increasing the memory of the machine by 256 Kbytes considerably relieves this situation.

Measuring LISP and REDUCE

There is a standard test and example program that is distributed with REDUCE, and that has been used for the major measurement. As there are no standard tests for LISP some of those that feature in the PSL Newsletters have been adapted [10]. The results are summarized in figure 2. As can be seen the REDUCE system is a respectable implementation while not being outstandingly fast. There is scope for a number of optimisations in the compiler, in particular in the keeping of NIL in a register for more of the time. While this is designed it has not yet seemed necessary to spend the time doing it. Similarly there are still parts of the garbage collector that would repay coding in assembler.

In measuring the LISP alone it is hard to consider just speed. This LISP was designed to have a highly checked interpreter as well as providing a range of extra features not normally found in any LISP system. On the Darkstar with 512Kbytes LISP regularly runs in 420Kbytes, of which about half is taken up with the base system and module handles on the compiler, prettyprinter and LISP-based reader. It has been possible to provide a severely cut-down version providing only Standard LISP functions and only small (24 bit) numbers that runs on a 256Kbyte system. We have found that on a 768Kbyte machine there are no store problems in the application areas so far attempted.

Conclusions

REDUCE is the widest used algebraic system, and with this implementation it should be possible to take algebra closer to the user, to his own personal workstation. Although this work has used a little known operating system (at least outside Cambridge) it does show the power of the M68000 processor when used as a 32 bit computer. When the 32 bit external bus version of this chip is available running at 12 MHz we should see at least a fourfold improvement on the times given here.

It should also be stressed that this is a full implementation of REDUCE, including the high energy physics package. It is hoped that the integration code [11] will be added soon. There is another REDUCE implementation for the M68000 due to Griss [12], which uses a larger memory machine. His speeds are about twice those of this system, due in part to the speed of the memory. The only other REDUCE implementation for a personal machine is that of Marti for the Z80 [13], which while showing remarkable power is not a full system.

In addition the underlying LISP system can form the basis of a cheaper personal LISP machine. We have already been using the LISP in research work for a parallel executing LISP machine [14], and expect shortly to implement some natural language and machine translation systems using it. It is to be hoped that this is only the beginning of a new style of computer algebra.

References

1. A C Hearn
"REDUCE Users' Manual"
University of Utah, 1975
2. J P Fitch & J Marti
"NLARGEing a Z80 microprocessor"
Proceedings of EUROCAM 82, Marseilles, Lecture Notes in Computer Science 144, pp249-255 (1982)
3. J Marti, A C Hearn, C Griss & M L Griss
"Standard LISP Report"
SIGSAM Bulletin 14 (1), pp23-43 (1980) & SIGPLAN Notices 14, pp48-68 (1979)
4. Sirius Microtech Systems
Darkstar Specification
5. M Richards et al.
"TRIPOS - A Portable Operating System for Mini-computers"
Software - Practice and Experience 9, pp513-526 (1979)
6. M Richards
"BCPL, A tool for compiler writing & system programming"
Proc. SJCC 34, pp557-566 (1969)
7. J P Fitch & A C Norman
"Implementing LISP in a High Level Language"
Software - Practice and Experience 7, pp713-25 (1977)
8. J P Fitch & A C Norman
"A Note on Compacting Garbage Collection"
Computer Journal 21 pp31-4 (1979)
9. A C Hearn & M L Griss
"The Portable LISP Compiler"
Software - Practice and Experience 11, pp541-605 (1980)
10. M L Griss
PSL Newsletters, University of Utah, 1980-83
11. J P Fitch
"User Based Integration Software"
Proceedings of SYMSAC 81, Snowbird, Utah, pp245-248 (1981)
12. M L Griss
Private communications
13. J Marti
Private communications
14. J P Fitch & J Marti
"The Bath Concurrent LISP Machine"
These proceedings

LISP Entry Sequence (with BCPL parts starred)

CALL2	CMPI.B	#tag,(B)	Check tag is as expected	
	BNE.S	CALLAPPLY2	if not go via APPLY/EVAL	
	MOVEA.L	(B),B	Pick up value cell of id	
	MOVEA.L	(SP)+.L	Recover link	**
	MOVEM.L	P/L/B,-12(P,D0,L)	Make the stack frame	**
	ADDA.L	D0,P	Move the stack pointer up	**
	MOVEM.L	D1/D2,(P)	Dump arguments on stack	**
	CMPA.L	G_FRINGE(G),P	Out of store for call?	
	BLE.S	GOGO	if not we can continue	
		Call garbage collector	
GOGO	JMP	(B)	Enter the function	**

Figure 1: LISP Entry sequence

Comparative Timings

	(1)	(2)	(3)	(4)	(5)
LISP					
Empty 10000	120	3	-	51	43
Slow empty 10000	1640	83	1750	1037	449
Reverse 10 length 3000	3760	87	796	-	1360
Length 100 length 3000	8340	476	2273	-	1812
Arithmetic 10000	21600	792	21827	2023	5080
Eval 10000	27180	772	6057	12806	3305
Vector 50	4840	-	-	516	-
Slow vector 50	-	1095	-	7391	-
REDUCE					
(1+x)**10	1220	47	46	-	416
Standard	235750	8630	8413	-	-

(1) Darkstar M68000, Cambridge LISP

(2) IBM 3081, Cambridge LISP

(3) IBM 3081, SLISP/360

(4) VAX 11/750, PSL (taken from PSL Newsletter)

(5) Honeywell Multics, MACLISP

All times in milliseconds

Figure 2: Timings of LISP and REDUCE

A NOTE ON THE COMPLEXITY OF CONSTRUCTING GRÖBNER-BASES

R.Buchberger

Mathematisches Institut
Universität Linz
A4040 LINZ, Austria

ABSTRACT

In the bivariate case, upper bounds for the degrees and the number of polynomials occurring in Gröbner-bases of polynomial ideals are given. In the case of the total degree ordering of monomials, the upper bound for the degrees is linear in the maximal degree of the polynomials in the given basis of the ideal. In the general case, the upper bound for the degrees is quadratic. The upper bound for the number of polynomials is linear in the minimal degree of the polynomials in the given basis. All the bounds are shown to be tight. The relevance of these bounds for constructive polynomial ideal theory is indicated.

Acknowledgement

This work was sponsored by the Austrian "Fonds zur Förderung der wissenschaftlichen Forschung" (Project Nr. 4567). Special thanks to D. Lazard who made a number of extremely valuable comments on the content of the paper (see below).

BACKGROUND AND MOTIVATION

The concept of Gröbner-bases for polynomial ideals has been introduced implicitly in /Buchberger 65, 70/ (where an algorithm for constructing such bases was established) and explicitly in /Buchberger 76a/. Gröbner-bases have since been treated in a number of papers, see /Buchberger, Loos 82, Section 12/ for a fairly complete bibliography and a brief introduction into the subject. We assume here that the reader is familiar with the concept, motivation, algorithmic construction and application of Gröbner-bases. (Also /Buchberger 79/ seems to be suitable as an easy introduction).

In the multivariate case, little is known about the complexity of constructing Gröbner-bases (and of similar algorithms). In the univariate case, the Gröbner-basis algorithm specializes to Euclid's algorithm whose complexity has been extensively studied (see /Loos 82/ for a survey). In the general case we have the complexity results of /Hermann 26/ for the degrees of polynomials in her standard bases (Hermann-bases, see also /Seidenberg 74/) and the recent results of /Cardoza, Lipton, Meyer 76/ and /Mayr, Meyer 81/ on the intrinsic space complexity of solving the uniform word problem for commutative semigroups (which is a special case of deciding membership for polynomial ideals given by bases). These results indicate that the construction of standard bases for polynomial ideals, intrinsically, is an exponential problem (if n , the number of variables, enters the complexity considerations).

Still, for practical applications it is important to investigate the complexity of constructing standard bases for fixed $n > 2$ and to establish bounds that are as tight as possible (and to improve the algorithms). In this paper, we consider $n = 2$. The Gröbner-basis algorithm is of particular interest in this case for practical applica-

tions, see /Guiver 82/ and /Sakata 81/. Also, tight bounds for the complexity (in particular for the degrees of the polynomials) of Gröbnerbases are interesting for a theoretical reason: the connection between resultants and Gröbner-bases is still not very well understood. There were two attempts to combine the use of resultants and of the reductions used in the Gröbner-basis algorithm in order to improve computations: /Schaller 79/ and /Pohst, Yun 81/. However, these computational considerations did not add to an understanding of the fundamental connection between the two concepts.

Recently, /Bayer 82/ has given an interesting approach to bringing the concepts together: Euclid's algorithm, Gauss' algorithm and the Gröbner-basis algorithm are viewed as special procedures of computing certain determinants ("resultants") in the univariate, linear multivariate, and general case, respectively. The present author intends to give a more specific connection between the concepts along the same lines in a future paper, by connecting one single matrix with a system of polynomials whose "normal form" by elementary row and column transformation answers questions about the solvability of the system and similar questions and whose transformation into normal form can be viewed as an application of the Gröbner-basis algorithm. The practicability of this approach heavily depends on the availability of (realistic) bounds for the degrees of the polynomials in Gröbner-bases. Essentially the same approach is pursued in /Lazard 83/.

This paper extends a result given in /Buchberger 79/ (an upper bound for the degrees of the polynomials in bivariate Gröbner-bases) to its "nearly" best possible form and gives some new results. D. Lazard meanwhile was able to totally fill the gaps between lower and upper bounds left open in this paper. We indicate his results in parentheses. The proofs of his results appear in /Lazard 83/, these proceedings. Still, we think that our proof methods, which are totally distinct from the algebraic geometry approach of Lazard, may present some interest in themselves and, furthermore, the combination of the methods may yield some new insights. Also, our Theorem 1 holds for all polynomials occurring during the execution of the Gröbner-basis algorithm whereas Lazard's version, I think, can be asserted for the final outcome of the algorithm only.

NOTATION

Let K be an arbitrary field. By $K[x,y]$ we denote the ring of bivariate polynomials over K and by $[x,y]$ the set of bivariate monomials. An "admissible" ordering of the monomials is a linear ordering \ll on $[x,y]$ satisfying:

$$t \ll t \quad (\text{for all monomials } t \neq 1) \quad \text{and}$$

$$s \ll t \Rightarrow s.u \ll t.u \quad (\text{for all monomials } s, t, u).$$

For $f \in K[x,y]$, $\text{LC}(f)$ and $\text{LM}(f)$ denote the leading coefficient and the leading monomial of f w.r.t. \ll (if \ll is clear from the context), $D(f)$ is the degree of f . For $s, t \in [x,y]$, $E^i(t)$ denotes the exponent of t at the i -th variable ($i=1,2$), and $\text{LCM}(s,t)$ is the least common multiple of s and t . We say that s divides t (t is a multiple of s) iff $E^1(s) < E^1(t)$ and $E^2(s) < E^2(t)$. For $F \subseteq K[x,y]$, $\text{ID}(F)$ is the ideal generated by F , $\text{MAXD}(F) = \max\{D(f) / f \in F\}$, $\text{MIND}(F) = \min\{D(f) / f \in F\}$. $\text{ID}(F)$ is zero-dimensional iff F , viewed as a system of algebraic equations, has only finitely many

solutions (see any text on algebraic geometry). G is a minimal Gröbner-basis for F iff G is a Gröbner-basis for F and deleting a polynomial in G destroys the property of being a Gröbner-basis for F ; a minimal Gröbner-basis is reduced iff all the polynomials in G are reduced w.r.t. the other polynomials in G (see /Buchberger 76b/).

Examples: The "total degree" ordering and the "purely lexicographical" ordering are admissible orderings. $\{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, \dots\}$ and $\{1, x, x^2, x^3, \dots, y, xy, x^2y, \dots, y^2, xy^2, x^2y^2, \dots\}$ is the set $[x, y]$ enumerated in the total degree and purely lexicographic ordering, respectively. Let $f := 5x^5y + 3xy^2 - xy$. Then $LC(f) = 5$, $LM(f) = x^5y$, if we use the total degree ordering, and $LC(f) = 3$, $LM(f) = xy^2$, if we use the purely lexicographical ordering. $D(f) = 6$. Let $s := x^5y$, $t := xy^2$, then $E^1(s) = 5$, $E^2(s) = 1$, $LCM(s, t) = x^5y^2$. s divides $LCM(s, t)$, but s does not divide t .

R E S U L T S

Theorem 1: Let F be a finite set of polynomials in $K[x, y]$. Then all the polynomials h occurring during the application of the Gröbner-basis algorithm to the input F (using total degree ordering of monomials), in particular the polynomials h in the final Gröbner-basis, satisfy

$$D(h) \leq 2 \cdot MAXD(F) \quad (\text{D. Lazard: } D(h) \leq 2 \cdot MAXD(F)-1).$$

(By the "Gröbner-basis algorithm" we mean the author's algorithm introduced in /Buchberger 65/ in the version described in /Buchberger 79/, where an "overlap-lemma" ("Criterion 2" and "Criterion 3" in /Buchberger 79/) is used in order to detect situations in which certain "critical pairs" (S-polynomials) need not be considered in the course of the algorithm.)

Corollary 1: Let F be a finite set of polynomials in $K[x, y]$ and G a minimal Gröbner-basis for F w.r.t. the total degree ordering of monomials. Then

$$MAXD(G) \leq 2 \cdot MAXD(F) (-1).$$

Corollary 2: Let F be a finite set of polynomials in $K[x, y]$ such that $ID(F)$ is zero-dimensional and let G be a minimal reduced Gröbner-basis for F w.r.t. an arbitrary admissible ordering of monomials. Then

$$MAXD(G) \leq 4 \cdot MAXD(F)^2 \quad (\text{D. Lazard: } MAXD(G) \leq MAXD(F)^2, \\ \text{condition on zero-dimensionality can be dropped}).$$

Proposition 1: For every natural number d there is an $F \subseteq K[x, y]$ with $d = MAXD(F)$ such that for all Gröbner-bases for F (w.r.t. the total degree ordering of monomials) $MAXD(G) > 2d - 1$.

Proposition 2: For every natural number d there is an $F \subseteq K[x, y]$ with $d = MAXD(F)$ such that for all Gröbner-bases for F (w.r.t. the purely lexicographical ordering of monomials)

$$MAXD(G) > d^2 - d + 1 \quad (\text{D. Lazard: } MAXD(G) > d^2).$$

Theorem 2: Let F be a finite set of polynomials in $K[x, y]$ and G a minimal Gröbner-basis for F w.r.t. an arbitrary admissible ordering of monomials. Then

$$\|G\| \leq MIND(LM(F)) + 1.$$

Proposition 3: For every natural number d there is an $F \subseteq K[x,y]$ with $d = \text{MIND}(\text{LM}(F))$ such that for all Gröbner-bases G for F (w.r.t. an arbitrary admissible ordering) $\|G\| \geq d + 1$.

Remarks:

The upper bound obtained in Theorem 1 should be compared with the following upper bound for the degrees of polynomials for Hermann-bases H for F (see /Hermann 26/):

$$\text{MAXD}(H) \leq \text{MAXD}(F) + \text{MAXD}(F)^2.$$

(The correction in /Seidenberg 74/ of Hermann's results indicates that Hermann's bound should even read $\text{MAXD}(F)^4$ in this case). The comparison gives some optimism that Hermann's upper bounds can be improved essentially also in the case $n \geq 3$.

Repeating the arguments in /Buchberger 79/, the following upper bound for the number of steps (in the uniform cost measure) for constructing a Gröbner-basis G for F may be obtained from Theorem 1:

$$3/2 \cdot (\|F\| + 2 \cdot (\text{MAXD}(F) + 2)^2)^4.$$

From the proof of Corollary 2 one sees, that for arbitrary (not necessarily reduced) minimal Gröbner-bases G for F one has

$$\text{MAXD}(\text{LM}(G)) \leq 4 \cdot \text{MAXD}(F)^2.$$

/Schaller 79/ showed that, in the case of the purely lexicographical ordering, the degrees of polynomials occurring in (the computation of) a Gröbner-basis G for bivariate F can be bounded by $2 \cdot \text{MAXD}(F)^2$. Proposition 2 shows that this bound can not be improved essentially. Rather, the possibility of quadratic growth is essentially connected with the purely lexicographic ordering. Thus, this ordering, though indispensable for elimination purposes (see /Trinks 78/), computationally may have disadvantages when compared with the total degree ordering (see Theorem 1).

R E M A R K S A B O U T T H E P R O O F S

The intuitions for the proofs of Theorem 1 and Theorem 2 can be obtained from a geometrical representation of the bivariate monomials in the plane ((x^i, y^j) is represented as the point with cartesian coordinates (i, j) , see /Buchberger 79/) and a clear understanding of the Gröbner-basis algorithm, in particular the use of the "overlap-lemma". The formal verification of the geometrical and algorithmic intuitions, however, is tedious. In this paper we, therefore, give only a sketch of the proofs. We even must omit the precise definition of some auxiliary notions and rather rely on an intuitive understanding of some of the expressions used. All the formal details are contained in the technical report /Buchberger 82/.

P R O O F O F T H E O R E M 1

1. Observation: From /Buchberger 79/, /Buchberger, Winkler 79/ we know that all the polynomials h occurring during the application of the Gröbner-basis algorithm to the input F (using total degree ordering) satisfy

$D(h) \leq M(F) + W(F)$, where

$M(F) := \max \{ D(LCM(LM(f_1), LM(f_2))) / f_1, f_2 \text{ are "essential" w.r.t. } F \}$,

$W(F) := \min \{ E^1(LM(f)) / f \in F \} + \min \{ E^2(LM(f)) / f \in F \}$.

A pair of polynomials f_1, f_2 in F is "essential" iff the consideration of its "critical pair" (S -polynomial) can not be ruled out by the "overlap-lemma". $W(F)$, the "width" of F , is a measure for the "area left over" by $LM(F)$ ($:= \{LM(f) / f \in F\}$).

2. Observation: A tedious formal proof shows that $M(F)$ can be represented in the following form

$M(F) = \max(MC(F), MAXD(F))$, where

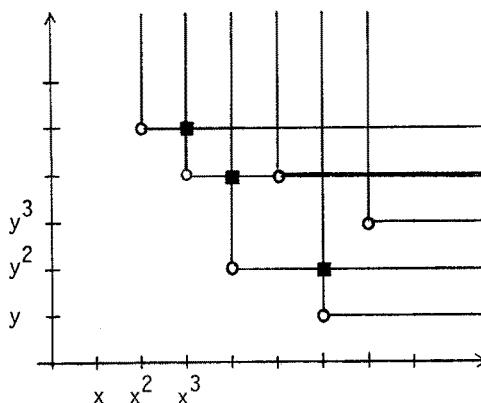
$MC(F) := \max \{ D(LCM(t_1, t_2)) /$

$t_1, t_2 \text{ are leading monomials of polynomials in } F,$

$t_1, t_2 \text{ lie on the "contour" of } F \text{ and}$

$\text{no other such monomial lies "between" them} \}$.

Graphically,



In this picture we see the leading monomials $x^6y, x^4y^2, x^3y^4, x^2y^5, x^7y^3, x^5y^4$ of a set F of polynomials. Only the first four of these monomials lie on the "contour". The least common multiples of neighbouring monomials on the contour are marked by a black square. From every point corresponding to a leading monomial two lines are drawn (one upwards and one to the right) embracing the area corresponding to monomials which are multiples of the given monomial. $MC(F)$ would be 8 in this example. (Formally, we call a finite set $T \subseteq [x,y]$ a "contour" iff there are no $s, t \in T$ such that $s \neq t$ and s divides t .)

3. Observation: One now proves $M(F) + W(F) \leq 2 \cdot MAXD(F)$. Essentially, the representation of $M(F)$ given above and the definition of $W(F)$ involve only the leading monomials of F . Also $MAXD(F) = MAXD(LM(F))$ (in the total degree ordering!). The proof of $M(F) + W(F) \leq 2 \cdot MAXD(F)$, therefore, can be carried out by establishing some lemmas on sets of bivariate monomials whose correctness and proof can easily be guessed from the geometrical interpretation. In more detail, we distinguish the cases $MC(F) \leq MAXD(F)$ and $MC(F) > MAXD(F)$.

In the first case we have

$$M(F) + W(F) = \text{MAXD}(F) + W(F) \stackrel{(1)}{\leq} \text{MAXD}(F) + \text{MAXD}(F) = 2 \cdot \text{MAXD}(F).$$

For (1) we need a lemma showing that, for arbitrary sets T of monomials, the width $W(T)$ is \leq the maximal degree $\text{MAXD}(T)$. The proof of this lemma is easy.

In the second case we have

$$(2) \quad M(F) + W(F) = MC(F) + W(F) = MC(F) + WC(F) \leq 2 \cdot \text{MAXDC}(F) \leq 2 \cdot \text{MAXD}(F).$$

Here, $WC(F)$ is the "width" of the set of leading monomials on the contour of $LM(F)$ and $\text{MAXDC}(F)$ is the maximal degree of these monomials. For (2) we need a lemma showing that the width of a set of monomials is determined by the monomials on the contour. The proof of this lemma is easy. For (3) we need a lemma showing that for contours T

$$MC(T) + WC(T) \leq 2 \cdot \text{MAXD}(T)$$

The proof of this lemma is a little more complicated, although still easy (draw a picture!)

PROOF OF COROLLARY 1

One minimal Gröbner-basis G for F w.r.t. the total degree ordering may be obtained by, first, computing a Gröbner-basis H for F (applying the Gröbner-basis algorithm to F) and then canceling polynomials in H whose leading monomials are multiples of other polynomials in H (see /Buchberger 76b/). By Theorem 1 we have $\text{MAXD}(H) \leq 2 \cdot \text{MAXD}(F)-1$ and, hence, $\text{MAXD}(G) \leq \text{MAXD}(H) \leq 2 \cdot \text{MAXD}(F)-1$. Furthermore, for arbitrary minimal Gröbner-bases G' , G'' for F: $LM(G') = LM(G'')$ (see /Buchberger 76b/) and, therefore $\text{MAXD}(G') = \text{MAXD}(G'')$. Hence, $\text{MAXD}(G) \leq 2 \cdot \text{MAXD}(F)-1$ for arbitrary minimal Gröbner-bases G for F.

PROOF OF COROLLARY 2

The residue class ring $V := K[x,y]/ID(F)$ is a vector space. If G is a Gröbner-basis for F w.r.t. an arbitrary admissible ordering R of monomials, then

$B_R := \{ C(t) \in [x,y] / t \text{ is in normal form w.r.t. } G \text{ (relative to R)} \}$, where $C(t)$ denotes the residue class of t w.r.t. $ID(F)$, is a linearly independent basis for V (see /Buchberger 65, 70/). In case F is zero-dimensional, V has finite vector space dimension. Every linearly independent basis for V, then, has the same number of elements. In particular, all sets B_R (for the different admissible orderings R) have the same number of elements. Now, let G be a minimal Gröbner-basis for F w.r.t. the total degree ordering R_0 and let F be zero-dimensional. By Corollary 1, (1) $\text{MAXD}(G) \leq 2 \cdot \text{MAXD}(F) =: d$.

Among the polynomials of G there must be two polynomials p and q such that $LM(p) = x^k$ and $LM(q) = y^l$ (otherwise B_{R_0} would be infinite, see /Buchberger 70/). Because of (1) we have

$$(2) \quad k, l \leq d.$$

The elements of B_{R_0} , therefore, can only consist of such $C(t)$, where t satisfies $E^1(t) \leq k, E^2(t) \leq l$. Hence,

$$(3) \quad |B_{R_0}| \leq k \cdot l \leq d^2.$$

Now we know that $|B_R| = |B_{R_0}|$ for arbitrary admissible orderings R and therefore

$$(4) \quad |B_R| \leq 4 \cdot \text{MAXD}(F)^2.$$

Assume now that G is a minimal reduced Gröbner-basis for F w.r.t. an arbitrary admissible ordering R and assume, furthermore, that $g \in G$ is such that $e := D(g) > 4 \cdot \text{MAXD}(F)^2$. Let s be a monomial occurring in g such that $D(s) = e$ and $i_1 = E^1(s)$, $i_2 = E^2(s)$, i.e. $e = i_1 + i_2$. Then for all monomials $t \mid s$ dividing s , t must be in B_R (otherwise g would not be part of a minimal reduced Gröbner-basis). There are $(i_1+1) \cdot (i_2+1) - 1 = (i_1+1) \cdot (e-i_1+1) - 1 = i_1 \cdot (e-i_1) + e > e > 4 \cdot \text{MAXD}(F)^2$, such monomials, i.e. by (4), there are more than $|B_R|$ such monomials, a contradiction!

P R O O F O F P R O P O S I T I O N 1

For $d \geq 3$ take

$$F := \{ \underline{xy^{d-1} \cdot x^d}, \underline{y^d} \} \quad (\text{the leading monomials are underlined}).$$

The Gröbner-basis algorithm yields

$$G' := \{ \underline{xy^{d-1} \cdot x^d}, \underline{y^d}, \underline{x^d y}, \underline{x^{2d-1}} \}.$$

G' is a minimal Gröbner-basis for F , $\text{MAXD}(G') = 2d-1$. If G is some other Gröbner-basis for F , then $\text{LM}(G)$ must contain all the underlined leading monomials in G' (see /Buchberger 76/). Hence, $\text{MAXD}(G) > \text{MAXD}(G') > 2d-1$ for arbitrary Gröbner-basis G for F . (For $d \leq 2$ it is easy to construct suitable examples F).

P R O O F O F P R O P O S I T I O N 2

We take Lazard's example

$$F := \{ \underline{y^d \cdot x}, \underline{y \cdot x^d} \}.$$

The Gröbner-basis algorithm yields $G' := \{ \underline{y \cdot x^d}, \underline{x^d \cdot x} \}$. G' is a minimal Gröbner-basis for F , $\text{MAXD}(G') = d^2$. By the same argument as above $\text{MAXD}(G) > \text{MAXD}(G') > d^2$ for all other Gröbner-bases G for F .

P R O O F O F T H E O R E M 2

(1) The basic structure of the proof is:

$$(2) \quad |G| = |\text{LM}(G)| \quad (3) \quad < \quad \text{MIND}(\text{LM}(G))+1 \quad (4) \quad < \quad \text{MIND}(\text{LM}(F))+1$$

↑ ↑ ↑

G is minimal LM(G) is G is a Gröbner-
 a contour basis for F

(2) For minimal Gröbner-bases G we know (see /Buchberger 76b/):

$$g_1, g_2 \in G, g_1 \neq g_2 \implies \text{LM}(g_1) \neq \text{LM}(g_2).$$

(3) In a minimal Gröbner-basis G , $\text{LM}(G)$ is a contour (see /Buchberger 76b/).

We need the following

Lemma: Let $M \subseteq [x, y]$, $t_0 \in M$. Then: M is a contour $\implies |M| < D(t_0)+1$.

Using this Lemma, we immediately obtain $|\text{LM}(G)| < \text{MIND}(\text{LM}(G))+1$.

(4) Since G is a Gröbner-basis, for all $f \in F$ there is a $g \in G$ such that $\text{LM}(g)$ divides $\text{LM}(f)$ (otherwise some $f \in F$ could not be reduced to 0 modulo G). Let f_0 be such that $D(\text{LM}(f_0)) = \text{MIND}(\text{LM}(F))$ and g_0 such that $\text{LM}(g_0)$ divides $\text{LM}(f_0)$. Then $\text{MIND}(\text{LM}(G)) < D(\text{LM}(g_0)) < D(\text{LM}(f_0)) = \text{MIND}(\text{LM}(F))$.

Again, the proof is reduced to the proof of a Lemma on certain sets of monomials, see

(3). Using the definitions

$$M_{1u} := \{ t \in M \mid E^1(t) \leq E^1(t_0), E^2(t) > E^2(t_0) \},$$

$$M_{u1} := \{ t \in M \mid E^1(t) > E^1(t_0), E^2(t) \leq E^2(t_0) \},$$

it is clear that M is the disjoint union of M_{1u} , $\{t_0\}$, and M_{u1} . Hence,

$$\|M\| = \|M_{1u}\| + 1 + \|M_{u1}\|.$$

Now,

$$(1) \|M_{1u}\| \leq E^1(t_0) \text{ and}$$

$$(2) \|M_{u1}\| \leq E^2(t_0).$$

From (1), (2) we get

$$\|M\| \leq E^1(t_0) + 1 + E^2(t_0) = D(t_0) + 1.$$

(1), (2) are easy, see /Buchberger 82/.

(Remark: The proof method used here is not applicable for $n \geq 3$. For example, starting from $t_0 := x^2yz$ we can define arbitrarily large contours M with $t_0 \in M$:

$$M := \{ x^2yz, x^t, x^{t-1}z, \dots, x^{t-1}z, z^t \} \quad (t \geq 4)$$

is a contour with $t+2$ elements.)

PROOF OF PROPOSITION 3

Take

$$F := \{ x^d, x^{d-1}y, \dots, xy^{d-1}, y^d \}.$$

F is a minimal Gröbner-basis (w.r.t. every admissible ordering). For any other Gröbner-basis G for F we have $\text{LM}(F) \subset \text{LM}(G)$ (see /Buchberger 76b/). Hence,

$$\|G\| \geq \|\text{LM}(G)\| \geq \|\text{LM}(F)\| = \|F\| = d+1.$$

References

Bayer, D.A., 82:

The Division Algorithm and the Hilbert Scheme. Harvard University,
Cambridge, Mass., Math. Deptmt: Ph.D. Thesis, June 1982.

Buchberger, B., 65:

An Algorithm for Finding a Basis for the Residue Class Ring of a
Zero-Dimensional Polynomial Ideal (German). Univ. of Innsbruck, Austria:
Math. Inst., Ph.D. Thesis 1965.

Buchberger, B., 70:

An Algorithmical Criterion for the Solvability of Algebraic Systems of
Equations (German). Aequationes mathematicae 4/3, 374-383 (1970).

Buchberger, B., 76a:

A Theoretical Basis for the Reduction of Polynomials to Canonical Form. ACM
SIGSAM Bull. 10/3, 19-29 (1976).

Buchberger, B., 76b:

Some Properties of Gröbner-Bases for Polynomial Ideals. ACM SIGSAM Bull.
10/4, 19-24 (1976).

Buchberger, B., 79:

A Criterion for Detecting Unnecessary Reductions in the Construction of
Gröbner-Bases. Proc. EUROSAM 1979, Marseille, Lect. Notes Comput. Sci. 72,
3-21 (1979).

Buchberger, B., 82:

Miscellaneous Results on Gröbner-Bases for Polynomial Ideals II. Technical Report, Dpmt. of Computer and Information Sciences, University of Delaware, to appear (1982).

Buchberger, B., Loos, R., 82:

Algebraic Simplification. In: Computer Algebra (B. Buchberger, G. Collins, R. Loos eds.), Springer, Wien-New York, 1982, 11-43.

Buchberger, B., Winkler, F., 79:

Miscellaneous Results on the Construction of Gröbner-Bases for Polynomial Ideals I. Technical Report Nr. 137, Mathematisches Institut, Universität Linz, Austria (1979).

Cardoza, E., Lipton, R., Meyer, A.R., 76:

Exponential Space Complete Problems for Petri Nets and Commutative Semigroups. Conf. Record of the 8th Annual ACM Symp. on Theory of Computing, 50-54 (1976).

Guiver, J.P., 82:

Contributions to Two-Dimensional System Theory. Univ. of Pittsburgh, Math. Depmt.: Ph.D. Thesis 1982.

Hermann, G., 26:

Die Frage der endlich vielen Schritte in der Theorie der Polynomideale. Math. Ann. 95, 736-788 (1926).

Lazard, D., 83:

Gröbner bases, Gaussian Elimination and Resolution of Systems of Algebraic Equations. These proceedings.

Loos, R., 82:

Generalized Polynomial Remainder Sequences. In: Computer Algebra (B. Buchberger, G. Collins, R. Loos eds.), Springer, Wien-New York, 1982, 115-137.

Mayr, E.W., Meyer, A.R., 81:

The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. M.I.T.: Lab. Comput. Sci. Rep. LCS/TM-199 (1981).

Pohst, M., Yun, D.Y.Y., 81:

On Solving Systems of Algebraic Equations Via Ideal Bases and Elimination Theory SYMSAC 1981, 206-211.

Sakata, S., 81:

On Determining the Independent Point Set for Doubly Periodic Arrays and Encoding Two-Dimensional Cyclic Codes and Their Duals. IEEE Trans. on Information Theory, IT-27/5, 556-565.

Schaller, S., 79:

Algorithmic Aspects of Polynomial Residue Class Rings. University of Wisconsin, Madison: Ph.D. Thesis, Comput. Sci. Tech. Rep. 370, 1979.

Seidenberg, A., 74:

Constructions in Algebra. Trans. AMS 197, 273-313 (1974).

Trinks, W., 78:

On B. Buchberger's method of Solving Algebraic Equations (German). J. Number Theory 10/4, 475-488 (1978).

..
GRÖBNER BASES, GAUSSIAN ELIMINATION AND
RESOLUTION OF SYSTEMS OF ALGEBRAIC EQUATIONS

D. Lazard
Mathématiques - Informatique
Université de Poitiers
F 86022 Poitiers Cedex

In the past few years, two very different methods have been developed for solving systems of algebraic equations : the method of Gröbner bases or standard bases [Buc 1, Buc 2, Tri, P.Y] and the one which I presented in Eurosam 79 [Laz 2, Laz 3] based on gaussian elimination in some matrices.

Although they look very different, they are, in fact, very similar, at least if we restrict ourselves to the first step of my method.

On the other hand Gröbner base algorithms are very close to the tangent cone algorithm of Mora [Mor].

All of these algorithms are related to Gaussian elimination.

In the first part of this paper, we try to develop all these relations and to show that this leads to improvements in some of these algorithms.

In the second part we give upper and lower bounds for the degrees of the elements of a Gröbner base. These bounds are based on projective algebraic geometry. The choice of the ordering appears to be critical : lexicographical orderings give Gröbner bases of high degree, while reverse lexicographical orderings lead to low degrees.

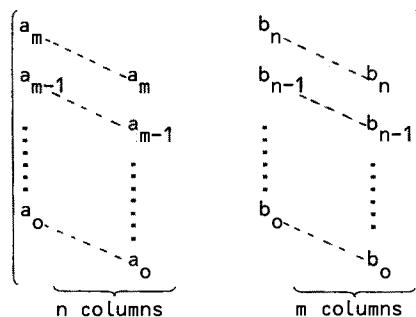
I - RESULTANT

To understand the above similarity, it is useful to begin with the simplest case of two univariate polynomials

$$\begin{aligned} A &= a_0 + a_1 X + \dots + a_m X^m & a_m &\neq 0 \\ B &= b_0 + b_1 X + \dots + b_n X^n & b_n &\neq 0 \end{aligned}$$

Here the standard method for computing Gröbner bases is exactly the Euclidean algorithm for computing GCDs and resultants.

The second way to compute the resultant consists of evaluating the Sylvester determinant



For this purpose, Gaussian elimination seems to have $O((m+n)^3)$ complexity, but a simple modification of it gives $O(mn)$ complexity and an algorithm strictly equivalent to Euclid's one :

The Gaussian method consists of subtracting b_n/a_m times the first column from the $(n+1)$ -th one. If $n \geq m$, the same computations allow us to subtract b_n/a_m times the i -th column from the $(n+i)$ -th for $i = 1, \dots, m$; this operation amounts to replacing the coefficients of B by those of $C := B - (b_n/a_m)X^{n-m}A$. Thus, the Sylvester determinant of A and B is reduced to a_m times the Sylvester determinant of A and C .

If we iterate this process, we simulate Euclid's algorithm in the reduction of a Sylvester matrix. On the other hand, Euclid's algorithm can be viewed as a way to save memory (and time) in the reduction of a Sylvester matrix.

Some years ago, I asked for such an economical method for elimination theory [Laz 1]; it appears that Gröbner base computation provides a solution.

II - GROBNER BASES AS INFINITE LINEAR BASES

Let $A = K[X_1, \dots, X_n]$ be a polynomial ring over a field K and $I = (f_1, \dots, f_k)$ an ideal generated by a finite set of polynomials f_1, \dots, f_k . As a K -vector space, I is generated by the mf_i 's where $i = 1, \dots, k$ and m runs through all the monomials $X_1^{a_1} \dots X_n^{a_n}$. On the base of A consisting of all the monomials, this generating set defines a (infinite) matrix which has the following properties :

- 1/ The non-zero entries of each column are finite in number and consist of coefficients of one of the f_i 's.
- 2/ Each row has a finite number of non-zero entries : if the row corresponds to some monomial m , then the non-zero entries must correspond to generators $m'f_i$ where m' is a monomial dividing m .

This finiteness property allows one to triangulate the matrix by column operations. This is a finite process for each pivot, which enumerates a base of the vector space I . Although the whole enumeration is not finite, one can give a finite description of a linear base of I ; such a description is provided by Gröbner bases.

For this purpose, we choose any total order on the monomials of A , which is compatible with products (i.e. $m \leq n \Rightarrow mp \leq np$) and mean by the leading monomial of a polynomial P , the greatest monomial which appears in P with non-zero coefficient ; this we denote $\text{lead}(P)$.

It is easy to prove that f_1, \dots, f_n is a Gröbner base for the ideal I if and only if the following set of polynomials is a base of the vector space I :

$$\{mf_i ; 1 \leq i \leq k, m \text{ monomial, } \text{lead}(mf_i) \text{ not multiple of } \text{lead}(f_j) \text{ for } j < i\}$$

There are three classical orderings which are used for computing Gröbner bases : the purely lexicographical ordering

$$1 < x < x^2 < x^3 < \dots < y < xy < x^2y < \dots < y^2 < xy^2 < \dots,$$

the total degree ordering

$$1 < x < y < x^2 < xy < y^2 < x^3 < x^2y < \dots$$

which satisfies $m \leq n \Rightarrow \text{degree}(m) \leq \text{degree}(n)$,

the reverse degree ordering

$$1 > x > y > x^2 > xy > y^2 > x^3 > x^2y > \dots$$

which satisfies $m \geq n \Rightarrow \text{degree}(m) \leq \text{degree}(n)$.

The first two are well-orderings and often used. The last one is useful when dealing with formal power series or local singularities [Mor].

For the latter, the termination criterion is not easy (see [Mor]), but it is the only one for which Gröbner base computation and Gaussian elimination are related exactly as in section I : for example, in the univariate case, the Sylvester matrix becomes the infinite matrix

$$\begin{array}{ccccccccc} a_0 & 0 & \dots & 0 & \dots & \dots & & & \\ a_0 & - & & & & & & & \\ a_1 & - & a_0 & - & & & & & \\ \vdots & & \vdots & & & & & & \\ a_m & - & a_1 & - & a_0 & - & & & \\ 0 & 0 & - & a_1 & - & & & & \\ 0 & & \vdots & & & & & & \\ 0 & & & & & & & & \\ \vdots & & & & & & & & \\ & & & & & & & & \end{array} \quad \begin{array}{ccccccccc} b_0 & 0 & \dots & 0 & \dots & \dots & & & \\ b_0 & - & & & & & & & \\ b_n & - & b_0 & - & & & & & \\ 0 & b_n & - & b_0 & - & & & & \\ 0 & 0 & - & b_0 & - & & & & \\ 0 & & \vdots & & & & & & \\ 0 & & & & & & & & \\ \vdots & & & & & & & & \\ & & & & & & & & \end{array}$$

In the two first cases however, the ordering on monomials is not compatible with any numbering of the rows of the above infinite matrix. It follows that our analogy between Gaussian elimination and Gröbner base computation does not work any longer unless we restrict the computations to finite submatrices. We shall do this in the following section.

III - HOMOGENEIZATION

In the rest of the paper, we shall study homogeneous polynomials. This is not really a restriction because every polynomial can be made homogeneous by adding a new variable :

$$x^2 + y^2 + 2x + 1 \rightarrow x^2 + y^2 + 2xt + t^2 ,$$

the computations can be done on homogeneous polynomials and the desired inhomogeneous result can be retrieved by setting the new variable to 1.

Homogeneous polynomials are as useful as projective geometry. For our purposes, the advantages are :

- a/ high degree intermediate computations are not cancelled in the result ;
- b/ the infinite matrix of the preceding section will appear as a direct sum of finite matrices ; thus with an appropriate order on the rows corresponding to a given degree the analogy between Gaussian elimination and a Gröbner base algorithm can be pursued ;
- c/ in the homogeneous case each degree appears as a separate domain and we never need to compare monomials of different degrees ; we only need total orderings on the monomials of each total degree which are compatible with respect to multiplication, i.e. $m < m' \Rightarrow mm'' < m'm''$; the inequality $m < m'$, which is generally required in Gröbner base algorithms, is no longer necessary.

If t is the homogenizing variable, the first and third orderings of the last section become lexicographical orderings in each degree, with respective orders

$$\begin{aligned} t &< x < y \\ \text{or} \quad y &< x < t \end{aligned}$$

on the variables. On the other hand, the second one becomes the reverse lexicographical ordering relative to

$$t < x < y$$

The reverse lexicographical ordering is the ordering on exponents such that $(a_1, \dots, a_n) < (b_1, \dots, b_n) \Leftrightarrow \exists k, a_k > b_k \text{ and } a_i = b_i \text{ for } i > k$.

These remarks have many computational implications :

A/ Computation of the tangent cones

The first and third orderings become isomorphic. It follows that the analogy between Mora's algorithm for the tangent cone and Buchberger's one for Gröbner bases

may be pursued much further than is asserted in [Mor] : every algorithm for a Gröbner base can be used for the tangent cone when working with homogenized polynomials. Some experimentation is needed to decide if this remark actually improves Mora's algorithm.

B/ Resolution of algebraic systems

The characterization of Gröbner bases of the last section may be used to make the following.

Définition. Let I be an ideal generated by homogeneous polynomials, and I^d the set of homogeneous polynomials in I which are of degree d . A set $F = (f_1, \dots, f_k)$ of homogeneous polynomials is called a Gröbner base of I for degree d or Gröbner base of I^d if

{ mf_i , m monomial, $i \in \{1, \dots, k\}$, $\text{lead}(mf_i)$ not a multiple of $\text{lead}(f_j)$ for $j < i$, $\text{degree}(mf_i) = d\}$.
is a linear base for I^d .

The two following facts are straightforward.

1/ F is a Gröbner base if and only if it is a Gröbner base in every degree.

2/ Every algorithm which computes Gröbner bases yields an algorithm which computes Gröbner bases in degree d by cancelling all computations in degrees greater than d .

In view of these remarks the first step of the algorithm of [Laz 2, Laz 3] (called "Réduction de la matrice ϕ " or "Reduction of the numerical part") may be modified as follows.

(R1) Compute a Gröbner base in degree DD for the ideal generated by the input polynomials (after homogenization). Let (F_1, \dots, F_k) be the result.

(R2) Let NL be the number of monomials of degree DD and M be the matrix with NL rows, whose columns are the coefficients of the polynomials mf_i , $i=1, \dots, k$, such that m is a monomial, mf_i is of degree DD , $\text{lead}(mf_i)$ is not a multiple of $\text{lead}(F_j)$ for $j < i$. The matrix MM [Laz 2] or CC [Laz 3] is the matrix of relations between the rows of the (triangular) matrix M .

This modification shows the relationship between my algorithm which uses Gaussian reduction and Gröbner base algorithms. Some experimentation is needed to decide which method is the best.

C/ The choice of the ordering on monomial has many repercussions on the complexity of the computation of a Gröbner base :

With the lexicographical ordering, the element of the Gröbner base with minimal leading term depends on a small number of variables (at most one plus the dimension of the variety of the ideal) and has a high degree (in general the degree of the variety of the ideal which is essentially the product of the degrees of the generators of the ideal).

On the other hand, with a reverse lexicographical ordering, the element of the Gröbner base with minimal leading term depends on many variables and has small degrees except in the last variable.

IV - BOUNDS ON DEGREE

It is clear that the complexity of Gröbner base algorithms depends on the degrees of the elements of the Gröbner base.

The above remark shows that these degrees depend strongly on the choice of ordering, the extreme cases being lexicographical and reverse lexicographical orderings. These collapse for homogeneous polynomials in two variables.

Some notation is required : let I be the ideal of $K[X_1, \dots, X_n]$ generated by (f_1, \dots, f_k) , \tilde{I} the homogeneous ideal of $K[X_0, \dots, X_n]$ generated by the homogenized polynomials $\tilde{f}_1, \dots, \tilde{f}_k$ where \tilde{f}_i is the homogenized polynomial corresponding to f_i , i.e. $\tilde{f}_i = X_0^{d_i} f_i(\frac{X_1}{X_0}, \dots, \frac{X_n}{X_0})$ with $d_i = \text{degree}(f_i)$.

We call $\dim(I) \leq \text{Dim}(\tilde{I})$ the dimension of the algebraic set V (resp. \tilde{V}) of common zeros of the f_i 's in an algebraic closure of K (resp. of the common projective zeros of the \tilde{f}_i 's). It is the maximum of the i 's such that *most* linear varieties of dimension $n-i-1$ do not intersect the algebraic set : here and in the following *most* or *in general* will mean always except on a Zariski closed set. The degrees $\deg(I) \leq \text{Deg}(\tilde{I})$ are the number of points (with multiplicities) of intersection of this algebraic set with a linear variety of dimension $n-i$ in *general* position. The inequalities between $\deg(I)$ and $\text{Deg}(\tilde{I})$ or $\dim(I)$ and $\text{Dim}(\tilde{I})$ come from what happens at infinity ; *in general* they are equalities.

Finally, let d_1, \dots, d_k be the degrees of the f_i and choose the numbering so that $d_1 \geq d_2 \geq \dots$

Proposition 1. (Bezout's theorem) If $r = \dim(I)$ (resp. $\text{Dim}(\tilde{I})$) we have $\deg(I) \leq d_1 d_2 \dots d_{n-r}$ (resp. $\text{Deg}(\tilde{I}) \leq d_1 d_2 \dots d_{n-r}$). Equalities hold when $k = n-r$ (in general for $\deg(I)$ and always for $\text{Deg}(\tilde{I})$).

Proposition 2. The projection of the algebraic set V (resp. \tilde{V}) of dimension r into a linear variety of dimension $r+1$ is, in general, a hypersurface of degree $\deg(I)$ (resp. $\deg(\tilde{I})$).

Corollary 1. After most linear changes of variable the ideal I (resp. \tilde{I}) contains a polynomial of degree $\deg(I)$ (resp. $\text{Deg}(\tilde{I})$) depending only on $X_{n-r}, X_{n-r+1}, \dots, X_n$ where $r = \dim(I)$ (resp. $\text{Dim}(\tilde{I})$) and does not contain polynomials of degree less than $\deg(I)$ (resp. $\text{Deg}(\tilde{I})$) in these variables.

Proof. Project into the linear variety with equations $(x_0 =) x_1 = \dots = x_{n-r-1} = 0$. The polynomial is the equation of the hypersurface of proposition 2.

Theorem 1. Let I be an ideal of $K[x_1, \dots, x_n]$ generated by polynomials f_1, \dots, f_k of degrees d_1, \dots, d_k ($k \leq n$). In most cases, every Gröbner base for the lexicographical ordering or for the third ordering of section II contains a polynomial with degree $d_1 d_2 \dots d_k$.

This is an immediate consequence of the preceding considerations. It would be useful to prove that every element of a reduced minimal Gröbner base has a degree less than or equal to this bound. This can be done in the case where $\dim(I) = 0$ with the method of [Buc 3].

V - BOUNDS ON THE DEGREE. REVERSE LEXICOGRAPHICAL ORDERING

In the preceding section we have seen that lexicographical orderings lead to Gröbner bases of high degree. Here we show that reverse lexicographical orderings lead to better bounds. We shall work only in the homogeneous case, even if the results apply in the non homogeneous case. Thus let I be an ideal of $K[x_0, \dots, x_n]$ (K a field) generated by homogeneous polynomials f_1, \dots, f_k with degrees d_1, \dots, d_k such that $d_1 \geq d_2 \dots \geq d_k$.

We set $A := K[x_0, \dots, x_n]/I$ and denote by A^d the set of elements of A which are classes of homogeneous polynomials of degree d .

We shall use freely facts from commutative algebra which can be deduced an exercices from [Kap] (for example). See also [Laz 3, Laz 4].

Lemma 1. There exists an integer i_0 such that $\text{ann}(x) \cap A^i = 0$ for $i \geq i_0$ and for most elements x of A^1 (here $\text{ann}(x) = \{y \in A ; xy = 0\}$). If $k \leq n$ then $i_0 = 0$.

Conjecture 1. If $k > n$ we can take $i_0 = d_1 + \dots + d_{n+1} - n$.

Proposition 3. If $\dim(I) \leq 0$ or $n \leq 2$, the above conjecture is true.

The first assertion is a part of [Laz 3, th. 3.3] ; the case $n = 2$ follows from the fact that if $g = \text{GCD}(f_1, \dots, f_k)$, the ideal J generated by the f_i/g 's satisfies $\dim(J) \leq 0$.

Define $\text{depth}(A)$ recursively by 0 if there is no $x \in A^1$ such that $\text{ann}(x) = 0$, or $1 + \text{depth}(A/xA)$ if $x \in A^1$ and $\text{ann}(x) = 0$. If $\text{depth}(A) = \dim(I) + 1$, the ring A is Cohen-Macaulay (this may be a definition) ; if $k \leq n$, then $\text{depth}(A) \geq 1 + n - k$.

We are now able to state our main result.

Theorem 2. Suppose one of the following conditions holds :

- i) Conjecture 1 is true ;
- ii) $\text{depth}(A) \geq \text{Dim}(I)$;
- iii) $\text{depth}(A) \geq n-2$
- iv) $\text{Dim}(I) \leq 0$
- v) $n \leq 2$ ($n+1$ is the number of homogeneous variables).

Then, after most linear changes of variable, the elements of any minimal reduced Gröbner base for the reverse lexicographical ordering have degree at most $d_1 + \dots + d_{r+1} - r$ where $r = n - \text{depth}(A)$ (We have always $r < k$, the number of polynomials).

We need the following lemma.

Lemma 2. With the same hypothesis, after most linear changes of variable, if

y_0, \dots, y_n are the new variables and $s = \text{Dim}(I)$, then

a/ every monomial of degree $d_1 + \dots + d_{r+1} - r$ is congruent modulo I to an element of $y_{n-s} A + y_{n-s+1} A + \dots + y_n A$.

b/ If z is a homogeneous polynomial of degree $d > d_1 + \dots + d_{r+1} - r$ such that $z \in I \cap (y_{n-s} A + \dots + y_n A)$ then $z \in y_{n-s} I + \dots + y_n I$.

Proof of the theorem from the lemma : let $D = d_1 + \dots + d_{r+1} - r$; any monomial m of degree D not depending on y_{n-s}, \dots, y_n satisfies a relation $G_m := m - \sum_{i=n-s}^n y_i g_i \in I$ and is the leading term of this element of I (recall that the reverse lexicographical order is defined by

$$y_0^{a_0} \dots y_n^{a_n} < y_0^{b_0} \dots y_n^{b_n} \text{ iff } \exists i \quad a_i > b_i, \quad a_{i+1} = b_{i+1}, \dots, a_n = b_n$$

Let G a Gröbner base and G' the set of polynomials of G which are of degree at most D ; we shall prove that G' is also a Gröbner base : let $g \in I$ be a homogeneous polynomial of degree $d \geq D$; if its leading term does not depend on y_{n-s}, \dots, y_n , it is a multiple of the leading term of some G_m and thus of the leading term of some element of G' . If $g \in y_{n-s} A + \dots + y_n A$, its leading term is a multiple of the leading term of some element of I of degree $d-1$ (assertion b/); and an induction shows that in all cases, the leading terms of elements of I are multiples of the leading terms of elements of G' ; this fact implies that G' is a Gröbner base.

Proof of the lemma :

1/ If $\text{Dim}(I) \leq 0$: this is proved in [Laz 3, th. 3.3 and p. 106].

2/ If $n \leq 2$: dividing by the GCD of the generators of I we get an ideal J s.t. $\text{Dim}(J) = 0$; applying the lemma to J and multiplying by the GCD, gives the result.

3/ If $\text{depth}(A) > 0$, after most linear changes of variable, y_n is such that $\text{ann}(y_n) = 0$. Let $\bar{A} = A/y_n A$. We have the exact sequence

$$0 \rightarrow A^{d-1} \xrightarrow{y_n} A^d \rightarrow \bar{A}^d \rightarrow 0$$

where $\xrightarrow{y_n}$ is multiplication by y_n . Some elementary diagram chasing permits us to deduce the lemma for A from the lemma for \bar{A} and this proves ii) and iii) from iv) and v).

4/ If $\text{depth}(A) = 0$ and conjecture 1 is true : the same argument as above works because we do not need the injectivity of $\xrightarrow{y_n}$ for $d \leq d_1 + \dots + d_{n+1} - n$.

Conjecture 2. The conclusion of the theorem 2 is true even if conjecture 1 is not true.

Conjecture 3. The conclusion of the theorem 2 is true for all linear changes of variable i.e. without any change of variable.

Remarks.

1/ We have not really used the particular ordering we have chosen, but only the following two conditions :

$$m < m' \Rightarrow mm'' < m'm''$$

every monomial not containing y_{n-s}, \dots, y_n

is greater than every monomial containing some of these variables.

2/ The results apply to the non homogeneous case if the hyperplane at infinity is in a general position, and this gives not only bounds on the degrees of Gröbner bases, but also on the degrees of their expressions as linear combinations of the f_i .

3/ Our bounds are the best possible : take $f_1 = x_1^{d_1}$ and, for $i > 1$, $f_i = x_{i-1}^{d_i-1} x_i^{d_i}$.
We conclude with a result which does not need any reference to "general position".

Theorem 3. Let I be a (not necessarily homogeneous) ideal in $K[X_1, \dots, X_n]$ generated by f_1, \dots, f_k of degrees d_1, \dots, d_k s.t. $d_1 \geq d_2 \geq \dots \geq d_k$. Choose any ordering on the monomials such that $m \leq m' \Rightarrow (mm'' \leq m'm'' \text{ and } \text{degree}(m) \leq \text{deg}(m'))$. Suppose one of the following conditions holds :

i) $n \leq 2$;

ii) $\text{Dim}(\tilde{I}) \leq 0$ where \tilde{I} is the ideal generated by the \tilde{f}_i 's homogenized from the f_i 's.

Then the polynomials of every minimal reduced Gröbner base have degrees at most $d_1 + \dots + d_{n+1} - n + 1$ with $d_{n+1} = 1$ if $k = n$.

Proof. As above, dividing the f_i by their GCD reduces case i) to case ii). Let $\tilde{A} := K[X_0, \dots, X_n]/\tilde{I}$ and \tilde{A}^d be its homogeneous part of degree d ; the results of [Laz 3, p. 106] prove that there exists $\tilde{y} \in \tilde{A}^1$ s.t. the multiplication $\tilde{A}^{d-1} \xrightarrow{\tilde{y}} \tilde{A}^d$ is surjective for $d \geq d_1 + \dots + d_n - n + 1$.

Let $A = K[X_1, \dots, X_n]/I$ and A_d be the image in A of the set of polynomials of degree at most d . We have $A_d \subset A_{d+1}$ for every d . Substituting 1 for X_0 gives a surjection $\tilde{A}^d \rightarrow A_d$. If y is the image of \tilde{y} under this substitution, the commutative diagram

$$\begin{array}{ccc} \tilde{A}^{d-1} & \xrightarrow{\tilde{y}} & \tilde{A}^d \\ \downarrow & & \downarrow \\ A_{d-1} & \xrightarrow{y} & A_d \end{array}$$

shows that y is a surjection and $\dim_K A_d \leq \dim_K A_{d-1}$ for $d \geq d_1 + \dots + d_n - n + 1$. The inclusion $A_{d-1} \subset A_d$ thus gives equality and every monomial of degree $d_1 + \dots + d_n - n + 1$ is congruent modulo I to a polynomial of lower degree. It follows that the elements of degree at most $d_1 + \dots + d_n - n + 1$ of any reduced Gröbner base are also a Gröbner base by the same argument as in the proof of theorem 2.

Remark. Assertion i) of theorem 3 is a slight improvement of a result of Buchberger [Buc 3]. Theorems 2 and 3 proceed from the goal of unifying Buchberger's result with results of algebraic geometry [Laz 3] : the following theorem can be deduced from theorem 3.3 of [Laz 3] with the same reduction steps as in theorem 2.

Theorem 4. *With same hypotheses and notation as in theorem 2, let $P(d)$ be the Hilbert polynomial, i.e. the unique polynomial in d such that $P(d) = \dim_K A^d$ for d large enough. We have $P(d) = \dim_K A^d$ for $d \geq d_1 + \dots + d_{r+1} - n$.*

As in the case of theorem 2 we conjecture that the result is true even if conjecture 1 is not true.

REFERENCES

- [Bay] D.A. Bayer. The division algorithm and the Hilbert scheme, Ph D, Harvard Univ., 1982.
- [Buc 1] B. Buchberger. Ein algorithmissches Kriterium fur die Lösbarkeit eines algebraischen Gleichungssystems. Aequationes Matematicae 4 (1970), p. 374-383.
- [Buc 2] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of Gröbner bases, EUROSAM'79, Lect. Notes in Comp. Sc. n° 72 (1979), p. 3-21.
- [Buc 3] B. Buchberger. A note on the complexity of constructing Gröbner bases. These proceedings.
- [Kap] I. Kaplansky. Commutative rings. Allyn and Bacon (Boston, 1970).
- [Laz 1] D. Lazard. Algèbre linéaire sur $K[X_1, \dots, X_n]$ et élimination. Bull. Soc. Math. France 105 (1977), p. 165-190.
- [Laz 2] D. Lazard. Systems of algebraic equations. EUROSAM 79, p. 88-94.
- [Laz 3] D. Lazard. Résolution des systèmes d'équations algébriques. Theor. Comp. Sciences 15 (1981), p. 77-110.

- [Laz 4] D. Lazard. Commutative Algebra and Computer Algebra, EUROCAM'82, Lect. Notes in Comp. Sc. n° 144 (1982), p. 40-48.
- [Mor] F. Mora. An algorithm to compute the equations of tangent cones, EUROCAM'82, p. 158-165.
- [P.Y.Y.] M. Pohst, D.Y.Y. Yun. On solving systems of algebraic equations via ideal bases and elimination theory SYMSAC 1981, p. 206-211.
- [Tri] W. Trinks, Ueber B. Buchberger Verfahren, Systeme algebraischer Gleichungen zu lösen, J. of Number Theory 10 (1978), p. 475-488.

Appendix

During the conference, F. Mora gave me the following counter-example to conjecture 3:
 Every Gröbner basis of the prime ideal $(x^{n+1} - y^{n-1} z^T, xz^{n-1} - y^n, x^n y - z^n T)$ contains the polynomial $y^{n^2+1} - z^n T$ when assuming to have a reverse lexicographical ordering such that $x > z > y > T$.

THE COMPUTATION OF THE HILBERT FUNCTION

Ferdinando Mora
Istituto di Matematica
Università di Genova
Genova, Italy

H. Michael Möller
FB Mathematik und Informatik
FernUniversität Hagen
D 5800 Hagen 1, BRD

The recent development of Computer Algebra caused a renewal of interest in constructive methods in Commutative Algebra and Algebraic Geometry and made some methods like working with Gröbner bases (briefly: G-bases) to powerful computational tools.

Before most of the authors still interested in constructive methods (for instance Macaulay, Hermann, van der Waerden, Gröbner, partially Seidenberg, and more recently Renschuch) used the following approach. By working only with projective and non affine varieties, which means - from an algebraic point of view - to deal exclusively with homogeneous ideals, they were able to transform questions about ideals and modules into questions about vectorial spaces and then to solve them by simply solving linear systems [3, 4, 8]. A typical application of this method is the investigation of the Hilbert function $H(t, I)$, which measures the dimension of the vectorial space of all polynomials in an ideal I having a fixed degree t or at most degree t . This Hilbert function allows deeper insight in the structure of ideals, cf. R.P. Stanley [9], and can even be applied in interpolation theory, cf. H.M. Möller [6].

Published algorithms for Hilbert function computation [3, 4, 8] follow this approach and require (symbolic) computation of finite free resolutions. B. Buchberger applies in his thesis [1], in which he investigated also G-bases, a new tool to Hilbert function computation. Avoiding any symbolic calculations, we restate here the original algorithm of Buchberger and give a proof of it in terms of resolutions. We present two modifications of the algorithm, both of them requiring also only manipulations on integers, and discuss the complexity of the different algorithms.

1. Hilbert function of graded rings.

1.1 A commutative ring with identity R is graded by \mathbb{N} , if $R = \bigoplus_{n \in \mathbb{N}} R_n$, R_n abelian subgroups, $R_s R_t \subset R_{s+t}$. If R is graded, an R -module M is graded iff $M = \bigoplus_{n \in \mathbb{N}} M_n$, M_n abelian subgroups, $R_s M_t \subset M_{s+t}$. An element $f \in R$ ($f \in M$), $f \neq 0$, is called homogeneous of degree t ($\deg(f) = t$), iff $f \in R_t$ ($f \in M_t$). Any element $f \in R$ ($f \in M$), $f \neq 0$, can be splitted in a unique way into a finite sum of non-zero homogeneous elements of different degree:

$$(1.1) \quad f = \sum_{j=1}^r f_{ij}, \deg(f_{ij}) = i_j, i_1 < i_2 \dots < i_r .$$

A submodule U of M is called homogeneous iff for any $f \in U$ a representation (1.1) exists with $f_{ij} \in U$, i.e. iff U is generated by homogeneous elements. A morphism $f: M \rightarrow M'$ between graded R -modules is called homogeneous of degree zero (we will call it, with abuse of notation, simply "homogeneous") iff $f(M_i) \subseteq M'_i$ for each i .

1.2. The polynomial ring $P := K[x_0, \dots, x_n]$ is graded in the usual way

$$P_t = \{f \in P : f \text{ homogeneous, } \deg(f) = t\} .$$

We can associate to any degree vector, i.e. to any r -tuple of nonnegative integers $(i_1, \dots, i_r) \in \mathbb{N}_0^r$, a natural graduation on P^r by putting

$$(P^r)_t = \{(f_1, \dots, f_r) : f_j \in P_{t-i_j} \text{ if } t \geq i_j, f_j = 0 \text{ otherwise}\}$$

in other words any graduation is assigned on P^r if a degree i_j is associated to any element e_j of the canonical basis of P^r . If I is an homogeneous ideal of P , $I = \bigoplus I_t$, I_t a K -vectorial subspace of P_t , then $R = P/I$ is graded in a canonical way by putting $R_t = P_t/I_t$. R_t is a finite dimensional K -vectorial space. Under these assumptions it is possible to define the Hilbert function of R

$$H(R, t) := \dim_K(R_t) .$$

1.3. Under these assumptions there is an homogeneous free resolution of the graded P -module R , i.e. an exact sequence:

$$(1.2) \quad 0 \longrightarrow P_s \xrightarrow{f_s} P_{s-1} \longrightarrow \dots \xrightarrow{f_2} P_1 \xrightarrow{f_1} P_0 \xrightarrow{f_0} R \longrightarrow 0$$

where each P_i is graded, and each f_i is homogeneous.

Hence $f_0(P) = R$, f_s is injective, $\text{Ker } f_i = \text{Im}(f_{i+1})$ $i = 0 \dots s-1$. If $\{e_{ij} : i=1, \dots, r_j\}$, $j=1 \dots s$, denotes the canonical basis of P^r_j and $(d_{1j}, \dots, d_{r_j j})$ the degree vector associated to the graduation on P^r_j , then $\deg(f_j(e_{ij})) = d_{ij}$.

1.4. Since the f_j 's are homogeneous, then (1.2) "splits" into exact sequences of K -vectorial spaces of finite dimension:

$$(1.3) \quad 0 \longrightarrow (P_s)_t \xrightarrow{f_{st}} (P_{s-1})_t \xrightarrow{f_{s-1,t}} \dots \xrightarrow{(P_2)_t \xrightarrow{f_{2t}} (P_1)_t \xrightarrow{f_{1t}} P_t} R_t \longrightarrow 0$$

where f_{jt} is the restriction of f_j , for each t . Then by iterated application of the dimension formula:

$$(1.4) \quad H(R, t) = \dim_K R_t = \dim_K P_t + \sum_{i=1}^s (-1)^i \dim_K (P_i)_t .$$

Using $\dim_K P_t = \binom{t+n}{n}$, $\dim_K (P_{d_j})_t = \sum_{j=1}^{r_i} \dim_K P_{t-d_j j_i}$ (where $P_{t-d_j j_i} := 0$ if $d_j > t$) and denoting

$$\binom{t+n}{n}_+ = \begin{cases} \binom{t+n}{n} & \text{if } t+n \geq 0 \\ 0 & \text{if } t+n < 0 \end{cases}$$

we have:

$$(1.5) \quad H(R, t) = \binom{t+n}{n} + \sum_{i=1}^s (-1)^i \sum_{j=1}^{r_i} \binom{t+n-d_j i}{n}_+$$

1.5. $H(R, t)$ is a polynomial for $t \geq \max(d_j) - n$. It is called the characteristic polynomial or Hilbert polynomial of R :

$$(1.6) \quad x(R, t) = \binom{t+n}{n} + \sum_{i=1}^s (-1)^i \sum_{j=1}^{r_i} \binom{t+n-d_j i}{n}$$

There are different formulations for $x(R, t)$ whose coefficients have a theoretical meaning [4]:

$$(1.7) \quad x(R, t) = h_0 \binom{t}{d} + \dots + h_i \binom{t}{d-i} + \dots + h_{d-1} \binom{t}{1} + h_d \quad (\text{Hilbert})$$

$$(1.8) \quad x(R, t) = k_0 \binom{t+d}{d} + \dots + k_i \binom{t+d-i}{d-i} + \dots + k_{d-1} \binom{t+1}{1} + k_d \quad (\text{Severi})$$

$$(1.9) \quad x(R, t) = \binom{t+d}{d} + p_0 \binom{t+d-1}{d} + \dots + (-1)^i \binom{t+d-i-1}{d-i} p_i + \dots + (-1)^d p_d \quad (\text{Marchionna})$$

The integers h_0, \dots, h_d in (1.7) are called Hilbert-coefficients, the integers p_1, \dots, p_d are genus numbers. Especially, $p_d = (-1)^d (h_d - 1)$ represents the virtual arithmetic genus of I , where $R = P/I$. The order of I is $h_0 (= k_0 = 1 + p_0)$ and d , the degree of $x(R, t)$, is the Krull dimension of I .

The formulas (1.6)-(1.9) can be interpreted as representations of a polynomial in t of degree $\leq n$ (or $\leq d$) using different bases of the polynomial space. Hence, by simple basis transformations the representations (1.7)-(1.9) and many others can be obtained from (1.6).

1.6. If $R = \bigoplus R_t$ is a graded ring and R_t is a finite dimensional K -vectorial space for each t , the Hilbert function of R can be defined as in 1.2. However the situation outlined above is quite general, because of the following results [9]: Any graded ring R which is noetherian and satisfies $R_0 = K$, is generated by a finite set of homogeneous elements $\{y_0, \dots, y_m\}$; then there is a surjection $p: P = K[x_0, \dots, x_m] \rightarrow R$ defined by $p(x_i) = y_i$; if we impose the additional

assumption:

$$(1.10) \quad y_i \in R_1 \quad \text{for each } i ,$$

$I = \text{Ker } p$ is homogeneous, $R = P/I$ is among the rings considered in 1.2-1.5 .

2. The G-resolution of a monomial ideal

2.1. Denoting by \mathbb{N}_0^{n+1} the set of nonnegative integers, we order \mathbb{N}_0^{n+1} and the set of terms $T = \{x_0^{i_0} \dots x_n^{i_n} : i_j \in \mathbb{N}_0\}$ by the graduated lexicographical term ordering $<_T$:

$$x_0^{i_0} \dots x_n^{i_n} <_T x_0^{j_0} \dots x_n^{j_n} : \Leftrightarrow (i_0, \dots, i_n) <_T (j_0, \dots, j_n)$$

: \Leftrightarrow the first non-zero component of $(\sum j_v - i_v, j_0 - i_0, \dots, j_n - i_n)$ is positive.

P is naturally graded by $<_T$, defining

$$P_{\underline{i}} := \{cx_0^{i_0} \dots x_n^{i_n} : c \in K\}, \underline{i} := (i_0, \dots, i_n) \in \mathbb{N}_0^{n+1} .$$

P^r can then be made a T -graded P -module by assigning a T -degree $\alpha_k \in \mathbb{N}_0^{n+1}$ to any element e_k of its canonical basis. Then $(P^r)_{\underline{i}} = \{(f_1, \dots, f_r) : f_j = c_j \cdot x_0^{j_0} \dots x_n^{j_n}, c_j \in K, \text{ if } j_v = i_v - (\alpha_j)_v \geq 0, v=0, \dots, n ; f_j = 0 \text{ otherwise}\}$.

The definitions of 1.1 carry on naturally to this graduation, the T -graduation; we will speak then of T -homogeneous elements, of T -degree, of T -homogeneous modules and T -homogeneous morphisms. By a T -homogeneous free resolution of the T -graded ring R we will mean a resolution (1.2) in which each P^r_i is T -graded and each f_i is T -homogeneous. If $0 \neq f \in P^r$ and $f = \sum_{v=1}^s f_{i_v}$, $f_{i_v} \neq 0$, $f_{i_v} \in (P^r)_{i_v}$, $i_s <_T i_{s-1} <_T \dots <_T i_1$ is the unique splitting of f into T -homogeneous summands, we will denote $M_T(f) := f_{i_1}$. If $0 \neq f \in P$ then $M_T(f) = c x_0^{i_0} \dots x_n^{i_n}$ where $T\text{-deg } f = (i_0, \dots, i_n)$ and $c \in K \setminus \{0\}$. For any submodule $U \subset P^r$, $M_T(U) = \{M_T(f) : 0 \neq f \in U\} \cup \{0\}$ is a T -homogeneous submodule of P^r .

If I is an ideal of P , a Gröbner basis for I can be defined as a basis $\{f_1, \dots, f_t\}$ of I s.t. $\{M_T(f_i) : i=1, \dots, t\}$ generates $M_T(I)$.

A Gröbner basis of I is irreducible iff for any i, j , $i+j$, $M_T(f_j)$ does not divide $M_T(f_i)$. Algorithms to compute a Gröbner basis for I from a basis for it are discussed in [2].

2.2. Theorem [5]:

If I is an homogeneous ideal of P (under the usual graduation) and $J = M_T(I)$, $R = P/I$, $S = P/J$, then $H(R, t) = H(S, t)$ for each t .

2.3. Let then I an homogeneous ideal of P , $\{\phi_1, \dots, \phi_s\}$ a Gröbner basis of I , s.t. w.l.o.g. $M_T(\phi_i)$ has coefficient 1. Let $J = M_T(I)$.

We will use the following notations:

$$r_k := \binom{s}{k}$$

$$\tilde{I}_k := \{(i_1, \dots, i_k) : 1 \leq i_1 < \dots < i_k \leq s\}$$

$$T(i_1, \dots, i_k) := \text{l.c.m. } (M_T(\phi_{i_1}), \dots, M_T(\phi_{i_k})) \in T$$

$$\tau(i_1, \dots, \cancel{i_j}, \dots, i_k) := (-1)^j T(i_1, \dots, \cancel{i_j}, \dots, i_k).$$

$$D(i_1, \dots, i_k) := \deg(T(i_1, \dots, i_k)).$$

We index the components of P^{r_k} by the elements of \tilde{I}_k .

We will write then P^k instead of P^{r_k} and denote $\{e_{(i_1, \dots, i_k)}\}$ the canonical basis of P^k under this indexing.

Impose on P^k the T -graduation defined by $\text{Tdeg}(e_{(i_1, \dots, i_k)}) := \text{Tdeg}(i_1, \dots, i_k)$. Define then for $k > 1$ $d_k: P^k \rightarrow P^{k-1}$ as follows:

$$(2.1) \quad d_k(e_{(i_1, \dots, i_k)}) = \sum_j \tau(i_1, \dots, \cancel{i_j}, \dots, i_k) e_{(i_1, \dots, \cancel{i_j}, \dots, i_k)};$$

let $d_0: P \rightarrow P/J$ the canonical projection. Define d_1 by $d_1(e_i) := M_T(\phi_i)$.

2.4. Lemma:

For each k , denoting $\phi_{(i_1, \dots, i_k)} := d_k(e_{(i_1, \dots, i_k)})$:

- 1) d_k is homogeneous and

$$d_k(e_{(i_1, \dots, i_k)}) \in (P^{k-1})_{\underline{i}} \text{ with } \underline{i} := \text{Tdeg } T(i_1, \dots, i_k)$$

- 2) $\phi_{(i_1, \dots, i_k)}$ is T -homogeneous

- 3) $\{\phi_{(i_1, \dots, i_k)}\}$ is a basis for $\text{Ker}(d_{k-1})$.

Proof: By induction on k 1) and 2) are immediate.

$$\begin{aligned}
 d_{k-1}(\phi(i_1, \dots, i_k)) &= \sum_{\mu, \nu} [(-1)^\mu \frac{T(i_1 \dots i_k)}{T(i_1 \dots \cancel{i_\mu} \dots i_k)} (-1)^{\nu-1} \frac{T(i_1 \dots \cancel{i_\nu} \dots i_k)}{T(i_1 \dots \cancel{i_\nu} \dots \cancel{i_\mu} \dots i_k)} \\
 &\quad + (-1)^\nu \frac{T(i_1 \dots i_k)}{T(i_1 \dots \cancel{i_\nu} \dots i_k)} \cdot (-1)^\mu \frac{T(i_1, \dots, \cancel{i_\mu}, \dots, i_k)}{T(i_1, \dots, \cancel{i_\mu}, \dots, \cancel{i_\nu}, \dots, i_k)}] \\
 &\quad \cdot e(i_1 \dots \cancel{i_\mu} \dots \cancel{i_\nu} \dots i_k) = 0 .
 \end{aligned}$$

So $\text{Im}(d_k) \subset \text{Ker}(d_{k-1})$. Let then $\psi \in \text{Ker}(d_{k-1})$. Since, inductively each d_i is T-homogeneous, $\text{Ker}(d_{k-1})$ is T-homogeneous, so we may suppose ψ homogeneous.

Let $\psi = \sum \psi(i_1, \dots, i_{k-1}) e(i_1, \dots, i_{k-1})$. Let (j_1, \dots, j_{k-1}) be the lexicographically greatest index s.t. $\psi(j_1, \dots, j_{k-1}) \neq 0$.

Since $d_{k-1}(\psi) = \sum \psi(i_1, \dots, i_{k-1}) \phi(i_1, \dots, i_{k-1}) = 0$ also its (j_2, \dots, j_{k-1}) component is zero and the non-zero contribute given to it by $\phi(j_1, \dots, j_{k-1})$ must be cancelled out by, at least, the contribute of another $\phi(j'_1, \dots, j'_{k-1})$.

Then, necessarily $j'_1 < j_1$, $j'_\alpha = j_\alpha$ if $\alpha > 1$. Also $T(j'_1, j_1, \dots, j_{k-1})$ divides $x_0^{j'_0} \dots x_n^{j'_n}$, where $\underline{j} = \text{Tdeg}(\psi)$. So $c \in K \setminus \{0\}$, $\psi_0 \in T$ exist in such a way that $\psi' = \psi - c\psi_0 \phi(j'_1, j_1, \dots, j_{k-1}) \in \text{Ker}(d_{k-1})$, $\text{Tdeg}(\psi') = \text{Tdeg}(\psi)$ unless $\psi' = 0$ and the last non-zero component of ψ' is lexicographically less than (j_1, \dots, j_{k-1}) . Then an inductive argument can be given to show $\psi \in \text{Im}(d_k)$.

2.5. Corollary

$$(2.2) \quad 0 \rightarrow P \xrightarrow{\tilde{I}_s} P \xrightarrow{d_s} P \xrightarrow{\tilde{I}_{s-1}} \dots P \xrightarrow{\tilde{I}_2} P \xrightarrow{d_2} P \xrightarrow{\tilde{I}_1} P \xrightarrow{d_1} P \xrightarrow{d_0} P/J \rightarrow 0$$

is a T-resolution, called the G-resolution of P/J .

2.6. Remark

In [7], after a definition of Gröbner bases for P -modules is given, a generalization of the construction of 2.4 is given which produces a T-resolution for P/I , I any ideal, in which the $\phi(i_1, \dots, i_k)$'s are a Gröbner basis for $\text{Ker}(d_{k-1}) = \text{Im}(d_k)$. That's why (2.2) is called a G-resolution.

2.7. Algorithm:

```

u: = D(1,...,s);
for i = 1 to u do
    mi: = 0;
m0: = 1 ;
for k = 1 to s do
    Ĩ: = Ĩk ;
    while Ĩ ≠ ∅ do
        let (i1,...,ik) ∈ Ĩ ;
        Ĩ: = Ĩ \ {(i1,...,ik)} ;
        t: = D(i1,...,ik) ;
        mt: = mt + (-1)k .
H(P/I,t): = H(P/J,t): =  $\sum_{j=0}^u m_j \binom{t+n-j}{n}_+$  .

```

2.8. Correctness of the algorithm comes out directly from 2.5 and (1.7). If we assume that no cost arises from storing and retrieving terms, the complexity is asymptotically $s \cdot 2^{s-1}$ for large s as easy computations show. However there is wide space for improvements: if $T(i_1, \dots, i_k) = T(i_1, \dots, \cancel{x}, \dots, i_k)$, the contribute of the two tuples cancel out each other. Improved algorithms based on this observations are discussed in the following.

3. Two algorithms for Hilbert function computation

3.1. We denote $\tilde{I}: = \bigcup_k \tilde{I}_k$,

$$\tilde{T}: = \{T(i_1, \dots, i_k) : (i_1, \dots, i_k) \in \tilde{I}\} ,$$

$$\tilde{I}_\tau: = \{(i_1, \dots, i_k) \in \tilde{I} : T(i_1, \dots, i_k) = \tau\} ,$$

$$\mu_\tau: = \sum_k (-1)^k * (\tilde{I}_\tau \cap \tilde{I}_k) .$$

Then $m_t = \sum \mu_\tau$, where the summation is extended over all τ 's satisfying $\deg(\tau) = t$.

The first algorithm presented here computes the μ_τ 's by iteration on s , the cardinality of the basis. Let primed symbols \tilde{I}'_k , \tilde{I}' , \tilde{I}'_τ , \tilde{T}' , μ'_τ be used with respect to the ideal generated by $T(1), \dots, T(s-1)$, i.e. consider just k -tuples with indices $< s$, and unprimed symbols for the ideal generated by $T(1), \dots, T(s)$.

Then obviously,

$$\tilde{T}_\tau = \tilde{T}_\tau \cup \{(i_1, \dots, i_k, s) : T(i_1, \dots, i_k) = \sigma, \text{l.c.m.}(\sigma, T(s)) = \tau\},$$

$$\mu_\tau = \mu'_\tau - \sum_{\sigma} \mu'_{\sigma} , \text{ where } \sigma \text{ satisfies l.c.m. } (\sigma, T(s)) = \tau .$$

This shows the correctness of the following algorithm.

3.2. Algorithm

```

 $\tilde{T} := \{1\}; \mu_1 := 1;$ 
 $\text{for } i := 1 \text{ to } s \text{ do}$ 
 $\quad \tilde{T}' := \tilde{T};$ 
 $\quad \text{for each } \sigma \in \tilde{T}' \text{ do}$ 
 $\quad \quad \mu'_{\sigma} := \mu_{\sigma};$ 
 $\quad \text{for each } \sigma \in \tilde{T}' \text{ do}$ 
 $\quad \quad \tau := \text{l.c.m. } (\sigma, T(i));$ 
 $\quad \quad \text{if } \tau \in \tilde{T} \text{ then } \mu_{\tau} := \mu_{\tau} - \mu'_{\sigma}$ 
 $\quad \quad \text{else } \tilde{T} := \tilde{T} \cup \{\tau\};$ 
 $\quad \mu'_{\tau} := -\mu'_{\sigma}.$ 

```

At the k -th step, \tilde{T}' contains at most $(k-1)^{n+1}$ elements, since any $\sigma \in \tilde{T}'$ is a l.c.m. of a subset of $\{T(1), \dots, T(k-1)\}$, such that at most $k-1$ different choices for any of the powers of x_0, \dots, x_n in σ exist. The computation of all $\tau = \text{l.c.m. } (\sigma, T(k))$ requires no more than $(n-1)(k-1)^{n+1}$ comparisons and the tests $\tau \in \tilde{T}$ are approximately better than $k^{n+1} \log k$ if a good sorting method like Heapsort is used. Considering no cost for storing and retrieving terms, the total amount is therefore asymptotically for $s \rightarrow \infty$ in the worst case $(n+1) \sum_{k=2}^s [(k-1)^{n+1} + k^{n+1} \log k] < 2(n+1) \sum_{k=2}^s k^{n+1} \log k \sim s^{n+2} \log s$.

3.3. The second algorithm is based on the following observation. Denoting $\deg_v(x_0^v \dots x_n^v) := i_v, v=0, \dots, n$, we know $\tau \in \tilde{T}$ iff a set $\{i_1, \dots, i_k\}$ exists, such that for all $v \in \{0, \dots, n\}$

$$(3.1) \quad \deg_v(T(i_\mu)) = \deg_v(\tau) \text{ for a } \mu \in \{1, \dots, k\},$$

$$(3.2) \quad \deg_v(T(i_\ell)) \leq \deg_v(\tau) \text{ for all } \ell \in \{1, \dots, k\}.$$

In the algorithm each call of NEXTINDEX enlarges the set $\{i_1, \dots, i_k\}$, whereas in J the components v are stored, for which (3.1) is not yet satisfied. The set K contains all $j \notin \{i_1, \dots, i_k\}$ satisfying $\deg_v(T(j)) \leq \deg_v(\tau)$, $v=0, \dots, n$

The algorithm terminates since each call of NEXTINDEX reduces the cardinality of K by 1 and of J by at least 1. In case $K = \emptyset$ and $J \neq \emptyset$ we have $\tau \notin \tilde{T}$. In case $J = \emptyset$ and $K \neq \emptyset$ there is a $i_0 \notin \{i_1, \dots, i_k\}$ satisfying also $\deg_v(T(i_0)) \leq \deg_v(\tau)$, $v=0, \dots, n$. Then the subsets of \tilde{T}_τ containing i_1, \dots, i_k are splitted in those containing also i_0 and in those without i_0 ; hence their contribute in μ_τ cancels pair by pair.

3.4. Algorithm

S: = $\{(a_0, \dots, a_n) : a_v = \deg_v(T(\mu)) \text{ for } \mu \in \{1, \dots, s\}\}$;

While $S \neq \emptyset$ do

let $(a_0, \dots, a_n) \in S$;

$\tau := x_0^{a_0} \dots x_n^{a_n}$;

$S := S - \{(a_0, \dots, a_n)\}$;

$J := \{0, \dots, n\}$;

$K := \{k : T(k) \text{ divides } \tau\}$;

$w := 1$;

NEXTINDEX (J, K, w) .

Procedure NEXTINDEX (J, K, w):

While $\{k \in K : \deg_{j_1}(T(k)) = \deg_{j_1}(\tau)\} \neq \emptyset$ do

$w := -w$;

let $k \in K$ s.t. $\deg_{j_1}(T(k)) = \deg_{j_1}(\tau)$;

$K := K - \{k\}$;

$J' := \{v \in J : \deg_v(T(k)) = \deg_v(\tau)\}$;

$J := J - J'$;

if $K = \emptyset$ and $J = \emptyset$ then $\mu_\tau := \mu_\tau + w$;

if $K \neq \emptyset$ and $J \neq \emptyset$ then NEXTINDEX (J, K, w) ;

$J := J \cup J'$.

3.5. To obtain an asymptotic estimate for the complexity of algorithm 3.4 for large s , we analyse at first the amount of the computations for a fixed μ_τ . Before the first call of NEXTINDEX, the computation of K requires at most $s(n+1)$ comparisons. (At this stage also all pairs (k, j) are stored satisfying $\deg_j(T(k)) = \deg_j(\tau)$.) Each call of NEXTINDEX is proportional to the cardinality of $\{k \in K : \deg_{j_1}(T(k)) = \deg_{j_1}(\tau)\}$ times the amount of the next call of NEXTINDEX. The

iterative calls of NEXTINDEX require hence in the worst case constant times $k_0 \dots k_n$ comparisons, where $k_v := \#\{k \in K : \deg_v(T(k)) = \deg_v(\tau)\}$. Since $\#S \leq s^{n+1}$, the preparations for the first calls of NEXTINDEX require totally at most $(n+1)s^{n+2}$ comparisons. The amount of all NEXTINDEX computations is apart of a constant factor in the worst case only $\sum_k k_0 \dots k_n = s^{n+1}$, the latter identity displays the fact that terms coincide, if different $T(i)$ have the same $\deg_v(T(i))$. Hence, again not considering the cost for storing and retrieving, the asymptotic behaviour of amount for $s \rightarrow \infty$ is determined by the first calls of NEXTINDEX yielding asymptotically s^{n+2} .

3.6. Both algorithms of this paragraph can be applied to obtain reduced T-resolutions. At detailed discussion of this fact is beyond the aim of our paper.

4. Example

4.1. We consider an arbitrary coordinate ring of Macaulay's curves [8, § 8.2]:

$R = K[t_0^s, t_0^{s-1}t_1, t_0t_1^{s-1}, t_1^s]$. R is graded by $R_t = \{f(t_0, t_1) \in R : f \text{ homogeneous, } \deg f = ts\}$. Then $R = K[x_0, x_1, x_2, x_3]/I$, where I has the G-basis

$$\begin{aligned} f_1 &= x_1x_2 - x_0x_3 \\ f_i &= x_1^{s+1-i}x_3^{i-2} - x_0^{s-i}x_2^{i-1} \quad , \quad i = 2, \dots, s-1 \\ f_s &= x_2^{s-1} - x_1x_3^{s-2} \quad . \end{aligned}$$

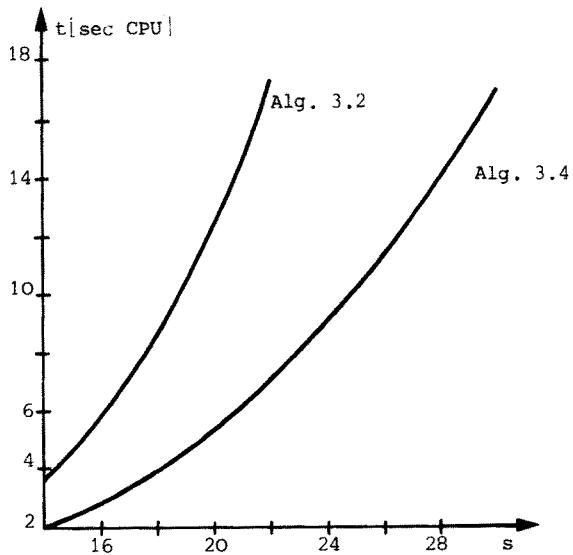
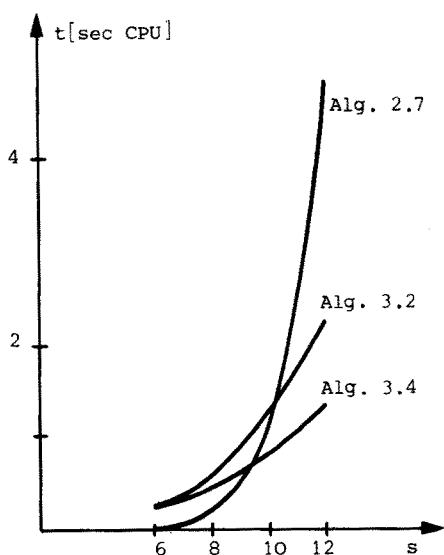
The coefficients of the Hilbert function are

$$m_0 = 1, m_2 = 1, m_{s-1} = -s+1, m_s = 2s-4, m_{s+1} = -s+3 \quad ,$$

and $m_i = 0$ for all other i 's. Therefore for all $t \in \mathbb{N}_0$

$$\begin{aligned} H(R, t) &= \binom{t+3}{3} - \binom{t+1}{3} - (s-1)\binom{t-s+4}{3} + (2s-4)\binom{t-s+3}{3} - (s-3)\binom{t-s+2}{3} + \\ &= \begin{cases} (t+1)^2 & \text{for } 0 \leq t < s-2 \\ st+1 & \text{for } t \geq s-2 \end{cases} . \end{aligned}$$

4.2. We tested the computational amount for finding the m_i 's with algorithms 2.7, 3.2, and 3.4 on the Hagen IBM 370/165 and obtained the following figures



Acknowledgement. We want to thank J. Brinker, FernUniversität Hagen, for his assistance to implement the algorithms.

REFERENCES

- [1] B. Buchberger, Ph. D. Thesis, Univ. Innsbruck, 1965.
- [2] B. Buchberger, A criterion for detecting unnecessary reductions in the construction of Gröbner-bases, Springer LN in Comp. Sci. Nr. 72, 1979, 3-21.
- [3] W. Gröbner, Moderne algebraische Geometrie, Springer-Verlag, Wien-Innsbruck 1949.
- [4] W. Gröbner, Algebraische Geometrie, Bibliographisches Institut Mannheim, vol I 1968, vol II 1970.
- [5] F.S. Macaulay, Some properties of enumeration in the theory of modular systems, Proc. London Math. Soc. 26, 1927, 531-555.
- [6] H.M. Möller, Mehrdimensionale Hermite-Interpolation und numerische Integration, Math. Z. 148, 1976, 107-118.
- [7] H.M. Möller and F. Mora, New constructive methods in classical ideal theory, submitted for publication.
- [8] B. Renschuch, Elementare und praktische Idealtheorie, Deutscher Verlag der Wissenschaften, Berlin 1976.
- [9] R.P. Stanley, Hilbert functions of graded algebras, Advances in math. 28, 1978, 57-83.

An Algorithm for Constructing Detaching Bases in the
Ring of Polynomials over a Field

Franz Winkler
Institut für Mathematik
Arbeitsgruppe CAMP
Johannes Kepler Universität
A-4040 Linz, Austria

Abstract

Most ideal theoretic problems in a polynomial ring are extremely hard to solve, if the ideal is given by an arbitrary basis. B. Buchberger, 1965, was the first to show that for polynomials over a field it is possible to construct a "detaching" basis from a given arbitrary one, such that the problems mentioned above become easily soluble. Other authors (e.g. M. Lauer, 1976, and S.C. Schaller, 1979) have considered different coefficient domains. In this paper we investigate a method, developed by C.Sims and C.Ayoub, for constructing "detaching" bases in the ring of polynomials over \mathbb{Z} , where the power products are ordered lexicographically. We show that the method also works for polynomials over a field, with only weak conditions on the ordering of the power products. New proofs of correctness and termination are presented. Furthermore we are able to improve the complexity behaviour of Ayoub's algorithm for the case of polynomials over a field.

1. Introduction and problem specification

Let K be a field. Then $K[x_1, \dots, x_n]$, the ring of polynomials over K in n indeterminates, is a Noetherian ring [vdW70]. This means that every ideal I in $K[x_1, \dots, x_n]$ is generated by a finite basis F ($I = \text{ideal}(F)$). If we are given a finite basis F for an ideal I in $K[x_1, \dots, x_n]$, a great number of problems still remain extremely difficult to solve. Among these problems are the problem of deciding whether a given polynomial belongs to I , the problem of deciding whether the polynomial ideal has dimension zero, the reduction of polynomials to canonical forms with respect to the ideal I and many more (compare [Wi78]). Therefore it is essential to compute from the basis F a "detaching" basis G , such that G generates the same ideal as F , but G makes the solutions of these problems easy.

In the following we assume that we have a total linear ordering $<_t$ on pp_n , the set of power products of the indeterminates x_1, \dots, x_n , which satisfies the two conditions

(T1) $1=x_1^{e_1} \dots x_n^{e_n} \leq_t p$ for all $p \in pp_n$ and

(T2) $p \cdot q_1 <_t p \cdot q_2$ for all $p, q_1, q_2 \in pp_n$ such that $q_1 <_t q_2$.

These two conditions imply that \leq_t is a Noetherian relation on pp_n .

Throughout this paper we use the following notations. If p is the power product $x_1^{e_1} \dots x_n^{e_n}$ in pp_n and $i \in \mathbb{N}$ then by $\text{rear}_i(p)$ we denote the power product $x_i^{e_i} \dots x_n^{e_n}$ (if $i > n$ then $\text{rear}_i(p) = 1$), and by $\deg_i(p)$ we denote e_i , the degree of p in x_i .

If f is a nonzero polynomial in $K[x_1, \dots, x_n]$, then $\text{ldpp}(f)$ is the greatest power product in pp_n which has a nonzero coefficient in f . $\text{lcc}(f)$ is the coefficient of $\text{ldpp}(f)$. $\text{lct}(f) = \text{lcc}(f)/\text{ldpp}(f)$. $\text{red}(f) = f - \text{lct}(f)$.

Following Buchberger's notation, for an arbitrary subset F of $K[x_1, \dots, x_n]$ we define a reduction relation $\xrightarrow{1,F}$ on $K[x_1, \dots, x_n]$:

$f \xrightarrow{1,F} g$ iff there is a power product p , which occurs with coefficient $a \neq 0$ in f , and a nonzero polynomial h in F such that p is a multiple of $\text{ldpp}(h)$ and
 $g = f - \frac{a}{\text{lcc}(h)} \cdot \frac{p}{\text{ldpp}(h)} \cdot h$.

By \xrightarrow{F} we denote the reflexive transitive closure of $\xrightarrow{1,F}$.

\xrightarrow{F} is a Noetherian relation, so a chain of reductions starting with a polynomial f terminates with some g such that g cannot be reduced further. In this case we say that g is a simplified version of f with respect to F . Clearly $g \equiv f$ modulo the ideal generated by F .

We say that $f \in K[x_1, \dots, x_n]$ is reduction unique with respect to F if there is a unique simplified version of f w.r.t. F .

- If every $f \in K[x_1, \dots, x_n]$ is reduction unique w.r.t. F
- then we call F a detaching basis (Gröbner-basis or complete basis)
- for $\text{ideal}(F)$.

Lemma 1.1: Let F be a finite subset of $K[x_1, \dots, x_n]$. If for every polynomial f in $\text{ideal}(F)$ $f \xrightarrow{F} 0$ then F is a detaching basis for $\text{ideal}(F)$.

In [Bu65], [Bu70], and [Bu76] B.Buchberger presented an algorithm for constructing a detaching basis G for an ideal I in $K[x_1, \dots, x_n]$, for which some basis F is given. The main step in this algorithm is to take two polynomials f and g in the basis, compute the least common multiple p of $\text{ldpp}(f)$ and $\text{ldpp}(g)$, reduce p to some h_1 using f and to some h_2 using g and compute simplified versions h_1' and h_2' of h_1 and h_2 . If $h_1' \neq h_2'$ the new polynomial $h_1' - h_2'$ is added to the basis.

Other authors ([La76a], [La76b], [Sc79]) have considered different coefficient domains. In [Si78] C.Sims presented an algorithm for constructing a basis for an ideal in $\mathbb{Z}[x]$, which allows to decide whether a given polynomial is contained in the ideal.

His work was extended to multivariate polynomials over \mathbb{Z} by C.Ayoub in [Ay80]. Ayoub proves her result only for the case where $<_t$ is the lexicographic ordering on the power products. This ordering is sufficient for deciding whether a given polynomial is contained in an ideal I . But when detaching bases should be used for simplifying polynomials with respect to polynomial side relations, it is desirable to have a wider range of possible definitions of the notion of "simpler" to choose from.

We present an algorithm for computing detaching bases for polynomial ideals in $K[x_1, \dots, x_n]$, where the underlying ordering $<_t$ on the power products has to satisfy only the two conditions (T1) and (T2). Detailed proofs of the various lemmata can be found in [Wi82].

We hope that this paper helps to understand the relations between Buchberger's and Ayoub's algorithms for completing bases for polynomial ideals.

2. A first algorithm for constructing detaching bases

A finite set F of polynomials in $K[x_1, \dots, x_n]$ is called staggered, if $0 \notin F$ and for $f, g \in F, f \neq g$, we have $\text{ldpp}(f) \neq \text{ldpp}(g)$.

Lemma 2.1: For every finite set of polynomials F in $K[x_1, \dots, x_n]$, a finite set of polynomials G can be constructed, such that $\text{ideal}(F) = \text{ideal}(G)$ and G is staggered.

Proof: Let F' be $F - \{0\}$. As long as there are two different polynomials f, g in F' with $\text{ldpp}(f) = \text{ldpp}(g)$, we carry out the following process:

compute $h = f - (\text{lde}(f)/\text{lde}(g)) \cdot g$. If $h=0$ then delete f from F' . Otherwise replace f by h in F' .

Obviously the ideal generated by F' remains unchanged during this process.

The process terminates after a finite number of steps, since the leading power products of the polynomials in F' decrease with respect to the Noetherian ordering $<_t$. Finally we get a set of polynomials F' which generates the same ideal as F and is staggered. So we let $G = F'$. *

For a finite, staggered set F in $K[x_1, \dots, x_n]$ we define the following tower of admissible pairs in $\text{pp}_n \times K[x_1, \dots, x_n]$:

$$F[0] := \{(1, f) \mid f \in F\} \text{ and for } 1 < i < n$$

$$F[i] := F[i-1] \cup \{(p \cdot x_j^s, f) \mid (p, f) \in F[i-1], \max(f, F, i), s \in \mathbb{N}\},$$

where $\max(f, F, i)$ iff $(\forall g \in F)(\text{rear}_{i+1}(\text{ldpp}(g)) = \text{rear}_{i+1}(\text{ldpp}(f)) \Rightarrow \deg_i(\text{ldpp}(g)) < \deg_i(\text{ldpp}(f)))$.

For the proof of the main theorem we will need the following lemmata.

Lemma 2.2: Let F be a finite, staggered subset of $K[x_1, \dots, x_n]$, $p, q \in pp_n$, $f \in F$. If $(p, f), (q, f) \in F^{[n]}$ then $(p \cdot q, f) \in F^{[n]}$.

Lemma 2.3: Let F be a finite, staggered subset of $K[x_1, \dots, x_n]$, $f \in F$, $1 \leq m \leq n$, $(q, g) \in F^{[n]}$, $f \neq g$ and $ldpp(x_m \cdot f) = ldpp(q \cdot g)$.

Then the exponents of x_m, \dots, x_n in q are 0.

Based on $F^{[n]}$ we define the set of reductors $F^{(n)}$ by $F^{(n)} = \{p \cdot f \mid (p, f) \in F^{[n]}\}$.

Lemma 2.4: Let F be a finite, staggered subset of $K[x_1, \dots, x_n]$.

Then $F^{(n)}$ is also staggered.

By $\text{mod}_K(F)$ let us denote the K -module generated by F for any $F \subseteq K[x_1, \dots, x_n]$, i.e. $\text{mod}_K(F) = \{a_1f_1 + \dots + a_mf_m \mid \text{for } m \in \mathbb{N}_0, a_1, \dots, a_m \in K, f_1, \dots, f_m \in F\}$. Now we are ready to state the fundamental theorem for the construction of detaching bases.

Theorem 2.1: Let F be a finite, staggered subset of $K[x_1, \dots, x_n]$.

Then the following two assertions are equivalent:

- (i) $x_i \cdot f \in \text{mod}_K(F^{(n)})$ for all $f \in F$, $1 \leq i \leq n$,
- (ii) $p \cdot f \in \text{mod}_K(F^{(n)})$ for all $f \in F$, $p \in pp_n$.

Proof: Obviously (ii) implies (i), since (i) is a special case of (ii).

It remains to show that (i) implies (ii). This we prove by induction on $ldpp(p \cdot f)$ with respect to the Noetherian relation $<_t$.

Suppose that for some $p^* \in pp_n$ we know that

(IH1) if $ldpp(p \cdot f) <_t p^*$ then $p \cdot f \in \text{mod}_K(F^{(n)})$ for all $f \in F$, $p \in pp_n$.

From the induction hypothesis (IH1) we have to show

(1) if $ldpp(p \cdot f) = p^*$ then $p \cdot f \in \text{mod}_K(F^{(n)})$ for all $f \in F$, $p \in pp_n$.

We prove (1) by induction on p with respect to the lexicographic ordering $<_l$ on pp_n (which is a Noetherian relation).

Suppose that for some $\underline{p} \in pp_n$ we know that

(IH2) if $\underline{p} <_l \underline{p}'$ and $ldpp(\underline{p} \cdot f) = \underline{p}'$ then $\underline{p} \cdot f \in \text{mod}_K(F^{(n)})$ for all $f \in F$, $\underline{p} \in pp_n$.

From the induction hypothesis (IH2) we have to show

(2) if $ldpp(\underline{p} \cdot f) = \underline{p}'$ then $\underline{p} \cdot f \in \text{mod}_K(F^{(n)})$ for all $f \in F$.

If for all indices m , $1 \leq m \leq n$, such that $\deg_m(\underline{p}) \neq 0$ we have $(x_m, f) \in F^{[n]}$, then by Lemma 2.2 $(\underline{p}, f) \in F^{[n]}$ and hence $\underline{p} \cdot f \in F^{(n)} \subseteq \text{mod}_K(F^{(n)})$.

Otherwise there is an index m , $1 \leq m \leq n$, such that $\deg_m(\underline{p}) \neq 0$ and $(x_m, f) \notin F^{[n]}$.

Because of (i) we have $x_m \cdot f \in \text{mod}_K(F^{(n)})$, i.e. there are $l \in \mathbb{N}$, $a_1, \dots, a_l \in K - \{0\}$, $g_1, \dots, g_l \in F^{(n)}$ such that

$$x_m \cdot f = \sum_{j=1}^l a_j \cdot g_j \quad \text{and } g_i \neq g_k \text{ for } i \neq k.$$

Because of Lemma 2.4 $ldpp(g_i) \neq ldpp(g_k)$ for $i \neq k$. W.l.o.g. we assume

$ldpp(g_j) <_t ldpp(g_1)$ for $1 < j \leq l$.

$$\underline{p} \cdot f = (\underline{p}/x_m) \cdot x_m \cdot f = a_1 \cdot (\underline{p}/x_m) \cdot g_1 + \sum_{j=2}^1 a_j \cdot (\underline{p}/x_m) \cdot g_j.$$

All the power products occurring in $\sum_{j=2}^1 a_j \cdot (\underline{p}/x_m) \cdot g_j$ are less than p^* (here we need the property (T2) of \prec_t), so by the induction hypothesis (IH1)

$$\sum_{j=2}^1 a_j \cdot (\underline{p}/x_m) \cdot g_j \in \text{mod}_K(F^{(n)}).$$

So it remains to show that $(\underline{p}/x_m) \cdot g_1 \in \text{mod}_K(F^{(n)})$.

$g_1 = q \cdot g$ for some $(q, g) \in F^{[n]}$, where by lemma 2.3 $\deg_r(q) = 0$ for $m < r < n$.

Now we set $q' = (\underline{p}/x_m) \cdot q$ and get $(\underline{p}/x_m) \cdot g_1 = q' \cdot g$.

But $q' \prec_1 \underline{p}$, so by the induction hypothesis (IH2)

$$(\underline{p}/x_m) \cdot g_1 = q' \cdot g \in \text{mod}_K(F^{(n)}).$$

This completes the proof of (2), (1) and (i) \Rightarrow (ii). •

Now it is easy to show that condition (i) in theorem 2.1 is equivalent to $\text{mod}_K(F^{(n)}) = \text{ideal}(F)$. Using this equivalence together with lemma 1.1 one can prove that (i) is a sufficient condition for F being a detaching basis.

- Theorem 2.2: Let F be a finite, staggered subset of $K[x_1, \dots, x_n]$.
- If $x_i \cdot f \in \text{mod}_K(F^{(n)})$ for all $1 \leq i \leq n$, $f \in F$, then F is a detaching basis
- for $\text{ideal}(F)$.

For a set of polynomials F in $K[x_1, \dots, x_n]$ we introduce the notion of restricted reducibility modulo F : $f \xrightarrow{1, r, F} g$ iff there is a power product p , which occurs with coefficient $a \neq 0$ in f , and a polynomial $h \in F$ such that $p = \text{lpp}(h)$ and

$$g = f - \frac{a}{\text{lcc}(h)} \cdot h.$$

By $\xrightarrow{r, F}$ we denote the reflexive transitive closure of $\xrightarrow{1, r, F}$.

$\xrightarrow{1, r, F}$ is a Noetherian relation, so a chain of reductions of a polynomial f has to terminate with some g , such that g cannot be reduced further. In this case we say that g is a restricted simplified version of f modulo F . Clearly $g \equiv f$ modulo the ideal generated by F .

Lemma 2.5: Let F be a finite, staggered subset of $K[x_1, \dots, x_n]$, $f \in K[x_1, \dots, x_n]$. Then $f \in \text{mod}_K(F^{(n)})$ if and only if $f \xrightarrow{r, F^{(n)}} 0$.

In order to be able to prove the termination of our algorithm, we need to introduce the notion of triangularity: a staggered subset F of $K[x_1, \dots, x_n]$ is called triangular iff for all $f \in F$, $1 \leq i \leq n$ there exists a polynomial g in $F^{(n)}$ such that $\text{lpp}(g) = \text{lpp}(x_i \cdot f)$.

Lemma 2.6: If F is a finite, staggered subset of $K[x_1, \dots, x_n]$, then in finitely many steps a finite, triangular subset G of $K[x_1, \dots, x_n]$ can be constructed such that $\text{ideal}(F) = \text{ideal}(G)$.

Proof: Initially we let G be F . As long as there are $f \in G$, $1 \leq i \leq n$ such that there is no $g \in G^{(n)}$ with $\text{ldpp}(g) = \text{ldpp}(x_i \cdot f)$, we add $x_i \cdot f$ to G .

The process terminates, since for every k , $1 \leq k \leq n$, no polynomial h is added such that $\deg_k(\text{ldpp}(h)) > \max\{\deg_k(\text{ldpp}(g)) \mid g \in F\}$. •

Lemma 2.7: If F is triangular, then for every $p \in pp_n$, $f \in F$, there is a $g \in F^{(n)}$ such that $p \cdot \text{ldpp}(f) = \text{ldpp}(g)$.

Lemma 2.8: If F is a finite, triangular subset of $K[x_1, \dots, x_n]$ and the nonzero polynomial h is irreducible modulo $_{1,r,F(n)}$, then there is no $f \in F$ such that $\text{ldpp}(h)$ is a multiple of $\text{ldpp}(f)$.

Now we can state a first version of the algorithm for constructing a detaching basis G for $\text{ideal}(F)$:

$G \leftarrow \text{detbl}(F)$

[Algorithm for constructing a detaching basis, 1.version. F is a finite subset of $K[x_1, \dots, x_n]$. G is a finite detaching basis for $\text{ideal}(F)$]

(1) Let G be a finite, triangular basis for $\text{ideal}(F)$;

[an algorithm for constructing such a basis can be extracted from the proofs of Lemma 2.1 and Lemma 2.6]

(2) Set $C \leftarrow \{(i, f) \mid 1 \leq i \leq n, f \in G, (x_i \cdot f) \notin G^{(n)} \text{ and } x_i \cdot f \notin G\}$;

(3) while $C \neq \emptyset$ do

 {Choose $(i, f) \in C$;

 Let h be a restricted simplified version of $x_i \cdot f$ modulo $G^{(n)}$;

 [an algorithm is given in [Ay80]]

if $h \neq 0$ then {Set $G' \leftarrow G \cup \{h\}$;

 Let G be a finite, triangular basis for $\text{ideal}(G')$;

 Set $C \leftarrow \{(i, f) \mid 1 \leq i \leq n, f \in G, (x_i \cdot f) \notin G^{(n)} \text{ and } x_i \cdot f \notin G\}$;

else Delete (i, f) from C };

return •

The correctness of the algorithm follows from theorem 2.2 and lemma 2.5. Now let us consider the problem of termination. If detbl would not terminate, this would mean that it continuously adds polynomials h_1, h_2, \dots to the basis, where by lemma 2.8 no h_i is a multiple of some h_1, \dots, h_{i-1} . Such a sequence of polynomials, however, cannot exist [Bu70]. So detbl has to terminate.

A rather annoying property of detbl is that whenever a new polynomial is added to the basis G in (3), then all the reductions of pairs (i, f) done so far are rendered useless and $x_i \cdot f$ has to be reduced anew. This is because adding a new polynomial to the basis may totally destroy the structure of $G^{(n)}$. So it would be desirable to modify the basic concept in such a way that each pair (i, f) must be considered only once.

3. Improvement of the algorithm

The cost for improving detbI has to be paid in notational complexity. There are two major changes in notation. Firstly we consider sequences of polynomials rather than sets of polynomials, i.e. we consider "ordered" bases. Secondly the set of reductors $F^{(n)}$ for a basis F is defined somewhat differently, so that adding new polynomials to F does not destroy the structure of $F^{(n)}$ but merely add new reductors.

We define staggeredness for sequences of polynomials as in chapter 2 - a sequence of polynomials in $K[x_1, \dots, x_n]$ is called staggered if every element of F occurs only once in F and the set of elements in F is staggered - and again we can prove that for every finite sequence F of polynomials in $K[x_1, \dots, x_n]$ in finitely many steps a finite staggered sequence G can be constructed such that F and G generate the same ideal.

A substitution σ is a mapping from $\{0, x_1, \dots, x_n\}$ into \mathbb{N}_0 such that $\sigma(0)=0$.

Let $(a, f), (b, g)$ be pairs in $X_n \times K[x_1, \dots, x_n] - \{0\}$, where X_n is the set $\{(a_1, \dots, a_n) \mid a_i = 0 \text{ or } a_i = x_i \text{ for } 1 \leq i \leq n\}$. We say that (a, f) and (b, g) are unifiable, if there are substitutions σ_f and σ_g such that $x_1^{\sigma_f(a_1)} \dots x_n^{\sigma_f(a_n)} \cdot \text{lpp}(f) = x_1^{\sigma_g(b_1)} \dots x_n^{\sigma_g(b_n)} \cdot \text{lpp}(g)$.

For a finite, staggered sequence of polynomials F in $K[x_1, \dots, x_n]$ we define the sequence_of_multipliers \tilde{F} as follows:

if $\text{length}(F)=0$ then $\tilde{F}=\emptyset$. If $F=Gh$ then $\tilde{F}=\tilde{G} \circ (a_1, \dots, a_n)$, where for $1 \leq k \leq n$ $a_k=x_k$ if $\max(h, F, k)$ and (G_j, G_j) and $((a_1, \dots, a_{k-1}, x_k, 0, \dots, 0), h)$ are not unifiable for $1 < j < \text{length}(G)$, and $a_k=0$ otherwise. (\circ denotes the operation of adding a last element to a sequence.)

The set_of_reductors F^* for a finite, staggered sequence of polynomials F is defined as $F^*=\{p \cdot F_j \mid (p, j) \in F^\approx\}$, where $F^\approx=\{(x_1^{\sigma(a_1)} \dots x_n^{\sigma(a_n)}, j) \mid (a_1, \dots, a_n)=F_j, \sigma \text{ a substitution, } 1 \leq j \leq \text{length}(F)\}$.

Lemma 3.1: Let F be a finite, staggered sequence in $K[x_1, \dots, x_n]$, $p, q \in ppn$, $1 \leq j \leq \text{length}(F)$.

If $(p, j), (q, j) \in F^\approx$ then $(p \cdot q, j) \in F^\approx$.

A staggered sequence F in $K[x_1, \dots, x_n]$ is called unambiguous if $(\forall (p, j), (q, k) \in F^\approx) ((p, j) \neq (q, k) \implies p \cdot \text{lpp}(F_j) \neq q \cdot \text{lpp}(F_k))$.

Lemma 3.2: Let F be a finite, unambiguous sequence in $K[x_1, \dots, x_n]$, h a nonzero polynomial in $K[x_1, \dots, x_n]$ such that there is no $g \in F^*$ with $\text{lpp}(h)=\text{lpp}(g)$. Then Foh is unambiguous.

Lemma 3.3: Let F be a finite staggered sequence of polynomials in $K[x_1, \dots, x_n]$. Then there is a permutation π such that $(F_\pi(1), \dots, F_\pi(\text{length}(F)))$ is unambiguous.

Proof: We induct on $l=\text{length}(F)$. For $l=0$ F is already unambiguous.

Now let $l>1$. We choose $j \in \{1, \dots, l\}$ such that $\text{ldpp}(F_j)$ is not a multiple of any $\text{ldpp}(F_i)$ for $1 < i < l$, $i \neq j$. Let $F' = (F_1, \dots, F_{j-1}, F_{j+1}, \dots, F_l)$. By the induction hypothesis there is a reordering G' of F' such that G' is unambiguous.

So $G=G' \circ F_j$ is a reordering of F and by lemma 3.2 G is unambiguous. •

Lemma 3.4: Let F be a finite, unambiguous sequence of polynomials in $K[x_1, \dots, x_n]$. Then F^* is a staggered set of polynomials.

Again we define the notion of triangularity: an unambiguous sequence F is called triangular if for all i, j , $1 < i < n$, $1 < j < \text{length}(F)$, there exists a pair $(q, k) \in F^\approx$, $k < j$, such that $\text{ldpp}(x_i \cdot F_j) = \text{ldpp}(q \cdot F_k)$.

Lemma 3.5: Let F be a finite, triangular sequence in $K[x_1, \dots, x_n]$, h a nonzero polynomial in $K[x_1, \dots, x_n]$ such that there is no $g \in F^*$ with $\text{ldpp}(h) = \text{ldpp}(g)$, and m such that $\deg_m(\text{ldpp}(h)) > \max\{\deg_m(\text{ldpp}(F_i)) \mid 1 < i < \text{length}(F)\}$.

Then $(x_m, \text{length}(F)+1) \in (F \circ h)^\approx$ and $F \circ h$ is unambiguous.

Lemma 3.6: Let F be a finite triangular sequence in $K[x_1, \dots, x_n]$, h a nonzero polynomial in $K[x_1, \dots, x_n]$ such that there is no $g \in F^*$ with $\text{ldpp}(h) = \text{ldpp}(g)$.

Then there are polynomials $h_1, \dots, h_m \in K[x_1, \dots, x_n]$, $h_m = h$, such that $G = F \circ h_1 \circ \dots \circ h_m$ is triangular and $\text{ideal}(F \circ h) = \text{ideal}(G)$.

Proof: Let $g=h$. As long as there is an index m such that there is no $f \in (F \circ g)^*$ with $\text{ldpp}(x_m \cdot g) = \text{ldpp}(f)$, set $g=x_m \cdot g$. The process terminates, since by lemma 3.5 $\deg_m(\text{ldpp}(g))$ cannot surpass $\max\{\deg_m(\text{ldpp}(h)), \max\{\deg_m(F_i) \mid 1 < i < \text{length}(F)\}\}$.

By lemma 3.2 $F \circ g$ is unambiguous. F is triangular and for every i , $1 < i < n$, there is a $(p, j) \in (F \circ g)^\approx$ with $\text{ldpp}(x_i \cdot g) = \text{ldpp}(p \cdot (F \circ g)_j)$, so $F \circ g$ is triangular. So we let $h_1=g$. Iterating this process we get the desired h_2, \dots, h_m .

Clearly $\text{ideal}(F \circ h) = \text{ideal}(G)$. •

Lemma 3.7: Let F be a finite, unambiguous sequence in $K[x_1, \dots, x_n]$.

Then there is a finite, triangular sequence G such that every polynomial in G is a multiple of some polynomial in F and $\text{ideal}(G) = \text{ideal}(F)$.

Proof: Because of the proof of lemma 3.3 we may assume that $\text{ldpp}(F_j)$ is not a multiple of $\text{ldpp}(F_i)$ for $1 < i < j < \text{length}(F)$.

We induct on $l=\text{length}(F)$. If $l=0$ then obviously F is triangular.

So let $l>1$. For $F'=(F_1, \dots, F_{l-1})$ by the induction hypothesis there is a finite, triangular sequence G' such that every polynomial in G' is a multiple of some polynomial in F' and $\text{ideal}(G') = \text{ideal}(F')$. By lemma 3.6 there are multiples h_1, \dots, h_m of F_l such that $G=G' \circ h_1 \circ \dots \circ h_m$ is triangular and $\text{ideal}(G) = \text{ideal}(F)$. •

Lemma 3.8: Let F be a finite, triangular sequence in $K[x_1, \dots, x_n]$.

Then for every $p \in pp_n$, $1 \leq j \leq \text{length}(F)$ there is a pair $(q, F_k) \in F^\approx$, $k < j$, such that $\text{ldpp}(p \cdot F_j) = \text{ldpp}(q \cdot F_k)$.

Lemma 3.9: Let F be a finite, triangular sequence in $K[x_1, \dots, x_n]$ and h a nonzero polynomial in $K[x_1, \dots, x_n]$ such that there is no $g \in F^*$ with $\text{ldpp}(h) = \text{ldpp}(g)$. Then there is no j , $1 \leq j \leq \text{length}(F)$, such that $\text{ldpp}(h)$ is a multiple of $\text{ldpp}(F_j)$.

With these new notations, a theorem analogous to theorem 2.1 holds.

Theorem 3.1: Let F be a finite, triangular sequence in $K[x_1, \dots, x_n]$.

Then the following two assertions are equivalent:

- (i) $x_i \cdot F_j \in \text{mod}_K(F^*)$ for all $1 \leq j \leq \text{length}(F)$, $1 \leq i \leq n$,
- (ii) $p \cdot F_j \in \text{mod}_K(F^*)$ for all $1 \leq j \leq \text{length}(F)$, $p \in pp_n$.

Proof: Obviously (ii) implies (i), since (i) is a special case of (ii).

It remains to show that (i) implies (ii). This we prove by induction on $\text{ldpp}(p \cdot F_j)$ with respect to the Noetherian relation $<_t$.

Suppose that for some $p^* \in pp_n$ we know that

(IH1) if $\text{ldpp}(p \cdot F_j) <_t p^*$ then $p \cdot F_j \in \text{mod}_K(F^*)$ for all $p \in pp_n$, $1 \leq j \leq \text{length}(F)$.

From the induction hypothesis (IH1) we have to show

(1) if $\text{ldpp}(p \cdot F_j) = p^*$ then $p \cdot F_j \in \text{mod}_K(F^*)$ for all $p \in pp_n$, $1 \leq j \leq \text{length}(F)$.

We prove (1) by induction on j .

If $j=1$ then $(p, 1) \in F^\approx$ and therefore $p \cdot F_1 \in F^\approx \subseteq \text{mod}_K(F^*)$.

Now suppose that for some j^* , $2 \leq j^* \leq \text{length}(F)$ we know that

(IH2) if $\text{ldpp}(p \cdot F_j) = p^*$ and $j < j^*$ then $p \cdot F_j \in \text{mod}_K(F^*)$
for all $p \in pp_n$, $1 \leq j \leq \text{length}(F)$.

From the induction hypothesis (IH2) we have to show

(2) if $\text{ldpp}(p \cdot F_{j^*}) = p^*$ then $p \cdot F_{j^*} \in \text{mod}_K(F^*)$ for all $p \in pp_n$.

If for all indices m , $1 \leq m \leq n$, such that $\deg_m(p) \neq 0$ we have $(x_m, j^*) \in F^\approx$, then by lemma 3.1 $(p, j^*) \in F^\approx$ and hence $p \cdot F_{j^*} \in F^\approx \subseteq \text{mod}_K(F^*)$.

Otherwise there is an index m , $1 \leq m \leq n$, such that $\deg_m(p) \neq 0$ and $(x_m, j^*) \notin F^\approx$.

Because of (i) we have $x_m \cdot F_{j^*} \in \text{mod}_K(F^*)$, i.e. there are $l \in \mathbb{N}$, $a_1, \dots, a_l \in K - \{0\}$, $g_1, \dots, g_l \in F^*$ such that

$$x_m \cdot F_{j^*} = \sum_{j=1}^l a_j \cdot g_j \quad \text{and } g_i \neq g_k \text{ for } i \neq k.$$

Because of lemma 3.4 $\text{ldpp}(g_i) \neq \text{ldpp}(g_k)$ for $i \neq k$. W.l.o.g. we assume

$\text{ldpp}(g_j) <_t \text{ldpp}(g_1)$ for all $2 \leq j \leq l$.

$$p \cdot F_{j^*} = (p/x_m) \cdot x_m \cdot F_{j^*} = a_1 \cdot (p/x_m) \cdot g_1 + \sum_{j=2}^l a_j \cdot (p/x_m) \cdot g_j.$$

All the power products occurring in $\sum_{j=2}^l a_j \cdot (p/x_m) \cdot g_j$ are less than p^* , so by the induction hypothesis (IH1)

$$\sum_{j=2}^l a_j \cdot (p/x_m) \cdot g_j \in \text{mod}_K(F^*).$$

It remains to show that $(p/x_m) \cdot g_1 \in \text{mod}_K(F^*)$.

$g_1 = q \cdot F_k$ for some $(q, k) \in F^\approx$. Since $\text{ldpp}(x_m \cdot F_j^*) = \text{ldpp}(q \cdot F_k)$ and F is triangular, k must be less or equal to j^* . But $j^* = k$ is impossible, because $(x_m, j^*) \notin F^\approx$. So $k < j^* - 1$. Therefore $(p/x_m) \cdot g_1 = (p/x_m) \cdot q \cdot F_k \in \text{mod}_K(F^*)$ by the induction hypothesis (IH2). This completes the proof of (2), (1) and (i) \Rightarrow (ii). •

In analogy to chapter 2 it turns out that (i) in theorem 3.1 is equivalent to $\text{mod}_K(F^*) = \text{ideal}(F)$. Using this equivalence together with lemma 1.1 one can prove that (i) is a sufficient condition for F being a detaching basis.

- Theorem 3.2: Let F be a finite, triangular sequence in $K[x_1, \dots, x_n]$.
- If $x_i \cdot F_j \in \text{mod}_K(F^*)$ for all $1 \leq j \leq \text{length}(F)$, $1 \leq i \leq n$, then F is a detaching basis for $\text{ideal}(F)$.

As in chapter 2 we need a method for deciding whether $f \in \text{mod}_K(F^*)$ for a polynomial f in $K[x_1, \dots, x_n]$ and a finite, triangular sequence F .

Lemma 3.10: Let F be a finite, unambiguous sequence in $K[x_1, \dots, x_n]$ and f a polynomial in $K[x_1, \dots, x_n]$.

Then $f \in \text{mod}_K(F^*)$ if and only if $f \xrightarrow{r, F^*} 0$.

Lemma 3.11: Let F be a finite, unambiguous sequence in $K[x_1, \dots, x_n]$, h a polynomial in $K[x_1, \dots, x_n]$ such that $\text{ldpp}(h) \neq \text{ldpp}(g)$ for all $g \in F^\approx$.

If $f_1 \xrightarrow{r, F^*} f_2$ then $f_1 \xrightarrow{r, (F \cup h)} f_2$, for any $f_1, f_2 \in K[x_1, \dots, x_n]$.

Lemma 3.11 is the key observation for reducing every polynomial $x_i \cdot G_j$ only once in the subsequent algorithm for constructing detaching bases.

Now we are ready to state the improved version of the algorithm:

$G \leftarrow \text{detb2}(F)$

[Algorithm for constructing a detaching basis, 2-version. F is a finite sequence in $K[x_1, \dots, x_n]$. G is a finite detaching basis for $\text{ideal}(F)$]

(1) Let G be a finite, triangular basis for $\text{ideal}(F)$;

[an algorithm for constructing such a basis can be extracted from the proofs of lemma 3.3 and lemma 3.7]

(2) Set $C \leftarrow \{(i, j) \mid 1 \leq i \leq n, 1 \leq j \leq \text{length}(G), (x_i, j) \notin G^\approx \text{ and } x_i \cdot G_j \notin G\}$;

(3) while $C \neq \emptyset$ do

{Choose $(i, j) \in C$;

Let h be a restricted simplified version of $x_i \cdot G_j$ modulo G^* ;

if $h \neq 0$ then {Construct h_1, \dots, h_{m-1} such that $G' = G \cup h_1 \cup \dots \cup h_{m-1}$ is a triangular basis for $\text{ideal}(G \cup h)$;

[use the method described in the proof of lemma 3.6]}

```

Set C ← C ∪ {(i,j) | 1 ≤ i ≤ n, length(G)+1 ≤ j ≤ length(G)+m,
                    (xi,j) ∈ G' and xi.G'j ∉ G'};
Set G ← G';
Delete (i,j) from C;
return •

```

The correctness of the algorithm follows from theorem 3.2 and lemma 3.10. The proof of termination of detb2 is analogous to the one for detb1. One merely has to use lemma 3.9 instead of lemma 2.8.

Example:

We consider the ideal in $\mathbb{Z}_5[x,y,z]$ which is generated by the set of polynomials $F = \{ xy^2z + 3x^2z, xy^3 + xz + 2y, xyz + 2y^2 \}$.

As the linear ordering \prec_t on the set of power products we choose the graduated lexicographic ordering ([Bu79]).

A detaching basis for $\text{ideal}(F)$ is computed first by Buchberger's algorithm GB with criterion 3 (compare [Bu79]) and then by the algorithm detb2.

GB generates the sequence of polynomials (G1)-(G13)

(G1) $xy^2z + 3x^2z$	(G8) $x^2y + 3xz + y$
(G2) $xy^3 + xz + 2y$	(G9) $xz^2 + xy^2 + 2yz$
(G3) $xyz + 2y^2$	(G10) $xz + 2y$
(G4) $y^3 + x^2z$	(G11) xy^2
(G5) $x^3z + 4xz + 3y$	(G12) xy
(G6) x^2y^2	(G13) y^2 .
(G7) $x^2z + 2xy$	

13.12/2 = 78 S-polynomials have to be considered for reduction, but only 21 reductions have to be carried out according to criterion 3.

The algorithm detb2 generates the sequence of polynomials (G'1)-(G'16)

(G'1) $xy^2z + 3x^2z$	(G'9) $x^2y + 3xz + y$
(G'2) $xy^3 + xz + 2y$	(G'10) $xz^2 + xy^2 + 2yz$
(G'3) $xyz + 2y^2$	(G'11) $xz + 2y$
(G'4) $y^3z + x^2z^2$	(G'12) xy^2
(G'5) $y^3 + x^2z$	(G'13) xy
(G'6) $x^3z + 4xz + 3y$	(G'14) y^2z^2
(G'7) x^2y^2	(G'15) y^2z
(G'8) $x^2z + 2xy$	(G'16) y^2 .

23 of the 48 polynomials $v.f$, $v \in \{x,y,z\}$, $f \in G'$, have to be reduced to normal form during the execution of detb2.

Since the reductions are the most time consuming steps in either GB and detb2, the efficiency of the two algorithms for this example is fairly the same (the two extra reductions in detb2 are counterbalanced by the number of tests for criterion 3 in GB).

Unfortunately, up to now it has not been possible to derive upper bounds on the computing time of detb2. The same is true for Buchberger's algorithm for constructing detaching bases (except for n=2, see [Bu79],[BW79]). One has to wait for an implementation of detb2 in order to be able to compare the run time efficiencies of the two algorithms.

References

- [Ay80] C.Ayoub: On Constructing Bases for Ideals in Polynomial Rings over the Integers, Research Report, Dept.Math., Pennsylvania State Univ., 1980
- [Bu65] B.Buchberger: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal, Ph.D. Dissertation, Univ. Innsbruck, 1965
- [Bu70] B.Buchberger: Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems, *Aequ.math.*, vol.4/3, pp.374-383, 1970
- [Bu76] B.Buchberger: A Theoretical Basis for the Reduction of Polynomials to Canonical Forms, *ACM SIGSAM Bull.* 39, pp.19-29, Aug.1976
- [Bu79] B.Buchberger: A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner-Bases, *Proc. EUROSAM'79*, pp.3-21, June 1979
- [BW79] B.Buchberger, F.Winkler: Miscellaneous Results on the Construction of Gröbner-Bases for Polynomial Ideals, *Techn.Rap. Nr. 137*, Inst. f. Math., Univ. Linz, June 1979
- [La76a] M.Lauer: Canonical Representatives for Residue Classes of a Polynomial Ideal, *Proc. 1976 ACM Symp. on Symbolic and Algebraic Computation*, pp.339-345, Aug.1976
- [La76b] M.Lauer: Kanonische Repräsentanten für die Restklassen nach einem Polynomideal, Diplomarbeit, Univ. Kaiserslautern, 1976
- [Sc79] S.C.Schaller: Algorithmic Aspects of Polynomial Residue Class Rings, Ph.D. Dissertation, Univ. Wisconsin-Madison, 1979
- [Si78] C.Sims: The Role of Algorithms in the Teaching of Algebra, in: M.F.Newman (ed.): *Topics in Algebra*, Springer Lecture Notes in Math., Nr.697, pp.95-107, 1978
- [vdW70] B.L.van der Waerden: *Modern Algebra*, vol.2, New York, Ungar, 1970
- [Wi78] F.Winkler: Implementierung eines Algorithmus zur Konstruktion von Gröbner-Basen, Diplomarbeit, Univ. Linz, 1978
- [Wi82] F.Winkler: An Algorithm for Constructing Detaching Bases in the Ring of Polynomials over a Field, *Techn.Rap. Nr. CAMP 82-20.0*, Inst. f. Math., Univ. Linz, December 1982

On the problem of Behā Eddīn 'Amūlī and the
computation of height functions

Horst G. Zimmer
Fachbereich 9 Mathematik
Universität des Saarlandes
D-6600 Saarbrücken

1. In his work "Essence of the art of computing" the islamic mathematician Behā Eddīn 'Amūlī (1547-1622) poses the problem^{*)} of solving the system of quadratic equations - a special case of Euler's equations -

$$x^2 + x + 2 = y^2, \quad x^2 - x - 2 = z^2. \quad (1.1)$$

This system appears among seven problems then believed to be unsolvable, one of them being Fermat's last theorem in the cubic case.

As opposed to Fermat's last theorem however, the above quadratic equations do have rational solutions, e.g. the obvious integral triple

$$(-2, 2, 2)$$

or the fractional triples

$$\left(-\frac{17}{16}, \frac{23}{16}, \frac{7}{16}\right), \quad \left(\frac{34}{15}, \frac{46}{15}, \frac{14}{15}\right)$$

found already during the 19th century. C.J. Scriba calculated a whole bunch of other solutions and asked the question as to whether one could give a complete set of rational triples solving (1.1).

This was shown to be possible in [1] by my students Folz, Kretschmer and Reichert. We shall report about their approach and show how to generalize the corresponding height computations. Since in particular the Néron-Tate height is computable in a sufficiently effective manner, the Behā Eddīn equations can be solved completely in the rationals \mathbb{Q} by making use of a computer.

On putting

$$\xi := x - z, \quad \eta := y - x \quad (1.2)$$

and eliminating x one derives from (1.1) the cubic equation

$$-(2\xi - 1)\eta^2 - 2(\xi^2 + 2)\eta + (\xi^2 + 4\xi) = 0.$$

^{*)}This problem was communicated to the author by C.J. Scriba who performed some hand calculations relating to it.

Application of the birational transformation

$$\xi := \frac{Y-3X+207}{6X+342}, \quad \eta := \frac{-3Y+X^2+69X-450}{-6Y+36X-216} \quad (1.3)$$

turns this into the Weierstrass normal form, with coefficients

$$a := -5211 = -3^3 \cdot 193, \quad b := 31050 = 2 \cdot 3^3 \cdot 5^2 \cdot 23,$$

$$Y^2 = X^3 + aX + b \quad (1.4)$$

of an elliptic curve E over \mathbb{Q} having discriminant

$$\Delta := 4a^3 + 27b^2 = -2^8 \cdot 3^{16} \cdot 7^2 \quad (1.5)$$

and absolute invariant

$$j := 12^3 \frac{4a^3}{\Delta} = \frac{193^3}{3^4 \cdot 7^2}.$$

By the Mordell-Weil theorem the group of rational points $E(\mathbb{Q})$ of E over \mathbb{Q} is a direct sum of the finite group $E_{\text{tor}}(\mathbb{Q})$ of torsion points and a free group $E_{\text{fr}}(\mathbb{Q}) \cong \mathbb{Z}^r$ with finitely many basis elements whose number r is called the rank of E over \mathbb{Q} , i.e.

$$E(\mathbb{Q}) = E_{\text{tor}}(\mathbb{Q}) \oplus E_{\text{fr}}(\mathbb{Q}).$$

Therefore the group $E(\mathbb{Q})$ can be completely determined and hence (1.4) completely solved in \mathbb{Q} by computing the torsion group $E_{\text{tor}}(\mathbb{Q})$, the rank r and a set of r basic rational points of infinite order on E . By virtue of the transformations (1.2) and (1.3) the solutions of (1.4) in \mathbb{Q} can then be traced back to those of the Behā Eddin equations (1.1).

Proposition 1. $E_{\text{tor}}(\mathbb{Q}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$.

Specifically, we have

$$E_{\text{tor}}(\mathbb{Q}) = \{\mathbf{0}, P_1, P_2, P_3\}$$

with points of order two

$$\mathbf{0} = (\infty, \infty), \quad P_1 = (-75, 0), \quad P_2 = (6, 0), \quad P_3 = (69, 0)$$

respectively corresponding to the rational solutions

$$(\infty, \infty, \infty), \quad (-2, -2, 2), \quad (\infty, \infty, \infty), \quad (-2, 2, -2)$$

of the equations (1.1).

The points of order two on E are obtained by solving the right hand side of (1.4). Hence $E_{\text{tor}}(\mathbb{Q})$ contains a group isomorphic to $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. Now, on reducing the curve E modulo 5 where E has good

reduction in accordance with (1.5), we see that the order

$$|E_{\text{tor}}(Q)| \leq 8.$$

Thus $E_{\text{tor}}(Q)$ can at most contain a point of order four. However the only points of order four on E with rational x -coordinates are

$$(33, \pm 324i), \quad (-183, \pm 2268i),$$

hence are complex and do not belong to $E_{\text{tor}}(Q)$. This proves proposition 1.

Proposition 2. $r = 1$.

Corollary. $E(Q) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}$.

Basically, this solves the problem of Behā Eddīn provided one is able to exhibit a generator of the infinite part $E_{\text{fr}}(Q)$ of $E(Q)$. In fact, it is shown in [1] that

$$E_{\text{fr}}(Q) = \langle Q \rangle$$

with the basic point of infinite order

$$Q = (-3, 216)$$

corresponding to the rational solution

$$\left(\frac{34}{15}, \frac{46}{15}, \frac{14}{15}\right)$$

of the equations (1.1). Then the system of all linear combinations over \mathbb{Z} of the points P_1, P_2, P_3 and Q on E leads to a complete set of rational solutions of the Behā Eddīn equations (1.1) via the birational transformations (1.2) and (1.3), except for the finitely many solutions of (1.1) lost when applying the transformations. For instance, the points

$$-2Q = (150, -1620), \quad -Q+P_2 = (573, -13608)$$

on (1.4) give rise to the rational solutions

$$\left(-\frac{17}{16}, \frac{23}{16}, \frac{7}{16}\right), \quad (-2, 2, 2)$$

of (1.1) mentioned at the beginning.

For proving proposition 2 one first shows that the point $Q = (-3, 216)$ of $E(Q)$ has indeed infinite order thus establishing the inequality $r \geq 1$. But Q cannot be a torsion point by proposition 1.

The crucial point in establishing proposition 2 is the proof of the inequality $r \leq 1$. This is accomplished in [1] by a method of Tate

(cf. [2]) as follows. One applies the birational transformation

$$X = 3^2 x' + 6, \quad Y = 3^3 y' \quad (1.6)$$

to (1.4) in order to change the equation of E into the generalized Weierstrass normal form

$$y'^2 = x'^3 + a'x'^2 + b'x' \quad (1.4')$$

with coefficients $a' := 2$, $b' := -63$, defining an isomorphic elliptic curve E' over \mathbb{Q} with points of order two

$$\mathfrak{O}' = (\infty, \infty), \quad P_1' = (-9, 0), \quad P_2' = (0, 0), \quad P_3' = (7, 0)$$

and point of infinite order

$$Q' = (-1, 8).$$

Now E' is 2-isogenous to the elliptic curve E'' over \mathbb{Q} defined by the generalized Weierstrass normal form

$$y''^2 = x''^3 + a''x''^2 + b''x'' \quad (1.4'')$$

with coefficients $a'' := -2a' = -4$, $b'' := a'^2 - 4b' = 256$. For the normal forms (1.4') and (1.4'') of E' and E'' over \mathbb{Q} the familiar Mordell-Weil homomorphism attains the shape

$$\begin{aligned} \varphi': E'(\mathbb{Q}) &\longrightarrow \mathbb{Q}^\times / \mathbb{Q}^{\times 2} \\ P' = (x', y') &\longmapsto \begin{cases} 1 \cdot \mathbb{Q}^{\times 2} & \text{if } x' = \infty \\ b' \cdot \mathbb{Q}^{\times 2} & \text{if } x' = 0 \\ x' \cdot \mathbb{Q}^{\times 2} & \text{if } x' \neq 0, \infty \end{cases} \end{aligned}$$

and analogously φ'' for E'' . Then one has the fundamental relation

$$2^r = \frac{|\varphi'E'(\mathbb{Q})| \cdot |\varphi''E''(\mathbb{Q})|}{4}.$$

Since the images of φ' and φ'' can be constructively determined, this relation yields the desired estimate $r \leq 1$ for the rank of E over \mathbb{Q} (cf. [1]).

The decisive assertion in solving Behā Eddīn's problem is made in

Proposition 3. $E_{fr}(\mathbb{Q}) = \langle Q \rangle$.

2. This is the stage where height computations come into play. We confine ourselves to the field of rational numbers \mathbb{Q} as ground field even though everything we shall discuss can be extended to an arbitrary algebraic number field K in place of \mathbb{Q} proceeding along the lines of [5].

Let for $z \in \mathbb{Q}$ the absolute value v_p , corresponding to a prime p of \mathbb{Q} or to $p = \infty$, be given by

$$v_p(z) = -\log|z|_p ,$$

where $| \cdot |_p$ denotes the normalized p -adic valuation or ordinary absolute value on \mathbb{Q} according as $p \neq \infty$ or $p = \infty$. We set for any E over \mathbb{Q} defined by (1.4)

$$\mu_p := \min\left\{\frac{1}{2}v_p(a), \frac{1}{3}v_p(b)\right\} \quad (2.1)$$

as p ranges over all primes p of \mathbb{Q} including ∞ . Then the local Weil height of a point $P = (x, y) \in E(\mathbb{Q})$ is

$$d_p(P) := \begin{cases} -\frac{1}{2} \min\{\mu_p, v_p(x)\} & \text{if } P \neq \emptyset \\ -\frac{1}{2}\mu_p & \text{if } P = \emptyset \end{cases} \quad (2.2)$$

and the global Weil height of P is

$$d(P) := \sum_p d_p(P) \quad (2.3)$$

the sum being taken over all primes p of \mathbb{Q} including ∞ . Writing the coordinates of a point $P = (x, y) \in E(\mathbb{Q})$ in the shape

$$x = \frac{\xi^2}{\zeta^2}, \quad y = \frac{\eta}{\zeta^3} \quad (2.4)$$

for relatively prime integers $(\xi, \zeta) = (\eta, \zeta) = 1$ we also introduce the modified Weil height of P setting

$$d_\infty(P) := \begin{cases} -\frac{1}{2} \min\{\mu_\infty + v_\infty(\zeta^2), v_\infty(\xi)\} & \text{if } P \neq \emptyset \\ -\frac{1}{2}\mu_\infty & \text{if } P = \emptyset \end{cases} \quad (2.5)$$

The global Néron-Tate height of $P \in E(\mathbb{Q})$ can now be defined on the basis of the Weil heights by the limit formulae

$$\delta(P) := \lim_{n \rightarrow \infty} \frac{d(2^n P)}{2^{2n}} = \lim_{n \rightarrow \infty} \frac{d_\infty(2^n P)}{2^{2n}} . \quad (2.6)$$

With the abbreviating notations

$$\mu := -\sum_p \mu_p \quad \text{and} \quad \alpha := -v_\infty(2) = \log 2 \quad (2.7)$$

we then obtain the estimates (see [4], [5], [6])

$$-\frac{2}{3}\alpha \leq d(P) - \delta(P) \leq \frac{5}{3}\alpha + \mu \quad (2.8)$$

and

$$-\frac{2}{3}\alpha \leq d_\infty(P) - \delta(P) \leq \frac{5}{3}\alpha + \mu - \frac{1}{2}\mu_\infty \quad (2.8_\infty)$$

for the difference of the Weil heights and the Néron-Tate height on $E(\mathbb{Q})$. Note in connection with (2.8_∞) that $-\mu_\infty \geq 0$ by (2.1).

The Néron-Tate height is a decisive tool in treating diophantine problems. At first it is a quadratic form (see [6], [7]).

Proposition 4. For $P, Q \in E(\mathbb{Q})$, we have

$$\delta(P+Q) + \delta(P-Q) = 2\delta(P) + 2\delta(Q). \quad (2.9)$$

This implies the

Corollary. For $P \in E(\mathbb{Q})$ and $m \in \mathbb{N}$, we have

$$\delta(mP) = m^2 \delta(P). \quad (2.10)$$

The quadratic form δ on $E(\mathbb{Q})$ leads to a symmetric bilinear form λ via

$$\lambda(P, Q) := \frac{1}{2} \{ \delta(P+Q) - \delta(P) - \delta(Q) \} \quad (2.11)$$

for any $P, Q \in E(\mathbb{Q})$. In view of propositions 1 and 3 the functions δ and λ yield the following important characterizations (see [7]).

Proposition 5.

$$P \in E_{\text{tor}}(\mathbb{Q}) \quad \text{iff} \quad \delta(P) = 0. \quad (2.12)$$

$$Q_1, \dots, Q_n \in E_{\text{fr}}(\mathbb{Q}) \text{ are linearly independent over } \mathbb{Z} \text{ iff} \\ \det(\lambda(Q_\mu, Q_v))_{\mu, v=1, \dots, n} \neq 0. \quad (2.13)$$

If in particular E has rank r and $Q_1, \dots, Q_r \in E(\mathbb{Q})$ form a basis of $E_{\text{fr}}(\mathbb{Q})$, the quantity

$$\mathfrak{R} := |\det(\lambda(Q_\mu, Q_v))_{\mu, v=1, \dots, r}| > 0$$

is called the regulator of E over \mathbb{Q} .

We are now in a position to prove proposition 3. If $Q = (-3, 216)$ were not a generator of the infinite part of $E(\mathbb{Q})$ there would be an-

other point $R_1 = (x_1, y_1) \in E(\mathbb{Q})$ such that

$$Q = mR_1 \quad \text{for some } m \in \mathbb{N}, m > 1. \quad (2.14)$$

We shall employ the height functions on $E(\mathbb{Q})$ in order to get estimates for its first coordinate x_1 and then show that within the range of x_1 there does not exist any other rational point R_1 on E thus establishing proposition 3.

One way of doing this is the method of "infinite descent" used in the proof of the Mordell-Weil theorem (see [4], [5]). It yields for the modified Weil height of a potential point $R_1 \in E(\mathbb{Q})$ with (2.14) the upper bound

$$d_\infty(R_1) \leq 1 + \left(\frac{5}{3}\alpha + \mu - \frac{1}{2}\mu_\infty \right) + 2(2r-1)r\alpha + (2r-1)rd_\infty(Q) \quad (2.15)$$

which is slightly stronger than the one given in [4]. Since $r = 1$ by proposition 2 we obtain

$$d_\infty(R_1) \leq 1 + \frac{11}{3}\alpha + \mu - \frac{1}{2}\mu_\infty + d_\infty(Q). \quad (2.15')$$

Here we have by (2.1), (1.4) and (2.7)

$$\mu_\infty = -\frac{1}{2}\log 5211 \quad \text{and} \quad \mu = -\mu_3 - \mu_\infty = \frac{1}{2}\log 579.$$

By (2.4), (2.5) we get for the modified Weil height of Q

$$d_\infty(Q) = -\frac{1}{2}\min\{\mu_\infty + v_\infty(1), v_\infty(-3)\} = \frac{1}{4}\log 5211.$$

Altogether, by (2.7), this gives us

$$d_\infty(R_1) \leq 1 + \frac{11}{3}\log 2 - \log 3 + \log 5211 \approx 11.001,454,43.$$

Since, by virtue of (2.5),

$$d_\infty(R_1) = \frac{1}{2}\max\left\{\frac{1}{2}\log 5211 + 2\log|\zeta_1|_\infty, \log|\xi_1|_\infty\right\} \quad (2.16)$$

if we write the coordinates x_1, y_1 of R_1 in terms of relatively prime integers ξ_1, η_1, ζ_1 as in (2.4), we end up with the upper bounds

$$|\xi_1|_\infty \leq 3,595,356,037 \quad \text{and} \quad |\zeta_1|_\infty \leq 7,057. \quad (2.17)$$

These bounds are too coarse in order to be used in a search procedure for points $R_1 \in E(\mathbb{Q})$ satisfying (2.14).

A different way of denying the existence of a point $R_1 \in E(\mathbb{Q})$ such that (2.14) holds is as follows. From the estimate (2.8 _{∞}) we gather by virtue of the property (2.10) of δ that we have

$$-\frac{2}{3}\alpha + \frac{1}{m^2} d(Q) \leq d_\infty(R_1) \leq \frac{5}{3}\alpha + \mu - \frac{1}{2}\mu_\infty + \frac{1}{m^2} d(Q) . \quad (2.18)$$

Thus we shall obtain new bounds for the modified Weil height d_∞ of the unknown point $R_1 \in E(\mathbb{Q})$ as soon as we shall be able to compute the Néron-Tate height δ of the known point $Q = (-3, 216) \in E(\mathbb{Q})$. This latter can be done approximately on the basis of the limit relation (2.6). This time we rely on the estimate (2.8) instead of (2.8_o) since it gives us sharper bounds. Applying (2.8) to a sufficiently large multiple mQ of the point Q and again utilizing property (2.10) of δ we conclude that

$$\frac{-\frac{5}{3}\alpha - \mu + d(mQ)}{m^2} \leq \delta(Q) \leq \frac{\frac{2}{3}\alpha + d(mQ)}{m^2} . \quad (2.19)$$

The size of the chosen $m \in \mathbb{N}$ depends on the capacity of the computer used for calculating the multiples of Q . Since it is less expensive to duplicate points we take $m := 2^n$ and choose $n := 7$, hence $m = 128$. The calculations were carried out by means of the SAC-2/ALDES algorithm system.

The point $128Q = (x_{128}, y_{128})$ has first coordinate

$$x_{128} = \frac{\xi_{128}}{\zeta_{128}^2} \approx \frac{1.724,406,122 \cdot 10^{5310}}{5.872,809,533 \cdot 10^{5307}}$$

thus giving the Weil height

$$d(128Q) = \frac{1}{2} \log \xi_{128} \approx 6,113,635,865 .$$

Hence the value to be used in (2.19) is

$$\frac{1}{128^2} d(128Q) \approx 0.373,146,672 . \quad (2.20)$$

Therefore, with $m = 128$, (2.19) reads numerically

$$0.372,882,078 \leq \delta(Q) \leq 0.373,174,925 \quad (2.21)$$

thus yielding the desired approximation of the Néron-Tate height. Now (2.14) can at most be true for $m \geq 3$. Choosing $m = 3$ and inserting the upper and lower bounds of (2.21) for $\delta(Q)$ into (2.18) leads to the estimate

$$-0.420,634,240 \leq d_\infty(R_1) \leq 6.516,992,185 . \quad (2.22)$$

By (2.16) this gives for the coordinates of R_1 the upper bounds

$$|\xi_1|_\infty \leq 457,706 \quad \text{and} \quad |\zeta_1|_\infty \leq 79 \quad (2.17')$$

improving (2.17). This is still not good enough for a search procedure. However we can further strengthen the bound for $|\zeta_1|$ to $\zeta_1 = 1$ by the lemma of E. Lutz (cf. [3]) stating that if R_1 had a fractional first coordinate so would Q by relation (2.14). Hence we are left with looking for integers x_1, y_1 such that

$$|x_1| \leq 457,706 \quad (2.23)$$

and $R_1 = (x_1, y_1) \in E(\mathbb{Q})$.

Since the polynomial on the right hand side of (1.4) assumes negative values outside the open intervals $(-75, 6)$ and $(69, \infty)$ we conclude that

$$-75 < x_1 < 6 \quad \text{or} \quad 69 < x_1. \quad (2.24)$$

Now we pass to the isomorphic curve E' over \mathbb{Q} defined by (1.4'). The point $R'_1 = (x'_1, y'_1) \in E'(\mathbb{Q})$ corresponding to $R_1 = (x_1, y_1) \in E(\mathbb{Q})$ via the transformation (1.6) now satisfies the conditions

$$-9 < x'_1 < 0 \quad \text{or} \quad 7 < x'_1 \leq 50,856 \quad (2.25)$$

arising from (2.23) and (2.24). Congruence considerations show that furthermore

$$x'_1 \equiv 0, 1 \text{ or } 3 \pmod{4} \quad \text{and} \quad x'_1 \equiv 0, 2, 5 \text{ or } 6 \pmod{7}$$

which together with (2.25) reduces the amount of values x'_1 searched for to about 21,800. It took four hours of computing time on a HP-41C to check these values turning up no such point $R'_1 \in E'(\mathbb{Q})$ and hence no such point $R_1 \in E(\mathbb{Q})$. This proves proposition 3.

We remark that it would probably have been better to work from scratch with the normal form (1.4') instead of (1.4) hoping to derive this way better bounds than the above. Note that (1.4') is the global minimal model of E .

3. What impact does the example of Behā Eddīn have on the general case? At first we must be able to approximately calculate the Néron-Tate height δ on the rational point group $E(\mathbb{Q})$ of an arbitrary elliptic curve E over \mathbb{Q} .

The method proposed in [4], [5] is based on the relation

$$\delta(P) = d_\infty(P) + \lim_{n \rightarrow \infty} \sum_{v=1}^n \frac{d_\infty(2^{v-1}P, 2^{v-1}P)}{2^{2v}}$$

with

$$d_\infty(2^{v-1}P, 2^{v-1}P) := d_\infty(2^v P) - 4d_\infty(2^{v-1}P) \quad (3.1)$$

estimated according to (cf. [6])

$$-5\alpha - 3\mu + 2\mu_\infty \leq d_\infty(2^{v-1}P, 2^{v-1}P) \leq 2\alpha - \frac{1}{2}\mu_\infty . \quad (3.2)$$

Since we have

$$\delta(P) - d_\infty(P) - \sum_{v=1}^n \frac{d_\infty(2^{v-1}P, 2^{v-1}P)}{2^{2v}} = \delta(P) - \frac{d_\infty(2^n P)}{2^{2n}}$$

the estimates (3.2) lead to the inequalities

$$\frac{1}{2^{2n}} (-\frac{5}{3}\alpha - \mu + \frac{2}{3}\mu_\infty) \leq \delta(P) - \frac{d_\infty(2^n P)}{2^{2n}} \leq \frac{1}{2^{2n}} (\frac{2}{3}\alpha - \frac{1}{6}\mu_\infty) . \quad (3.3)$$

However the bounds in (3.3) are coarser than those used in the Behá Eddín example. If we employ instead the formula

$$\delta(P) = d_\infty(P) + \sum_{v=1}^n \frac{d_\infty(2^{v-1}P, 2^{v-1}P)}{2^{2v}} + \frac{\delta(2^n P) - d_\infty(2^n P)}{2^{2n}}$$

formed in analogy to [7] and use the estimates (2.8 _{∞}) and (3.2) we end up with the inequalities

$$(-\frac{5}{3}\alpha - \mu + \frac{1}{2}\mu_\infty) + \frac{1}{6}\mu_\infty (1 - \frac{1}{2^{2n}}) \leq \delta(P) - d_\infty(P) \leq \frac{2}{3}\alpha - \frac{1}{6}\mu_\infty (1 - \frac{1}{2^{2n}})$$

which are coarser than (2.8 _{∞}).

Therefore the best approach for calculating the Néron-Tate height seems to be the one employed in the Behá Eddín example. It is based on the second limit relation (2.6). Denoting the bounds in (2.8 _{∞}) by

$$c_1 := -\frac{5}{3}\alpha - \mu + \frac{1}{2}\mu_\infty , \quad c_2 := \frac{2}{3}\alpha \quad (3.4)$$

we infer from (2.8 _{∞}) by virtue of property (2.10) of δ that

$$\frac{c_1 + d_\infty(2^n P)}{2^{2n}} \leq \delta(P) \leq \frac{c_2 + d_\infty(2^n P)}{2^{2n}} \quad (3.5)$$

with bounds converging to $\delta(P)$ by (2.6). The rate of convergence is seen from the following estimation of the differences of two consecu-

tive bounds in (3.5) belonging to $n-1$ and n . Using the notation of (3.1) we have

$$\frac{\frac{1}{2}\mu_\infty}{2^{2n}} \leq \pm \frac{d_\infty(2^{n-1}P, 2^{n-1}P) - 3c_v}{2^{2n}} \leq \frac{7\alpha + 3\mu - 2\mu_\infty}{2^{2n}} \quad (3.6)$$

with the plus or minus sign according as $v = 1$ or 2 respectively.

Comparison of (3.3) and (3.4), (3.5) shows that (3.5) gives the better bounds because, as observed earlier, $-\mu_\infty \geq 0$. Hence the latter approach is the more effective one. Yet the numerical results of section 2 show that convergence of the bounds in (3.4) towards $\delta(P)$ in accordance with (2.6) can be rather slow. Also it is tedious to compute the multiples $2^n P$ of a given point $P \in E(\mathbb{Q})$ since the coordinates grow fast with n (see the formulas preceding (2.20)). At any rate the speed of convergence is measured by (3.6).

The above estimates could be adjusted to the ordinary Weil height d instead of the modified Weil height d_∞ and would even involve somewhat sharper bounds. In fact the computations of section 2 were carried out mainly with d instead of d_∞ . Yet we have decided to work with d_∞ in this section because in general d_∞ is easier to calculate. For the calculation of d_∞ requires only knowledge of the representation (2.4) of the coordinates of rational points and not their prime decompositions necessary for computing d . Of course, the difference between d_∞ and d on a multiple of a point $2^n P$ disappears for sufficiently high n if divided by 2^{2n} as is clear from (2.6).

It remains to consider the problem of finding r basis elements of infinite order in $E(\mathbb{Q})$ for an elliptic curve E of rank r over \mathbb{Q} .

If one knows already r independent points of infinite order $Q_1, \dots, Q_r \in E(\mathbb{Q})$ as in the Beha Eddin example it is basically possible according to the infinite descent method (cf. [4], [5]) to find r basis points of infinite order $R_1, \dots, R_r \in E(\mathbb{Q})$ among the $R \in E(\mathbb{Q})$ satisfying (cf. (2.15) above)

$$d_\infty(R) \leq 1 + (\frac{5}{3}\alpha + \mu - \frac{1}{2}\mu_\infty) + 2(2r-1)r\alpha + (2r-1)r \max_{v=1, \dots, r} \{d_\infty(Q_v)\}. \quad (3.7)$$

For finding a basis of $E_{fr}(\mathbb{Q})$ starting from the known points $Q_1, \dots, Q_r \in E(\mathbb{Q})$ one has to minimize the determinant

$$\det(\lambda(Q_\mu, Q_v))_{\mu, v=1, \dots, r}$$

in accordance with (2.13) of proposition 5. Representing the Q_μ in

terms of the generators R_v as linear combinations over \mathbb{Z} according to

$$Q_{\mu^t} = \sum_{v=1}^r m_{\mu^t v} R_v \quad (\mu^t = 1, \dots, r) \quad (3.8)$$

we get a regular matrix over \mathbb{Z}

$$M = (m_{\mu^t v})_{\mu^t, v=1, \dots, r} \quad (3.8')$$

whose determinant gives the index of the subgroup generated by Q_1, \dots, Q_r in $E(\mathbb{Q})$:

$$[E(\mathbb{Q}) : \langle Q_1, \dots, Q_r \rangle] = |\det M| . \quad (3.9)$$

The determinants of the sets of points are related by

$$\det(\lambda(Q_{\mu^t}, Q_v)) = \det(m_{\mu^t v})^2 \cdot \det(\lambda(R_{\mu^t}, R_v)) . \quad (3.10)$$

This relation guides the required minimization which finally leads to the regulator of E over \mathbb{Q} .

One can try to generalize the method of section 2, leading there to the bounds (2.22) for d_∞ on a possible basis point R_1 , in order to find corresponding bounds for d_∞ of possible generators R_1, \dots, R_r instead of the above (3.7). By (3.9) we would then have

$$\det M \cdot R_{\mu^t} = \sum_{v=1}^r n_{\mu^t v} Q_v \quad (\mu^t = 1, \dots, r) \quad (3.11)$$

with a regular matrix over \mathbb{Z}

$$N = (n_{\mu^t v})_{\mu^t, v=1, \dots, r} \quad (3.11')$$

such that

$$M^{-1} = \frac{N}{\det M} . \quad (3.12)$$

We set

$$M^{-1} := M' = (m'_{\mu^t v})_{\mu^t, v=1, \dots, r} \quad (3.13)$$

to obtain a matrix over \mathbb{Q} .

Let us now apply (2.8 _{∞}) to the potential basis points of infinite order R_1, \dots, R_r and utilize (3.11). This yields

$$\begin{aligned} -\frac{2}{3}\alpha + \frac{1}{\det M^2} \cdot \delta \left(\sum_{v=1}^r n_{\mu^t v} Q_v \right) &\leq d_\infty(R_{\mu^t}) \\ &\leq \frac{5}{3}\alpha + \frac{1}{2}\mu_\infty + \frac{1}{\det M^2} \cdot \delta \left(\sum_{v=1}^r n_{\mu^t v} Q_v \right) . \end{aligned} \quad (3.14)$$

The quadratic form property (2.9) of δ and the definition (2.11) of λ enable us to evaluate in (3.14)

$$\delta\left(\sum_{v=1}^r n_{\mu v} Q_v\right) = \sum_{v, \rho=1}^r n_{\mu v} n_{\mu \rho} \lambda(Q_v, Q_\rho)$$

thus entailing by (3.11'), (3.12), (3.13)

$$\frac{1}{\det M^2} \cdot \delta\left(\sum_{v=1}^r n_{\mu v} Q_v\right) = \sum_{v, \rho=1}^r m'_{\mu v} m'_{\mu \rho} \lambda(Q_v, Q_\rho). \quad (3.15)$$

Therefore if we are given r independent points of infinite order $Q_1, \dots, Q_r \in E(\mathbb{Q})$ generating a subgroup of rank r in $E(\mathbb{Q})$ and want to find a basis R_1, \dots, R_r of the full group $E(\mathbb{Q})$ we shall have to check for all possible regular integral matrices M in (3.8') arising from (3.8) if there are points $R_\mu \in E(\mathbb{Q})$ satisfying (3.14) with the Néron-Tate height δ evaluated from (3.15) and (3.13) by virtue of the approximation method described at the beginning of this section. Since δ is computable on the known points Q_1, \dots, Q_r so is the bilinear form λ defined by (2.11). Reduction methods may be applied in order to cut down the number of matrices M to be checked as in the case $r = 1$ of section 2. Perhaps (3.14) supplies better bounds than those of (3.7) obtained by the infinite descent method. This was at least the case for $r = 1$ in the Behā Eddīn example.

In conclusion we remark that, as pointed out earlier, the above exposition can be extended without trouble to elliptic curves E over an algebraic number field K in place of \mathbb{Q} proceeding along the lines of [5].

References

- [1] H.G. Folz, T. Kretschmer, M. Reichert, Zu dem Problem von Behā Eddīn 'Amūlī. Manuscript, Saarbrücken 1982.
- [2] J. Tate, Rational points on elliptic curves. Philips Lectures, Haverford College, 1961.
- [3] H.G. Zimmer, Die Néron-Tateschen quadratischen Formen auf der rationalen Punktgruppe einer elliptischen Kurve. J. Number Theory 2 (1970), 459-499.
- [4] H.G. Zimmer, On Manin's conditional algorithm. Calculateurs en Math., Limoges, 1975. Bull. Soc. Math. France, Mémoire 49-50 (1977), 211-224.
- [5] H.G. Zimmer, Generalization of Manin's conditional algorithm. Proc. of the 1976 ACM Sympos. Symb. Alg. Comp., Yorktown Heights, New York, 1976, 285-299.

- [6] H.G. Zimmer, On the difference of the Weil height and the Néron-Tate height. *Math. Z.* 147 (1976), 35-51.
- [7] H.G. Zimmer, Lokale Höhenfunktionen auf elliptischen Kurven. *Abh. Braunschweig. Wiss. Ges.* 33 (1982), 253-260.

A PROCEDURE FOR DETERMINING ALGEBRAIC

INTEGERS OF GIVEN NORM

U. Fincke and M. Pohst

Mathematisches Institut
Universität Düsseldorf
Universitätsstr. 1
4 Düsseldorf, West Germany

1. Introduction:

Let F be an algebraic number field of degree n . Let s denote the number of real, $2t$ the number of complex conjugates of F ordered in the usual way such that

$$F^{(1)}, \dots, F^{(s)} \subseteq \mathbb{R}, F^{(s+t+j)} = \overline{F^{(s+j)}} \subseteq \mathbb{C} \quad (1 \leq j \leq t).$$

Let R be a subring of the ring of algebraic integers in F which contains \mathbb{Z} and is a free \mathbb{Z} -module of rank n :

$$R = \mathbb{Z} w_1 + \dots + \mathbb{Z} w_n.$$

One of the key problems in algebraic number theory is the solution of norm equations: For given $K \in \mathbb{Z}^{\geq 2}$ we want to determine $\beta \in R$ with

$$(1) |N(\beta)| = |\prod_{i=1}^n \beta^{(i)}| = K,$$

where $\beta^{(1)}, \dots, \beta^{(n)}$ denote the conjugates of $\beta = \beta^{(1)}$.

This problem for example arises in all class number and class group computations.

Our method of solving (1) bases on K. Mahler's "ceilings" (introduced in [5]) which allow to transform the product in (1) into a positive definite quadratic form depending on n real parameters $\underline{\lambda} = (\lambda_1, \dots, \lambda_n)$. A solution of (1) uniquely determines $\underline{\lambda}$. By appropriate methods of mathematical optimization we obtain only a small number of vectors $\underline{\lambda}$ which possibly yield a solution of (1). For each such $\underline{\lambda}$ it is easily tested, whether it corresponds to a solution of (1).

2. Relaxation and separation of the problem.

Each $\beta \in R$ has a presentation

$$(2) \quad \beta = x_1 w_1 + \dots + x_n w_n \quad (x_j \in \mathbb{Z}, 1 \leq j \leq n).$$

Writing $\langle \underline{w}^{(i)}, \underline{x} \rangle := \sum_{j=1}^n x_j w_j^{(i)} = \beta^{(i)} \quad (1 \leq i \leq n)$ for abbreviation,

(1) is equivalent to

$$(3) \quad |\beta| = \left| \prod_{i=1}^s \langle \underline{w}^{(i)}, \underline{x} \rangle \right| \prod_{i=s+1}^{s+t} \left| \langle \underline{w}^{(i)}, \underline{x} \rangle \right|^2 = K$$

$$(\underline{x} \in \mathbb{Z}^n, \underline{x} \neq \underline{0}).$$

If (3) has a solution, then - as a consequence of the following lemma - there is also a solution \underline{x} of (3) for which the coordinates x_i of \underline{x} are bounded. Because of $x_i \in \mathbb{Z} \quad (1 \leq i \leq n)$ there are only finitely many possibilities for \underline{x} .

Lemma 1. Let $\varepsilon_1, \dots, \varepsilon_{s+t-1}$ be a system of independent units of R and (3) be solvable. Then there exists

$\beta = x_1 w_1 + \dots + x_n w_n \in R$ and $L_i, U_i \in \mathbb{R}^{>0} \quad (1 \leq i \leq n)$ satisfying (1) and

$$(4) \quad L_i \leq |\beta^{(i)}| \leq U_i$$

$$\text{for } L_i := \exp\left(\frac{1}{n} \log K - \frac{1}{2} \sum_{j=1}^{s+t-1} |\log|\varepsilon_j^{(i)}||\right),$$

$$U_i := \exp\left(\frac{1}{n} \log K + \frac{1}{2} \sum_{j=1}^{s+t-1} |\log|\varepsilon_j^{(i)}||\right)$$

$$(1 \leq i \leq n).$$

A proof of Lemma 1 is given in [1]. We easily derive bounds for the coefficients $x_j \quad (1 \leq j \leq n)$ of β , for example, via a dual basis of R .

Since the amount of computations necessary to solve (3) strongly depends on the size of $L_i, U_i \quad (1 \leq i \leq n)$, $\varepsilon_1, \dots, \varepsilon_{s+t-1}$ are to be chosen such that

$$\sum_{j=1}^{s+t-1} |\log|\varepsilon_j^{(i)}|| \quad (1 \leq i \leq n)$$

respectively their product, become small. Thus it would be of advantage, if $\varepsilon_1, \dots, \varepsilon_{s+t-1}$ were fundamental units of R . However, it is very time consuming to show that $s + t - 1$ independent units are already

fundamental units. Hence, in general we will be satisfied with $s+t-1$ independent units which are fundamental with high probability. They can be computed by the methods of [6], [7]. In the sequel we always assume that $\varepsilon_1, \dots, \varepsilon_{s+t-1}$ are independent units of R and L_i, U_i ($1 \leq i \leq n$) the constants defined in Lemma 1.

The next lemma transforms the problem of solving (3) into one which turns out to be easier to handle.

Lemma 2. Let (3) be solvable. Then there exists $\beta = x_1 w_1 + \dots + x_n w_n = \langle \underline{w}, \underline{x} \rangle \in R$ satisfying (1) and (4), and $\lambda \in (\mathbb{R}^{>0})^{s+t}$ subject to

$$(5) \quad \prod_{i=1}^s \lambda_i \prod_{i=s+1}^{s+t} \lambda_i^2 = 1,$$

$$(6) \quad \sum_{i=1}^s \lambda_i |\langle \underline{w}^{(i)}, \underline{x} \rangle|^2 + 2 \sum_{i=s+1}^{s+t} \lambda_i |\langle \underline{w}^{(i)}, \underline{x} \rangle|^2 = nK^{2/n},$$

$$(7) \quad U_i^{-2} K^{2/n} \leq \lambda_i \leq L_i^{-2} K^{2/n} (1 \leq i \leq s+t).$$

Proof. According to Lemma 1 there exists $\beta = \langle \underline{w}, \underline{x} \rangle \in R$ satisfying $|N(\beta)| = K$ and (4). For this algebraic integer β we define:

$$\lambda_i := |\langle \underline{w}^{(i)}, \underline{x} \rangle|^{-2} K^{2/n} \in \mathbb{R}^{>0} (1 \leq i \leq n).$$

Then (5), (6), (7) are an immediate consequence of (3), (4). □

Remark. The transformation of (3) into (5), (6) follows an idea of K. Mahler [5].

Instead of solving (3) directly we try to solve (6) subject to (4), (5), (7) instead. This turns out to be easier, since we can prove that it suffices to consider the quadratic form in (6) for very few vectors $\lambda = (\lambda_1, \dots, \lambda_{s+t})$.

Theorem 1. Let (3) be solvable and λ, R_i, S_i ($1 \leq i \leq n$) defined by:

$$(8) \quad \lambda \in \mathbb{R}^{>1} \text{ is a (unique) zero of}$$

$$(1-h(\lambda)) \lambda^{h(\lambda)} + h(\lambda) \lambda^{h(\lambda)-1} - (1 + \frac{1}{K})^{2/n} = 0$$

$$\text{for } h(\lambda) := \frac{\lambda}{\lambda-1} - \frac{1}{\log \lambda},$$

$$(9) \quad R_i := \lfloor \frac{2}{\log \lambda} (\frac{1}{n} \log K - \log U_i) \rfloor,$$

$$S_i := \lceil \frac{2}{\log \lambda} (\frac{1}{n} \log K - \log L_i) \rceil = -R_i (1 \leq i \leq n).$$

Then there exists $\beta = \langle \underline{w}, \underline{x} \rangle \in R$ with $|N(\beta)| = K$ satisfying

$$(10) \quad \sum_{i=1}^n \lambda^{r_i} |\langle \underline{w}^{(i)}, \underline{x} \rangle|^2 \leq n(K+1)^{2/n},$$

where r_1, \dots, r_n are rational integers subject to

$$(11a) \quad \sum_{i=1}^n r_i = 0,$$

$$(11b) \quad R_i \leq r_i \leq S_i \quad (1 \leq i \leq n),$$

$$(11c) \quad r_{s+t+i} - r_{s+i} \in \{0, 1\} \quad (1 \leq i \leq t) \text{ and}$$

$$\#\{i | r_{s+t+i} - r_{s+i} = 1, 1 \leq i \leq t\} \in \{0, 1\}.$$

Proof. It is easily seen that $h(\lambda) := \frac{\lambda}{\lambda-1} - \frac{1}{\log \lambda}$ is in $(0, 1)$ for $\lambda > 1$. Then for $x \in (0, 1)$, $\lambda > 1$ the function $f(x, \lambda) := (1-x)\lambda^x + x\lambda^{x-1}$ is strictly increasing in λ for fixed x and concave in x for fixed λ with a maximum for $x_\lambda := \frac{\lambda}{\lambda-1} - \frac{1}{\log \lambda}$. Therefore $f(x_\lambda, \lambda)$ is strictly increasing in λ . Because of $f(x_{1+}, 1+) = 1$ we obtain a unique zero λ of $f(x_\lambda, \lambda) - (1 + \frac{1}{K})^{2/n} = 0$, hence (8).

We define $y_i := K^{-2/n} |\beta^{(i)}|^2$ ($1 \leq i \leq n$) for $\beta \in R$ satisfying the conditions of Lemma 2. We note that $\prod_{i=1}^n y_i = 1$ because of $|N(\beta)| = K$. We can represent y_i by λ in the form

$$y_i = \lambda^{-\tilde{r}_i + \tilde{\varepsilon}_i} (0 \leq \tilde{\varepsilon}_j < 1, \tilde{r}_i \in \mathbb{Z}, \sum_{i=1}^n \tilde{r}_i = \sum_{i=1}^n \tilde{\varepsilon}_i = 0, \\ 1 \leq j \leq n-1, 1 \leq i \leq n).$$

By subtracting 1 from $\tilde{\varepsilon}_i$ for suitable indices $i \in \{1, \dots, n-1\}$ and changing $\tilde{r}_i, \tilde{\varepsilon}_n, \tilde{r}_n$ correspondingly, we obtain $r_1, \dots, r_n \in \mathbb{Z}$, $a \in (0, 1)$ and $\varepsilon_1, \dots, \varepsilon_n \in [a-1, a]$ such that $y_i = \lambda^{-r_i + \varepsilon_i} (1 \leq i \leq n)$ and (11a), (11c) are satisfied. Then (11b), (9) are an easy consequence of (7).

Finally we prove (10). Namely,

$$\begin{aligned} \sum_{i=1}^n \lambda^{r_i} y_i &= \sum_{i=1}^n \lambda^{\varepsilon_i} \leq \sum_{i=1}^n ((1-(a-\varepsilon_i))\lambda^a + (a-\varepsilon_i)\lambda^{a-1}) \\ &\leq n(1-h(\lambda))\lambda^{h(\lambda)} + n h(\lambda)\lambda^{h(\lambda)-1} \\ &= n(1 + \frac{1}{K})^{2/n}. \end{aligned}$$

The first inequality follows from the convexity of the exponential function, the second by $f(x, \lambda) \leq f(x_\lambda, \lambda)$, and the last equation by (8).

□

For the computations it is of advantage to split the matrix of coefficients of the quadratic form in (10):

Lemma 3. The quadratic form in (10) satisfies

$$(12) \quad \sum_{i=1}^n \lambda^{r_i} |\langle \underline{w}^{(i)}, \underline{x} \rangle|^2 = \underline{x} \underline{U} \underline{\lambda}^2 \underline{U}^T \underline{x} \text{ for}$$

$$\underline{U}^T := \begin{pmatrix} \underline{w}^{(1)} \\ \vdots \\ \underline{w}^{(s)} \\ \operatorname{Re} \underline{w}^{(s+1)} \\ \operatorname{Im} \underline{w}^{(s+1)} \\ \vdots \\ \operatorname{Re} \underline{w}^{(s+t)} \\ \operatorname{Im} \underline{w}^{(s+t)} \end{pmatrix} \quad \text{and}$$

$$\underline{\lambda} := (v_1, \dots, v_s, z_1, \dots, z_{2t}), \quad \underline{D}_{\underline{\lambda}} = \operatorname{diag}(v_1, \dots, v_s, z_1, \dots, z_{2t}),$$

where

$$v_i := \lambda^{r_i/2} (1 \leq i \leq s), \quad z_{2i-1} = z_{2i} := (\lambda^{r_{s+i}} + \lambda^{r_{s+t+i}})^{1/2} \quad (1 \leq i \leq t)$$

and

$$\operatorname{Re} \underline{w}^{(s+i)} := (\operatorname{Re} w_1^{(s+i)}, \dots, \operatorname{Re} w_n^{(s+i)}),$$

$$\operatorname{Im} \underline{w}^{(s+i)} := (\operatorname{Im} w_1^{(s+i)}, \dots, \operatorname{Im} w_n^{(s+i)}) \quad (1 \leq i \leq t).$$

Proof. By straightforward computation.

3. Worst case analysis.

The number of quadratic forms (12) which must be considered is roughly bounded by

$$(13) \quad (t+1) \left(3 - \frac{4 \log K}{n \log \lambda} + \frac{4 \log B}{\log \lambda} \right)^{s+t-1}$$

in case $U_i \leq B$ ($1 \leq i \leq n$). This follows from the conditions for r_i ($1 \leq i \leq n$) of Theorem 1. We note that (13) yields a reasonable bound only in case λ is not close to 1. This imposes conditions for the size of n and K .

For each quadratic form Q_λ obtained we need to determine the set

$$S_\lambda := \{\underline{x} \in \mathbb{Z}^n \setminus \{0\} \mid nK^{2/n} \leq \underline{x} U D_\lambda^{\text{tr}} \underline{x}^{\text{tr}} \leq n(K+1)^{2/n}\}$$

according (12) and (10). Also by (10) S_λ can contain only vectors \underline{x} with $|N(\langle \underline{w}, \underline{x} \rangle)| \leq K+1$ and the probability for $|N(\langle \underline{w}, \underline{x} \rangle)| = K+1$ ($\underline{x} \in S_\lambda$) is practically 0. Hence, S_λ contains only very few points which are then good candidates for solving (3).

Theorem 2. Denote $D_0 := \text{diag}(1, \dots, 1, \sqrt{2}, \dots, \sqrt{2}) = D_\lambda|_0$ and $V^{(0)} = (v_{ij}^{(0)})_{1 \leq i, j \leq n} := (D_0 U^{\text{tr}})^{-1}$. Let $v, B \in \mathbb{R}^{>0}$ such that $|v_{ij}^{(0)}| \leq v$ ($1 \leq i, j \leq n$) and $U_i \leq B$ ($1 \leq i \leq n$). Then for sufficiently large B the number of arithmetic operations of our procedure is roughly bounded by

$$(14) \quad (t+1) \left(3 - \frac{4 \log K}{n \log \lambda} + \frac{4 \log B}{\log \lambda} \right)^{s+t-1} \frac{1}{2} \left(\frac{n+1}{2} \right)^{\frac{n+1}{2}} B n v \lambda^{\frac{1}{2}} \left(1 + \frac{1}{K} \right)^{\frac{n}{2}} + 1 \left(\frac{n}{2} \right)^{\frac{n}{2}} 2(n+1)n.$$

Proof. The first two factors come from the number (13) of quadratic forms to be considered. The last two factors contain the number of necessary operations for the transition from one quadratic form to the next, for the reduction algorithm applied to each quadratic form and, finally for the enumeration procedure.

Let $V^{(\lambda)} := (D_\lambda U^{\text{tr}})^{-1}$ with D_λ defined as in Lemma 3. We denote the rows of $V^{(\lambda)}$ by $\underline{v}_1^{(\lambda)}, \dots, \underline{v}_n^{(\lambda)}$ and set $T := \max\{S_i \mid 1 \leq i \leq n\}$ for abbreviation. Applying the reduction algorithm of [4] to the rows of an arbitrary regular matrix $V \in \mathbb{R}^{n,n}$ we get its row-reduced version \tilde{V} . Because of $|v_{ij}^{(0)}| \leq v$, $|v_{ij}^{(\lambda)}| \leq v \lambda^{T/2}$ ($1 \leq i, j \leq n$) and (1.12) of [4] we obtain

$$(15) \quad \|\tilde{\underline{v}}_i^{(\phi)}\|^2 \leq 2^{n-1} n v^2 \quad (1 \leq i \leq n)$$

$$(16) \quad \|\tilde{\underline{v}}_i^{(\lambda)}\|^2 \leq 2^{n-1} n \lambda^T v^2 \quad (1 \leq i \leq n),$$

where $\|\cdot\|$ denotes the Euclidean norm in \mathbb{R}^n .

Furthermore, Proposition 1.6 from [4] yields

$$(17) \quad |\det \underline{v}^{(o)}|^2 \leq \prod_{i=1}^n \|\tilde{\underline{v}}_i^{(o)}\|^2 \leq 2^{n(n-1)/2} |\det \underline{v}^{(o)}|^2, \text{ and for the row-reduced matrix } \tilde{\underline{v}}^{(\lambda)}$$

$$(18) \quad |\det \underline{v}^{(\lambda)}|^2 \leq \prod_{i=1}^n \|\tilde{\underline{v}}_i^{(\lambda)}\|^2 \leq 2^{n(n-1)/2} |\det \underline{v}^{(\lambda)}|^2,$$

where the bounds in (18) depend indeed not on T . By 1.12 Remark, denoting the Euclidean norm of the smallest vector in the lattice generated by the rows of $\underline{v}^{(o)}$ (resp. $\underline{v}^{(\lambda)}$) by $M_1^{(o)}$ (resp. $M_1^{(\lambda)}$), we also obtain lower bounds for $\|\tilde{\underline{v}}_i^{(o)}\|^2$ and $\|\tilde{\underline{v}}_i^{(\lambda)}\|^2$, $(1 \leq i \leq n)$

$$(19) \quad \|\tilde{\underline{v}}_i^{(o)}\|^2 \geq (M_1^{(o)})^2 \geq \frac{1}{(2^n n v^2)^{n-1}} |\det \underline{v}^{(o)}|^2, \quad (1 \leq i \leq n)$$

and considering the transition matrix from $\underline{v}^{(\lambda)}$ to $\underline{v}^{(o)}$

$$(20) \quad \|\tilde{\underline{v}}_i^{(\lambda)}\|^2 \geq (M_1^{(\lambda)})^2 \geq \frac{1}{\lambda^T (2^n n v^2)^{n-1}} |\det \underline{v}^{(o)}|^2, \quad (1 \leq i \leq n).$$

Now we can estimate the number $N^{(\lambda)}$ of vectors $\underline{x} \in \mathbb{Z}^n$ which must be tested for solving (3) after the reduction of the quadratic form

$$\underline{x}^T \underline{U} \underline{D}^2 \underline{\lambda}^{-1} \underline{U}^T \underline{x}.$$

By [2], [3] we have

$$N^{(\lambda)} \leq \frac{1}{2} \prod_{i=1}^n (2 (\lfloor \|\tilde{\underline{v}}_i^{(\lambda)}\|^2 n^{(K+1)^2/n} \rfloor)^{1/2} + 1).$$

For T (e.g. B) sufficiently large we conclude from the relations (16), (20) and (18)

$$\leq \frac{1}{2} (2 (2^{n-1} \lambda^T n v^2 n^{(K+1)^2/n})^{1/2} + 1)^{n/2} =: N. \text{ Because of}$$

$$T \leq \frac{\log(B^2)}{\log(\lambda)} + 1 - \frac{\log(K^{2/n})}{\log(\lambda)} \text{ we finally obtain}$$

$$(21) \quad N^{(\lambda)} \leq \frac{1}{2} (2^{\frac{n+1}{2}} B n v \lambda^{1/2} (1 + \frac{1}{K})^{1/n} + 1)^{n/2}.$$

Furthermore it is easily seen that the analysis whether a possible solution $\underline{x} \in \mathbb{Z}^n$ actually solves (3) requires at most

$$(22) \quad 2n^2 + n - 1$$

arithmetic operations.

Next the number of arithmetic operations of the reduction algorithm [4] applied to our special problem must be investigated. Using the notations of [4] we have

$$(23) \quad D \leq (n \lambda^T v^2)^{\frac{n(n-1)}{2}}$$

and by (20)

$$(24) \quad d_i \geq \left(\frac{3}{4}\right)^{i(i-1)/2} \left(\frac{1}{\lambda^T (2^n - n v^2)^{n-1}} |\det v^{(o)}|^2\right)^i, \quad (1 \leq i \leq n).$$

If B is sufficiently large, the number of arithmetic operations needed by the reduction algorithm of [4] is in our case bounded by

$$(25) \quad cn^4 \log(n v^2 B^4 \lambda^2) \text{ with some fixed constant } c \in \mathbb{R}^{>0}.$$

This is of a lower order of magnitude than the product of (21) and (22).

In addition the number of arithmetic operations for a change of the quadratic form and for an initialization of the algorithm are negligible.

□

On the other hand for $U_i = B$ ($1 \leq i \leq n$) the enumeration of all $\underline{x} \in \mathbb{Z}^n \setminus \{0\}$ subject to (4) of Lemma 1 (the usual procedure of solving (3)) yields $\frac{1}{2}((2[Bnv]+1)^n - 1)$ vectors $\underline{x} \in \mathbb{Z}^n$ whose corresponding norms must be calculated. (Here we assume that the sum of the absolute values of the elements of each row of the matrix

$$((w_j^{(i)})_{1 \leq i, j \leq n})^{-1} \text{ is of size } vn.$$

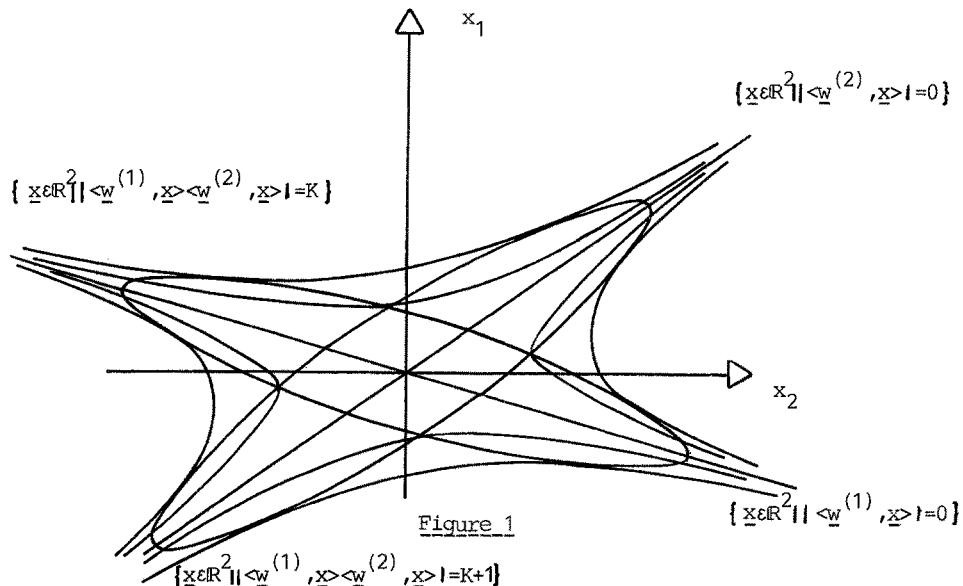
Thus the number of arithmetic operations needed by the usual enumeration procedure is greater than

$$\frac{1}{2}((2[Bnv] + 1)^n - 1)(2n - 1)$$

and in the interesting case of large B , this is about the square of the number of arithmetic operations needed by our procedure.

4. Geometrical interpretation of our method.

The usual way to solve (3) considers all lattice points $\underline{x} \in \mathbb{Z}^n$ subject to (4) of Lemma 1, whereas we take only those for which $|N < \underline{w}, \underline{x} >| \leq K$. The region of \mathbb{R}^n in which they are contained is then suitable covered by ellipsoids as shown in Figure 1 for $n = 2$.



References.

- [1] S.I. Borewics und I.R. Safarevic, Zahlentheorie,
Birkhäuser-Verlag, Basel und Stuttgart
1966, pp. 134-141.
- [2] U. Dieter, How to Calculate Shortest Vectors in a Lattice,
Math. Comp., v. 29, 131,
(1975), pp. 827-833.
- [3] D.E. Knuth, The art of computer programming, vol.2,
Addison-Wesley, sec.ed. (1981), p.95.
- [4] A.K. Lenstra, H.W. Lenstr Jr., L. Lovász, Factoring Polynomials
with Rational Coefficients,
Math. Ann. 261 (1982), 515-534.
- [5] K. Mahler, Inequalities for Ideal Bases in Algebraic Number Fields,
J. Austral. Math. Soc. 4, (1964), pp. 425-447.
- [6] M. Pohst u. H. Zassenhaus, An effective number geometric method
of computing the fundamental units
of an algebraic number field,
Math. Comp., v. 31, 1977, pp. 754-770.
- [7] M. Pohst, P. Weiler u. H. Zassenhaus, On effective computation
of fundamental units I,II,
Math. Comp., v. 38, 1982, pp. 275-329.

COMPUTATION OF INTEGRAL SOLUTIONS OF A SPECIAL
TYPE OF SYSTEMS OF QUADRATIC EQUATIONS

Michael Pohst
Mathematisches Institut
Universität Düsseldorf
Universitätsstr. 1
4 Düsseldorf, FRG

1. Abstract.

Let S, T be $n \times n$ integral matrices and one of them positive definite. We develop a method to decide whether a solution $X \in \mathbb{Z}^{n \times n}$ of $X^T S X = T$ exists and, if the answer is affirmative, an algorithm for the computation of X . Some applications in algebraic number theory and lattice construction are presented.

2. Introduction.

Let $S = (s_{ij})$, $T = (t_{ij})$ be $n \times n$ integral matrices and one of them positive definite. To solve

$$(2.1) \quad X^T S X = T \quad (X \in \mathbb{Z}^{n \times n})$$

we construct a set of invariants for S, T which either shows that (2.1) has no solution over the integers or finally provides such a solution.

The elementary divisors of S, T yield the first invariant, since their computation is comparatively easy. They are obtained upon computing the Smith normal form of S, T , for which a practical algorithm is given in [1]. If the elementary divisors of S, T do not coincide, (2.1) is certainly unsolvable.

From Section 3 on we view S, T as inner product matrices for the basis vectors of n -dimensional lattices $\mathcal{L}, \mathcal{L}'$. Then a solution X of (2.1) describes a lattice isomorphism which can be computed from its action on the vectors of \mathcal{L} of small length.

In case the entries of S, T are of large absolute value we first apply reduction theory to the bases of $\mathcal{L}, \mathcal{L}'$ in Section 4 in order to obtain matrices \tilde{S}, \tilde{T} which are similar to S, T but easier to handle, i.e. we obtain

$$(2.2) \quad M, N \in GL(n, \mathbb{Z}) : \tilde{S} = M^{\text{tr}} S M, \tilde{T} = N^{\text{tr}} T N$$

as well as M^{-1}, N^{-1} and instead of (2.1) we consider

$$(2.3) \quad Y^{\text{tr}} \tilde{S} Y = \tilde{T} \quad (Y \in \mathbb{Z}^{n \times n}).$$

It is obvious that each solution Y of (2.3) provides a solution

$$(2.4) \quad X = MYN^{-1}$$

of (2.1), and (2.1) is unsolvable if and only if (2.3) is unsolvable. Hence, from Section 5 on we can assume that S, T are inner product matrices of reduced bases $(\underline{a}_1, \dots, \underline{a}_n), (\underline{b}_1, \dots, \underline{b}_n)$ of \mathcal{F}, \mathcal{F} , respectively.

Then the numbers of vectors of "small" length in \mathcal{F}, \mathcal{F} provide a second invariant in Section 5. Namely, since S, T represent the inner products of reduced basis vectors of \mathcal{F}, \mathcal{F} , respectively, the diagonals of S, T must coincide by now and any solution X of (2.1) must map the i -th basis vector onto one of the same length ($1 \leq i \leq n$). Let $l_1 < \dots < l_r$ be all lengths occurring among s_{11}, \dots, s_{nn} , then we need to compute all lattice vectors of lengths l_i in \mathcal{F}, \mathcal{F} ($1 \leq i \leq r$). If the numbers s_i, t_i of vectors of length l_i in \mathcal{F}, \mathcal{F} , respectively, do not coincide, (2.1) is certainly unsolvable.

In Section 6 we distinguish vectors \underline{x} of \mathcal{F} (respectively \mathcal{F}) with respect to their "type". Because of the basis reduction in Section 4 a candidate for the k -th basis vector must have inner products with basis vectors \underline{a}_j ($1 \leq j < k$) whose absolute values are bounded by $\frac{1}{2}\|\underline{a}_j\|$. Hence, the numbers of those vectors of length l_i ($1 \leq i \leq r$) for which the absolute value of their inner product with \underline{x} is m ,

$$0 \leq m \leq \frac{l_i}{2}, \text{ yield further invariants of } S, T, \text{ respectively.}$$

If all these invariants of S, T coincide we are convinced that (2.1) is solvable and start to construct X by an appropriate search method. A useful tool in this context is the so-called finger-print (see also [2] for a special case) which is derived in Section 7. The essential algorithm for the computation of X is then given in Section 8. We conclude with some remarks concerning a version for $S, T \in \mathbb{R}^{n \times n}$ and give a few examples.

The electronic computations for Example 3 of Section 9 were carried out on the CDC Cyber 76 of the Rechenzentrum of the Universität zu Köln.

3. S, T and their lattices \mathcal{T}, \mathcal{S} .

We consider two possibilities of viewing S, T as inner product matrices of appropriate lattices. By e_1, \dots, e_n we denote the canonical basis of column vectors of \mathbb{Z}^n , where e_i has all coordinates zero except the i -th being one ($1 \leq i \leq n$). For S positive definite the mapping

$$(3.1) \quad \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^{>0} : (\underline{x}, \underline{y}) \rightarrow \langle \underline{x}, \underline{y} \rangle_S := \underline{x}^{\text{tr}} S \underline{y}$$

defines an inner product on the n -dimensional Euclidean space \mathbb{R}^n .

Then (2.1) can be interpreted as the task of computing a basis

b_1, \dots, b_n of \mathbb{Z}^n ,

$$(3.2) \quad b_j = \sum_{i=1}^n x_{ij} e_i \quad (1 \leq j \leq n),$$

which satisfies

$$(3.3) \quad \langle b_i, b_j \rangle_S = t_{ij} \quad (1 \leq i, j \leq n).$$

Hence, we are searching for a basis of \mathbb{Z}^n for which the inner products of the basis vectors are given by the entries of T . The second possibility is to consider abstract n -dimensional lattices

$$(3.4) \quad \mathcal{T} = \mathbb{Z} s_1 + \dots + \mathbb{Z} s_n, \quad \mathcal{S} = \mathbb{Z} t_1 + \dots + \mathbb{Z} t_n$$

and the usual inner product

$$(3.5) \quad \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R} : (\underline{x}, \underline{y}) \rightarrow \langle \underline{x}, \underline{y} \rangle = \underline{x}^{\text{tr}} \underline{y}.$$

We stipulate

$$(3.6) \quad \langle s_i, s_j \rangle = s_{ij}, \quad \langle t_i, t_j \rangle = t_{ij} \quad (1 \leq i, j \leq n).$$

Then (2.1) describes a lattice isomorphism between \mathcal{T} and \mathcal{S} , t_j being the image of the vector

$$(3.7) \quad b_j := \sum_{i=1}^n x_{ij} s_i$$

of \mathcal{T} ($1 \leq j \leq n$).

We note that in both cases the columns of any solution X of (2.1) are the coefficient vectors of the vectors b_j of a new basis b_1, \dots, b_n defined by (3.2), (3.7), respectively, and we can identify the j -th column of X with a basis vector b_j of length $\|b_j\| := t_{jj} = \langle b_j, b_j \rangle$. Generally we shall assume the second point of view.

4. Reduction theory and vectors of small length.

Let $\mathcal{F} = \mathbb{Z} s_1 + \dots + \mathbb{Z} s_n$ be an n-dimensional lattice in \mathbb{R}^n with inner product matrix $S = (s_i^T s_j)_{1 \leq i, j \leq n}$. We search for bases a_1, \dots, a_n of \mathcal{F} for which the basis vectors are of small length. For $\underline{x} = (x_1, \dots, x_n)^T, \underline{y} = (y_1, \dots, y_n)^T \in \mathbb{R}^n$ we define

$$(4.1) \quad \underline{x} < \underline{y} : \Leftrightarrow \begin{aligned} (\|\underline{x}\| < \|\underline{y}\| \vee (\|\underline{x}\| = \|\underline{y}\| \wedge \exists j \in \{1, \dots, n\} : \\ x_i = y_i \quad (1 \leq i < j) \wedge x_j > y_j)). \end{aligned}$$

This yields a total ordering on \mathbb{R}^n which induces a total ordering of bases $(a_1, \dots, a_n), (b_1, \dots, b_n)$ of \mathcal{F} via

$$(4.2) \quad (a_1, \dots, a_n) < (b_1, \dots, b_n) \quad : \Leftrightarrow \exists j \in \{1, \dots, n\} : a_i = b_i \quad (1 \leq i < j) \wedge a_j < b_j.$$

A lowest basis with respect to this ordering is called reduced, its basis vectors are uniquely determined. A method for the computation of a reduced basis is given in [3].

Since our main interest is to obtain a basis of \mathcal{F} for which the entries of the corresponding inner product matrix are small in absolute value, it usually suffices to apply only pair-reduction to the basis vectors s_1, \dots, s_n . For each pair s_i, s_j ($1 \leq i < j \leq n$) we reduce the orthogonal projection of s_j into $\mathbb{R} s_i$ modulo $\mathbb{Z} s_i$. Thus we easily get a basis $\tilde{s}_1, \dots, \tilde{s}_n$ of \mathcal{F} which is pairwise reduced (for the algorithm see [3]), i.e.

$$(4.3) \quad \tilde{s}_1 < \tilde{s}_2 < \dots < \tilde{s}_n \text{ and } 2| < \tilde{s}_i, \tilde{s}_j > | \leq \|\tilde{s}_i\| \quad (1 \leq i < j \leq n).$$

Since pairwise reduced bases are in most cases also reduced this procedure usually suffices to yield bases a_1, \dots, a_n of \mathcal{F} , b_1, \dots, b_n of \mathcal{F} such that the corresponding inner product matrices S, T satisfy

$$(4.4) \quad s_{11} = t_{11} \leq s_{22} = t_{22} \leq \dots \leq s_{nn} = t_{nn}, \quad 2|s_{ij}| \leq s_{ii}, \\ 2|t_{ij}| \leq t_{ii} \quad (1 \leq i < j \leq n).$$

Hence, we need to determine a reduced basis for \mathcal{F}, \mathcal{T} only in case $s_{ii} = t_{ii}$ ($1 \leq i \leq n$) is violated for the inner product matrices S, T of pairwise reduced bases of \mathcal{F}, \mathcal{T} , respectively. Hence, after transition to pairwise reduced or reduced bases, if necessary, (4.4) holds or (2.1) is certainly unsolvable.

5. Computation of vectors of small length.

From now on we assume (4.4) for the entries of S, T . Let $1 \leq l_1 < l_2 < \dots < l_r = n$ such that

$$(5.1) \quad s_{11} = \dots = s_{l_1 l_1} < s_{l_1 + 1, l_1 + 1} = \dots = s_{l_2 l_2} < \dots \\ < s_{l_{r-1} + 1, l_{r-1} + 1} = \dots = s_{l_r l_r}.$$

Only vectors $\underline{x} \in \mathcal{T}$ of length $\|\underline{x}\| = s_{ii}$ can be mapped onto the i -th basis vector of \mathcal{Y} . Hence, we need to compute

$$(5.2) \quad \mathcal{T}_i = \{\underline{x} \in \mathcal{T} \mid \|\underline{x}\| = s_{l_i l_i}\} \quad (1 \leq i \leq r). \\ \mathcal{T}_i = \{\underline{x} \in \mathcal{Y} \mid \|\underline{x}\| = s_{l_i l_i}\}$$

This is easily done using Cholesky's method [3]. Of course, we only store one of the vectors \underline{x} . The numbers

$$(5.3) \quad s_i := \#\mathcal{T}_i, \quad t_i := \#\mathcal{Y}_i \quad (1 \leq i \leq r)$$

are further invariants of \mathcal{T}, \mathcal{Y} , respectively. In case $(s_1, \dots, s_r) * (t_1, \dots, t_r)$ the equation (2.1) is certainly unsolvable. Therefore we assume $s_i = t_i$ ($1 \leq i \leq r$) in the sequel. To diminish the number of candidates $\underline{x} \in \mathcal{T}_i$ which can possibly be mapped onto one of the basis vectors $b_{l_{i-1}+1}, \dots, b_{l_i}$ we partition each \mathcal{T}_i into suitable subsets in the next section.

6. Inner products types.

since the bases $(\underline{a}_1, \dots, \underline{a}_n)$ of \mathcal{T} , $(\underline{b}_1, \dots, \underline{b}_n)$ of \mathcal{Y} are pairwise reduced the vectors $\underline{a}_i, \underline{b}_i$ satisfy

$$(6.1) \quad |\underline{a}_i^T \underline{a}_j| \leq \frac{1}{2} \|\underline{a}_i\|, \quad |\underline{b}_i^T \underline{b}_j| \leq \frac{1}{2} \|\underline{b}_i\| \quad (1 \leq i < j \leq n).$$

Therefore it is useful to define the type of a vector $\underline{x} \in \mathcal{T}_i$ ($1 \leq i \leq r$) (\mathcal{Y}_i analogously):

$$(6.2) \quad t(\underline{x}) := (t(\underline{x}, 0), \dots, t(\underline{x}, [\frac{s_{ii}}{2}])), \text{ where} \\ t(\underline{x}, j) := \#\{\underline{y} \in \mathcal{T}_i \mid |\underline{x}^T \underline{y}| = j\} \quad (0 \leq j \leq \frac{s_{ii}}{2}).$$

All vectors $\underline{x} \in \mathcal{T}_i$ which are of the same type as $\underline{a} \in \mathcal{T}_i$ belong to the equivalence class

$$(6.3) \quad T_i(\underline{a}) := \{\underline{x} \in \mathcal{T}_i \mid t(\underline{x}) = t(\underline{a})\}.$$

Obviously, $T_i(\underline{a})$ is a union of orbits of \mathcal{T}_i under the action of the automorphism group of \mathcal{T} . The types $t(\underline{a})$ and the type frequencies $\# T_i(\underline{a})$ are further important invariants for \mathcal{T}, \mathcal{T} , respectively.

We note that for large s_i the partitioning of \mathcal{T}_i into equivalence classes of vectors of the same type is very time consuming and therefore not always recommendable.

7. The finger-print matrix.

To obtain a solution X of (2.1) we compute vectors

$$(7.1) \quad \underline{x}_i = \sum_{j=1}^n x_{ji} \underline{a}_j \in \mathcal{T} \quad (1 \leq i \leq n) \text{ subject to}$$

$$(7.2) \quad \underline{x}_v \in \mathcal{T}_j \quad (l_{j-1} < v \leq l_j; \quad l_0 = 0; \quad 1 \leq j \leq r) \text{ and}$$

$$(7.3) \quad \underline{x}_\mu^T \underline{x}_v = t_{\mu v} \quad (1 \leq \mu, v \leq n).$$

We assume that we have already determined $\underline{x}_1, \dots, \underline{x}_k$ ($1 \leq k < n$) subject to (7.2), (7.3). Then it can happen that there is no \underline{x}_{k+1} satisfying the required conditions. Therefore we refine our search method in two ways. We build up X in a more suitable order of succession such that the number of candidates for the next column is as small as possible. For example, we start choosing \underline{x}_v ($v \in \{1, \dots, n\}$) such that $\# T(\underline{x}_v)$ is as small as possible. Also we choose \underline{x}_v at each step such that there are still candidates for the next column of X to be filled. Both tasks are solved by introducing the so-called finger-print matrix (see also [2], where a special case is considered).

We define a new order of the basis vectors (i_1, \dots, i_n) and sets

$C(i_j, k)$ ($1 \leq k \leq n, 1 \leq j \leq n-1$) inductively by:

Let i_1 be the smallest index such that $\# T(b_{i_1}) \leq \# T(b_k)$ ($1 \leq k \leq n$).

If i_1, \dots, i_j are determined ($1 \leq j \leq n-1$), let

$$C(i_j, k) := \begin{cases} \emptyset & \text{for } k \in \{i_1, \dots, i_j\}, \\ \{\underline{x} \in T(b_k) \mid b_{i_v}^T b_k = b_{i_v}^T \underline{x} \quad (1 \leq v \leq j)\} & \text{otherwise.} \end{cases}$$

In case $j < n-1$ let i_{j+1} be the smallest index such that $\# C(i_j, i_{j+1})$ is minimal among all $\# C(i_j, k) > 0$. For $j = n-1$ let i_n be the remaining

element of $\{1, \dots, n\} \setminus \{i_1, \dots, i_{n-1}\}$. The finger-print (of γ) then is the matrix $C = (c_{ij}) \in \mathbb{Z}^{(n-1) \times n}$ with $c_{ij} = \# C(i, j)$ ($1 \leq i, j \leq n; i \neq i_n$).

Thus the finger-print of γ determines the order (i_1, \dots, i_n) in which the columns of a solution X of (2.1) are to be determined, and the entries $\# C(i_j, i_{j+1})$ are the number of candidates for column i_{j+1} of X ($1 \leq j \leq n-1$). Note that the number of candidates for column i_1 is $\# T(b_{i_1})$ which is not an entry of the finger-print.

8. The algorithm.

Input: Matrices $S, T \in \mathbb{Z}^{n \times n}$ and S positive definite.

Output: Either "No solution" is printed or a solution $X \in \mathbb{Z}^{n \times n}$ of $X^T S X = T$.

- Step 1: (Elementary divisors) Compute the elementary divisors c_1, \dots, c_n of S and d_1, \dots, d_n of T . If $c_i \neq d_i$ for some i ($1 \leq i \leq n$), then go to 11.
- Step 2: (Reduction) Reduce the entries of S, T as described in Section 4 and store the transformation matrices M, N^{-1} ((2.2) - (2.4)). (The reduced versions of S, T are also denoted by S, T in the sequel.) If $s_{ii} \neq t_{ii}$ for some i ($1 \leq i \leq n$), go to 11.
- Step 3: (Vectors of small length) Determine l_1, \dots, l_r according to (5.1) and γ_i ($1 \leq i \leq r$) of (5.2), set $t_i = \#\gamma_i$.
- Step 4: (Finger-print of T) Compute the vector types in each γ_i as well as the type frequencies and the finger-print matrix C for T and the new order (i_1, \dots, i_n) introduced in Section 7.
- Step 5: (Types and type frequencies of S) Determine γ_i ($1 \leq i \leq r$) according to (5.2) and set $s_i = \#\gamma_i$. If $s_i \neq t_i$ for some i ($1 \leq i \leq r$), go to 11.
Else compute the types of the vectors in γ_i ($1 \leq i \leq r$) as well as the corresponding type frequencies. If they do not coincide with those of γ_i , go to 11.
- Step 6: (Initialization of the computation of X) Set $k \leftarrow 1, \tilde{k} \leftarrow 2, i_o \leftarrow 0$. Compute the set $C(i_o, i_1)$ of all $s \in \gamma$ of type $t(s_{i_1})$. Choose $s_{i_1} \in C(i_o, i_1)$ and set $C(i_o, i_1) \leftarrow C(i_o, i_1) \setminus \{s_{i_1}\}$.

Step 7: (Further extension of s_{i_1}, \dots, s_{i_k} possible?)

For $\mu = 1, \dots, n$, $\mu \notin \{i_1, \dots, i_k\}$ compute $C(i_k, \mu) := \{\underline{x} \in \mathbf{T} \mid \underline{x}$ of type $t(\underline{x})_i < s_{i_\mu}, \underline{x} > = t_{i_\mu \mu} \quad (1 \leq \mu \leq k)\}.$

In case $\#C(i_k, \mu) = c_{i_k \mu}$ for $\mu = 1, \dots, n, \mu \notin \{i_1, \dots, i_k\}$, go to 8, else to 9.

Step 8: (Increase k) Choose $s_{i_{\tilde{k}}} \in C(i_k, i_{\tilde{k}})$ and replace $C(i_k, i_{\tilde{k}})$ by $C(i_k, i_{\tilde{k}}) \setminus \{s_{i_{\tilde{k}}}\}$. In case $\tilde{k} = n$ go to 12; otherwise set $k \leftarrow \tilde{k}, \tilde{k} \leftarrow \tilde{k} + 1$ and go to 7.

Step 9: (Replace s_{i_k}) In case $k = 0$ go to 11.

For $k > 0$ consider $C(i_{k-1}, i_k)$. If $C(i_{k-1}, i_k) = \emptyset$, go to 10. Else set $\tilde{k} \leftarrow k$, $k \leftarrow k - 1$ and go to 8.

Step 10: (Decrease k) Set $\tilde{k} \leftarrow k$, $k \leftarrow k - 1$ and go to 9.

Step 11: (No solution) Print "No solution" and terminate.

Step 12: (Print solution) Let $y_{v i_j}$ ($1 \leq v \leq n$) be the coordinates of s_{i_j} with respect to the reduced basis of \mathbf{T} computed in Step 2 ($1 \leq j \leq n$). For $Y = (y_{v \mu})_{1 \leq v, \mu \leq n}$ set $X = M Y N^{-1}$ with the M, N^{-1} computed in Step 2. Print X .

9. Remarks and applications.

Of course there are some possibilities of speeding up the computations by refining single steps of the algorithm. A few useful remarks in this direction can be found in [2]. Of greater interest, however, is the chance of applying our method also in case S, T are in $\mathbb{R}^{n \times n}$ instead of $\mathbb{Z}^{n \times n}$. Then of course the elementary divisor test is to be eliminated. But reduction theory is still valid and yields analogous results except for the inner products now being real numbers. Hence, the definition of vector types must be modified. This has the nice side effect that, in general, the type frequencies will be much smaller. Since the computations of reduced bases then use floating point numbers we must take care of round-off errors. In special cases - for example when dealing with algebraic numbers - the result can be checked by integral calculations.

Among other useful applications our procedure is of great use in checking whether two monic irreducible polynomials of $\mathbb{Z}[x]$ generate

the same algebraic number field (see Example 1 below) and to construct the automorphism group of a lattice (choosing $S = T$, see [2]).

Example 1. The polynomials $f(x) = x^3 - 7x - 7$ and $g(x) = x^3 + x^2 - 2x - 1$ both generate cubic fields F, G by adjoining to \mathbb{Q} a zero α of $f(x)$, β of $g(x)$, respectively. Since all zeros are real, the inner product is given by the trace. The discriminant matrices for $(1, \alpha, \alpha^2), (1, \beta, \beta^2)$ are

$$S = \begin{vmatrix} 3 & 0 & 14 \\ 0 & 14 & 21 \\ 14 & 21 & 98 \end{vmatrix}, \quad T = \begin{vmatrix} 3 & -1 & 5 \\ -1 & 5 & -4 \\ 5 & -4 & 13 \end{vmatrix}, \quad \text{respectively.}$$

By pair reduction we obtain the reduced versions

$$\tilde{S} = \begin{vmatrix} 3 & -1 & 1 \\ -1 & 5 & 2 \\ 1 & 2 & 5 \end{vmatrix}, \quad \tilde{T} = \begin{vmatrix} 3 & -1 & -1 \\ -1 & 5 & -2 \\ -1 & -2 & 5 \end{vmatrix}, \quad \text{and}$$

$\tilde{X} = \text{diag}(1, 1, -1)$ solves $\tilde{X}^{\text{tr}} \tilde{S} \tilde{X} = \tilde{T}$. Hence $\beta \mapsto \alpha^2 - \alpha - 5$ yields an isomorphism between $\Phi(\alpha)$ and $\Phi(\beta)$. The discriminant of $F \simeq G$ is $d = \det S = 49$.

Example 2. A zero α of the polynomial $f(x) = x^3 - x + 1$ generates a non totally real cubic field $F = \mathbb{Q}(\alpha)$ of discriminant $d = -23$. A(n-integral) basis of F is given by $1, \alpha, \alpha^2$. The inner product of $\alpha_1, \alpha_2 \in F$ is

now defined by $\text{Tr}(\alpha_1 \bar{\alpha}_2) := \sum_{j=1}^3 \alpha_1^{(j)} \overline{\alpha_2^{(j)}}$, where the sum is over all

conjugates of F and $\bar{-}$ denotes complex conjugation. The discriminant matrix for $1, \alpha^2 - 1, \alpha$ is

$$T = \begin{vmatrix} 3 & -1 & 0 \\ -1 & 3.21930 & -1.32473 \\ 0 & -1.32473 & 3.26464 \end{vmatrix} \quad (|d| = \det T)$$

which is already pairwise reduced.

An application of the real version of our algorithm to

$$S = \begin{vmatrix} 35.13550 & 25.45335 & 19.12706 \\ 25.45335 & 49.13340 & 2.90932 \\ 19.12706 & 2.90932 & 14.33792 \end{vmatrix}$$

yields the solution

$$X = \begin{vmatrix} -4 & 9 & -7 \\ 2 & -4 & 3 \\ 5-11 & 9 \end{vmatrix}$$

of $X^{\text{tr}} S X = T$.

Example 3. A procedure for constructing integral lattices (see [2]) with prescribed minimum length $M = 4$ yields the following two inner product matrices in dimension 14:

S =

T =

$$\left(\begin{array}{cccccccccccccc} 4 & 2 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 2 & 4 & -2 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & -2 & 4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 4 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 \\ -2 & -2 & 0 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 4 & -2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 4 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 4 & 1 & 1 & -1 & 1 \\ -1 & -2 & 1 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 4 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 4 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 1 & 1 & -1 \end{array} \right) \quad \left(\begin{array}{cccccccccccccc} 4 & 2 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -2 \\ 2 & 4 & -2 & -2 & 0 & -2 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & -1 \\ 0 & -2 & 4 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -2 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & 0 & 4 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ -2 & -2 & 0 & 0 & 0 & 4 & -2 & 0 & -1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -2 & 4 & -2 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 4 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 4 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 4 & 1 & -2 & 1 & 1 \\ -1 & -2 & 1 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 4 & -2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -2 & 4 & 1 \\ -2 & -1 & 0 & 0 & 0 & 1 & -1 & 1 & 1 & 1 & 1 & -1 & 1 & 4 & 0 \end{array} \right)$$

Their elementary divisors are $1^8 2^4 4 \cdot 12$. Both matrices are pairwise reduced. In Step 3 of our algorithm we obtain $r = 1$, $l_1 = 14$, $t_1 = 2 \cdot 711$.

There are four types of vectors in \mathcal{T}_1 :

- 2*(270,320,120) of frequency 2*288,
- 2*(318,256,136) of frequency 2*36,
- 2*(252,346,112) of frequency 2*384,
- 2*(324,256,130) of frequency 2*3.

The new order of succession is (2,5,3,1,4,6,7,8,9,10,12,14,11,13).

Steps 6-12 of the algorithm of Section 8 provide the solution

X =

$$\left(\begin{array}{cccccccccccccc} -9 & -11 & 1 & 9 & 1 & 4 & 4 & -2 & 3 & -3 & 6 & 0 & 0 & 7 \\ 54 & 66 & -10 & -53 & -2 & -21 & -25 & 10 & -17 & 16 & -36 & -2 & 2 & -44 \\ 34 & 42 & -6 & -34 & -2 & -13 & -16 & 6 & -11 & 10 & -23 & -1 & 1 & -28 \\ 26 & 32 & -5 & -26 & -1 & -10 & -12 & 5 & -8 & 8 & -18 & -1 & 1 & -21 \\ 16 & 20 & -3 & -16 & -1 & -6 & -8 & 3 & -5 & 5 & -11 & -1 & 0 & -13 \\ 34 & 42 & -7 & -34 & 0 & -13 & -16 & 6 & -10 & 9 & -23 & -1 & 1 & -28 \\ 22 & 28 & -5 & -23 & 0 & -8 & -11 & 4 & -7 & 6 & -15 & -1 & 1 & -18 \\ 12 & 15 & -3 & -12 & 0 & -4 & -6 & 2 & -4 & 3 & -8 & 0 & 0 & -10 \\ 5 & 6 & -1 & -5 & 0 & -2 & -2 & 1 & -1 & 1 & -3 & 0 & 0 & -4 \\ 3 & 4 & -1 & -3 & 0 & -1 & -2 & 1 & -1 & 1 & -2 & 0 & 0 & -2 \\ 3 & 4 & -1 & -3 & 0 & -1 & -2 & 1 & -1 & 1 & -2 & 0 & 0 & -3 \\ 2 & 2 & 0 & -2 & 0 & -1 & 0 & 0 & -1 & 1 & -1 & 0 & 0 & -2 \\ 2 & 3 & -1 & -2 & 0 & -1 & -1 & 1 & -1 & 0 & -2 & 0 & 0 & -2 \\ 1 & 2 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \end{array} \right).$$

References.

- [1] G. Havas and L. Sterling, "Integer matrices and Abelian groups",
Symbolic and Algebraic Computation
(E. W. Ng, ed.),
Lecture Notes in Computer Science 72,
Springer Verlag (1979), 431-451.
- [2] W. Plesken and M. Pohst, "Constructing integral lattices with
prescribed minimum I", to appear.
- [3] M. Pohst, "On the computation of lattice vectors
of minimal length, successive minima
and reduced bases with applications",
ACM SIGSAM Bulletin vol. 15, no. 1
(1981), 37-44.

Factorisation of Sparse Polynomials

J.H. Davenport¹

Equipe d'Analyse Numérique

Laboratoire I.M.A.G., B.P. 68

38402 Saint Martin d'Hères France

Abstract. Sparse polynomials $x^n \pm 1$ are often treated specially by the factorisation programs of computer algebra systems. We look at this, and ask how far this can be generalised. The answer is that more can be done for general binomials than is usually done, and recourse to a general purpose factoriser can be limited to "small" problems. but that general trinomials and denser polynomials seem to be a lost cause. We are concerned largely with the factorisation of univariate polynomials over the integers, being the simplest case.

Introduction

The following theorem (see, e.g., van der Waerden [1949] pp. 160-162) is well-known, and is used as a short-cut by several computer algebra systems (the author knows that it is used in MACSYMA [Bogen et al. 1977] and REDUCE [Hearn, 1973]), even though its use is not always mentioned in the descriptions (Wang, [1978] and Moore & Norman [1981], respectively).

Theorem. The factorisation of a polynomial $x^n - 1$ into irreducible polynomials over the rational numbers is given by the product of $f_d(x)$ over d dividing n , where f_d is the minimal polynomial of a primitive d -th root of unity.

This is an extremely important short-cut practically (we can contrast the time taken by MACSYMA² to factorise $x^{20}-1$ of 351 ms with the time taken for x^{20-220} of 7555 ms). We note that, in general, the only method of computing the f_d is to divide $x^d - 1$ by the f_e corresponding to factors e of d . f_p and f_{pq} (p, q distinct prime powers) have simple enough forms, but the general f_d is complicated, and can have arbitrarily large coefficients [Vaughan, 1974]. The factorisation of $x^n \pm 1$ follows

1. Permanent address: Emmanuel College, Cambridge, England.
2. All MACSYMA timings are taken from MACSYMA 296 running under MULTICS 9.1 on the CII-Honeywell-Bull 68/80 DPS3 at the Centre Interuniversitaire de Calcul de Grenoble. Repeated attempts at the same calculation indicate that the timings are accurate to $\pm 5\%$.

from that of x^{2n-1} , and is important because it enables us to dispose of polynomials like x^4+1 , which, while irreducible over the integers, has consistent factorisations modulo all the primes [Knuth, 1981, p. 437, ex. 12].

It is also of interest from the point of view of complexity theory, since all known factorising algorithms have average running time at least $O_A(d^2)$, where d is the degree of the polynomial. But for the sub-class of polynomials of the form x^d-1 , d is exponential in the complexity n of the problem, since it can be represented in binary. This gives us a $O_A(2^{2n})$ algorithm for factoring these polynomials by a general method, while we can do better using even a very naive factorising algorithm ($O_A(d^{1/2})$, which can be regarded as

$$O_B(d^{1/2} \log d \log \log d \log \log \log d),$$

to the point where generating the answer (or even printing it) is more expensive than the factorisation, since the cofactor of $x-1$ in the complete factorisation of x^p-1 contains p non-zero terms.

Of course, it comes as no surprise to computer algebraists to observe that it is more efficient to solve sparse problems by methods which take advantage of their sparsity. The rest of this paper is concerned with the question of such special-purpose methods for other classes of sparse polynomials.

I am grateful to A. Bremner, A. Schinzel and B.M. Trager for their helpful discussions, to the referee for his suggestions, and to B. Tuckerman and S. Wolfram for several examples.

Binomials (I)

If we consider more general binomials, we note at once that we can reduce the problem from ax^n+bx^m to $yn-m+an-m-1b$, where $x=y/a$, though in practice it might be more efficient to remember the factorisation of the (possibly rather large) constant term generated. The reducibility of these polynomials is covered by a theorem of Capelli [1898], whose statement we reproduce from Schinzel [1982, Theorem 21, p. 91].

Capelli's Theorem. $yn-a$ is reducible in k if and only if either:

- a) for some prime divisor p of n , $a=bP$ for some b in k ; or
- b) 4 divides n and $a=-4b^4$ for some b in k .

The first case is the obvious generalisation of the factorisation of x^p-1 observed in the introduction, while the second corresponds to the "anomalous" factorisation $x^4+4=(x^2+2x+2)(x^2-2x+2)$.

Since factorisation of numbers can be performed relatively quickly (we note the algorithm of Schnorr [1981], which is asymptotically faster than ne for arbitrarily

small e) and since determining whether an integer less than 2^k is a perfect p -th power can be performed (for fixed p ; it is necessary to allow for the size of p in general, but this adds a factor dominated by $\log^2 p$) in asymptotically the same time as multiplication of such numbers (currently $O(k \log k \log \log k)$ [Caviness, 1975], this leads to a fast algorithm for determining whether a binomial is irreducible or not, of asymptotically the same complexity as the factorisation of the exponent.

In fact we can do somewhat better than this: we can determine an irreducible factor of the polynomial, if it is not irreducible, in the same time. This relies on the following theorem of Schinzel [1963], whose proof we reproduce because of its inherent interest, illustrating as it does why binomials are easier than general polynomials, as well as the obscurity of its original appearance.

Schinzel's Theorem. If x^n-a does factor according to (b), but not according to (a) in Capelli's Theorem, its factors according to (b) are irreducible.

Proof. $x^n-a = (x^{2k}-2bx^{k+2}b^2)(x^{2k}+2bx^{k+2}b^2)$, where $n=4k$ and $a=-4b^4$. Suppose $g(x)$ divides one of these factors properly, where the degree of g is j . For each of the roots z_i of g , we have that $z_i^{4k}=-4b^4$. Hence $\text{abs}(z_i)^j=2b^2$. Since g_0 , the constant term of g , is the product of the j roots of g , we have that $\text{abs}(g_0)^{2k}=(2b^2)^j$. $2b^2$ is then a perfect m -th power, say c^m , where $m=2k/(j, 2k)$, and is greater than 1 since g is a proper factor. Since 2 is not a square, m is odd and we can generalise the ordinary factorisation of x^m+1 to a factorisation of $(x^{n/m})^m+(c^2)^m = x^n-a$, contradicting our initial assumption about the absence of type (a) factorisation.

Thus, if we first perform all the type (a) factorisations we can, and then perform a type (b) factorisation if possible, we are guaranteed an irreducible factor of the initial polynomial. Furthermore, the time of the initial test is clearly dominant, and we have an algorithm to find an irreducible factor in asymptotically the same required to factorise the exponent. More precisely, if $F(n)$ is the time required to factorise an n -bit number, and $M(n)$ is the time required to multiply two n -bit numbers together, the algorithms of this section are bounded by

$$O(\max(F(e), e M(c))),$$

where there are e bits in the exponent and c bits in the constant coefficient. The extra factor of e comes from the fact that we need to determine if the constant coefficient is a p -th power for all primes p dividing the exponent, of which there are at most e . For fixed $d=e+c$, the second term is approximately $O(d^2 \log d)$, while the first term is super-polynomial in d .

Binomials (II)

Determining the complete factorisation is, inevitably, more expensive for the cyclotomic polynomials, simply because of the size of the answer. Surprisingly, for binomials that are not truly cyclotomic (i.e. have a constant term other than ± 1), there is a polynomial-time factorisation algorithm, even though the theory of general binomial factorisation is more complex than one would think.

The reason for this extra complexity is that we are not guaranteed that the non-binomial factors of decomposition (a) are irreducible, and, indeed, they need not be. Consider, for example, the polynomial x^6+27 , which factors (a) as:

$$(x^2+3)(x^4-3x^2+9).$$

Since 27 is not a sixth power we would not expect the second term to factor, but in fact it does, as $(x^2-3x+3)(x^2+3x+3)$. Similar phenomena can be observed for all the odd primes, e.g.:

$$x^{10}-3125 = (x^2-5)(x^4-5x^3+15x^2-25x+25)(x^4+5x^3+15x^2+25x+25),$$

where again we would not expect the non-binomial term to factorise.

We can write x^6+27 as x^6-a^6 , where $a^2=-3$, and this then factors into the product of linear factors over $\mathbb{Z}[a, 11/6]$. But this field is in fact of degree 2 over \mathbb{Z} , since $11/6=(1+a)/2$. Hence it must factor into the product of quadratic factors over \mathbb{Z} . The same phenomenon will occur for all odd primes, and is, indeed, also behind the "anomalous" factorisation of x^4+4 observed in the previous section, because of the following result (see, e.g., Lang [1970], Theorem 6, pp. 76-77).

Theorem. $\pm p$ is a perfect square in $\mathbb{Z}[1^{1/p}]$, where the ambiguous sign is the quadratic residue $(-1/p)$.

Similarly, it is possible that other field inclusions will give rise to other factorisations of the non-binomial parts of the "obvious" factorisation, but this can only occur to a limited extent, as the following theorem shows.

Theorem. Let b , an integer greater than 1, be a perfect k -th power, but not a perfect l -th power for l greater than k . Then the factorisation of $x^n\pm b$ can be obtained from substituting $y=x^n/l$ in the factorisation of $y^l\pm b$, where l is the g.c.d. of n and $2k$.

Proof. We shall use the notation and results of Schinzel [1978], as specialised to the case $Q=\mathbb{Q}$. The definition of b implies that $e(b, \mathbb{Q})=k$ (or k without its factors of 2 in the case of radicals of negative numbers), and then Lemma 2 [op. cit.] implies that $E(b, \mathbb{Q})$ divides $2k$. Lemma 5 op. cit. can then be applied to the polynomial $\Phi(x) = x\pm b$, and this yields the statement of the theorem.

The theorem can be generalised to algebraic number fields, but the question of determining whether numbers are exact powers and so on become more complicated.

Hence the following algorithm.

Factorise (n, b) [returns a factorisation of $x^n - b$, b not a unit]

```
[1]   b1:=abs(b); k:=1
[2]   for i :=1:log2 b1 do:
[2.1]     while power(b1,i) do
[2.1.1]       k:=k*i; b1:=root(b1,k)
[3]   l:=gcd(n,2k)
[4]   f:=Cyclotomic-factorise (x^l-b);
[5]   f1:=for p in f collect
[5.1]     if binomialp(p) then p else factor(p)
[6]   return subst(x^n/l for x in f1)
```

k , and hence l , is obviously at most $O(\log_2 b)$. Even if we ignore the improvement presented by the use of Cyclotomic-factorise, we thus have a polynomial of degree (or several polynomials, the sum of whose degree is) $O(\log b)$ to factorise, and hence, using a polynomial-time factoring algorithm [Lenstra et al., 1982], the total time is a polynomial in $\log b$ and $\log n$ (in fact, n is only used in steps [3] and [6], and there is no requirement to factor it). This leads to the result that binomials where the constant term is not a unit are easier to factorise than the cyclotomic polynomials themselves.

There are obviously a number of practical improvements that can be made to this algorithm, such as: remembering the decomposition of b ; use of modular tests to determine if $b1$ could possibly be an i -th power; restricting i to primes. Even without these or the cyclotomic-factorise improvement, this algorithm is clearly a great improvement on the use of a general factoriser: see the following table (where the entry "new time" refers to the time for the factorisation in step 5.1 above, assuming that no use is made of the cyclotomic factorisation in step 4).

value of n	4	8	12	16	20	23	24	25
Obvious time for $x^n - 4$	160	241	771	886	2908	4990	3809	4483
New time for $x^n - 4$	160	160	160	160	160	0	160	0
Obvious time for $x^n + 4$	165	222	528	584	3136	4946	2851	4478
New time for $x^n + 4$	165	165	165	165	165	0	165	0

value of n	10	15	20	22
Obvious time for $x^n - 32$	1073	684	6273	2312
New time for $x^n - 32$	1073	203	1073	81

The use of cyclotomic-factorise does speed the factorisation, even if the results of the cyclotomic factorisation are not, in fact, irreducible. Some figures are given in the following table.

Polynomial	Time for complete factorisation.	Time after elimination.
x^{20-210}	6800	1642 (I)
x^{10-25}	3831	839 (I)
x^{20+310}	16804	1209 (I)
x^{10-55}	855	643 (R)
x^{20-510}	8216	1349 (R)
x^{14+77}	4452	1283 (R)

All times are in milli-seconds, and (I) indicates that the residue after elimination of obvious factors was irreducible, (R) that it had further factors.

Unary Polynomials

By this title, we mean polynomials all of whose non-zero coefficients are ± 1 . Hence the theorem quoted in the introduction is a complete characterisation of the factorisation of unary binomials. Unary trinomials [Ljunggren, 1960; Tverberg, 1960] and quadrinomials [Ljunggren, 1960] have also been investigated. The results are that the only factorisations of these polynomials are those that correspond to roots of unity: a statement that the next two theorems make more precise.

Ljunggren-Tverberg Theorem. The polynomial

$$f(x) = x^{nd} + x^{md} + x^e$$

where n and m are coprime, $e, e' = \pm 1$, n greater than or equal to $2m$, is irreducible except for the cases when 3 divides $n+m$ and one of: n, m both odd and $e=1$; n even and $e'=1$; m even and $e=e'$. In these cases $f(x)$ is the product of an irreducible polynomial and $x^{2d+e} m d e' n d x d + 1$.

The latter polynomial is either $(x^{3d}+1)/(x^d+1)$ or $(x^{3d}-1)/(x^d-1)$, depending on the sign of the coefficient of x^d , and so can be solved by the techniques for $x^n \pm 1$. The case n less than $2m$ can be solved by writing $y=1/x$, and factorising the resulting polynomial. This analysis of cases is sufficiently laborious, and for the case of quadrinomials we shall not descend to such depths.

Ljunggren's Theorem. The polynomial

$$f(x) = x^{nd} \pm x^{md} \pm x^{pd} \pm 1,$$

where $(n, m, p) = 1$, $(n, m-p) = a$, $(m, n-m) = b$, $(p, n-m) = c$ and $n > m+p$, is the product of an irreducible polynomial and the polynomials defining the roots of unity which are also roots of f , which are to be found among the divisors of

$$x^{ad} \pm 1, x^{bd} \pm 1, x^{cd} \pm 1.$$

This gives rise to the obvious algorithm of taking the g.c.d. of f with these six polynomials and decomposing such products in accordance with the general

procedures for factors of x^n-1 , being assured that the residue is irreducible. Of course, it would be possible to perform a complete analysis into cases, as for trinomials, but that appears somewhat tedious.

This gives, in principle, relatively fast algorithms for the factorisation of these unary sparse polynomials, analogous to the factorisation of unary binomials frequently implemented. As one example of the difference in speed, we note that the Ljunggren-Tverberg theorem enables us to factorise $x^{40}+x^8+1$ immediately, while MACSYMA takes 64432 milliseconds to find the factorisation. Similarly, Ljunggren's theorem enables us to factorise $x^{30}+x^{20}+x^{10}+1$ in 164 milliseconds, compared with MACSYMA's 28300. The question is whether such special-purpose methods are justified. This can only be answered by a study of the types of polynomials that users actually want to factorise, and no such study has, as far as I know, been done.

General Trinomials

It is a consequence of Schinzel's Theorem that every binomial has an irreducible factor with at most three terms, and our fast algorithms for determining an irreducible factor of a binomial was based on a constructive proof of this result. We recall that the proof was based on the fact that all the roots had the same absolute value, a technique that, regrettably, does not generalise. Schinzel [1963] has conjectured that a similar result is true for trinomials, or, more precisely:

Schinzel's Conjecture: there exists a constant K such that every trinomial over the rational numbers has an irreducible factor with at most K terms.

The following identity (attributed by Schinzel [1963; corrected 1983] to Mrs. H. Smyczek) shows that $K > 6$:

$$x^{10}-12x^2-196 = (x^5+2x^4+2x^3-4x^2-10x-14)(x^5-2x^4+2x^3+4x^2-10x+14)$$

(both factors are irreducible by Eisenstein's criterion). Bremner [1981] has recently demonstrated that $K > 8$, by producing a polynomial¹ $x^{14}+ax^2-b$ whose irreducible factors are two dense polynomials of degree 7. Since a constructive proof of this conjecture would appear to be necessary for even an algorithm to find an irreducible factor of trinomials, we may conclude that special-purpose algorithms for this problem are currently inaccessible.

As regards complete factorisation, we note that, since a cyclotomic polynomial

1. We remark that a has 86 digits and b has 99. The fact that one has to search so far for such an example renders it, in the author's opinion, less likely that there is a simple theory which will explain the factorisation of trinomials.

of degree n divides a trinomial of degree $2n$, we have at least linear cost in the degree (and hence exponential in the complexity) for the operation of generating the complete factorisation. If we do not have any cyclotomic factors, then the methods of Schinzel [1978] can be applied to trinomials provided that they can be written as polynomials of low degree in powers of x . Hence they are very suitable for $x^{nm}+ax^m+b$, but are no use for $x^{nm+1}+ax^n+b$. Thus it would appear that, in the present state of knowledge, deterministic algorithms for general sparse polynomials are not available.

As regards non-deterministic methods, one might hope that the probabilistic methods of Zippel [1981] could be applied to this problem. They can not yield a truly polynomial time algorithm, though, because they rely on a "dense" factorisation modulo p , using, say, Berlekamp's algorithm, and hence have a cost $O(n^3)$. However, they might be applied to speed up programs in practice.

Other results

There are a wide variety of other results known on the factorisation of sparse polynomials, both univariate and multivariate. Many of them are contained in the book of Schinzel [1982]. It is the author's contention that they are unlikely to be applicable to a general-purpose algebra system, but they are nevertheless both interesting and applicable to a range of polynomials types which may occur in special problems.

Univariate over the Integers. Schinzel [1962] contains a theorem similar to the Ljunggren-Tverberg theorem quoted above for polynomials of the form $x^m \pm 2x^n \pm 1$ except that in the cases $m/n = 2/7$ and $5/7$ the non-cyclotomic factor is the product of two irreducible polynomials. Mikusiński & Schinzel [1964] provide a (completely impractical) bound on the size of the special ratios m/n (which is the key element determining whether we can deduce the behaviour of a given polynomial from that of one of lower degree) for the factorisation of $x^m \pm px^n \pm 1$, and there are further generalisations, especially to polynomials of the form $x^m \pm x^n \pm p$.

Univariate over other Fields. Capelli's theorem is valid for all fields (though the test for perfect powers is often more complicated), and Schinzel's Theorem holds over fields of characteristic 0 in which 2 is not a square. Regrettably, reduction of sparse polynomials over algebraic number fields to polynomials over the integers tends not to preserve sparsity.

Multivariate Polynomials. Ehrenfeucht & Pełczyński [unpublished; see Schinzel, 1963] have shown that polynomials of the form $f(x)+g(y)+h(z)$ are irreducible over all fields of characteristic zero. For polynomials in two variables, the situation

is more complicated: $x-y$ divides $f(x)-f(y)$, but there are reducible polynomials of the form $f(x)+g(y)$ which do not satisfy this paradigm, for example

$$x^4-4x^2+y^4-4y^2+4 = (x^2+2\sqrt{2}xy+y^2-2)(x^2-2\sqrt{2}xy+y^2-2)$$

[Davenport, Lewis & Schinzel, 1961]. Fried [1973] has shown, inter alia, that this can not happen over \mathbb{Q} for indecomposable¹ f and g . Ehrenfeucht [see Schinzel, 1978, Corollary 3, p. 94] has shown that $f(x)+g(y)$ is irreducible if the degrees of f and g are coprime.

Schinzel [1973; theorem 22 of Schinzel, 1982] produces a general criterion for "structural irreducibility" of a sparse (more than half as many variables as terms) polynomial, in the sense that no choice of non-zero coefficients can make a polynomial of the given shape reducible. We can also note that Eisenstein's criterion is valid x -adically as well as p -adically, in the following form [van der Waerden, 1949, p.74] (further generalisations also exist):

Proposition. A polynomial $f(x,y)$ in $K[x,y]$ is irreducible if x does not divide the coefficient of the highest power of y , x does divide all other coefficients, and x^2 does not divide the constant term.

Conclusions.

Several algebra systems contain special methods for the factorisation of unary binomials, which are far faster than the general factoring methods they possess. There are two possible ways in which these techniques could be generalised: towards arbitrary binomials and towards unary trinomials and quadrinomials. Both generalisations are possible, and yield substantial performance improvements. In the absence of any statistics on the type of polynomials that factoring routines encounter in practice, it is hard to know whether they are justified by their frequency of application, but the extension to arbitrary binomials seems justified by the greater uniformity it would bring to the performance of the factorisers, which currently exhibit wildly varying times for equivalent problems: the factorisation of x^n-a^m .

Generalisation to a significantly wider class of univariate polynomials seems impossible at the moment, though there exist algorithms for specialised classes of multivariate polynomials which may or may not appear in practice.

From the complexity-theoretic point of view, we note that factoring a sparse non-cyclotomic binomial is not significantly harder than factoring a dense one of

1. Indecomposability is necessary as is shown by the example [Schinzel, 1983]:

$$x^4-4x^2+4y^4-8y^2+4 = (x^2+2xy+y^2-2)(x^2-2xy+y^2-2).$$

The left-hand side can be written as $p(x^2-2)+p(2y^2-2)$, where $p(z)=z^2-2$.

the same complexity.

References

- Aho, Hopcroft & Ullman, 1974
 Aho, A.V., Hopcroft, J.E. & Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
- Bogen et al., 1977
 Bogen, R.A. et al., MACSYMA Reference Manual (version 9), M.I.T Laboratory for Computer Science, 1977.
- Bremner, 1981
 Bremner, A., On Reducibility of Trinomials. Glasgow Math. J., 22(1981), pp. 155-156. Zbl. 464.12001. MR 82i:10079.
- Capelli, 1898
 Capelli, A., Sulla Riduttibilità delle equazioni algebriche. Nota secunda. Rend. Accad. Sc. Fis. Mat. Soc. Napoli, Ser. 3, 4(1898), pp. 84-90.
- Caviness, 1975
 Caviness, B.F., More on Computing Roots of Integers. SIGSAM Bulletin 9(1975) 3, pp. 18-20, 29.
- Davenport, Lewis & Schinzel, 1961
 Davenport, H., Lewis, D.J. & Schinzel, A., Equations of the form $f(x)=g(y)$. Quart. J. Math. (Oxford) (2) 12(1961) pp. 102-104. H. Davenport, Collected Papers, Academic Press, 1977, Vol. IV, pp. 1711-1719.
- Fried, 1973
 Fried, M., The Field of Definition of Function Fields and a Problem in the Reducibility of Polynomials in Two Variables. Illinois J. Math. 17(1973) pp. 128-146.
- Hearn, 1973
 Hearn, A.C., REDUCE-2 User's Manual. Report UCP-19, University of Utah, 1973.
- Knuth, 1981
 Knuth, D.E., The Art of Computer Programming, vol. II, Seminumerical Algorithms (2nd. ed.). Addison-Wesley, 1981.
- Lang, 1970
 Lang, S., Algebraic Number Theory. Addison-Wesley, 1970.
- Lenstra et al., 1982
 Lenstra, A.K., Lenstra, H.W., Jr. & Lovász, L., Factoring Polynomials with Rational Coefficients. Preprint IW 195/82, Afdeling Informatica. Mathematisch Centrum, Amsterdam.
- Ljunggren, 1960
 Ljunggren, W., On the Irreducibility of Certain Trinomials and Quadrinomials. Math. Scand 8(1960) pp. 65-70.
- Mikusiński & Schinzel, 1964
 Mikusiński, J. & Schinzel, A., Sur la réductibilité de certains trinômes. Acta Arithmetica 9(1964) pp. 91-95.
- Moore & Norman, 1981

- Moore,P.M.A. & Norman,A.C., Implementing a Polynomial Factorization and GCD Package. Proc. SYMSAC 81, ACM, New York, 1981, pp. 109-116.
- Schinzel,1962
 Schinzel,A., Solution d'un problème de K. Zarankiewicz sur les suites de puissances consécutives de nombres irrationnels. Colloq. Math. 9(1962), pp. 291-296.
- Schinzel,1963
 Schinzel,A., Some Unsolved Problems on Polynomials. Matematicka Biblioteka 25(1963) pp. 63-70.
- Schinzel, 1973
 Schinzel,A., A General Irreducibility Criterion. J. Indian Math. Soc. (N.S.) 37(1973) pp. 1-8.
- Schinzel, 1978
 Schinzel,A., Reducibility of lacunary polynomials III. Acta Arithmetica 34(1978) pp. 227-266.
- Schinzel,1982
 Schinzel,A., Selected Topics on Polynomials. Universiy of Michigan Press, Ann Arbor, Michigan, 1982.
- Schinzel,1983
 Schinzel,A., Private Communication.
- Schnorr,1981
 Schnorr,C.P., Refined Analysis and Improvement On Some Factoring Algorithms. Proc. 8th. Colloquium on Automata, Languages and Programming (Springer Lecture Notes in Computer Science 115, 1981), pp. 1-15. Zbl. 469.68043.
- Tverberg,1960
 Tverberg,H., On the Irreducibility of the Trinomials x^n+x^m+1 . Math. Scand. 8(1960) pp. 121-126.
- van der Waerden,1949
 van der Waerden,B.L., Modern Algebra, vol. I. Ungar, New York, 1949 (trans. from Moderne Algebra, 2nd. ed., Springer, 1937)
- Vaughan,1974
 Vaughan,R.C., Bounds for the Coefficients of Cyclotomic Polynomials. Michigan Math. J. 21(1974), pp. 289-295.
- Wang,1978
 Wang,P.S., An Improved Multivariable Polynomial Factorising Algorithm. Math. Comp. 32(1978) pp. 1215-1231. Zbl. 383.10035.
- Zippel,1981
 Zippel,R.E., Newton's Iteration and the Sparse Hensel Algorithm. Proc. SYMSAC 81, ACM, New York, 1981, pp. 68-72.

**Early Detection of True Factors in Univariate
Polynomial Factorization**

Paul S. Wang*

Department of Mathematical Sciences
Kent State University
Kent, Ohio, USA 44242

ABSTRACT

An algorithm which recovers a rational number from its image modulo a suitable prime or prime power is used in the p-adic lifting stage of univariate polynomial factorization to detect the formation of true factors. Such detections are usually made long before the modulus reaches the coefficient bound. The early removal of factors results in a reduced lifting problem and a lower coefficient bound. Algorithms are specified. Actual computer timing data are included.

1. Introduction

Recent research results of Lenstra et al. [5] have shown that factoring univariate polynomials over the rational numbers is of polynomial-time complexity. As part of his thesis, Kaltofen [4] has shown a similar complexity property for the factorization of multivariate polynomials. These are important theoretical findings. The polynomial-time algorithms developed in these studies are not competitive in practice with the factoring algorithms based on the Berlekamp and Hensel algorithms (referred to as p-adic algorithms in the sequel). Versions of the p-adic factoring algorithms are implemented on such symbolic computation systems as SCRATCHPAD [2], REDUCE [3],[8]

* Work reported herein has been supported in part by the National Science Foundation under Grant MCS 82-01239 and in part by the Department of Energy under Grant DE-AS02-ER7602075-A010.

and MACSYMA [10],[13]. Thus, continued improvements in the p-adic algorithms are of significance, even though such improvements do not make them polynomial-time algorithms.

Let $f(x)$ be a primitive polynomial with integral coefficients.

$$f(x) = c_n x^n + \dots + c_1 x + c_0$$

We shall consider the problem of obtaining the irreducible factors of $f(x)$ over the integers, \mathbb{Z} . Since repeated factors can be removed through polynomial gcd computations involving $f(x)$ and its first derivative, we assume that $f(x)$ has no repeated factors (i.e. $f(x)$ is squarefree). Modern computer-based p-adic factoring systems use the following algorithm.

- (1) Choose a small integer prime p in \mathbb{Z} such that $f_0(x) = f(x) \pmod{p}$ remains squarefree in the field \mathbb{Z}_p .
- (2) The irreducible factors of $f_0(x)$ over \mathbb{Z}_p are obtained using Berlekamp's algorithm for small fields.
- (3) The factors of $f_0(x)$ are then p-adically lifted to factors of $f(x)$ modulo p^j for successively larger integers $j > 1$.
- (4) As p^j becomes greater than some predetermined bound, B , on the size of the coefficients of the divisors of $f(x)$, the actual factors of $f(x)$ can be derived from the factors in the ring \mathbb{Z}_{p^j} .

The coefficient bound B depends on the degree and the coefficients of $f(x)$. Let b_i be the coefficient of the x^i term in any divisor $Q(x)$ of $f(x)$ then, according to Mignotte [5],

$$|b_i| \leq \binom{\deg(Q)}{i} \|f\|, \quad \|f\| = (\sum |c_i|^2)^{1/2}$$

This bound is quite sharp [7]. For the purpose of factorization, the

value used for B can be derived from the above as

$$B = 2 \left(\frac{n}{n/2} \right) |f|.$$

Because the bound B has to cover the worst case, it is usually very much larger than necessary in practice. This can cause several extra iterations in step (3) with calculations involving ever larger coefficients.

A method is described to obtain true factors of $f(x)$ over \mathbb{Z} as the lifting is being carried out. That is, at the end of each iteration in step (3), the modulo p^j factors will be examined and any which leads to a true factor will be removed from further lifting. Each true factor thus found reduces the problem and the time required to perform the overall factorization. In section 2 a rational coefficient recovering algorithm is described which is needed for the early detection of true factors that is discussed next in section 3. Actual machine timing data using the VAXIMA system, a version of MACSYMA for VAX-UNIX, are contained in section 4.

2. Recovering a Rational Number from Its Modular Image

A brief description of an algorithm for recovering a rational number a/b from its image modulo a suitable prime or prime power is given. This algorithm is the key to our ability to detect true factors early. The discovery of this application came through research in p -adic algorithms for polynomial partial fraction expansions (PFE) [12]. Unlike factorization, the PFE of a rational function with integral coefficients involve rational numbers. It is not surprising that a p -adic PFE algorithm will need an algorithm for recovering rational numbers from their modular images.

We state a theorem and an algorithm with an example for illustration. The proofs are omitted. Interested readers are referred to

[11] and [12] for more details.

Theorem : Given positive integers c and m , $c < m$, if there exists a reduced rational number a/b such that

- (1) $a/b \equiv c \pmod{m}$,
- (2) $a^2 < m/2$,
- (3) $b^2 < m/2$,

then (i) a/b is unique, and (ii) a and b can be computed from c and m by the following algorithm.

```

ALGORITHM R (c, m)
R1. m2 := SQRT(m/2);
R2. s := (1, 0, m); t := (0, 1, c);
R3. WHILE t3 <= m2 DO
    {q := s3 / t3; r := s - q*t; s := t; t := r};
R4. IF ABS(t2) >= m2 THEN ERROR("a/b does not exist");
R5. a := SIGN(t2)*t3; b := ABS(t2);
R6. RETURN((a,b)).
```

The above theorem and algorithm effectively allow us to conduct computations involving rational numbers in a field or ring \mathbb{Z}_m and recover the result with rational coefficients later. Note for any a/b to be recovered, we must also make sure that b and m are relatively prime. Otherwise, b^{-1} does not exist in the ring \mathbb{Z}_m .

Consider an example where $m = 23^5 = 6436343$ and $c = 4521075$. Using ALGORITHM R, we set m_2 to 1794, s to the vector $(1, 0, 6436343)$ and t to $(0, 1, 4521075)$ initially. As we carry out the algorithm, the vector t obtains the final value $(-262, 494, 29)$. Since both 29 and 494 are less than m_2 , the rational number returned is $29/494$.

If a smaller modulus is used, e.g. $m = 23^4 = 279841$, we observe

$$29/494 = 43619 \pmod{23^4}.$$

However, the same rational number can not be obtained by ALGORITHM R given $c = 43619$ and $m = 279841$ because 494 is greater than $(m/2)^{1/2}$. The application of ALGORITHM R in such cases will lead either to an error in step R4, as this case would, or to some other a/b .

3. Early Detection of True Factors

An algorithm is presented which enables the detection of true factors of the given polynomial $f(x)$ as the p -adic lifting progresses. This algorithm relies on the rational coefficient recovering algorithm described in the previous section.

At the end of the j th iteration in the p -adic lifting phase (step 3 in section 1), we have

$$f(x) = u_1(x)u_2(x)\dots u_r(x) \pmod{p^j}, \quad j \geq 1,$$

where p is the prime used in lifting and the u_i are irreducible factors of $f(x)$ modulo p^j . Each $u_i(x)$ will be examined to determine whether a true factor can be derived from it at this point. If $u_i(x)$ is not an extraneous factor and the modulus is large enough then the corresponding true factor can always be derived by the early detection algorithm. Any true factors found are removed from the lifting process which also results in a lower coefficient bound. The lifting is continued with the remaining factors.

If $u(x)$ is any factor of $f(x)$ modulo m , algorithm F is used to derive from $u(x)$ a true factor of $f(x)$ over \mathbb{Z} if possible. The algorithm either returns the true factor found or false if not found.

Note that we expect the denominator of any rational number recovered to divide $lc(f)$. Such denominators are relatively prime to the modulus, m , which is a power of a small prime that does not divide

$\text{lc}(f)$. If $u(x)$ does not lead to a true factor using algorithm F, then either $u(x)$ is an extraneous factor or m is not large enough. If $u(x)$ is not an extraneous factor and $u_0(x)$ is the corresponding true factor of $f(x)$, then u_0 can be derived from $u(x)$ if the rational coefficients of $\text{lc}(u_0)^{-1} * u_0$ can all be recovered with algorithm R.

```

ALGORITHM F(f(x),u(x),m)

F1. IF (both f and u are monic) THEN
     IF (u divides f over Z) THEN RETURN(u)
     ELSE RETURN(false);

F2. IF (u is not monic) THEN u := lc(u)-1 * u (mod m);

F3. m2 := SQRT(m / 2);

F4. FOR each coefficient, c > m2, of u DO
     (i) a/b := R(c, m2),
         if the call to algorithm R fails
         then return (false);
     (ii) IF (b does not divide lc(f)) THEN RETURN(false);
     (iii) replace the coefficient c in u(x) by a/b;

F5. /* at this point the polynomial u may */
/* contain rational coefficients */
Clear the denominators of u.

F6. IF (u divides f) THEN RETUEEN(u) ELSE RETURN(false).

```

All factors found are divided out of $f(x)$. A new coefficient bound is computed from the reduced $f(x)$ and the p-adic lifting continues to the step $j+1$ with the remaining $u_i(x)$.

Note that if the given polynomial $f(x)$ is monic, then early detection reduces to the trivial task of trial division which can be done without algorithm R. The early detection of true factors does introduce some additional computation in the lifting stage, although algorithm R is very fast. To lessen this overhead, we may want to apply the detection technique only when the modulus becomes reasonably large. The "engineering" question of where in the p-adic lifting process to begin detecting true factors is interesting. Let r be the number of factors being lifted. Several reasonable options are : (1) when the modulus is bigger than the r -th root of the maximum-size

coefficient in $f(x)$, (2) when the modulus is larger than the r th root of B and (3) just before the modulus exceeds single precision. Experimental evidence is still accumulating for each of these options.

4. An example

Consider factoring the polynomial $f(x)$ over the integers where

$$\begin{aligned} f(x) = & 90090x^7 + 115962x^6 - 708513x^5 - 780790x^4 + 773310x^3 \\ & + 441398x^2 - 352292x + 64680 \end{aligned}$$

The polynomial $f(x)$ is squarefree and its irreducible factorization in \mathbb{Z}_{17} are obtained by an algorithm due to Berlekamp [1]. Thus,

$$f(x) = (7x^3 - 8x^2 + x - 7)(x^2 + 8x - 4)(x^2 + 6x - 2) \pmod{17}$$

Note that the three factors are unique up to units in \mathbb{Z}_{17} . Therefore, the value of each coefficient can be quite different from the above yet the factorization modulo 17 still holds. One iteration of the p -adic lifting algorithm will result in the following.

$$\begin{aligned} f(x) = & (-78x^3 + 111x^2 + 52x - 126) \\ & (x^2 - 111x - 21)(x^2 - 96x - 104) \pmod{17^2} \end{aligned}$$

The factors of $f(x)$ modulo 17^2 are derived from the corresponding ones modulo 17. The coefficient bound, B , for this example is 17199492. However, using the early detection algorithm, an actual factor of $f(x)$ over \mathbb{Z} can be found now.

Let us examine the factor

$$-78x^3 + 111x^2 + 52x - 126$$

After being multiplied by the inverse of -78 in the ring \mathbb{Z}_{289} , this factor becomes

$$x^3 - 57x^2 - 97x + 135$$

If this form is the image of a true factor $h(x)$ of $f(x)$ then each coefficient is congruent to the corresponding rational coefficient of $\text{lc}(h)^{-1}h(x)$. Thus, we can find $h(x)$ by recovering the rational numbers from the modulo 289 coefficients.

We found $-57 \equiv 4/5 \pmod{289}$. Similarly, the coefficient of the x term, -97 , is $-2/3 \pmod{289}$ and the constant term $2/15 \pmod{289}$. After clearing the denominators we obtain

$$15x^3 + 12x^2 - 10x + 2,$$

which divides $f(x)$ over \mathbb{Z} (a true factor). At this point, applying the same techniques to the remaining two factors does not lead to true factors. This is because, the integers involved in the true factors exceed $(289/2)^{1/2}$. In this example, those factors will be found as soon as the modulus becomes large enough.

5. Timing Comparisons

To test the effect of the early detection algorithm on the running times of univariate polynomial factorization, many examples were run with the detection mechanism enabled or disabled. Six examples are included here in Table 1. These examples were run on the VAXIMA system (a version of MACSYMA on the DEC-VAX) at Kent State University. The computer is a VAX-11/780 with an effective memory cycle time of about 300 nanoseconds.

Times shown in Table 1 are total factorization times in seconds with garbage collection time included. The difference made by the early detection is rather remarkable considering it is only a small part of the over-all univariate factoring algorithm. The problems (1) through (5) are listed in factored form here. Problem 6 is the degree 40 univariate polynomial that arose in SIGSAM problem 7. This polynomial also appeared in a Scientific American article on Computer Algebra [9].

- $$(1) \quad f(x) = (906x^4 + 126x^3 - 504x^2 + 1661)(784532x^2 + 36909x - 1314692)$$
- $$(2) \quad g(x) = (300x^2 + 655x - 456)(391x^3 + 34x - 46)(38751x^4 + 187x - 9784362)$$
- $$(3) \quad (15x^3 + 12x^2 - 10x + 2)(42x^2 + 14x - 87)(1435x^2 + 722x - 981)$$
- $$(4) \quad (234x^5 + 98x^4 - 1234x + 8)(192837x^5 - x^3 + 1287x^2 - 1)(999x^3 + 765x^2 - x - 1)$$
- $$(5) \quad f(x)g(x)$$

TABLE 1

Timing Comparisons

Prob.	with early detection	without early detection
1	1.18	1.53
2	5.38	11.23
3	2.3	3.07
4	12.1	22.4
5	21.78	35.61
6	124.78	242.46

For these examples, it is interesting to know whether there are any extraneous factors, what primes (p) are used, what are the coefficient bounds (B), and the final values of the lifted modulus (M). We list such information for each problem below.

problem 1 : B = 13679360556, p = 5, M = 390625.

The (mod 5) factors are

$$2x^4 + 2x^3 + 2x^2 + 2, \quad x^2 + 2x - 1$$

problem 2 : B = 64961591070240, p = 7, M = 33232930569601.

The (mod 7) factors are

$$-x^3 - 2, \quad x^2 + 2x - 2, \quad x^2 + 3x + 1, \quad x - 2, \quad x$$

problem 3 : B = 62770176, p = 17, M = 83521.

The (mod 17) factors are

$$7x^3 - 8x^2 + x - 7, \quad x^2 + 6x + 4, \quad x + 2, \quad x + 4$$

problem 4 : B = 10730435211020, p = 5, M = 152587890625.

The (mod 5) factors are

$$x^5 + 2x^4 - x + 2, \quad x^3 + x + 1, \quad x^2 - x + 1, \quad x^2 + 2, \quad x + 1$$

problem 5 : B = 1645392911484647797394720, p = 19,

$$M = 288441413567621167681.$$

The (mod 19) factors are

$$\begin{aligned} & -3x^4 + 6x^3 - 5x^2 + 4, \quad x, \quad x - 7, \\ & x^4 - 6x + 3, \quad x^3 - 9x + 1, \quad x^2 - 9x - 2 \end{aligned}$$

problem 6 : B = 37051482265257197570443532376, p = 19,

$$M = 16983563041.$$

The (mod 19) factors are

$$\begin{aligned} & 6x^{10} - 4x^9 + 5x^7 + 3x^6 - x^5 - 4x^4 + 2x^3 - 2x^2 - 3x + 7, \\ & x^{10} + 2x^9 - 4x^8 - x^7 + 9x^6 + 4x^5 + 6x^4 - 2x^3 - 8x^2 + 7x + 9, \\ & x^{10} - 8x^9 + x^8 + 2x^7 - 5x^6 + 7x^5 - 6x^4 + 3x^3 - 6x^2 + 4x - 3, \\ & x^6 - 5x^4 + 7x^3 + 4x^2 + 7x - 7, \\ & x^4 + 2x^3 - 5x^2 - 5x - 1 \end{aligned}$$

6. References

- [1] E. R. Berlekamp, "Factoring Polynomials over Finite Fields," Bell System Tech. J., V. 46, 1967, pp. 1853-1859.
- [2] J. H. Griesmer, R. D. Jenks and D. Y. Yun, "SCRATCHPAD User's Manual," IBM Research Report RA70, June 1975.
- [3] A. C. Hearn, REDUCE II a system for Algebraic Computation, The University of Utah, Computer Sciences Department, 1979.
- [4] E. Kaltofen, "On the Complexity of Factoring Polynomials with integer Coefficients," Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, New York, December 1982.

- [5] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring Polynomials with Rational Coefficients," Report 82-05, Amsterdam Mathematisch Instituut, Universiteit van Amsterdam, 1982.
- [6] M. Mignotte, "An Inequality about Factors of Polynomials," Mathematics of Computation, Vol. 28, 1974, pp. 1153-1157.
- [7] M. Mignotte, "Some Inequalities About Univariate Polynomials," Proceedings of the 1981 ACM symposium on Symbolic and Algebraic Computation, Snowbird, Utah , August 5-7, 1981, pp. 195-199.
- [8] P. M. A. Moore and A. C. Norman, "Implementing a polynomial factorization and GCD package," Proceedings, ACM Symposium on Symbolic and Algebraic Computation, August 1981, pp. 109-113.
- [9] R. Pavelle, M. Rothstein and J. Fitch, "Computer Algebra," Scientific American, Vol 245 No. 6, December 1981, pp. 136-152.
- [10] P. S. Wang, "Parallel p-adic Constructions in the Univariate Polynomial Factoring Algorithm," Proceedings, MACSYMA Users' Conference (1979), Cambridge, MA, MIT pp. 310-318.
- [11] P. S. Wang, M. Guy, and J. Davenport, "p-adic Reconstruction of Rational Numbers," ACM SIGSAM Bulletin, Vol. 16, May 1982, pp. 2-3.
- [12] P. S. Wang, "A p-adic Algorithm for Univariate Partial Fractions," Proceedings of the 1981 ACM symposium on Symbolic and Algebraic Computation, Snowbird, Utah , August 5-7, 1981, pp. 212-217.
- [13] MACSYMA Reference Manual version nine, the MATHLAB group, Laboratory for Computer Science, MIT, Camb. MA. 02139

On the Complexity of Finding Short Vectors in Integer Lattices*

by

Erich Kaltofen

University of Toronto
Department of Computer Science
Toronto, Ontario M5S 1A4, Canada

Abstract. In [Lenstra, A., et al. 82] an algorithm is presented which, given n linearly independent n -dimensional integer vectors, calculates a vector in the integer lattice spanned by these vectors whose Euclidean length is within a factor of $2^{(n-1)/2}$ of the length of the shortest vector in this lattice. If B denotes the maximum length of the basis vectors, the algorithm is shown to run in $O(n^6(\log B)^3)$ binary steps. We prove that this algorithm can actually be executed in $O(n^6(\log B)^2 + n^5(\log B)^3)$ binary steps by analyzing a modified version of the algorithm which also performs better in practice.

1. Introduction

Various diophantine problems can be reduced to the problem of finding a short or a shortest vector in an integer lattice. Among them are integer linear programming with a fixed number of variables [Lenstra, H. 81], integer polynomial factorization [Lenstra, A., et al. 82], [Kaltofen 82], and the breaking of some variants of the Merkle-Hellman knapsack encryption scheme [Lagarias 82], [Odlyzko 82]. A. Lenstra, H. Lenstra and L. Lovász [Lenstra, A., et al. 82] present an algorithm which, given a set of n linearly independent n -dimensional integer vectors whose Euclidean lengths are bounded by B , finds a vector in the \mathbb{Z} -span of these vectors the length of which is within a factor of $2^{(n-1)/2}$ of the length of the shortest vector in this lattice. Their algorithm is shown to run in $O(n^6(\log B)^3)$ binary steps provided that classical integer arithmetic is used. In this paper we present a modified version of this algorithm which not only seems to perform better in practice, but we also prove that this version runs in $O(n^6(\log B)^2 + n^5(\log B)^3)$ binary steps when using classical integer arithmetic. With our result one can prove that factoring an integer polynomial f of degree n can be accomplished in $O(n^{11} + n^8(\log |f|)^3)$ binary steps, where $|f|$ is the length of the coefficient vector of f (cf. [Lenstra, A., et al. 82]).

If one employs fast integer multiplication the running times compare as follows.

* The research for this paper has been partially supported by the Connaught Fund, Grant # 3-370-126-80.

Using a $O(K^{1+\varepsilon})$ complex K -digit times K -digit integer multiplication procedure where ε is a small constant > 0 (cf. [Knuth 81, p. 280]): $O(n^5(\log B)^{2+\varepsilon} + n^{5+\varepsilon}(\log B)^2)$ (ours) vs. $O(n^{5+\varepsilon}(\log B)^{2+\varepsilon})$ ([Lenstra, A., et al. 82]).

Using a $O(K \log K \log \log K)$ complex K -digit times K -digit integer multiplication procedure (cf. [Schönhage and Strassen 71]): $O(n^5(\log B)^2 \log K \log \log K)$ with $K = n + \log B$ (ours) vs. $K = n \log B$ ([Lenstra, A., et al. 82]). Notice that in this case both bounds are asymptotically equal and our improvement only lowers the constant multiplier for the given complexity.

In order to make this paper sufficiently self-contained we shall repeat some of the arguments presented in [Lenstra, A., et al. 82] as well as adopt most of the notation used there. Let $b_1, \dots, b_n \in \mathbb{Z}^n$ be linearly independent (over \mathbb{Q} , the rationals). By b_1^*, \dots, b_n^* we denote the orthogonalization of this basis, namely

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \quad 1 \leq i \leq n \quad (1.1)$$

$$\mu_{ij} = \frac{(b_i, b_j^*)}{|b_j^*|^2}, \quad 1 \leq j < i \leq n \quad (1.2)$$

where (\cdot, \cdot) denotes the scalar product and $|\cdot|$ the square norm. The basis b_1, \dots, b_n is called *reduced* if

$$|b_k^*|^2 \geq \frac{1}{2} |b_{k-1}^*|^2 \text{ for } 1 < k \leq n. \quad (1.3)$$

Notice that our notion of reduced basis is weaker than the one given in [Lenstra, A., et al. 82].

Lemma 1 (cf. [Lenstra, A., et al. 82, Proposition 1.11]): Let $b_1, \dots, b_n \in \mathbb{Z}^n$ form a reduced basis. Then for any non-zero vector $x \in \mathbb{Z}b_1 + \dots + \mathbb{Z}b_n$

$$2^{n-1} |x|^2 \geq |b_1|^2.$$

Hence $|b_1|$ is within a factor of $2^{(n-1)/2}$ of the norm of the shortest vector in the lattice spanned by b_1, \dots, b_n .

Proof: Let $x = \sum_{i=1}^l r_i b_i = \sum_{i=1}^l r_i^* b_i^*$ with $r_i \in \mathbb{Z}$, $r_i^* \in \mathbb{Q}$ and $r_l \neq 0$, $1 \leq l \leq n$. Since b_i^* is orthogonal to b_i for $i = 1, \dots, l-1$, $(x, b_l) = r_l (b_l, b_l) = r_l^* (b_l^*, b_l^*)$. But $(b_l, b_l) = (b_l^*, b_l^*) \neq 0$. Therefore, $r_l^* = r_l$, which is an integer $\neq 0$. It follows that $|x|^2 = \sum_{i=1}^l (r_i^*)^2 |b_i^*|^2 \geq (r_l^*)^2 |b_l^*|^2 \geq |b_l^*|^2 \geq |b_1^*|^2 / 2^{l-1}$, the last inequality by using (1.3) inductively. Since $b_1^* = b_1$ the lemma is proven. \square

2. The Basis Reduction Algorithm

[Given n linearly independent vectors b_1, \dots, b_n with integer entries this algorithm transforms this basis into a reduced one.]

- (I) [Initialize the arrays μ and β such that μ_{ij} , $1 \leq j < i \leq n$, and $\beta_l = |b_l^*|^2$, $1 \leq l \leq n$, satisfy (1.1) and (1.2):]

FOR $i \leftarrow 1$ TO n DO

[The following loop is skipped for $i = 1$.]

FOR $j \leftarrow 1$ TO $i-1$ DO

$$\mu_{ij} \leftarrow \frac{1}{\beta_j} \left((b_i, b_j) + \sum_{l=1}^{j-1} \mu_{jl} \mu_{il} \beta_l \right).$$

$$\beta_i \leftarrow |b_i^*|^2 - \sum_{l=1}^{i-1} \mu_{il}^2 \beta_l.$$

$k \leftarrow 2$.

- (L) [At this point μ and β correspond to the orthogonalization of b_1, \dots, b_n . Moreover b_1, \dots, b_{k-1} is reduced, i.e.

$$|b_l^*|^2 \geq \frac{1}{2} |b_{l-1}^*|^2 \text{ for } 1 < l \leq k-1, \text{ and } |\mu_{ij}| \leq \frac{1}{2} \text{ for } 1 \leq j < i \leq k-1. \quad (2.1)$$

IF $k=n+1$ THEN RETURN (b_1, \dots, b_n).

IF $k=1$ THEN $k \leftarrow 2$; GOTO step (L).

Call algorithm S given below.

[Now (2.1) is also valid for $i = k$, i.e.

$$|\mu_{kj}| \leq \frac{1}{2} \text{ for } 1 \leq j < k. \quad (2.2)$$

IF $\beta_k \geq \frac{1}{2} \beta_{k-1}$ THEN $k \leftarrow k+1$; GOTO step (L).

[At this point

$$|b_k^*|^2 < \frac{1}{2} |b_{k-1}^*|^2 \leq \left[\frac{3}{4} - \mu_{k,k-1}^2 \right] |b_{k-1}^*|^2 \quad (2.3)$$

the last inequality because of (2.2) for $j = k-1$.]

Interchange b_{k-1} and b_k and update the arrays μ and β such that (1.1) and (1.2) is satisfied for the new order of basis vectors.

[As we will explain below, the only entries which change are β_{k-1} , β_k and μ_{ij} , $i = k-1, k$, $1 \leq j < i$, as well as $\mu_{i,k-1}$, μ_{ik} , $k < i \leq n$. Let γ and ν denote the updated contents of β and μ , then, according to [Lenstra, A., et al. 82, 1.22]

$$\gamma_{k-1} = \beta_k + \mu_{k,k-1}^2 \beta_{k-1}, \quad \nu_{k,k-1} = \frac{\mu_{k,k-1} \beta_{k-1}}{\gamma_{k-1}}, \quad (2.4)$$

$$\gamma_k = \beta_{k-1} - \nu_{k,k-1}^2 \gamma_{k-1} = \frac{\beta_{k-1} \beta_k}{\gamma_{k-1}}, \quad (2.5)$$

$$\left. \begin{aligned} \nu_{i,k-1} &= \mu_{i,k-1} \nu_{k,k-1} + \mu_{ik} \beta_k / \gamma_{k-1} \\ \nu_{i,k} &= \mu_{i,k-1} - \mu_{ik} \mu_{k,k-1} \end{aligned} \right\} \text{for } k < i \leq n, \quad (2.6)$$

$$\nu_{k-1,j} = \mu_{kj}, \nu_{k,j} = \mu_{k-1,j} \text{ for } 1 \leq j < k-1. \quad (2.7)]$$

$k \leftarrow k-1$; GOTO step (L). \square

Algorithm S:

[This algorithm replaces b_k by $b_k - \sum_{l=1}^{k-1} \lambda_l b_l$, $\lambda_l \in \mathbb{Z}$, such that the new μ_{kj} satisfy (2.2):]

(M) [Initialize the modulus:]

$B_k \leftarrow \max\{\beta_1, \dots, \beta_k\}; M \leftarrow \left\lceil \sqrt{(k+3)B_k} \right\rceil$. [M can also be chosen the least power of the radix larger than that.]

(L1) FOR $m \leftarrow k-1$ DOWNTO 1 DO step (L2)

(L2) [Make μ_{km} absolutely smaller than $\frac{1}{2}$:]

IF $|\mu_{km}| \geq \frac{1}{2}$ THEN

$r \leftarrow \text{ROUNDED}(\mu_{km})$. [ROUNDED(x) denotes largest integer z s.t. $|z-x| \leq \frac{1}{2}$.]

Replace b_k by $(b_k - rb_m) \bmod M$. (2.8)

[Notice that in this case $\lambda_m = r$, otherwise $\lambda_m = 0$.]

[Readjust μ_{ki} for $i \leq j < m$:]

FOR $j \leftarrow 1$ TO $m-1$ DO $\mu_{kj} \leftarrow \mu_{kj} - r\mu_{mj}$. (2.9)

$\mu_{km} \leftarrow \mu_{km} - r$.

ENDIF

(B) Balance the residual entries of $b_k \bmod M$ such that each modulus $> \frac{1}{2}M$ is replaced by its negative equivalent $\leq \frac{1}{2}M$. \square

Remark: If k is decremented in the main loop (L), (2.2) and (2.7) imply that $|\mu_{kj}| \leq \frac{1}{2}$ for the new index k . Hence the subsequent call to algorithm S has no effect and therefore can be omitted. In practice, one should keep a switch which is set/reset as k is in-/decremented, and only call the subprocedure S if this switch is set.

3. Partial Correctness of the Reduction Algorithm

That step (I) computes the correct μ 's and β 's follows by substituting $b_j^* = b_j + \sum_{l=1}^{j-1} \mu_{jl} b_l^*$ into (1.2). We first focus on the subprocedure S. Since the vector $\sum_{m=1}^{k-1} \lambda_m b_m$ in the subspace spanned by b_1, \dots, b_{k-1} is subtracted from b_k , neither b_i^* , $1 \leq i \leq n$, nor μ_{ij} , $i \neq k$, $1 \leq j < i$, change. By (1.2), the new μ_{kj} can be computed as

$$\mu_{kj} = \frac{1}{|b_j^*|^2} (b_k - \sum_{m=1}^{k-1} \lambda_m b_m, b_j^*) = \frac{(b_k, b_j^*)}{|b_j^*|^2} - \lambda_j - \sum_{m=j+1}^{k-1} \lambda_m \mu_{mj}.$$

observing that $(b_j, b_j^*) = |b_j^*|^2$. But this is exactly the value computed within the loops (L1) and (L2). After these loops are executed

$$|\mu_{kj}| \leq \frac{1}{2} \text{ for } 1 \leq j < k.$$

Therefore the new vector b_k satisfies

$$|b_k|^2 = |b_k^*|^2 + \sum_{j=1}^{k-1} \mu_{kj}^2 |b_j^*|^2 \leq B_k + (k-1) \frac{B_k}{4} = \frac{k+3}{4} B_k. \quad (3.1)$$

Thus the updated entries of b_k will be absolutely bounded by $\frac{1}{2}\sqrt{(k+3)B_k}$ and it suffices to compute these entries modulo an integer of twice this bound to obtain their true values in step (B).

The partial correctness of algorithm R can now be immediately concluded by the use of the invariant given at the label (L). The invariance of this statement is also easily established. The counter k is incremented only if $|b_k^*|^2 \geq \frac{1}{2}|b_{k-1}^*|^2$. Subprocedure S does not change any b_i^* or μ_{ij} for $i < k$. Hence with (2.2) the statement is true including the additional subscript. If the counter is decremented, b_i and μ_{ij} , $1 \leq i \leq k-2$, $1 \leq j < i$, are not at all affected by the updates (2.4) - (2.7), which implies the invariance in this case as well.

4. Complexity Analysis

Following the ingenious argument laid out in [Lenstra, A., et al. 82] we shall first show that algorithm R must terminate by proving that k can be decremented at most $O(n^2 \log B)$ times, where

$$B = \max\{|b_1|^2, \dots, |b_n|^2\}.$$

Of course, k can be incremented only an additional n times more which shows that the main loop (L) is executed $O(n^2 \log B)$ times.

Let d_l denote the Gramian of the vectors $\{b_1, \dots, b_l\}$ that is

$$d_l = \det((b_i, b_j))_{1 \leq i, j \leq l} = \prod_{i=1}^l |b_i^*|^2 \text{ for } 1 \leq i \leq n. \quad (4.1)$$

(cf. [Gantmacher 59, Sec. 9.3].) From (1.1) it follows that all b_i^* initially satisfy $|b_i^*|^2 \leq |b_l|^2 \leq B$. Hence at the beginning

$$d_l \leq B^l \text{ for } 1 \leq l \leq n. \quad (4.2)$$

The only time throughout the algorithm any of the Gramians changes is when some $|b_i^*|^2$ are updated. This only happens when k is decremented, that is, when b_{k-1} and b_k are interchanged. By (4.1) it follows that the Gramians d_l are invariant to the order of the vectors b_1, \dots, b_l , hence the exchange does not affect any d_l with $l \neq k-1$. For $d_{k-1} = \prod_{i=1}^{k-1} |b_i^*|^2$, the only of the $|b_i^*|^2$, $1 \leq i \leq k-1$, being updated is $|b_{k-1}^*|^2$, which is, by (2.4) and (2.3), replaced by

$$|b_k^*|^2 + \mu_{k,k-1}^2 |b_{k-1}^*|^2 < \frac{3}{4} |b_{k-1}^*|^2. \quad (4.3)$$

Therefore the new value of d_{k-1} is at least $\frac{3}{4}$ times smaller than the old one,

and since no other d_l changes, the same is true for the product $\prod_{l=1}^{n-1} d_l$. However, throughout the algorithm, this product remains a positive integer, which, by (4.2), is initially not greater than $B^{(n-1)n/2}$. Thus, the number of times k can be decremented is at most the number of times $\prod_{l=1}^{n-1} d_l$ can be reduced by the factor $\frac{3}{4}$, which is bounded by

$$\left\lfloor \frac{(n-1)n}{2} \log_{4/3} B \right\rfloor = O(n^2 \log B).$$

This establishes our initial claim.

Before we can derive the binary complexity of algorithm R, we need to estimate the lengths of intermediately computed integers. Since the entries in β and μ are rationals this includes predetermining their denominators. From (4.1) it follows that

$$|b_i^*|^2 = \frac{d_l}{d_{l-1}} \text{ for } 2 \leq l \leq n. \quad (4.4)$$

It can be easily established (cf. [Lenstra, A., et al. 82, 1.29]) that

$$d_j \mu_{ij} \in \mathbf{Z} \text{ for } 1 \leq j < i \leq n.$$

Therefore all calculated denominators have, by (4.2), $O(n \log B)$ digits.

We now estimate $|b_i|^2$, $|b_i^*|^2$ and μ_{ij} throughout algorithm R. Initially

$$|b_i^*|^2 \leq B \text{ for } 1 \leq i \leq n, \quad (4.5)$$

and it is easy to see that this condition remains true during the algorithm. The only changes to $|b_i^*|^2$ occur in (2.4) and (2.5). But

$$0 < \gamma_{k-1} < \frac{3}{4} \beta_{k-1} \leq \frac{3}{4} B < B,$$

by (4.3), and by (2.5),

$$\gamma_k = \beta_{k-1} - \nu_{k,k-1}^2 \gamma_{k-1} < \beta_{k-1} \leq B.$$

Therefore, the new values of $|b_{k-1}^*|^2$ and $|b_k^*|^2$ are not greater than B . This also implies that during algorithm R

$$|b_i|^2 \leq nB \text{ for } 1 \leq i \leq n. \quad (4.6)$$

The set $\{b_1, \dots, b_n\}$ is only changed by the subprocedure S, where b_k is updated. From (3.1) we conclude that the new value of $|b_k|^2$ satisfies

$$|b_k|^2 \leq \frac{k+3}{4} B_k < nB,$$

the latter inequality by (4.5). Finally, we conclude that at the label (L)

$$\mu_{ij}^2 \leq \frac{|b_i|^2}{|b_j^*|^2} = \frac{d_{j-1} |b_i|^2}{d_j} \leq d_{j-1} |b_i|^2 \leq nB^j \text{ for } 1 \leq j < i \leq n,$$

using Schwarz' inequality on (1.2), then (4.4), (4.2) and (4.6) (cf. [Lenstra, A. et al. 82, 1.35]). It now follows from specification of algorithm R and (2.4) - (2.7)

that any intermediate integer or any rational numerator has at most $O(n \log B)$ digits. Therefore, each arithmetic operation in algorithm R takes $O(n^2(\log B)^2)$ binary steps, using classical integer arithmetic routines.

Now let T_S denote the binary complexity of the subprocedure S. The arithmetic complexity of step (I) is $O(n^3)$, that of step (L), excluding the call to algorithm S, $O(n^3 \log B)$. Therefore, the binary complexity of the whole reduction is

$$O(n^5(\log B)^3 + T_S n^2 \log B) \quad (4.7)$$

with classical arithmetic. It remains to determine T_S .

Lemma 2: Let $0 < K \leq N$ be integers. To multiply a K -digit integer with an N -digit integer or to compute the integer quotient and remainder of a $K+N$ -digit integer divided by an N -digit integer has binary complexity $O(M(K)N/K)$ where $O(M(K))$ is the binary complexity of multiplying two K digit integers.

Proof: The classical complexity as well as the one for fast multiplication can be found in [Knuth 82, Sec. 4.3, esp. Sec. 4.3.3, Exercise 13]. To establish the fast division complexity, we observe that by grouping the digits of the dividend into integers of K digits, we can determine the quotient by just considering the first few groups [Knuth 82, loc. cit.]. This amounts to dividing integers of size $O(K)$ and then multiplying $O(N/K)$ integers of that size. Division and multiplication can be accomplished in $O(M(K))$ binary steps. \square

It is clear that the arithmetic complexity of algorithm S is $O(n^2)$. Each coordinate update of b_k in (2.8) as well as assignment (2.9) is executed that many times. We first focus on the binary complexity of the later, assignment (2.9). Upon entry to algorithm S,

$$|\mu_{kj}| < \sqrt{n2^n B} = C \text{ for } 1 \leq j < k. \quad (4.9)$$

To prove this, we use the invariant at label (L) and the fact that $|b_1^*| = |b_1| \geq 1$. For, by (4.6)

$$nB \geq |b_k|^2 = |b_k^*|^2 + \sum_{l=1}^{k-1} \mu_{kl}^2 |b_l^*|^2 > \mu_{kj}^2 |b_j^*|^2 \geq \mu_{kj}^2 \frac{|b_1^*|^2}{2^{j-1}} > \frac{\mu_{kj}^2}{2^n},$$

which proves (4.9). Let $\mu_{kj}^{(m)}$ denote μ_{kj} after (2.9) has been executed for the indices m and j , $j < m$. We can prove by induction that

$$|\mu_{kj}^{(m)}| < 2^{k-m} C. \quad (4.10)$$

For $m = k$, the $\mu_{kj}^{(k)}$ are the original μ_{kj} and (4.10) reduces to (4.9). We now deduce (4.10) for $m-1$ from (4.10) for m . By (2.9), $\mu_{kj}^{(m-1)} = \mu_{kj}^{(m)} - r \mu_{mj}$, $1 \leq j \leq m-2$, where r is the rounded value of $\mu_{k,m-1}^{(m)}$. Hence,

$$|r| < 2 |\mu_{k,m-1}^{(m)}| \leq 2^{k-m+1} C \quad (4.11)$$

and $|\mu_{mj}| \leq \frac{1}{2}$ by the invariant at label (L). Therefore, $|\mu_{kj}^{(m-1)}| \leq |\mu_{kj}^{(m)}| + |r| |\mu_{mj}| \leq 2^{k-m} C + \frac{1}{2} 2^{k-m+1} C = 2^{k-(m-1)} C$, as was to be shown. Thus,

throughout algorithm S

$$|\mu_{kj}| \leq 2^{k-1} C < \sqrt{n 2^{3n} B} \text{ for } 1 \leq j < k. \quad (4.12)$$

This means that r is of size $O(n + \log B)$ and by the above lemma $\text{ROUNDED}(\mu_{km})$ and the assignment (2.9) can be computed in $O((n + \log B)n \log B)$ binary steps. Obviously, the update of the coordinates of b_k in (2.8) can be achieved in fewer binary steps since $M \ll C$. In fact, by choosing M to be the least power of the radix larger than $\sqrt{(k+3)B}$, one can avoid the cost for the division by M when computing the modulus. Therefore,

$$T_S = O((n + \log B)n^3 \log B)$$

and the complexity for the whole algorithm, (4.7) and (4.8), becomes

$$O(n^6(\log B)^2 + n^5(\log B)^3),$$

assuming that the classical arithmetic procedures are employed. The running times cited in section 1 for fast integer arithmetic can be easily deduced from lemma 2 and the above analysis.

5. Conclusion

We have shown that the complexity bound in [Lenstra, A., et al. 82] for computing a short vector in an integer lattice is too pessimistic, in fact that the total exponent can be lowered by one. We claim that our version also has its practical merits. The main gain is during the update of b_k in (2.8), provided that M is chosen an appropriate power of the radix. If no modulus were taken, by (4.11) we would have to multiply with an r which could be $3n/2$ bits longer than $r \bmod M$. Since this multiplication might be executed $O(n^4 \log B)$ times the loss of time may become quite significant.

Other practical improvements have been suggested by A. Odlyzko [Odlyzko 82]. For example, the entries in μ and β could be made floating point numbers with extended precision. Then, whenever the loss of significant digits during roundoff becomes too great to decide $\beta_k \geq \frac{1}{2}\beta_{k-1}$ in algorithm R or calculate $\text{ROUNDED}(\mu_{km})$ in algorithm S, one can recompute the μ 's and β 's like in step (I) of algorithm R. This change has been successfully used to solve large problems by the VAXIMA version of the algorithm.

Acknowledgement. The presentation of this paper has greatly benefitted from the remarks of two unnamed referees.

REFERENCES

[Gantmacher 59]

Gantmacher, F. R.: Matrix Theory, vol. 1. New York: Chelsea 1959.

[Kaltofen 82]

Kaltofen, E.: A Polynomial-Time Reduction from Bivariate to Univariate Integral Polynomial Factorization. Proc. 23rd Symp. Foundations of Comp. Sci., IEEE 57-64 (1982).

[Knuth 81]

Knuth, D. E.: The Art of Computer Programming, vol.2, Seminumerical Algorithms, 2nd ed. Reading, MA: Addison Wesley 1981.

[Lagarias 82]

Lagarias, J.C.: The Computational Complexity of Simultaneous Diophantine Approximation Problems. Proc. 23rd Symp. Foundations of Comp. Sci., IEEE 32-39 (1982).

[Lenstra, A., et al. 82]

Lenstra, A. K., Lenstra, H. W., jr., Lovász, L.: Factoring Polynomials with Rational Coefficients. Report 82-05. Amsterdam: Mathematisch Instituut 1982.

[Lenstra, H. 81]

Lenstra, H.W., jr.: Integer Programming with a Fixed Number of Variables. Univ. Amsterdam: Math. Inst. Report 81-03, 1981.

[Odlyzko 82]

Odlyzko, A.M.: Private Communications 1982.

[Schönhage and Strassen 71]

Schönhage, A., and Strassen, V.: Schnelle Multiplikation grosser Zahlen. Computing 7, 281-292 (1971).

Factoring polynomials over algebraic number fields

A.K. Lenstra
Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

Abstract.

We present a polynomial-time algorithm for the factorization of univariate polynomials over algebraic number fields. Our algorithm is a direct generalization of the polynomial-time algorithm for the factorization of univariate polynomials over the rationals [7].

1. Introduction.

In [7] a polynomial-time algorithm was given to factor polynomials in one variable with rational coefficients. In this paper we generalize this result to polynomials in one variable with coefficients in an algebraic number field.

The existence of a polynomial-time algorithm for this problem is not surprising in view of [7]. Kronecker's idea of using norms reduced the problem to the factorization of univariate polynomials with rational coefficients, and in [5] it is shown that this reduction is polynomial-time. Here we pursue a direct approach to the factorization of polynomials over algebraic number fields. As suggested in [6: Section 5] we regard the irreducible factor we are looking for as an element of a certain integral lattice, and we prove that it is the 'smallest' element in this lattice. As we have seen in [7] this enables us to effectively compute this factor by means of a basis reduction algorithm for lattices.

The practical importance of our algorithm should not be overstated. This is mainly due to the rather slow basis reduction algorithm. For practical purposes we recommend the algorithm from [6], where the ideas of the lattice approach are combined with the well-known exponential-time factoring algorithm. In the algorithm from [6] the algebraic numbers are represented by their residues modulo a certain prime-power p^k . In the last step (trial divisions to determine the true factors), the algebraic num-

bers are restored in a unique way by means of a reduced basis of a certain integral lattice. Experiments have shown that this greatly reduces the running time (cf. [6]).

This paper is organized as follows. Section 2 contains some notation and definitions; furthermore we recall there some results from [7: Section 1]. Section 3 deals with the connection between factors and lattices; it generalizes the first part of [7: Section 2]. In Section 4 we give a global description of the factoring algorithm and we analyze its running time.

For a polynomial f we denote by δf the degree of f , by $lc(f)$ the leading coefficient of f , and f is said to be *monic* if $lc(f) = 1$.

2. Preliminaries.

Let the algebraic number field $\mathbb{Q}(\alpha)$ be given as the field of rational numbers \mathbb{Q} extended by a root α of a prescribed monic irreducible polynomial $F \in \mathbb{Z}[T]$, i.e. $\mathbb{Q}(\alpha) \cong \mathbb{Q}[T]/(F)$. This implies that the elements of $\mathbb{Q}(\alpha)$ can be represented as polynomials in α over \mathbb{Q} of degree $< \delta F$. We may assume that the degree of the minimal polynomial F is at least 2.

Similarly, we define $\mathbb{Z}[\alpha] \cong \mathbb{Z}[T]/(F)$ as the ring of polynomials in α over \mathbb{Z} of degree $< \delta F$, where multiplication is done 'modulo F '.

Let f be a monic polynomial in $\mathbb{Q}(\alpha)[X]$. In Section 4 we will describe how to choose a positive integer D such that

$$(2.1) \quad f \text{ and all monic factors of } f \text{ in } \mathbb{Q}(\alpha)[X] \text{ are in } \frac{1}{D}\mathbb{Z}[\alpha][X].$$

The algorithm to determine the irreducible factors of f in $\mathbb{Q}(\alpha)[X]$ that we will present, is very similar to the algorithm for factorization in $\mathbb{Z}[X]$ as described in [7]: first determine the factorization of f over some finite field ($\mathbb{Z}/p\mathbb{Z}$ in [7]), next extend this factorization to a factorization over a large enough ring ($\mathbb{Z}/p^k\mathbb{Z}$ in [7]), and finally use a lattice reduction algorithm to determine the factors over $\mathbb{Q}(\alpha)$. Therefore we first describe how to choose this finite field and this ring.

Let p be a prime number such that

$$(2.2) \quad p \text{ does not divide } D,$$

and let k be a positive integer. For $G = \sum_i a_i T^i \in \mathbb{Z}[T]$ and some integer l we denote by G_l or $(G \bmod p^l)$ the polynomial $\sum_i (a_i \bmod p^l) T^i \in (\mathbb{Z}/p^l\mathbb{Z})[T]$. In Section

4 we will see that we are able to determine p in such a way that we can compute a polynomial $H \in \mathbb{Z}[T]$ such that:

$$(2.3) \quad H \text{ is monic,}$$

$$(2.4) \quad H_k \text{ divides } F_k \text{ in } (\mathbb{Z}/p^k \mathbb{Z})[T],$$

$$(2.5) \quad H_1 \text{ is irreducible in } (\mathbb{Z}/p \mathbb{Z})[T],$$

$$(2.6) \quad (H_1)^2 \text{ does not divide } F_1 \text{ in } (\mathbb{Z}/p \mathbb{Z})[T].$$

It follows that H_1 divides F_1 in $(\mathbb{Z}/p \mathbb{Z})[T]$, and that $0 < \delta H \leq \delta F$.

This polynomial H , together with the prime number p and the integer k , gives us the possibility to construct the finite field and the ring we were looking for. We denote by q the prime-power $p^{\delta H}$ and by \mathbb{F}_q the finite field containing q elements. From (2.5) we derive that $\mathbb{F}_q \cong (\mathbb{Z}/p \mathbb{Z})[T]/(H_1)$. Remark that $\mathbb{F}_q \cong \{\sum_{i=0}^{\delta H-1} a_i \alpha_1^i : a_i \in \mathbb{Z}/p \mathbb{Z}\}$ where $\alpha_1 = (T \bmod (H_1))$ is a zero of H_1 . This enables us to represent the elements of \mathbb{F}_q as polynomials in α_1 over $\mathbb{Z}/p \mathbb{Z}$ of degree $< \delta H$. The finite field \mathbb{F}_q corresponds to $\mathbb{Z}/p \mathbb{Z}$ in [7]; we now define the ring that will play the role of $\mathbb{Z}/p^k \mathbb{Z}$ in [7]. Let $W_k(\mathbb{F}_q) = (\mathbb{Z}/p^k \mathbb{Z})[T]/(H_k)$ be a ring containing q^k elements. We have that $W_k(\mathbb{F}_q) = \{\sum_{i=0}^{\delta H-1} a_i \alpha_k^i : a_i \in \mathbb{Z}/p^k \mathbb{Z}\}$ where $\alpha_k = (T \bmod (H_k))$ is a zero of H_k . So elements of $W_k(\mathbb{F}_q)$ can be represented as polynomials in α_k over $\mathbb{Z}/p^k \mathbb{Z}$ of degree $< \delta H$, and $W_k(\mathbb{F}_q)$ can be mapped onto \mathbb{F}_q by reducing the coefficients of these polynomials modulo p . For $a \in W_k(\mathbb{F}_q)[X]$ we denote by $(a \bmod p) \in \mathbb{F}_q[X]$ the result of applying this mapping coefficient-wise to a . Remark that $W_1(\mathbb{F}_q) \cong \mathbb{F}_q$.

We now show how we map polynomials in $\frac{1}{D} \mathbb{Z}[a][X]$ to polynomials in $\mathbb{F}_q[X]$ and $W_k(\mathbb{F}_q)[X]$ respectively. Clearly, the canonical mapping from $\mathbb{Z}[T]/(F)$ to $(\mathbb{Z}/p^\ell \mathbb{Z})[T]/(H_\ell)$ defines a mapping from $\mathbb{Z}[a]$ to $W_\ell(\mathbb{F}_q)$, for $\ell = 1, k$. (Informally, this mapping works by reducing the polynomial in a modulo p^ℓ and $H_\ell(a)$.) For $a \in \mathbb{Z}[a]$ we denote by $(a \bmod (p^\ell, H_\ell)) \in W_\ell(\mathbb{F}_q)$ the result of this mapping. Finally, for $g = \sum_i \frac{a_i}{D} X^i \in \frac{1}{D} \mathbb{Z}[a][X]$ we denote by $(g \bmod (p^\ell, H_\ell))$ the polynomial $\sum_i ((D^{-1} \bmod p^\ell) a_i) \bmod (p^\ell, H_\ell) X^i \in W_\ell(\mathbb{F}_q)[X]$. Notice that $D^{-1} \bmod p^\ell$ exists due to (2.2).

We conclude this section with a result from [7: Section 1] that we will need here. Let $b_1, b_2, \dots, b_n \in (\frac{\mathbb{Z}}{D})^n$ be linearly independent; we restrict ourselves to the case that the $n \times n$ matrix having b_1, b_2, \dots, b_n as columns is upper-triangular. The i -dimensional lattice $L_i \subset (\frac{\mathbb{Z}}{D})^i$ with basis b_1, b_2, \dots, b_i is defined as $L_i = \sum_{j=1}^i \mathbb{Z} b_j = \{\sum_{j=1}^i r_j b_j : r_j \in \mathbb{Z}\}$; we put $L = L_n$.

(2.7) Proposition. (cf. [7: (1.11), (1.26), (1.37)]) Let $B \in \mathbb{Z}_{\geq 2}$ be such that $|Db_j|^2 \leq B$ for $1 \leq j \leq n$, where $||$ denotes the ordinary Euclidean length. There is an algorithm that determines a vector $\tilde{b} \in L$ such that \tilde{b} belongs to a basis for L , and such that $|\tilde{b}|^2 \leq 2^{n-1} |x|^2$ for every $x \in L$, $x \neq 0$; this algorithm takes $O(n^4 \log B)$ elementary operations on integers having binary length $O(n \log B)$. Furthermore, during the first $O(i^4 \log B)$ operations (on integers having binary length $O(i \log B)$), vectors \tilde{b}_i , belonging to a basis for L_i , are determined such that $|\tilde{b}_i|^2 \leq 2^{i-1} |x_i|^2$ for every $x_i \in L_i$, $x_i \neq 0$, for $1 \leq i \leq n$. \square

Informally, (2.7) states that we can find a reasonable approximation of the shortest vector in L in polynomial-time. Furthermore, during this computation, we find approximations of the shortest vectors of the lattices L_i without any time loss.

3. Factors and lattices.

This section is similar to the first part of [7: Section 2]. We formulate the generalizations of [7: (2.5), (2.6), (2.7), (2.13)] to polynomials over algebraic number fields. Let f, D, p, k, F , and H be as in Section 1. We put $n = \delta f$; we may assume that $n > 0$.

Suppose that we are given a polynomial $h \in \mathbb{Z}[\alpha][x]$ such that

$$(3.1) \quad h \text{ is monic,}$$

$$(3.2) \quad (h \bmod (p^k, H_k)) \text{ divides } (f \bmod (p^k, H_k)) \text{ in } W_k(\mathbb{F}_q)[x],$$

$$(3.3) \quad (h \bmod (p, H_1)) \text{ is irreducible in } \mathbb{F}_q[x],$$

$$(3.4) \quad (h \bmod (p, H_1))^2 \text{ does not divide } (f \bmod (p, H_1)) \text{ in } \mathbb{F}_q[x].$$

We put $\ell = \delta h$; so $0 < \ell \leq n$. In Section 4 we will see which extra conditions have to be imposed on p so that h can be determined.

Let $h_0 \in \frac{1}{D} \mathbb{Z}[\alpha][x]$ be the unique monic irreducible factor of f for which $(h \bmod (p, H_1))$ divides $(h_0 \bmod (p, H_1))$ in $\mathbb{F}_q[x]$ (or equivalently $(h \bmod (p^k, H_k))$ divides $(h_0 \bmod (p^k, H_k))$ in $W_k(\mathbb{F}_q)[x]$, cf. [7: (2.5)]).

(3.5) In the remainder of this section we fix an integer m with $\ell \leq m < n$. We define L as the collection of polynomials $g \in \frac{1}{D} \mathbb{Z}[\alpha][x]$ such that:

$$(i) \quad \delta g \leq m,$$

$$(ii) \quad \text{if } \delta g = m, \text{ then } \ell c(g) \in \mathbb{Z},$$

$$(iii) \quad (h \bmod (p^k, H_k)) \text{ divides } (g \bmod (p^k, H_k)) \text{ in } W_k(\mathbb{F}_q)[x].$$

We identify such a polynomial $g = \sum_{i=0}^{m-1} \sum_{j=0}^{\delta F-1} a_{ij} \alpha^j X^i + a_m X^m$ with the $(m \delta F + 1)$ -dimensional vector $(a_{00}, a_{01}, \dots, a_{0 \delta F-1}, \dots, a_{m-1 \delta F-1}, a_m)$. Using this identification it is not difficult to see that L is a lattice in $(\frac{Z}{D})^{m \delta F + 1}$. From the fact that H and h are monic ((2.3) and (3.1)) it follows that an upper-triangular basis for L is given by:

$$\begin{aligned} & \left\{ \frac{1}{D} p^k \alpha^j X^i : 0 \leq j < \delta H, 0 \leq i < \ell \right\} \cup \\ & \left\{ \frac{1}{D} \alpha^{j-\delta H} H(\alpha) X^i : \delta H \leq j < \delta F, 0 \leq i < \ell \right\} \cup \\ & \left\{ \frac{1}{D} \alpha^j h X^{i-\ell} : 0 \leq j < \delta F, \ell \leq i < m \right\} \cup \\ & \{ h X^{m-\ell} \}. \end{aligned}$$

We define the length $|g|$ of g as the ordinary Euclidean length of the vector identified with g ; the height q_{\max} of g is defined as $\max\{|a_{ij}|$. Similarly we define the length and the height of polynomials in $Z[T]$.

(3.6) Proposition. Let $b \in L$ satisfy

$$(3.7) \quad p^{k \ell \delta H / \delta F} > \left(D f_{\max} ((n+1) \delta F (1+F_{\max})^{\delta F-1})^{\frac{k}{2}} \right)^m \left(D b_{\max} ((m+1) \delta F (1+F_{\max})^{\delta F-1})^{\frac{k}{2}} \right)^n.$$

Then b is divisible by h_0 in $\mathbb{Q}(\alpha)[X]$ and in particular $\gcd(f, b) \neq 1$.

Proof. The proof is similar to the proof of [7: (2.7)]; we therefore omit the details.

Put $g = \gcd(f, b)$, and $e = \delta g$. Identify the polynomials

$$(3.8) \quad \{ \alpha^j X^i f : 0 \leq j < \delta F, 0 \leq i < \delta b - e \} \cup \{ \alpha^j X^i b : 0 \leq j < \delta F, 0 \leq i < n - e \}$$

with $(\delta F(n+\delta b-e))$ -dimensional vectors. The projections of these vectors on $\frac{1}{D} ZX^e + \frac{1}{D} Z \alpha X^e + \dots + \frac{1}{D} Z \alpha^{\delta F-1} X^e + \frac{1}{D} ZX^{e+1} + \dots + \frac{1}{D} Z \alpha^{\delta F-1} X^{n+\delta b-e-1}$ form a basis for a $(\delta F(n+\delta b-2e))$ -dimensional lattice M' . Using induction on j one proves that

$$(\alpha^j X^i f)_{\max} = (\alpha^j f)_{\max} \leq f_{\max} (1+F_{\max})^j,$$

so that, for $0 \leq j < \delta F$ and $0 \leq i < \delta b - e$,

$$|\alpha^j X^i f| \leq f_{\max} ((n+1) \delta F)^{\frac{j}{2}} (1+F_{\max})^j.$$

With Hadamard's inequality, and a similar bound on $|\alpha^j X^i b|$ we get

$$d(M') \leq \left((f_{\max} ((n+1) \delta F (1+F_{\max})^{\delta F-1})^{\frac{k}{2}})^m (b_{\max} ((m+1) \delta F (1+F_{\max})^{\delta F-1})^{\frac{k}{2}})^n \right)^{\delta F},$$

where $d(M')$ denotes the determinant of M' . With (3.7) this gives

$$(3.9) \quad d(M') < \frac{p^{k \ell \delta H}}{D^{(n+m) \delta F}}.$$

It is easy to prove that h_0 divides g in $\mathbb{Q}(\alpha)[X]$ if and only if $(h \bmod (p, H_1))$ divides $(g \bmod (p, H_1))$ in $\mathbb{F}_q[X]$ (cf. [7: (2.5)]). So assume that the latter is not

the case; we will derive a contradiction from this.

Let $v \in \frac{1}{D} \mathbb{Z}[\alpha][x]$ be some integral linear combination of the polynomials in (3.8) such that $\delta v < e + \ell$. It follows from our assumption that $(v \bmod (p^k, H_K)) = 0$ in $W_k(\mathbb{F}_q)[x]$ (cf. [7: (2.7)]). Therefore, if we regard $lc(v)$ as a polynomial in α , we have

$$(3.10) \quad lc(lc(v)) \equiv 0 \pmod{p^k} \quad \text{if } \delta lc(v) < \delta H.$$

Now choose a basis $b_{e0}, b_{e1}, \dots, b_{e\delta F-1}, b_{e+1,0}, \dots, b_{n+\delta b-e-1, \delta F-1}$ for M' such that $\delta b_{ij} = i$ and $lc(b_{ij}) = j$ for $e \leq i < n+\delta b-e$ and $0 \leq j < \delta F$, where $lc(b_{ij})$ is regarded as a polynomial in α . From (3.10) we derive that $lc(lc(b_{ij})) \equiv 0 \pmod{p^k}$ for $0 \leq j < \delta H$ and $e \leq i < e+\ell$. Since $lc(lc(b_{ij})) \in \frac{\mathbb{Z}}{D}$, we obtain $|lc(lc(b_{ij}))| \geq \frac{p^k}{D}$ for $0 \leq j < \delta H$ and $e \leq i < e+\ell$ and $|lc(lc(b_{ij}))| \geq \frac{1}{D}$ for $\delta H \leq j < \delta F$ or $e+\ell \leq i < n+\delta b-e$.

The determinant of M' equals the product of $|lc(lc(b_{ij}))|$, so that

$$d(M') \geq \frac{p^{k\ell\delta H}}{D^{(n+\delta b-2e)\delta F}} \geq \frac{p^{k\ell\delta H}}{D^{(n+m)\delta F}}.$$

Combined with (3.9) this is the desired contradiction. \square

(3.11) Proposition. (cf. [7: (2.13)]) Suppose that

$$(3.12) \quad p^{k\ell\delta H/\delta F} > \left(2^{n(m\delta F+1)} (n+1)^{n+m} (m+1)^n \left(\frac{2m}{m} \right)^n \delta F^{4n+m} (\delta F-1)^{n(\delta F-1)} (1+f_{\max}^{(n+m)(\delta F-1)} |\text{discr}(F)|^{-n})^{\frac{1}{2}} \cdot (Df_{\max}^{n+m} |F|^{2n(\delta F-1)}) \right),$$

where $\text{discr}(F)$ denotes the discriminant of F . Then we have $\delta h_0 \leq m$ if and only if (3.7) is satisfied with b replaced by \tilde{b} , where \tilde{b} results from applying (2.7) to L .

Proof. In [8] we show that the method sketched in [10] combined with results from [9] gives the following upper bound for the length of a monic factor of degree $\leq m$ of f in $\frac{1}{D} \mathbb{Z}[\alpha][x]$:

$$f_{\max} (2(n+1) \delta F^3 (\delta F-1)^{\delta F-1} \left(\frac{2m}{m} \right)^{\frac{1}{2}} |F|^{2(\delta F-1)} |\text{discr}(F)|^{-\frac{1}{2}}).$$

With (3.6) the proof follows immediately. \square

4. Description of the algorithm.

We describe how the results from the previous sections can be used to formulate a polynomial-time algorithm to factor $f \in \mathbb{Q}(\alpha)[x]$. First we present an algorithm that determines h_0 , given D, p, H and h . Let d be such that $f \in \frac{1}{d} \mathbb{Z}[\alpha][x]$.

(4.1) Suppose that a positive integer D , a prime number p , and polynomials $H \in \mathbb{Z}[T]$ and $h \in \mathbb{Z}[\alpha][X]$ are given such that (2.1), (2.2), (2.3), (2.5), (2.6), (3.1), (3.3) and (3.4), and (2.4) and (3.2) with k replaced by 1, are satisfied. We describe an algorithm that determines h_0 , the monic irreducible factor of f for which $(h \bmod (p, H_1))$ divides $(h_0 \bmod (p, H_1))$ in $\mathbb{F}_q[X]$.

Put $\ell = \delta h$; we may assume that $\ell < n$. We calculate the least positive integer k for which (3.12) holds with m replaced by $n-1$:

$$(4.2) \quad p^{k\ell \delta H/\delta F} > \left(2^{n((n-1)\delta F+1)} (n+1)^{2n-1} n^2 (n-1)^n \delta F^{5n-1} (\delta F-1)^{n(\delta F-1)} (1+F_{\max})^{(2n-1)(\delta F-1)} |\text{discr}(F)|^{-n} \right)^{\frac{1}{2}} \cdot (Df_{\max})^{2n-1} |F|^{2n(\delta F-1)}.$$

Next we modify H in such a way that (2.4) holds for the value of k just calculated. The factor $H_k = (H \bmod p^k)$ of $(F \bmod p^k)$ gives us the possibility to compute in $W_k(\mathbb{F}_q)$. Therefore we now modify h , without changing $(h \bmod (p, H_1))$, in such a way that (3.2) holds for the above value of k . The computations of the new H and h can both be done by means of Hensel's lemma [4: exercise 4.6.22; 11]; notice that Hensel's lemma can be applied because of (2.6) and (3.4).

Now apply Proposition (2.7) to the $(m \delta F + 1)$ -dimensional lattice L as defined in (3.5), for each of the values of $m = \ell, \ell+1, \dots, n-1$ in succession; but we stop as soon as for one of these values of m we find a vector \tilde{b} in L such that (3.7) is satisfied with b replaced by \tilde{b} . If such a vector is found for a certain value m_0 of m , then we know from (3.11) that $\delta h_0 \leq m_0$. Since we try the values $m = \ell, \ell+1, \dots, n-1$ in succession we also know that $\delta h_0 > m_0 - 1$, so $\delta h_0 = m_0$. By (3.6) h_0 divides \tilde{b} in $\mathbb{Q}(\alpha)[X]$ which implies, together with $\delta \tilde{b} \leq m_0$, that $\delta \tilde{b} = m_0$. From (3.5)(ii) and from the fact that h_0 is monic we find that $\tilde{b} = ch_0$, for some constant $c \in \mathbb{Z}$. Using that $h_0 \in L$ and that \tilde{b} belongs to a basis for L , we conclude that $c = \pm 1$, so that $\tilde{b} = \pm h_0$.

If on the other hand we did not find such a vector \tilde{b} in any of the lattices, then we know from (3.11) that $\delta h_0 > n-1$. This implies that $h_0 = f$. This finishes the description of Algorithm (4.1).

(4.3) Proposition. Denote by $m_0 = \delta h_0$ the degree of the irreducible factor h_0 of f that is found by Algorithm (4.1). Then the number of arithmetic operations needed by Algorithm (4.1) is $O(m_0(n^5 \delta F^6 + n^4 \delta F^6 \log(\delta F |F|) + n^4 \delta F^5 \log(Df_{\max}) + n^3 \delta F^4 \log p))$ and

the integers on which these operations are performed each have binary length

$$O(n^3 \delta F^3 + n^2 \delta F^3 \log(\delta F |F|) + n^2 \delta F^2 \log(Df_{\max}) + n \delta F \log p).$$

Proof. Let m_1 be the largest value of m for which Proposition (2.7) is applied; so $m_1 = m_0$ or $m_1 = m_0 - 1$. From (2.7) it follows that during the application of (2.7) to the $(m_1 \delta F + 1)$ -dimensional lattice, also approximations of shortest vectors were obtained for the $(m \delta F + 1)$ -dimensional lattices, for $\ell \leq m < m_1$. Therefore the number of arithmetic operations needed for the applications of (2.7) for $\ell \leq m \leq m_1$ is equal to the number of operations needed for $m = m_1$ only.

To analyze the latter we derive a bound B for the length of the vectors in the initial basis for L (cf. (3.5)). Assuming that the coordinates of the initial basis are reduced modulo p^k , we derive from (4.2), $|\text{discr}(F)| \geq 1$, $\delta H \geq 1$ and $\ell \geq 1$ that $\log B = O(n^2 \delta F^2 + n \delta F^2 \log(\delta F |F|) + n \delta F \log(Df_{\max}) + \log p)$. Combined with $m_1 = O(m_0)$ and (2.7) this yields the estimates given in (4.3).

It is straightforward to verify that the same estimates are valid for both applications of Hensel's lemma and for the computation of $\text{discr}(F)$ (cf. [2], [11]). \square

(4.4) We now describe how to choose D , p , H and h in such a way that Algorithm (4.1) can be applied. The algorithm to factor f into its monic irreducible factors in $\mathbb{Q}(\alpha)[x]$ then easily follows.

First we choose a positive integer D such that (2.1) holds, i.e. f and all monic factors of f in $\mathbb{Q}(\alpha)[x]$ are in $\frac{1}{D}\mathbb{Z}[\alpha][x]$. From [10] it follows that we can take $D = dc$, where d is such that $f \in \frac{1}{d}\mathbb{Z}[\alpha][x]$, and c is the largest integer such that c^2 divides $\text{discr}(F)$. This integer c however might be difficult to compute; therefore we take $D = d|\text{discr}(F)|$ as denominator, which clearly also suffices.

We may assume that the resultant $R(f, f') \in \mathbb{Q}(\alpha)$ of f and its derivative f' is unequal to zero, i.e. f has no multiple factors in $\mathbb{Q}(\alpha)[x]$. We determine p as the smallest prime number not dividing $D \cdot \text{discr}(F) \cdot R(f, f')$; so (2.2) is satisfied.

Using Berlekamp's algorithm [4: Section 4.6.2] we compute the irreducible factorization $(F \bmod p) = \prod_{i=1}^t (G_i \bmod p)$ of $(F \bmod p)$ in $(\mathbb{Z}/p\mathbb{Z})[T]$. This factorization does not contain multiple factors because $\text{discr}(F) \not\equiv 0 \pmod{p}$. Combined with $R(f, f') \not\equiv 0 \pmod{p}$ this implies that there exists an integer $i_0 \in \{1, 2, \dots, t\}$ such that $(R(f, f') \bmod (p, (G_{i_0} \bmod p))) \neq 0$; let H be such a polynomial G_{i_0} . We may assume

that H is monic, so that (2.3), (2.5), (2.6) and (2.4) with k replaced by 1 are satisfied.

Next we determine the irreducible factorization of $(f \bmod (p, H_1))$ in $\mathbb{F}_q[X]$ by means of Berlekamp's algorithm [1: Section 5], where $q = p^{\delta H}$ and $\mathbb{F}_q \cong (\mathbb{Z}/p\mathbb{Z})[T]/(H_1)$. (Notice that we use a modified version of Berlekamp's algorithm here, one that is polynomial-time in p and δH rather than polynomial-time in the number of elements of the finite field.)

Since f is monic the resultant $R(f, f')$ is, up to sign, equal to the discriminant of f , so that it follows from the construction of H that the discriminant of f is unequal to zero in \mathbb{F}_q . Therefore (3.4) holds for all irreducible factors $(h \bmod (p, H_1))$ of $(f \bmod (p, H_1))$ in $\mathbb{F}_q[X]$; we may assume that these factors are monic.

The algorithm to factor f now follows by repeated application of Algorithm (4.1).

(4.5) Theorem. The algorithm sketched above computes the irreducible factorization of any monic polynomial $f \in \frac{1}{d} \mathbb{Z}[\alpha][X]$ of degree $n > 0$. The number of arithmetic operations needed by the algorithm is $O(n^6 \delta F^6 + n^5 \delta F^6 \log(\delta F|F|) + n^5 \delta F^5 \log(df_{\max}))$, and the integers on which these operations are performed each have binary length $O(n^3 \delta F^3 + n^2 \delta F^3 \log(\delta F|F|) + n^2 \delta F^2 \log(df_{\max}))$.

Proof. It follows from [2] that the calculations of $R(f, f')$ and $\text{discr}(F)$ satisfy the above estimates. From Hadamard's inequality we obtain $|\text{discr}(F)| \leq \delta F^{\delta F} |F|^{2\delta F - 1}$; it follows that

$$\log D = O(\log d + \delta F \log(\delta F|F|)).$$

In order to give an upper bound for the height of $R(f, f')$, we use the result from [3]. Let A be a matrix having entries $A_{ij} = \sum_{\ell=0}^{\delta F-1} a_{ij\ell} T^\ell \in \mathbb{Z}[T]$, for $1 \leq i, j \leq m$, and some positive integer m . The determinant $d(A)$ of A is a polynomial of degree $\leq m(\delta F-1)$ in $\mathbb{Z}[T]$. According to [3] the length, and therefore the height, of $d(A)$ is bounded from above by

$$(\prod_{j=1}^m \sum_{i=1}^m (\sum_{\ell=0}^{\delta F-1} |a_{ij\ell}|)^2)^{\frac{1}{2}}.$$

Using this bound it is easily proved that the height of $d(A) \bmod F$ is bounded by

$$(\prod_{j=1}^m \sum_{i=1}^m (\sum_{\ell=0}^{\delta F-1} |a_{ij\ell}|)^2)^{\frac{1}{2}} (1+F_{\max})^{(m-1)(\delta F-1)}.$$

It follows that

$$(R(f, f'))_{\max} \leq (\sqrt{n+1} \delta F f_{\max})^{n-1} (\sqrt{n} \delta F n f_{\max})^n (1+F_{\max})^{(2n-2)(\delta F-1)},$$

where $R(f, f')$ is regarded as a polynomial in α . We find from the definition of D and p that

$$\prod_{q \text{ prime}, q < p} q \leq d |discr(F)| (R(df, df'))_{\max}$$

and this yields in a similar way as in [7] that

$$p = O(\log d + n \delta F \log(\delta F |F|) + n \log n + n \log(df_{\max})).$$

This implies that the computation of the prime number p , and the computation of the factorizations of $(F \bmod p)$ in $(\mathbb{Z}/p\mathbb{Z})[T]$ and $(f \bmod (p, H_1))$ in $\mathbb{F}_q[X]$ satisfy the estimates in (4.5). Theorem (4.5) now easily follows from the bounds on $\log D$ and p , and from the observation that a monic factor g of f in $\mathbb{Q}(\alpha)[X]$ satisfies $\log(g_{\max}) = O(\delta F \log(\delta F |F|) + n + \log(f_{\max}))$ (this follows from a bound similar to the one given in the proof of (3.11)). \square

References.

1. E.R. Berlekamp, Factoring polynomials over large finite fields, *Math. Comp.* 24 (1970), 713-735.
2. W.S. Brown, The subresultant PRS algorithm, *ACM Transactions on mathematical software* 4 (1978), 237-249.
3. A.J. Goldstein, R.L. Graham, A Hadamard-type bound on the coefficients of a determinant of polynomials, *SIAM Rev.* 16 (1974), 394-395.
4. D.E. Knuth, *The art of computer programming*, vol. 2, *Seminumerical algorithms*, Addison Wesley, Reading, second edition 1981.
5. S. Landau, Factoring polynomials over algebraic number fields is in polynomial-time, unpublished manuscript, 1982.
6. A.K. Lenstra, Lattices and factorization of polynomials over algebraic number fields, *Proceedings Eurocam 82, LNCS 144*, 32-39.
7. A.K. Lenstra, H.W. Lenstra, Jr., L. Lovász, Factoring polynomials with rational coefficients, *Math. Ann.* 261 (1982), 515-534.
8. A.K. Lenstra, Factoring polynomials over algebraic number fields, Report IW 213/82, Mathematisch Centrum, Amsterdam 1982.
9. M. Mignotte, An inequality about factors of polynomials, *Math. Comp.* 28 (1974), 1153-1157.
10. P.J. Weinberger, L.P. Rothschild, Factoring polynomials over algebraic number fields, *ACM Transactions on mathematical software* 2 (1976), 335-350.
11. D.Y.Y. Yun, The Hensel lemma in algebraic manipulation, MIT, Cambridge 1974; reprint: Garland Publ. Co., New York 1980.

THE CONSTRUCTION OF A COMPLETE MINIMAL SET OF CONTEXTUAL NORMAL FORMS

Monique Rice
Centre de Recherche en Informatique de Nancy
Campus scientifique. BP 239.
54506 Vandoeuvre-les-Nancy Cedex FRANCE

ABSTRACT

This paper introduces the concept of Complete Minimal Set of Contextual Normal Forms (C.M.S.C.N.F.).

The Church-Rosser theorem on confluent rewriting systems is extended to a hierarchical conditional confluent rewriting system.

An algorithm which computes C.M.S.C.N.F.s is presented and proved in this article, and several examples are developed.

INTRODUCTION

The purpose of this paper is to introduce the concept of Complete Minimal Set of Contextual Normal Forms (C.M.S.C.N.F.)
A context is a set of preconditions necessary for further developments.

In the first section, we recall the definitions and properties of the hierarchical conditional specifications (axioms and preconditions); we always work with the primitive or with the conditional specification.

In our approach, we consider the predicates and the operations separately. The terms are constructed from operations and variables.

The simple boolean expressions, the predicates and the contexts are built from a set of predicates, a set of elementary operations and a set of variables.

The logical connectors permit us to assemble booleans expressions.

The conditional rewriting is a tool which allows us to prove the Church-Rosser property (two elements are equal under an equational theory iff they can be rewritten into a third term) and thus the contextual equivalence of two terms.

In the 3rd section, we define a contextual normal form. In fact, we actually get a set of contextual normal forms.

In the 4th section, we find an algorithm to build this set.

So, the main idea in this paper is to construct this minimal set of contextual normal forms. We prove the contextual confluence by adapting the Knuth and Bendix completion algorithm, and we establish the contextual equivalence of two terms by using the Church-Rosser property which follows from the confluence of the Hierarchical Conditional Rewriting System (H.C.R.S.).

1. DEFINITIONS

In this section, we give some definitions which help us to understand what is the contextual rewriting.

The only elements that we use are :

- sorts
- operations defined on the sorts
- variables
- terms constructed with the sorts and the operations.

We also need logical connectors (AND , OR , NOT , \Rightarrow , \Leftarrow , TRUE, FALSE) because the contexts always are boolean expressions.

Def 1.1 (Signature with predicates)

A signature with predicates is the triple (S, F, P) with :

- S : a set of sort names
- F : a set of operation names with the following profile
 $s \leftarrow s_1, s_2, \dots, s_n$ with s in S and i in I
 I is the set of indexes
- P : a set of predicate names with the profile
 $\text{bool } \leftarrow s_1, s_2, s_3, \dots, s_n$

Notation 1.1

X is the set of variables.

$T(F, X)$ is the set of F -terms constructed with the variables of X and with the functional symbols from F .

$T(F, X)$ satisfies the fixed point equation which is

$T(F, X) = \{t / (t=x; x \in X) \text{ or } (t = f(t_1, t_2, \dots, t_n); f \in F \text{ and } t_i \in T(F, X)\}$

$BE(F, P, X)$ is the set of boolean expressions constructed from F -terms, predicates and logical connectors.

Example 1.1

Definition of the Infimum of two elements in a totally ordered set

```

Signature ORD
Sort : elem
Operations :
    Inf : elem  $\leftarrow$  elem , elem
Predicate :
    LE : bool  $\leftarrow$  elem , elem
```

Example 1.2

Definition of operations which allow us to build a set

```

Signature TE
Sorts : elem, ens , bool
Operations :
    Evide : ens  $\leftarrow$ 
    Ajt : ens  $\leftarrow$  ens, elem
    Ret : ens  $\leftarrow$  ens, elem
    App : bool  $\leftarrow$  ens, elem
Predicate :
    Eg : bool  $\leftarrow$  elem, elem
```

Def 1.2 (F-P algebra)

An F-P algebra A is defined as :

- a family (A_s) of non empty sets with s in S
- a family (f_A) of operations respecting the profiles and with f in F
- a family (P_A) of predicates on A respecting the profiles and with p in P .

Example 1.3

Natural numbers with classical order and classical Inf operation form an F-P algebra for the signature given in example 1.1

Def 1.3 (Primitive specification)

A primitive specification (E_0, A_0) based on a signature (S_0, F_0, P) is defined as a pair (E_0, A_0) where

- . E_0 is the set of equations from $T(F_0, X)$
- $E_0 = \{ G_0 = D_0 \}$ with G_0 and D_0 in $T(F_0, X)$
- . A_0 is the set of axioms of $BE(F_0, P, X)$; each axiom being a boolean expression.

We write B_0 the union A_0 , E_0 and $=_{B_0}$ the equivalence on boolean expressions derived from the axioms.

We can enlarge the primitive specification by adding sorts (we get a new set S including S_0), operations $(F_0 \subseteq F)$ and conditional equations $P \Rightarrow G=D$ where P is called a premise and is a primitive boolean expression (P in $BE(F_0, P, X)$)

Note :

F may contain symbols generating boolean results.

Def 1.4 (Hierarchical Conditional Specification (H.C.S.))

A H.C.S. (E_0, A_0, E_1) on a signature with predicates (S, F, P) is defined as :

- a primitive specification (E_0, A_0) on a sub-signature (S_0, F_0, P)
 E_0, A_0 being defined as above
- a set E_1 of conditional equations; $E_1 = \{ P \Rightarrow G=D \}$

Note :

This specification is hierarchical because, later, we always work on two levels of specification :

- a primitive specification
- a conditional specification.

Example 1.4 (The infimum specification)

- Primitive equation : nothing
- Axioms :
 - . $LE(x, x)$
 - . $LE(x, y)$ and $LE(y, z) \text{ IMP } LE(x, z)$
 - . $\text{not } (LE(x, y)) \text{ IMP } LE(y, x)$
- Conditional equations :
 - . $LE(x, y) \Rightarrow Inf(x, y)=x$
 - . $\text{not}(LE(x, y)) \Rightarrow Inf(x, y)=y$

Example 1.5 (Sets specification)

- Primitive equation : nothing
- Axioms :
 - . $Eg(x, x)$
 - . $Eg(x, y) \text{ IMP } Eg(y, x)$
 - . $(Eg(x, y) \text{ AND } Eg(y, z)) \text{ IMP } Eg(x, z)$
- Conditional equations :
 - . $\text{TRUE} \Rightarrow App(Evide, x) = \text{FALSE}$ (E1)
 - . $Eg(x, y) \Rightarrow App(Ajt(XX, x), y) = \text{TRUE}$ (E2)
 - . $\text{Not}(Eg(x, y)) \Rightarrow App(Ajt(XX, x), y) = App(XX, y)$ (E3)
 - . $\text{TRUE} \Rightarrow Ret(Evide, x) = Evide$ (E4)
 - . $Eg(x, y) \Rightarrow Ret(Ajt(XX, x), y) = Ret(XX, y)$ (E5)
 - . $\text{Not}(Eg(x, y)) \Rightarrow Ret(Ajt(XX, x), y) = Ajt(Ret(XX, y), x)$ (E6)
 - . $Eg(x, y) \Rightarrow App(Ret(XX, y), x) = \text{FALSE}$ (E7)
 - . $\text{Not}(Eg(x, y)) \Rightarrow App(Ret(XX, y), x) = App(XX, x)$ (E8)

Def 1.5 (Context)

A context C is either a premise or a conjunction of premises.

We pose the problem of satisfying the equivalence of two terms in a given context. We only work with boolean expressions we can evaluate in B0. We thus define two equivalence relations which are more and more rich. Note we consider ternary relations for taking into account premises.

Def 1.6 (Contextual congruence)

A contextual relation, denoted $\simeq()$, is a contextual congruence if it satisfies the following properties :

- . Reflexivity $M \simeq M (C)$
- . Symmetry $M \simeq M' (C) \text{ then } M' \simeq M (C)$
- . Transitivity $M \simeq M' (C) \text{ and } M' \simeq M'' (C') \text{ then } M \simeq M'' (C \text{ AND } C')$
- . Compatibility $M \simeq M' (C) \text{ then } f(\dots M \dots) \simeq f(\dots M' \dots) (C)$

We say that a contextual congruence is invariant if for every substitution s such that sC is a context, and if $M \simeq M' (C)$ then $sM \simeq sM' (sC)$.

A substitution is legal if it substitutes variables with primitive terms.

Proposition [REMY 82]

Given E , a set of conditional equations, there is a smallest contextual invariant congruence, denoted \sqsubseteq_E containing E .

But the contextual congruence is not sufficient to prove any conditional equation.

We say that two terms are equivalent in a context C if there is a covering of C by several contexts so that these two terms are congruent in each context.

Def 1.7 (BO-covering)

Given C a primitive context, a family (C_i) of contexts is a BO -covering of C if $C = \bigvee_{i=0}^n C_i$.

Example 1.6

Given B_0 from the example 1.4, the family (C_1, C_2) is a BO -covering of the context $C = LE(a,b)$ where

$$\begin{aligned} C_1 &: LE(a,b) \text{ AND } LE(b,c) \\ C_2 &: LE(a,b) \text{ AND NOT } LE(b,c) \\ (C_1 \text{ OR } C_2) &= \sqsubseteq_{B_0} (LE(a,b) \text{ AND } LE(b,c)) \text{ OR } (LE(a,b) \text{ AND NOT } LE(b,c)) \\ &= \sqsubseteq_{B_0} (LE(a,b)) \text{ AND } (LE(b,c) \text{ OR NOT } LE(b,c)) \\ &= \sqsubseteq_{B_0} (LE(a,b)) \text{ AND TRUE} \\ &= \sqsubseteq_{B_0} LE(a,b) \end{aligned}$$

Def 1.8 (Contextual equivalence)

Given (E_0, A_0, E_1) a hierarchical conditional specification.
Take E union of E_0 and E_1 .

Two terms M and N are E, BO equivalent in a context C (written $M \sqsubseteq_E, BO N$) if there is a BO -covering (C_i) of C so that for each i , we have $M \sqsubseteq_{C_i} N(C_i)$.

Example 1.7

Given E and B_0 the same as in the example 1.5,

$$\begin{aligned} App(Ret(Ajt(XX,a),b),c) &=_{E, B_0} App(Ret(XX,b),c) \text{ (C)} \quad (\text{using E5}) \\ &=_{E, B_0} App(XX,c) \text{ (C)} \quad (\text{using E8}) \\ \text{where } C &= (Eg(a,b) \text{ AND NOT}(Eg(b,c))) \end{aligned}$$

Conclusion :

The contextual equivalence permits us a generalization of the notions of rewriting system, and the confluence and Church Rosser properties.

2. HIERARCHICAL CONTEXTUAL REWRITING SYSTEM (H.C.R.S.)

At first, we define a H.C.R.S. which is only a hierarchical conditional specification in which the equations have been oriented in order to form the rewriting rules $P \Rightarrow G \rightarrow D$, which respect the condition $V(D) \subseteq V(G)$.

Notation : We note $V(G)$ the set of variables of G

Note :

G must contain at least one symbol from $F-F_0$ in order to not modify the primitive specification.

Then, we develop the contextual confluence notions and we establish the equivalence between the Church-Rosser and the contextual confluence property.

Def 2.1 (H.C.R.S.)

A H.C.R.S. (R_0, A_0, R_1) on a signature (S, F, P) is :

- a set R_0 of rewriting rules without preconditions from $T(F_0, X)$
- a set A_0 of axioms from $BE(F_0, P, X)$ so that (E_0, A_0) is a primitive specification. E_0 being the set of rules from R_0 which have been transformed into equations.
- a set R_1 of rules $P \Rightarrow G \rightarrow D$ with $V(D) \subseteq V(G)$.

Note:

We accept supplementary variables in P , which are intuitively, existentially quantified.

Notation :

$$\begin{aligned} B_0 &= E_0 \cup A_0 \\ R &= R_0 \cup R_1 \\ (R, B_0) &\text{ instead of } (R_0, A_0, R_1). \end{aligned}$$

Def 2.2 (Contextual rewriting)

Given (R, B_0) a H.C.R.S., two terms M and N and a (primitive) context C , we say that M is rewritten in N in the (primitive) context C (this is written $M \rightarrow_R^N (C)$) if :

- there is a rule $P \Rightarrow G \rightarrow D$ in R
- there is a substitution s and an occurrence u of $O(M)$ (the set of all the occurrences of M) and so that :
 - $M/u = sG$, $N = M[u \leftarrow sD]$,
 - C is a conjunction of contexts including sP and
 - s must be a legal substitution for the variables of P (by construction to obtain a (primitive) context) and arbitrary terms to the other variables of D and G .

Note :

We write \rightarrow_R^* () the reflexive and transitive closure of \rightarrow_R ().

Def. 2.3. Rewriting in an extended sense.

Given (R, B_0) a H.C.R.S., M is rewritten into M' in an extended sense, ($written M \xrightarrow{R, B_0} M'(C)$) if there is a B_0 -covering (C_i) of C so that for each i $M \xrightarrow{R} M'(C_i)$.

Def 2.4 (contextual convergence)

Given (R, B_0) a H.C.R.S., C a (primitive) context, two terms M_1 and M_2 are R, B_0 convergent in a context C (this is written $M_1 \downarrow_{R, B_0} M_2 (C)$) if there is a B_0 -covering (C_i) of C and if for each i , there is a term M_i so that $M_1 \xrightarrow{R} M_i (C_i)$ and $M_2 \xrightarrow{R} M_i (C_i)$. (See Fig 2.1)

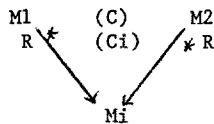
Def 2.5 (Contextual Church-Rosser property)

Given (R, B_0) a H.C.R.S., R and B_0 being defined as above, (R, B_0) has the contextual Church-Rosser property if the contextual relations $=_{E, B_0} (C)$ and $\downarrow_{R, B_0} (C)$ are equal. (See Fig. 2.2.)

Def 2.6 (Contextual confluence property)

Given (R, B_0) a H.C.R.S., (R, B_0) has the contextual confluence property if for each context C , each triple M, M_1, M_2 of terms so that $M \xrightarrow{R} M_1 (C)$ and $M \xrightarrow{R} M_2 (C)$ then $M_1 \downarrow_{R, B_0} M_2 (C)$ (See Fig. 2.3.)

We see that the contextual confluence is a special case of the Church-Rosser property.



with $\bigvee_i C_i = C$

Fig. 2.1.

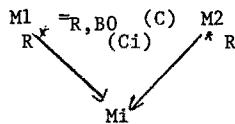


Fig. 2.2.

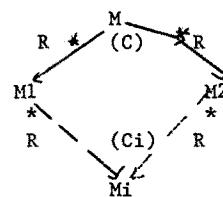


Fig. 2.3.

Theorem [REMY 82]

Given (R, B_0) a H.C.R.S., the following properties are equivalent :

- (R, B_0) has the contextual confluence property
- (R, B_0) has the Church-Rosser property.

As in the case of rewriting systems, confluence is an essential property. To make a rewriting system confluent, we use the Knuth and Bendix completion algorithm.

To achieve this, we examine the finite set of pairs (M_1, M_2) which are called critical pairs [Knuth and Bendix]

We test if the two critical pairs have the same normal form; if the normal forms are not equal, we add the pair of normal forms to the set of rewriting rules satisfying the finite termination property. Every time we have to choose the orientation. In the next section, we extend these notions to a H.C.R.S.

3. CONTEXTUAL NORMAL FORMS (C.N.F.)

Let us recall that a contextual term is a pair formed by a term and a (primitive) context.

Def 3.1 (Contextual normal forms)

Given (M, C) and (M', C') two contextual terms, (M', C') is the contextual normal form of (M, C) if :

- . M' cannot be rewritten, i.e. there is no rule $P \Rightarrow G \Rightarrow D$ in R which satisfies the rewriting conditions.
- . there is a family $(C'i)$ so that $C' = \bigvee_i C'i$ and for each i , $M \rightarrow_R M'(C'i)$ and $C'i = (C \text{ AND } C''i)$ for some $C''i$.

Note :

Given a contextual term (t, C) , there is a set of contextual terms (ti, Ci) which are normal forms of the contextual term (t, C) .

Def 3.2 (Complete Minimal Set of Contextual Normal Forms (C.M.S.C.N.F.))

A Set of Contextual Normal Forms (S.C.N.F.) of a term M and a (primitive) context C is a family (Mi, Ci) of the normal forms of (M, C) .

This set is complete (C.S.C.N.F.) if (Ci) is a BO-covering of C .

This set is minimal (C.M.S.C.N.F.) if for each i and j , $i \neq j$, we have $Mi \neq Mj$ and (Ci) is not a BO-covering of FALSE.

Note :

A priori, a contextual term can have several C.M.S.C.N.F.

If the system has the contextual confluence property, all the C.M.S.C.N.F. are equivalent.

Def 3.3 (Equivalent C.M.S.C.N.F.'s)

Given two C.M.S.C.N.F. (Mi, Ci) and $(M'j, C'j)$ of a contextual term (M, C) , they are BO-equivalent if there is a bijection on the set of indexes so that for each index, we have the equality of terms and the equivalence modulo BO of contexts :

$$\begin{aligned} f : I &\rightarrow J \\ Mi &= M'f(i) \text{ and } Ci =_{BO} C'f(i) \end{aligned}$$

To prove that the two contexts are BO-equivalent, we can use for instance the Hsiang resolution method. [Hsiang 82]

In the next section, we present a method to construct a C.M.S.C.N.F. of a contextual term (t, C) .

4 . CONSTRUCTION OF A COMPLETE MINIMAL SET OF C.N.F.

In this section, we propose an algorithm which builds a C.M.S.C.N.F. The H.C.R.S must satisfy some additional properties to prove this algorithm :

- Finite termination insures us of the existence of a contextual normal form.
- For each set of rules $P_i \Rightarrow G \rightarrow D_i$ in R, with same left-hand side G, we must have the disjunction of the premises P_i B0-equivalent to TRUE. (c.f. def. 4.2.)
- Uniformity of the set of variables for each premise. (c.f. def. 4.3.)

These two last conditions are not absolutely necessary to insure the existence of a C.M.S.C.N.F., but they are very usefull to prove the algorithm.

Def. 4.1. (Strong Finite Termination)

Given (R, B_0) a H.C.R.S., R has the property of strong finite termination if the rewriting system in which we have forgotten the premises has finite termination.

Notations :

Given a set of rules $P_i \Rightarrow G \rightarrow D_i$, with G fixed, we note :

PREM(G)	the set of premises of G.
REF(G)	the set of pairs (P_i, D_i) .
VP(G)	the set of variables of premises of G, $(VP(G) = \cup_i V(P_i))$.
VSS(V)	the set of valid substitutions which replace each variable of V with a term from $T(F_0, X)$ and which replace the other variables with a term from $T(F, X)$.

We introduce two new properties necessary to prove the algorithm.

Def 4.2. (Well-Covering)

A H.C.R.S. (R, B_0) is well-covering if for each term G of a rule, the disjunction of its premises is B0-equivalent to TRUE.
 $(TRUE=B_0 \vee_i P_i / P_i \in PREM(G))$.

Def 4.3. (Uniformity for the premises)

A H.C.R.S. (R, B_0) is uniform for the premises if for each left-hand side G of a rule in R, all the premises of PREM(G) have the same set of variables (which is indeed VP(G)).

Exemple 4.1.

The rewriting system associated with the Inf specification is strongly finite terminating, well-covering and uniform for the premises.

We design and prove a functional algorithm to construct a C.M.S.C.N.F.

Presentation of the algorithmNotation :

$LHS(R)$: the set of left-hand sides' G of the rules $P_i \Rightarrow G \rightarrow D_i$

Functional algorithm

```
C.M.S.C.N.F.(M:term,C:context) =
```

```
if C = FALSE
then ()
else
  if (M,C) is irreducible
  then return (M,C)
  else
    given G ∈ LHS(R) and an u ∈ O(M) and s ∈ VSS(VP(G))
    such that M/u = sG
    for each pair (Pi,Di) ∈ DEF(G)
      do
        Mi = M[u ← sDi]
        Ci = sPi AND C
      od
    rof
  return ⋆iC.M.S.C.N.F.(Mi,Ci)
fi
fi
```

where \star is defined as : given two C.M.S.C.N.F. (M_i, C_i) and (M'_j, C'_j) ,
 $(M_i, C_i) \star (M'_j, C'_j) = (M''k, C''k)$ with
 $(M''k, C''k) = (M_i, C_i)$ if $M'_j \neq M_i \vee j$
 $= (M'_j, C'_j)$ if $M'_j \neq M_i \vee i$
 $= (M_i, C_i \text{ OR } C'_j)$ if $\{ i, j / M_i = M'_j$

Proof-sketch :

1. Clearly, the algorithm terminates, because of the strong finite termination hypothesis.
2. By noetherian induction, C.M.S.C.N.F.(M,C) is a set of C.N.F.
 This is clearly true if (M,C) is irreducible and if not, by recurrence C.M.S.C.N.F.(M_i,C_i) is a S.C.N.F. for each (M_i, C_i) and thus for (M,C) .
3. This set is complete :
 By noetherian induction, if (M,C) is irreducible, it's trivial.
 If not, let $(M_i, C_i)_i$ be a family as in the algorithm.
 By induction, C.M.S.C.N.F.(M_i,C_i) is complete for each i , that is

$$\bigvee_j \bigwedge_{i=0}^n C_{ij} = \bigwedge_{i=0}^n C_i$$

$$\begin{aligned} \text{if C.M.S.C.N.F.}(M_i, C_i) &= (M_{ij}, C_{ij})_j \\ \text{therefore C.M.S.C.N.F.}(M, C) &= \star(M_{ij}, C_{ij})_{ij} \text{ and} \\ ij \quad C_{ij} &= \bigwedge_{i=0}^n C_i = \bigwedge_{i=0}^n C \text{ AND } (\bigwedge_i sP_i) \\ &= \bigwedge_{i=0}^n C \text{ AND TRUE (because of the well-covering of R)} \\ &= \bigwedge_{i=0}^n C \end{aligned}$$

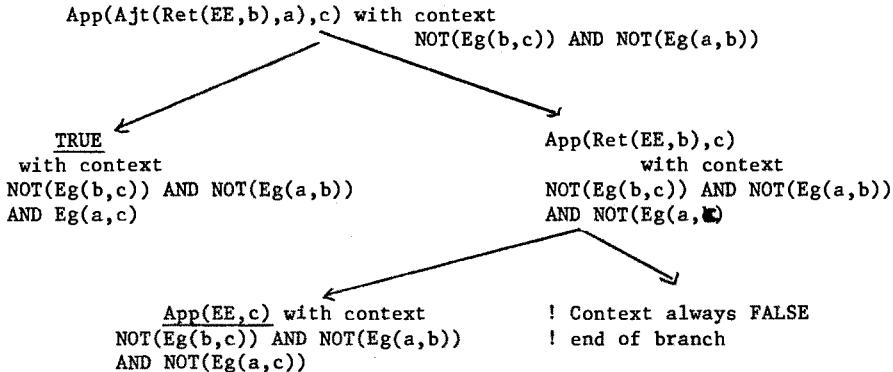
4. This set is minimal by construction of the \star .

A version of this algorithm has been programmed and tested in LISP. [RICE 82]

Exemple 4.2.

(signature c.f. ex. 1.2., rewriting rules c.f. ex. 1.5. where the equations have been oriented).

C.M.S.C.N.F.(App(Ajt(Ret(EE,b),a),c), NOT(Eg(b,c)) AND NOT(Eg(a,b)))



The normal forms have been underlined and the contexts computed along the branches.

5. CONCLUSION

When studying the H.C.R.S., the concept of C.N.F. is very important. It allows us to extend the Knuth and Bendix completion algorithm to the H.C.R.S

We proceed as follows :

- for each term we compute the CMSFNCs and we test if they are R,B0-equivalent.
If this is not the case, we must complete the H.C.R.S. until we get the R,B0-equivalence.
- now we can prove the confluence and thus the contextual equivalence of two terms in a context (Church-Rosser property).

We give some indications about what we want to do for extending our algorithm.

First we must implement the Hsiang boolean computing method (or some other method) to reduce the context into irreducible contexts.

So we can be sure that the CMSFNC is really minimal.

This also allows us to test the R,B0-equivalence of two CMSFNC of a contextual term.

BIBLIOGRAPHY

- HSIANG J.: Topics in automated theorem proving and program generation. (Ph. D. Thesis U. of Urbana, Illinois 82)
- HUET G.: Confluent reductions : abstract properties and applications for term rewriting systems.
(J. Assoc. Comp. Mach., 1980)
- HUET G.: A complete proof of correctness of the Knuth-Bendix completion algorithm.
(J.C.S.S. 23,1,27,4,pp. 797-821 (1980))
- HUET G. HULLOT J.M.: Proofs by induction in equational theories with constructors.
(Proc. 21th Symposium on Foundation of Computer Science 1980)
- KNUTH D. BENDIX P. : Simple word problems in universal algebras in computational problems in abstract algebras.
(Leech J. ed. Pergamon Press, pp. 263-297 (1970))
- LESCANNE P.: Computer experiments with REVE term rewriting system generators.
(proc. of 10th P.O.P.L. 1983)
- NELSON Nils: Artificial intelligence method (ch. 6 p. 156-183)
- REMY J.L.: Etude des systèmes de réécriture conditionnelle et application aux spécifications algébriques de types abstraits.
(Thèse d'Etat de l'I.N.P.L. Nancy 1982)
- REMY J.L.: Proving conditional identities by equation case reasoning rewriting and normalization.
(82 R085 CRIN82)
- RICE M.: Implantation d'un algorithme de calcul des ensembles complets minimaux de formes normales contextuelles pour un système de réécriture conditionnel.
(Rapport 82-R-075, C.R.I.N. 1982)

A KNOWLEDGE-BASED APPROACH TO USER-FRIENDLINESS IN SYMBOLIC COMPUTING

F. Gardin and J.A. Campbell
Department of Computer Science
University of Exeter
Exeter EX4 4QL, England

Abstract

An experiment in making a symbolic computing system tolerant of technical deficiencies in programming by novice users is outlined. The work uses a knowledge-base of information on general properties of symbolic computations, and a pattern-matcher which can specify instances of relevant properties in a user's program or inputs. Both of these components are implemented in LISP, and have been used in conjunction with REDUCE.

1. Introduction

It has been recognised for a long time that symbolic computing (SC) systems are not friendly to inexperienced users. This is probably an explanation for the observed facts that the use of these systems has spread rather slowly, and that the best way to recruit new users is for a human expert to demonstrate to them how a system can tackle their own problems. As it is not yet common for human experts to be delivered along with new copies of SC systems, the novice typically faces a system armed only with a users' manual, which explains the available commands, gives simple examples, and occasionally warns against particularly disastrous types of error. Unfortunately, the combined resources of a system and a manual do not give the user much relevant advice about computations which he or she expresses in ways which are technically error-free but which lead to unsatisfactory results like the message "no free storage left".

Experts can bypass novices' difficulties by making use of their special knowledge about SC, which is mostly knowledge whose applicability is not confined to any one SC system. We can distinguish static applications of the knowledge (to users' programs and inputs) and dynamic applications (to outputs and behaviour of programs); in practice, most static cases require significantly smaller amounts of computing resources and are easier to describe. Dynamic cases are of four types: in increasing order of difficulty, those dealing with outputs, run-time characteristics of a user's programs, patterns of calls of functions in the SC systems for which the programs are written, and other features of the systems. In the present work most of the examples considered are static, although we give instances of the first two dynamic cases. An example which illustrates that the third type of dynamic analysis is also practicable is presented elsewhere [3].

For SC, user-friendliness means minimally that the user should be protected from unhelpfully brief error messages and the worst effects of assuming that acceptable practices in numerical programming are also acceptable in SC. To do better than the minimum, one should provide him with on-the-job education about both phenomena, so that he can understand the special features of SC better as his amount of practice increases. For the present, our programs can handle only the output of a fixed explanatory message

for each change which they make or suggest in a user's program, but techniques from natural-language processing or a "cheap" imitation of it (e.g. [8]) may make more flexible outputs possible in future work.

2. Structure of the experimental program

The code which we are using is written in LISP and intended for interfacing with a LISP-based SC system. Our experiments have been carried out on REDUCE [4].

The central hypothesis is that all knowledge relevant to particular behaviours of an SC program for a given system can be expressed as "if a certain pattern occurs, take appropriate action". Therefore the primary element of our construction is a table of pattern-action (P-A) pairs. However, because of the aim of informing the inexperienced user of the reason for any action that is likely to be of educational value to him, provision is also made for a third "explanation" (E) component in any entry, which is used as described in the Introduction.

The form of specification of patterns must be general enough to allow description of all the types of both static and dynamic cases mentioned in the Introduction. In our previous work [3] on SC diagnostics, we have developed a LISP-based pattern-manipulating scheme which is now [2] named DMNPAT and which has the necessary expressive power. Pattern elements in the table of P-A-E triples are written in the DYNPAT syntax, for which examples are given below and in [3]. Use of a given P element in some context involving a user's program U causes actions A which modify U or (in isolated dynamic cases, involving tests on flags, which we have examined to date) in some of the functions of REDUCE itself. In the latter type of case we have run these functions of REDUCE interpretively, having modified them so as to convey information to the LISP code in the DYNPAT package that is concerned with identifying patterns, e.g. patterns of REDUCE function-calls. Analysis of such patterns can serve useful purposes even if we claim no special knowledge of the actual internal functions of an SC system: for a good example, see [3].

Each P-A-E triple is additionally labelled by its membership in one of more sets of information, e.g. STATIC (for triples whose effect can be explored before run-time), DYNAMIC (for triples which imply run-time analysis), FLAG (relevant to use of REDUCE flags), NOVICE (relevant to a program-transformation to correct faults of beginners' SC), SUBST (invoking user-requested substitutions). The labelling is intended to allow specific sets of rules to be applied by choice rather than by sequential search of the table, and to change the order of application according to circumstances.

Apart from this table, there is a separate two-way tabulation that relates symptoms or characteristics of the input program to the sets of P-A-E- triples which may refer to them. Thus a suitable symptom, e.g. "no output before free storage becomes exhausted", can generate an ordering in which the candidates for being the most relevant triples are scanned first.

The resemblance of the organisation of storage, particularly the latter tabulation, to the organisation of an expert or knowledge-based system is not coincidental. Most of the connections expressed in compact form in horizontal entries in the tables

are, when expanded or rewritten in English, everyday items of knowledge among people who design SC systems or advise beginners on their use. We organise the knowledge-base in this way to make changes and incremental additions easy and transparent.

Inputs are users' programs plus indications of defects to be overcome and of transformations or pattern-directed searches to be applied to the programs. Direct outputs are transformed programs accompanied by messages about the reasons for the transformations. Where these programs can then be submitted to REDUCE for execution, the effects of the execution may be regarded as secondary outputs that are suitable for new passes through the pattern-matching process if required.

3 Examples of knowledge and associated patterns

3.1 P-A-E triples

The general structure of patterns and explanations is indicated above. Actions involve either making modifications in a user's program or in setting up additional code to request trace or control information from the system which is to run that program. Both types of action are achieved by calls to LISP functions which are designed to carry them out, and which take the values of parts of successful matches to DYNPAT patterns as arguments. When a pattern P is matched successfully, the corresponding explanation E in the triple that includes P is printed out, and the action is then performed.

The descriptions below cover elements of triples, plus their set-labels, that express knowledge about behaviour of SC programs. General information on the DYNPAT syntax for P-elements is given in [2,3]. E is in each case a paraphrase of the description.

(a) Knowledge about silence

If a user's program is to compute and print out a family (particularly if indexed) of results, but exhausts storage without printing anything, it is probable that some of the member have been computed but that the program is organised in a structured way, with "compute all the results" followed by "print all the results". The cure is to transform the program to interleave the two types of operation, as far as possible. An expert in SC will sometimes need to recommend unstructured programming!

```
P: FOR (n (NOT (SUM OR DO))) (SUM OR DO) (n (NOT WRITE))
    (((WRITE) ((n ((char) AND (NOT ; )))) $B)) $A)
A: (MAPPING (SETQ X (LISTIFY B))
             (INSERT-AFTER-LAST UPROG (INIT (LIST 'SETQ (CAR X))))
             (STMT ('WRITE (CAR X)))))

```

(REMOVE UPROG (LISTIFY A))

Labels: NOVICE, NO-FREE-STG, OPTIMISE(NIL), STATIC

Above, UPROG is a global pointer to the user's program, NO-FREE-STG indicates that there is possible relevance of the triple to cases where exhaustion of storage is the difficulty (which can be notified directly via a declaration (PROBLEM NO-FREE-STG) accompanying the program, OPTIMISE(x) indicates that the action expresses an optimisation which, by itself, is of high (x = T) or low importance, \$A and \$B in P refer to

A and B in the action, and the action itself specifies the program-transformation to be carried out if UPROG matches P.

(b) Detection of an instance of recursive substitutions

In advising one novice user about unusual behaviour of a program, we traced the problem to the recursive use of LET substitutions. Because we have observed this programming style in other beginners, we believe that it should be searched for in any novice program which uses LET if troubles occur at run-time. As the cure for the problem depends on the user's intentions, there is no general action to fill in for A, and all that can be done is to print out an explanation of why recursive substitution is dangerous.

```
P: LET $A = (n (NOT ; )) $B (n (NOT ; )) ;
   LET $B = (n (NOT ; )) $A (n (NOT ; ))
A: NIL
Labels: NOVICE, SUBST, STATIC
```

(c) Removal of unnecessary evaluations inside procedure-calls

Pearce and Hicks [6] mention that calls to user-defined procedures f which are of the form $u := f(g(n,x))$, where g is a given expression depending on a loop index n and a symbolic quantity x, may be highly inefficient if each call generates new non-trivial symbolic results. They recommend the introduction of a new statement $v := f(g(N,x))$ just before the loop, and replacement of the u-binding statement by the result of the local substitution (via SUB in REDUCE) of n for the dummy symbolic quantity N in v. In the example below, the right-hand side of the original u-binding is more general: it contains $f(g(n,x))$ somewhere, rather than containing only $f(g(n,x))$.

```
P: (n PROCEDURE name) £ FOR $A := £ (n (NOT ((m (NOT := )) := £
   name £ , ))) (((n (NOT := )) := ) $E)
   (£ name "(" ((£ $A £) $B) ")") £ $D ) $C)
A: (SIGNAL T) (SETQ V (GENSYM)) (SETQ N (GENSYM))
(SETQ G (PATIFY (LIST 'SUB B N V)))
(REPLACE UPROG C (PATCON E G))
(SIGNAL "TYPE THE STATEMENT WHICH IS TO BE FOLLOWED BY THE EXPRESSION OF
FORM [*] ABOVE")
(INSERT-AFTER UPROG MSG (LIST 'SETQ V (LISTIFY (PATSUB N A D))))
Labels: NO-FREE-STG, STATIC, OPTIMISE(NIL)
```

The SIGNAL function in A is used when explanations prompt input (argument T implies yes/no input) from the user. Here, the explanation indicates the $v := \dots$ statement above with a label (*), and asks the user if automatic modification of the code is desired. As there can be complicated situations where the answer is "no" (e.g. if more than two loops are nested), the second self-explanatory SIGNAL is reached only if the user's response to the original explanation is "yes". Again, any quantity \$x in P above corresponds to x in A.

(d) Unrolling of fixed-size loops

Dongarra and Hinds [1] observe that there are advantages in replacing loops whose indices run over a small fixed range (say n) by the n instances of the corresponding code. Our use of SC systems suggests that this may be so if $n \leq 5$ and the loops are simple and contain references to arrays or procedures with more than one index or argument.

```
P: ((FOR $A := (num $B) STEP (num $C) UNTIL (num $D) DO ) $H )
   £ $E "(" $A "," £ ")" (( $A "," ) $G)

A: (SETQ X (FINDSTM UPROG G)) (COND ((GREATERP (DIFFERENCE D B) 5) (RETURN NIL)))
   (SETQ Z X) (SETQ Y B)

G (INSERT-AFTER UPROG Z (SETQ Z (SUBST Y A X)))
   (SETQ Y (PLUS Y C))
   (COND ((NOT (GREATERP Y D)) (GO G)))
   (REMOVE UPROG X)
   (REMOVE UPROG (LISTIFY H))

Labels: STATIC, TIME, OPTIMISE(NIL)
```

(e) Moving of substitutions towards their first points of use

Ideally a LET declaration of the form $x = y$ should occur immediately before x is to be used for the first time with the value y , if superfluous searches of the REDUCE list of substitutions are to be eliminated. To avoid using new syntactic information here, we give only the case of a single substitution as an argument of LET.

```
P: LET (( $A = (n (NOT "," )) $B) ; £ (((($ATOM $C) AND ($C EQ $A))
   (n (NOT ; )) $D)

A: (REMOVE UPROG (SETQ Y (LISTIFY B))
   (INSERT-BEFORE UPROG (FINDSTM-END UPROG D) Y)

Labels: SUBST, OPTIMISE(NIL), STATIC, TIME
```

(f) Removal of superfluous BEGIN-END constructions

This is a standard optimisation [6] if BEGIN-END or << - >> enclose only a single statement. The inefficiency may arise in a program if a novice is too eager to respect ALGOL-like conventions, or if he has deleted statements during debugging.

```
P: (((BEGIN ((n (NOT END)) (NOT ; )) $A) END) OR
   (<< ((n (NOT END)) (NOT ; )) $A ) >>) $B)

A: (SETQ UPROG (SUBST (LISTIFY A) (LISTIFY B) UPROG))

Labels: NOVICE, STATIC, OPTIMISE(T)
```

(g) Example of a rule concerning degree of expertise

A user is unlikely to mention the EXP flag in REDUCE unless he is sufficiently well informed to know something about the order in which REDUCE performs steps of evaluation. Therefore, if EXP occurs in his program, he is probably not a novice.

```
P: (" " OR "(" OR ")") EXP (" " OR "(" OR ")")

A: (SIGNAL T)
   (COND ((EQ SIG 'YES) (DEACTIVATE 'NOVICE)))
```

Label: PATH

Here, the tutorial-style explanation about EXP is accompanied by the question "Do you therefore agree that you are not a novice-level user?". The P and A examples here are very simple, but represent a first attempt to use knowledge relevant to the modelling of a user's characteristics. PATH is the label used for rules about how to apply lower-level rules.

(h) Other applications of P-A-E triples

Similar triples have been constructed to cover simple cases of whether or not to represent rational expressions in common-denominator form, dead-variable elimination, dead-procedure elimination, profiling of statements of a program, transforming array or procedure calls of simple forms $f(g(n))$ in loops on n to $f(n)$ by moving the effect of $g(n)$ into the loop bounds and step size, redundant use of variables, optimising changes in operator strengths, and superfluous repetition of procedure or array calls. These cases are discussed either in [6] or a recent book on FORTRAN optimisation by Metcalf [5]. Some of them, e.g. dead-variable elimination, are difficult in general and require much more attention before one can hope to design patterns for dealing with them which are both versatile and satisfactory from the standpoint of efficient pattern-matching computations.

3.2 Control of the use of P-A-E- triples

The sets of triples are searched at present through accesses by means of their labels. The orders of labels given in sec. 3.1 is significant: all triples with a given label in first place are ordered ahead of those with the label in second place, for example. We state that a label x is relevant for a given program by a declaration (PROBLEM x). In some cases, e.g. NO-FREE-STG, REDUCE responds to a problematical program by an output from which x is obvious, but does not deposit x or some $f(x)$ in a LISP data-structure that can be accessed by a diagnostic program. In our experiments, therefore, we have chosen x -values for analysis of a given program by examination of the output of REDUCE when run with that program.

If our choices of x select triples whose P-elements all fail to match a test program, the remaining triples are used sequentially in attempted matches to the program.

4. Concluding remarks

The object of the scanning of a user's program as described in sec. 3 is to modify it to remove examples of inefficiencies, and to present explanations of the reasons why the modifications are made. Because many inefficiencies characteristic of novices are local and relatively easy to describe, the use of P-A-E triples to achieve ends such as those in sec. 3.1 is quite practical. We have written about 35 examples of static or dynamic patterns which have relevance to automatic analysis of users' programs.

To extend the effectiveness of the method, we are now examining the following problems or questions.

(a) Identification of the set of information required from an SC system in order to diagnose and correct all reasonable instances of users' inefficiencies if their programs have failed to run satisfactorily.

(b) Adaptation or construction of at least one system to provide all this information for automatic access by a diagnostic package.

(c) Identification of the interactions between basic P-A-E- triples which allow significant additional phenomena in users' programs to be detected. Alternatively stated, identification of those phenomena which cannot be expressed in individual triples in the present notation.

(d) Generalisation of the "label" concept to allow label-like information, containing variables, to be attached to triples so that the observations from (c) can be represented in the data-base.

(e) Design of an interpreter that can make efficient use of the information in (d).

(f) As examples such as 3.1(c) show, there is now room for a P-element syntax that does not involve the bracketing complication of the present DYNPAT syntax (that arose from our original need to implement a pattern-matcher quickly with the minimum of parsing effort).

(g) While it is fairly easy to write DYNPAT patterns for genuine dynamic pattern-matching (e.g. against sequences of function-calls inside an SC system; an example relying on Weissman's symbolic mathematical simplifier [7] as a system is given in [3]), using them in conjunction with a system as complex as REDUCE requires solution of interfacing problems of a fully general (in the sense of (b)) type, and tight control so that breadth-first searches for patterns do not exhaust the available storage.

This work was supported in part by the Science and Engineering Research Council, U.K., through grant GR/B/0151.7. F. Gardin acknowledges the award of a University of Exeter Postgraduate Scholarship.

References

1. J.J. Dongarra and A.R. Hinds, *Software - Practice and Experience* 9, 219 (1979)
2. F. Gardin, "DYNPAT User's Manual", manual M-104, Dept. of Computer Science, University of Exeter (1982)
3. F. Gardin and J.A. Campbell, in Proc. 1981 Symposium on Symbolic and Algebraic Computation, SYMSAC '81 (ed. P.S. Wang). Association for Computing Machinery, New York (1981), p. 233.
4. A.C. Hearn, "REDUCE 2 User's Manual", UCP-19, Computational Physics Group, University of Utah (1973)
5. M. Metcalf, "FORTRAN Optimization" (Academic Press, London, 1982)
6. P.D. Pearce and R.J. Hicks, in Proc. 1981 Symposium on Symbolic and Algebraic

- Computation, SYMSAC '81 (ed. P.S. Wang). Association for Computing Machinery, New York (1981), p. 131.
7. C. Weissman, "LISP 1.5 Primer" (Dickenson Publishing Co., Belmont, California, 1967).
 8. J. Weizenbaum, Comm. A.C.M. 9, 36 (1966).

APPENDIX: A brief description of some of the DYNPAT pattern-matching primitives

- any arbitrary sequence of lower-case letters and digits (the first character being a letter) may be used as an unknown (e.g. x, x1, unknown2, name).
- any unknown can be bound to only one item of the subject pattern.
- any arbitrary sequence of upper-case letters and/or numbers, delimited by \$, is a variable.
- any variable can be bound to any arbitrary sequence of items of the subject pattern.
- any atom (in the LISP sense) can be used to represent items of the subject pattern provided that they do not coincide with reserved words in DYNPAT.
- (< number > < pattern >) is a quick way to represent a pattern formed by some <number> of repetitions of < pattern > .
- (n < pattern >) is the application of the "arbitrary" operator to < pattern > ; its meaning is an arbitrary number of repetitions (including 0) of the pattern.
- the infix operators AND, OR and EQ are reserved words, with the obvious meanings.
- # stands for any arbitrary sequence of tokens of the subject pattern.
- \$ATOM is the pattern which matches any (LISP-like) atom.
- non-alphanumeric characters or character-sequences stand for themselves except where they have ambiguous meanings in DYNPAT, e.g. blanks or brackets. Characters which are ambiguous in this sense should be surrounded by double-quote marks.
- the design of DYNPAT to specify and match patterns is modelled on that of SNOBOL. Further details of the syntax are given in [2] and [3].

COMPUTER ALGEBRA and VLSI,
prospects for cross fertilization.

J Smit.

Dep. of Electrical Eng., EF9274.
Twente University of Technology, POBox 217,
NL-7500AE Enschede, The Netherlands.

ABSTRACT.

Experimental single-chip LISP processors are already being build, but high performance processor-intensive architectures, which might be used for CPU-intensive tasks in computer algebra are still in the early stages of design. Parallelism will be needed to extend the power of computer algebra systems. Implementation of a parallel EVAL scheme is one of the most frequently mentioned options. The implementation of special purpose parallel hardware with a tight coupling between storage and computational units, was proposed recently under the name systolic array. The actual introduction of a flexible, general purpose, processor-intensive computer partition as a part of a large scale multiprocessor Computer Algebra system, depends heavily on the progress made in design and technology needed to develop these computational structures. This paper describes ideas for the design of a fully programmable processor-intensive computer partition, which can be (micro)programmed from a high level language.

1. Introduction:

Four aspects which need, among others, special care before the actual implementation of a VLSI-chip with special Computer Algebra (CA) features are:

- a. Selection of CA algorithms which are good candidates for VLSI implementation.
- b. Design of the desired computational structure in terms of hardware elements, such as storage locations (registers), datapaths (busses), and functional units.
- c. Description of the time and place dependent (micro)programming features supported by the proposed chipdesign.
- d. Development of (micro) programming tools for the chip(s).

2. Allocation of processors and memory.

The intrinsic computational power of a parallel CA system grows linearly with the number of processors used. Its intrinsic storage capacity grows also linearly with the number of storage locations needed, which lies somewhere between one and four megabytes of memory. A two megabyte computer system may be build from, say four, memory boards with 64 memory chips of 64Kbit each or one memory board with 256Kbit chips, when technological advances and the redundancy techniques to improve the yield of these chips become available.

The requirements concerning speed and functionality of a single VLSI processor in a multiprocessor system depend on the number of processors used and the task which they should execute. The multiple processors are individually connected to their own local memory over a local bus and interconnected in a suitable communication network over another bus. Table 1 shows some ways to allocate N processors on the main memory of a Multiple Instruction Multiple Data (MIMD) machine. Such configurations are well suited for memory intensive CA problems.

#	Processors	Memory bus	Interconnect bus
1	on a dedicated board	Connector	Connector
4	1 on each board	On board	Connector
32	8 on each board	On board	On board & Connector
256	1 on each memory chip	On chip	On board & Connector

Table 1.

There is a limit on the density of processors per megabyte if one allocates an equal amount of the available memory to each processor. This limit is reached if the size (and hence price) of the system becomes dominated by the number of processors used.

A multiprocessor system designed around 4 MC68000 processors is shown by Marti and Fitch [1]. More processors can be incorporated in a multiprocessor system if individual processors can be kept simple, as in [2]. This requires that system I/O, which involves interrupt handling and character processing, is allocated to a general purpose processor like an MC68000, such that the multiple processors can be designed to efficiently execute an object oriented language, such as LISP. The last alternative in which 256 processors are operated in a single system goes beyond the current limitations of hardware and software technology. Problems can be expected with production yields due to the large chip sizes due to the high overhead of interprocessor communication. Good scheduling algorithms will be needed to spread the data over the available processors, in order to solve the main software engineering problem.

Will the sketched MIMD computer system perform optimally for all CA algorithms? I.e. do we expect that all CA tasks are equally well served with a large amount of memory and a basically limited number of processors? The answer to this important question is definitely NO. There are CA algorithms with a high time complexity which require a relatively small memory partition. These algorithms will be identified in this paper and the hardware on which they can be run will be described in some more detail. The tools used to design and program the hardware for the mentioned processor-intensive computer partition will be described briefly.

3. Characterization of basic Computer Algebra algorithms.

Any algorithm which requires a maximum amount of M units of storage for intermediate results and a linear amount ($O(M)$) of computational steps when expressed in the storage requirement, on a uniprocessor system, should be executed in the memory-intensive system partition, as the time needed for the transition of the data from the memory-intensive partition to the processor-intensive partition is of order $O(M)$.

It is important to note here that not the length of the input for the algorithm, N , is of critical importance, but instead the maximum amount of

memory, M , needed by the algorithm in any intermediate stage. The calculation of the determinant of a sparse matrix, is an example of an algorithm which is often exponential in N , but linear in M , as M is itself also exponential in N .

Basic algorithms affected by this rule are:

1. Algorithms which depend on sorting of like terms and/or powers, like the algorithms used for the operators $+$, $-$, $*$, when applied to sparse polynomials, which require $O(M)$ steps when a hashing technique is used and $O(MxLog(M))$ if good sorting algorithms are used to conserve space.
2. Knowledge based algorithms, as required for transformations on symbolic expressions, such as needed for differentiation of arbitrary functions like LN, SIN and COS.

All algorithms with an $O(M^2)$ and higher complexity should be run on the processor-intensive partition of the CA machine. This includes:

3. Algorithms with a numeric nature, like:

- * and $/$, with for instance:
 - Simple numbers like: integers, fractions and floating point,
 - Extended precision numbers (BigNum and BigFloat),
 - Dense polynomials.
 as operands.

A threefold partitioning with subsystems which are optimal for algorithms with a complexity of $O(M)$, $O(M^2)$ and $O(M^3)$, can be seen as a straightforward extension of the introduced concepts.

4. Utilization of locality of reference.

The given partitioning of a CA system optimizes the system for the global aspects of locality of reference.

The same aspect, is the basis of performance improvements at the (macroscopic) processor level. This was certainly one of the design objectives which has lead to the Reduced Instruction Set Computer (RISC) [3,4] and its chip realisation, which utilizes a simple instruction set to make room for an on-chip hardware stack organized as multiple register sets with overlap of register sets for parameter passing to subroutines. The improved performance of this processor can be traced back to the reduced number of slow memory references as compared to the number of fast references to the hardware stack.

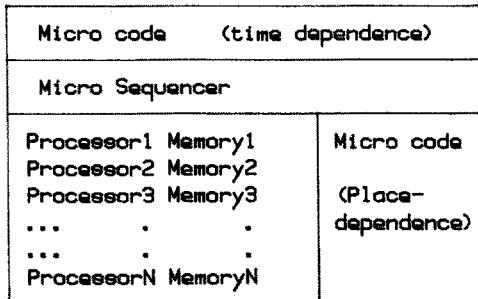
Single chip systems which are designed to execute a high computational load [5,6,7], such as required for digital signal processing, implement separate computational units for addition and multiplication. Pipelining of (micro) instructions results on such architectures in a very short (micro) program, which can be as short as 1/6 of the program-length needed for a single address machine [8]. Multiple memories and multiple datapaths are needed in these architectures to route several operands simultaneously. This pipelined approach results in a highly increased speed at a moderate chip area. The mentioned chips have the advantage of full (micro) programmability from a high level language [8,9].

Hard wired, Single Instruction Multiple Data (SIMD) systems were suggested by Kung [10] as multiprocessor elements in the processor-intensive part of a CA system. An assumed low cost of a chip is presumed as a justification for the use of a hard wired chip when he states: "The cost of a special purpose system must be low enough to justify it's limited applicability. For VLSI

chips the dominant costs are typically on design and testing".

Our experience with highly efficient compilers, which translate a given algorithm description automatically from a high level language, to microcode for a pipelined architecture, has encouraged us to postulate the feasibility of programmable processor-intensive computer partitions, which could cooperate with a memory-intensive computer partition to solve CA problems with the best available tools. Processor-intensive computer partitions might be used with processors placed in several interconnect structures, but a linear array of processors with up to 8 memory locations per processor and a restricted interconnect pattern is almost certainly a good starting point for trial designs.

The algorithms which are mapped on the processor-intensive partition of a computer can be viewed as realizations of a piece of dedicated parallel hardware. The word "realization" should be read here as an instantiation through a (micro) program with separate microwords for time and space dependent program specifications, as given in figure 2.



A linear (micro) programmable, processor-intensive array.

Figure 2.

A modelling and design language, dubbed MODL was used designed to describe, among other aspects, the essential timing aspects found in dedicated hard wired algorithms, as well as microprogrammed versions of these algorithms. The ideas behind this tool will be used as an introduction for the specification and programming techniques which are currently used with great success.

5. Modelling of hard-wired algorithms.

The Modelling and Design Language MODL, is a convenient high level language, with well chosen semantics which can be used by non experts to describe algorithms and hardware suited for direct implementation into silicon. An excellent behavioral correspondence between description and the corresponding VLSI design is thought to be of major importance for a hardware description language [11,12,13]. The syntactic form of the language used is important as far as it concerns the user-friendliness, but more important are adequate semantics, availability and the time needed to develop design tools. It will be clear that a LISP based system is a good choice for this purpose.

The two aspects of a VLSI design: 1. algorithm design and 2. hardware design have corresponding aspects in LISP datastructures, namely as algorithm and as data. The implementation of MODL in a LISP systems was rather straightforward, as a LISP compiler treats algorithms as data. The design tool for the VLSI designer, which is called a SILICON compiler, should map an algorithm into (mask) data. The microcoderecompiler for the processor-intensive computer

partition generates bitpatterns which are loaded into the writable control stores which determine the time and place dependent control of the processor structure. This process can be seen as a mapping from the given computational graph into the graph of the processor structure. Arbitrary processor-intensive algorithms, like BigNum multiplication and division can be mapped onto the processor structure, provided that the structure provides enough memory and an adequate amount of processors. The algorithms needed for basic multiplication and division algorithms are so regular and well understood, that it was found to be hardly illustrative to show these as an example. Instead a considerably less regular example is taken from the already advanced area of digital signal processing. This has the accompanying advantage that a hardware view as a dual of a software view is an accepted idea in this discipline and its widely available literature.

6. Hardware semantics:

Clocked synchronous hardware is characterized by the

- State and the
- Functional behaviour of the hardware.

The terms used in hardware descriptions for these semantic units are given in the left column of figure 3.

Hardware elements:	Descriptive elements:
1. BUS	Variable
2. REGISTER	Variable (CLOCKED)
3. SUBUNIT	MODEL definition
4. CLOCK	LOOP construct
5. CONTROL PATH	Conditional statements.
6. Primitive FUNCTION	FUNCTION definition.

Figure 3.

A proposed set of descriptive elements to be used in a design environment is given in the right column. The semantic rules 1. and 2. connect names to the resources in a design. The creative action of a design, is captured in the semantic rule number 3 which gets more attention in sections 7 and 11. Synchronous hardware is always driven by a periodic clock, which can be represented by an infinite loop in the design language. All statements used inside this loop are thought to execute in parallel. The statements within a branch of a conditional statement are executed in parallel whenever the condition is met. Semantic rule number 6. shows the need for primitive functions. One may think here of NAND and or NOR circuits as basic building blocks if the MODL language is used as a lowlevel hardware modelling and design language and of Arithmetic and Logic Units, or even complete Arithmetic units such as ADDERS, MULTIPLIERS etc. if MODL is used as a high level modelling tool or as a (micro)programming language.

Changed semantics:

- 1) Assignments to CLOCKED variables are actually assignments to the SLAVE associated with that variable.
- 2) Assignments MASTER:=SLAVE are placed at the end of the ClockLOOP.
- 3) Only one assignment can be made to a Bus-Variable during a clock period.
- 4) The value of a Bus variable is undefined at the start of the ClockLOOP.

7. Hardware instantiation.

A 4-bit counter can be defined with a bus variable or with a CLOCKED variable:

MODEL definition:

```
MODEL counter(X),           MODEL counter(X),
      CLOCKED(X),
      X:=MOD(X+1,16),
ENDMOD;                   ENDMOD;
```

These counters can be instantiated with a MODEL call and driven by a Clock-LOOP:

```
LOOP                  LOOP
  ..
  counter(Y),          counter(Z),
  ..
ENDLOOP,              ENDLOOP,
```

A timing analysis module, which is part of the design software package can now easily detect that the use of a non-clocked variable as a counter defines an algebraic loop and is hence invalid. This module will therefore issue an error message, showing the name of the bus on which the conflict occurs.

The expanded code for the, correct, CLOCKED version becomes:

```
LOOP
  ..
  ZSlave :=MOD(ZMaster+1,16),
  ..
  ZMaster:=ZSlave,
ENDLOOP,
```

So far, the need of consequent naming of variables and the distinction between CLOCKED and bus variables was discussed. But it is also important to observe that local variables, such as the counter value of a frequency divider should be retentive.

```
MODEL Divider (Output, MaxCount),
  WHEN counter (X) = MaxCount,
    X := 0, Output := 1,
    EXIT,
    Output := 0,
ENDMOD,
```

This aspect of retention of local values is exactly a reflection of the creative aspect of a MODEL call, it is an instantiation of resources.

The passing of actual arguments, which has a direct correspondence with wiring in hardware, is the next semantic topic which deserves extra attention.

8. Argument passing and wiring.

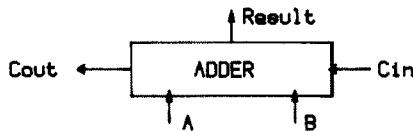
One may think of a SUBUNIT in a VLSI design as a box with wires which pass a cell boundary. This suggests the following semantic relations for parameter passing:

Parameter passed:	Wiring in cell boundary:
-------------------	--------------------------

- 1) Constant None.
- 2) normal VARIABLE Of specified bus.
- 3) CLOCKED VARIABLE Of specified register.
- 4) a MODEL call "Unnamed" wires pass the cell boundary.

The formal arguments of a MODEL are simple symbols or ARRAY elements. A parameter list may look like this:

MODEL ADDER(A,B,Cin,Result,Cout),



Modern VLSI layout design uses wiring on the four sides of the bounding box of a cell. This aspect of wiring by abutment will not be discussed here.

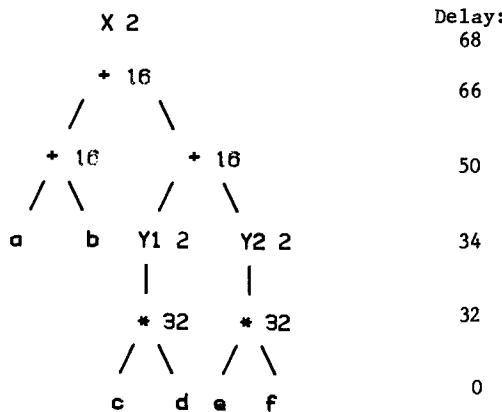
It should be noted that the architecture of the processor-intensive computer partition itself, can be modelled with MODL as well as the algorithms which should be microprogrammed on the processor structure.

9. Timing and critical path analysis.

The module TIM is designed to calculate the timing of a hardware model for all variables taking any CLOCKED declarations into account. The module PATH calculates in addition the critical path. Timing information of this kind is not only essential for the design of hardware, it is also expected to play an essential role in the allocation process of the microcode compiler for a processor-intensive computer partition.

```

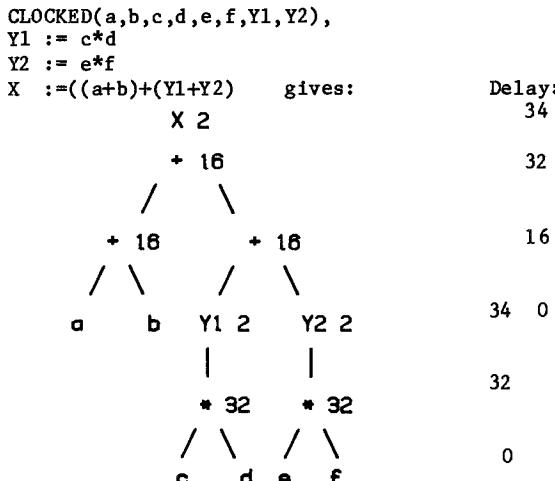
Example: CLOCKED(a,b,c,d,e,f),
          Y1 := c*d
          Y2 := e*f
          X := ((a+b)+(Y1+Y2))   gives:
  
```



The critical path is now represented by the symbolic expression:

(68 (X ++ (Y1 * (c) (d)) (Y2 * (e) (f))))

The effect of subtle changes in the use of CLOCKED can be seen from a slight extension of the previous example:



The critical path is now represented as:

(34 (X ++ (Y1) (Y2))) (Y1 * (c) (d)) (Y2 (* (e) (f))))

These subtle changes are known as pipelining.

10. Functional simulation.

Functional simulation is frequently considered to be a major task by those who are unfamiliar with LISP, but given the fact that the hardware description is given by a single symbolic expression, the output of the model expansion module MOD, which will be described into more detail in the next section, it is only a triviality. A call to the LISP function EVAL with the

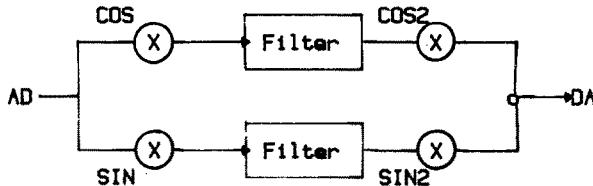
expanded modelbody as an argument is sufficient to simulate a hardware model of a given problem. This technique may be used to test an algorithm description in the memory-intensive part of a parallel CA system, without need to generate the microcode for the processor-intensive computer part.

A simulated analog to digital converter and a simulated digital to analog converter are additional basic ingredients of an already available support package for digital signal processing algorithms. The digital to analog converter is used to display simulation results. Part of the description of the algorithm is an accurate description of the arithmetic used.

The simulator is currently used as a useful tool to analyse the behavior of digital hardware, such as computer structures, digital filters, Bignum algorithms and the like.

11. A digital signal processing example.

A Weaver single sideband modulator which transfers a telephone channel to a primary group of an FDM transmission system is given schematically:



Where each Filter is build as a cascade of one first order filter and three second order filter sections.

The associated hardware/algorithm can be specified with the following description which is based on an implementation of a MODEL feature in muSIMP.

```

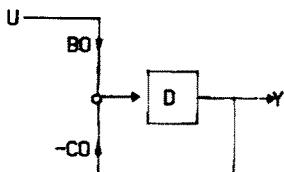
MODEL Weaver(),
  N:=0,
  M:=0,
  % tables: length: index interval:
  SIN, COS 0..20   N  0..2*pi
  SIN2, COS2 0..2   M  0..2*pi      %
  TABLE(SIN, COS, k1),
  TABLE(SIN2,COS2,k2),
  LOOP,
    X:=AD(),
    N:=ModN(N,EVAL(k1)),
    M:=ModN(M,EVAL(k2)),
    Filter(X*COS(N),Xout), Filter(X*SIN(N),Yout),
    DA:=Xout*COS2(M)+Yout*SIN2(M),
  ENDLOOP,
ENDMOD;

% Elliptic Lowpass filters,
Sample frequency: FS=42.0 Kc,
Passband edge: Fp= 1.7 Kc,
Stopband edge: Fs= 2.3 Kc,
Passband ripple: .5 DB
Stopband level: 60.0 DB
%
```

```

MODEL Filter(U,Y),
  First( U ,Y1, 4781711 E -8, -9043658 E -7 ),
  Second(Y1,Y2, 3025599 E -7, -5550694 E -7,
         -1872219 E -7, 9222696 E -7 ),
  Second(Y2,Y3, 5628951 E -8, -8856287 E -8,
         -1830456 E -6, 8544722 E -7 ),
  Second(Y3,Y, 5439686 E -7, -1022025 E -6,
         -1911246 E -6, 9771588 E -7 ),
ENDMOD;

```



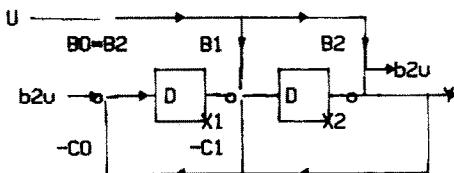
% 1St Order Filter Section

%

```

MODEL First(U,Y,B0,C0),
CLOCKED(Y),
  Y:=B0*U-C0*Y,
ENDMOD;

```



% 2nd Order Filter Section with B0=B2

%

```

MODEL Second(U,Y,B2,B1,C1,C0),
CLOCKED(X1,X2),
  X2:=-C1*(Y:=X2+(b2u:=B2*U))+X1+B1*U,
  X1:=-C0*Y+b2u,
ENDMOD;

```

The Weaver modulator described with the above text can now be used in conjunction with several hardware analysis, c.q. modelling modules:

1. TIM A hardware timing analysis module.
2. PATH A critical path analysis module.
3. TERM A functional simulator with output to the terminal.
4. PLOT A functional simulator with output to the plotter.
5. CMP A microcode generator for a dedicated signal processor to perform realtime simulations.
6. NEC A microcode generator which generates highly optimized code for the NEC uPD7720 signalprocessing chip.

Available as an RLISP program is:

7. CODOPT A code optimizer [14,15] for algebraic expressions as well as hardwired logic.

These tools will be used for the design of, and the microcode generation for, a processor-intensive chip.

Conclusion.

Programmable, processor-intensive, computer partitions which enhance algorithms with an arithmetic nature ($O(M^2)$ and higher complexity) are feasible extensions for the CA systems in the next years. The introduced MODL hardware description tools make it possible to refine the specification of a programmable processor-intensive computer partition. The available experience with codegenerators for highly pipelined microprogrammed processors is expected to be a rigid starting point for new investigations. The MODL description language is also an ideally suited tool for the specification of user programs which will be mapped onto the processor-intensive computer partition. The timing analysis module TIM may also be used either by the user or by the microcode compiler to improve the quality of the generated parallel algorithm.

REFERENCES

- [1] Marti, J. and Fitch, J. P., "The Bath Concurrent LISP Machine", These proceedings.
- [2] Sussmann, G. J. et al. Scheme 79, LISP on a chip, IEEE Computer, Vol 14, no 7. July 1981, pp 10-21.
- [3] Patterson, D. A. and Ditzel, D. R., The case for the Reduced Instruction Set Computer. Computer Architecture News, October 1980.
- [4] Patterson, D. A. and Sequin, C. H., Risc I: A Reduced Instruction Set VLSI Computer. Proceedings of the Eighth Annual Symposium on Computer Architecture (May 1981), Minneapolis Minnesota, pp 443-457.
- [5] Thompson, J. S. and Tewksbury S. K., LSI Signal Processor Architecture for Telecommunication applications. IEEE Acoustics Speech and Signal Processing, ASSP-30, number 4., pp 613-631.
- [6] Nishitani, T. et. al., A single chip digital signal processor for telecommunication applications. IEEE Journal of Solid-State Circuits, Vol. SC-16 (1981), pp 372-376.
- [7] NEC Catalog uPD7720 Digital Signal Processor, pp 519.
- [8] Smit, J., Herrmann, O.E. and Brederoo, R., Microprogramming from a High Level Language, theory and practice. Twente University of Technology Research Report 1231-AM-0683.
- [9] Smit, J., Herrmann, O.E. and Jansen, J. H., Flexible Tools to Microprogram Signalprocessors. Twente University of Technology Research Report 1231-AM-2282.
- [10] Kung, H.T., Use of VLSI in Algebraic Computation, Some suggestions. Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation. pp. 218-222.
- [11] Piloty, R., Barbacci, M., Borrione D., Dietmeyer, D., Hill, F. and Skelly, P. "CONLAN-A Formal Construction Method for Hardware Description Languages, Basic principles, Language derivation, Language application", Proc. of the National Computer Conference 1980, Vol 49, Anaheim, California, pp 209-236.
- [12] Special issue on Hardware Description languages, IEEE Computer Society, Computer, Vol. 10, No. 6., June 1977.
- [13] Lim, W.Y.P., "HISDL, A structure description language". Comm. of the ACM, Vol. 25, no 11, pp. 823-831.
- [14] Van Hulzen, J.A., Hulshof, B.J.A., Smit, J., "Code optimization facilities in the NETFORM context". Twente University of Technology: Memorandum nr.IW-368.
- [15] Van Hulzen, J.A., Breuers grow factor algorithm in computer algebra. Proc. of the 1981 ACM Symposium on Symbolic and Algebraic computation. pp. 100-105. (1981).

CODE OPTIMIZATION OF MULTIVARIATE POLYNOMIAL SCHEMES: A PRAGMATIC APPROACH

J.A. van Hulzen
Twente University of Technology
Department of Computer Science
P.O. Box 217, 7500 AE Enschede, the Netherlands.

Abstract: Heuristic, structure preserving algorithms are presented, based on Breuer's grow factor algorithm, for finding common subexpressions in a set of multivariate polynomials with integer coefficients. It is shown how these polynomials can be decomposed into a set of linear expressions and a set of monomials, interrelated via hierarchical structure information, the "noise". Then algorithms are shortly discussed to find common subexpressions, by reducing the arithmetic complexity of both sets, when viewing them as blocks of straightline code, and for communicating subresults between both sets, called "noise conquering". Further extensions are shortly indicated.

1. Introduction

In a previous paper [20] we argued that it can be profitable to consider a common subexpression search (css) as being an attempt to minimize the arithmetic complexity of a block of straightline code. Such a css can be relevant for simplification, for improving the readability of output and thus consequently also for preparing optimized code for numerical evaluation. However our algorithms were tailored for a class of problems, mainly occurring in electrical network theory [10,19]. This justified to assume the structure of the input, sets of multivariate polynomials, to be quite specific and restrictive. Although any internal representation, what so ever, is allowed, exponentiation and constant coefficients not equal to ± 1 are in principle excluded. Dropping such artificial restrictions allows to apply a css on more realistic input, thus broadening its applicability to sets of multivariate polynomials with integer coefficients. The main aspects of this more general algorithm are discussed in section 2. Essentially it is based on the assumption that the commutative law holds, for both addition and multiplication. Future extensions based on the distributive law are shortly indicated in section 3. Once such facilities are available a further extension, as to include elementary transcendental functions, can hardly be qualified as complicated.

But before presenting extensions of our original algorithm, denoted by T, we shortly review and discuss the original approach through an example, thus hoping to create an adequate context.

Let us assume to have an initial set E_0 of N_0 multivariate polynomials q_i in $\mathbb{Z}_2[v_0]$, where V_0 is a set of M_0 indeterminates. This initial set $E_0 = \{q_1, \dots, q_{N_0}\}$, the input for T, can be interpreted as the definition of a block of straightline code $B_0 = (P_0, V_0, U_0)$. Here P_0 is a list of assignment statements of the form $e_i \leftarrow v(q_i)$, where $v(q_i)$ is the value of q_i , obtained by replacing the elements of V_0 , when considered as set of inputnames, by a set of input values. $U_0 = \{e_1, \dots, e_{N_0}\}$ is simply a set of outputnames. The arithmetic complexity, taken over all permissible inputs, can be defined as a pair $AC_0 = (n_0^+, n_0^\omega)$, where n_0^ω stands for the number of binary ω -operations.

Example 1.1: Let $E_0 = \{q_1, q_2, q_3\}$ where

$$q_1 = a + b + c.d + (g + f).(a + b.c.d)$$

$$q_2 = a + f + c.d(g + f).(a + b.c.d)$$

$$q_3 = a + g.c.d(g + c.d).(a + b)$$

Then $V_0 = \{a, b, c, d, g, f\}$, $M_0 = 6$ and $AC_0 = (12, 14)$. \square

It is obvious that P_0 certainly not reflects the most optimal way to obtain output. Hence a css can be viewed as an attempt to minimize AC_0 .

An element q of E_0 is said to be of ω_k -type if ω_k is its leading operator and assuming $\omega_1 = *$ and $\omega_2 = +$. Each q has an arbitrary but finite number of operands, which are all different. This number is called the length $\lambda(q)$ of q . Obviously $\lambda(q) \geq 2$. If q has α operands from V_0 the other $\beta = \lambda(q) - \alpha$ are of ω_{3-k} type. Either $\alpha=0$ or $\beta=0$ is possible. Hence each q consists of a so called primitive part formed by elements of V_0 , concatenated by ω_k -symbols, and of a composite part formed by β expressions of ω_{3-k} -type, again concatenated by ω_k -symbols. This suggests to decompose E_0 into a description D_0 , consisting of two separate sets, one of ω_1 - and another of ω_2 -type primitives, both grouped together in a matrix, such that via additional structure information E_0 can be redesigned from D_0 . Figure 1 gives a simplified picture of D_0 , reflecting the actual implementation, based on a list representation for sparse matrices (If no confusion is possible π, σ are used instead of ω_1, ω_2). The rows of the $(B\omega_k)_0$ -matrices reflect which elements of V_0 are incident with which ω_k -type primitives. The integers in the $(\alpha\omega_k)_0$ -columns denote the lengths of these primitives. The $(\psi\omega_k)_0$ -columns consist of rownumbers of the $(B\omega_{3-k})_0$ -matrix, defining the corresponding fathers. The integers in the $(\beta\omega_k)_0$ -columns denote the lengths of the composite parts. The $(B\omega_{3-k})_0$ -rownumbers, corresponding with the description of these ω_{3-k} -type operands, are given in the $(\delta\omega_k)_0$ -lists. The integers in the $(\delta\omega_k)_0$ -rows finally, reflect the density of the corresponding elements of V_0 . A straightforward application of Breuer's grow factor algorithm [5], based on the use of weights defined by $(\delta\omega_k)_0, (\alpha\omega_k)_0$, as will be explained below, leads to the common

Σ	a	b	c	d	g	f	$\alpha\sigma$	$\psi\sigma$	$\beta\sigma$	$\delta\sigma$	Π	a	b	c	d	g	f	$\alpha\pi$	$\psi\pi$	$\beta\pi$	$\delta\pi$
	1	2	3	4	5	6						1	2	3	4	5	6				
q_1	1	1					2	0	2	(1, 2)	Π	1		1	1			2	1	0	
	2			1	1		2	2	0			2						0	1	2	(2, 3)
	3	1					1	2	1	(3)		3	1	1	1			3	3	0	
q_2	4	1			1		2	0	1	(4)		4		1	1			2	4	2	(5, 6)
	5			1	1		2	4	0			5	1	1	1			3	6	0	
	6	1					1	4	1	(5)		6		1	1	1		3	7	2	(8, 9)
q_3	7	1					1	0	1	(6)		7		1	1			2	8	0	
	8			1			1	6	1	(7)											
	9	1	1				2	6	0												
$\delta\sigma$	6	2		3	3						$\delta\pi$	2	6	6	1						

$$D_0 = (\begin{array}{ccccc} D\sigma_0 & , & H\sigma_0 & , & D\pi_0 \\ = (B\sigma_0, \alpha\sigma_0, \delta\sigma_0) & , (\psi\sigma_0, B\sigma_0) & , & (B\pi_0, \alpha\pi_0, \delta\pi_0) & , (\psi\pi_0, \beta\pi_0) \end{array})$$

Figure 1

subexpressions (cse's) $s_1=a+b$ and $p_1=c.d$. This illustrates that the D_0 -description allows an all level search. Once a cse is found it can be removed from its $B\omega_k$ -description, assuming however that, from that moment on, it is considered as new indeterminate, if necessary in both schemes, and that its description is added to the matrix it originated from. Hence detection of $p_1=c.d$ will result in the following resettings in $B\pi_0$:

$$\left. \begin{array}{l} - \alpha\pi(i) \leftarrow \alpha\pi(i)-1 \\ - B\pi(i,j) \leftarrow 0; B\pi(i,7) \leftarrow 1; B\pi(8,j) \leftarrow 1 \\ - \delta\pi(j) \leftarrow \delta\pi(j)-5 \end{array} \right\} \quad i = 1, 3, \dots, 7 \text{ and } j = 3, 4.$$

Since $\alpha\pi(1)=1$ (primitive reduced to new indeterminate), $\psi\pi(1)>0$ and $\beta\pi(1)=0$ (now contained as new indeterminate in (a)primitive sum(s) elsewhere) p_1 is also introduced in $D\sigma_0$. After these resettings the new description, D_1 , is used for further optimizations. Clearly both $(B\omega_k)_1$ -matrices are larger and sparser, implying that after some steps further searches will become useless. Hence T is an iterative process, based on reductions of binary ω_k -operations. This leads to

Definition 1.1: Let AC_i be the arithmetic complexity of E_i , $i \geq 0$. A transformation $T: E_i \rightarrow E_{i+1}$ is called an optimization iff $AC_i > AC_{i+1}$. The set $E_\ell = T^\ell(E_0)$, $\ell \geq 0$, is called an optimal description of E_0 with respect to T iff $AC_0 > AC_1 > \dots > AC_\ell = AC_{\ell+1}$. \square

Since we operate on D_i , instead of E_i directly, we consider T as a composite function such that $E_\ell = T(E_{\ell-1}) = R^{-1} T_B^{\ell} R(E_{\ell-1}) = R^{-1} T_B^{\ell} R(E_0)$. The value of ℓ depends

on a termination condition, related to AC, say profit. Assuming $\text{profit}_\ell = \text{false}$ and $\text{profit}_i = \text{true}$ for $i=0, \dots, \ell-1$, gives:

$$R : E_0 \rightarrow (D_0, \text{profit}_0)$$

$$T_\beta : (D_i, \text{profit}_i) \rightarrow (D_{i+1}, \text{profit}_{i+1}), i=0, \dots, \ell-1.$$

$$R^{-1} : (D_\ell, \text{profit}_\ell) \rightarrow E_\ell$$

As suggested by the example we also have to consider the extended Breuer algorithm, now denoted by T_β , as a composite function: $T_\beta = M\pi \cdot T\sigma \cdot M\omega \cdot T\pi$.

The function $T\omega_k$ is supposed to be applied on the pair $((D\omega_k)_i, (p\omega_k)_i)$, $i \geq 0$, where $(p\omega_k)_i$ denotes the local profit criterium, used to possibly select an ω_k -type cse. As already indicated, it might be possible that this cse can be added as new indeterminate (column) to the other matrix as well, before the iterative search, now based on the $(i+1)$ -th version of the original description, is continued. These modifications are achieved by the functions $M\omega_k$. This part of the overall process is shortly called "noise"conquering. It is obvious that it is only based on the assumption that the commutative law holds for both operators. When also assuming the validity of the distributive law, i.e. when deciding to locally factor out gcd's of π -type terms of σ -type expressions, the restructuring activities, covered by $M\omega_k$, have to be extended. The profit criteria are embodied in the Breuer searches, denoted by $T\omega_k$. Again $T\omega_k$ can be considered as a composite function: $T\omega_k = U\omega_k \cdot C\omega_k \cdot E\omega_k$.

Application of $E\omega_k$ is required to eliminate redundancy from $(B\omega_k)_i$. Since ω_k has at least two operands, rows and columns with weight 1 can obviously be disregarded, implying that a repeated "removal" might lead to the conclusion that no further profit can be gained. $C\omega_k$ achieves the detection of a cse. Each column has a weight, denoting the number of occurrences of its corresponding indeterminate in ω_k -type primitives, and stored in $(\delta\omega_k)_i$. The weightfactors of all columns with maximal weight are computed. The weightfactor wf_k of column γ_k is the sum of the weights, given in $(\alpha\omega_k)_i$, of those rows, which share a 1 with γ_k . They deliver a "preview" of the structure of potential parents and allow to explore local 1-density. Therefore the most left column with maximal weightfactor is selected to obtain a submatrix of $(B\omega_k)_i$ for further processing. This process consists of a repetitive application of the same selection criterium. This results in stepwise extending the set J of column-indices and shrinking the set P of indices of parents (of the cse). The profit gained is $ws = (|J|-1) \cdot (|P|-1) > 0$. It reflects the reduction in binary ω_k -operations. The search is terminated when ws does not longer increase. $U\omega_k$, finally, carries out the updating operations, required for the incorporation of the cse-information: Its description as new row, its occurrences as new indeterminate via a new column, its removal from the matrix-description and resettings of the weights.

This approach implies that R^{-1} , responsible for the construction of E_ℓ , the optimized description of E_0 , has to contain a module for ordering of the cse's, which ought to

precede the rewritten, and now condensed, N_0 original expressions.

It is easily verified, when taking the set E_0 from example 1.1, that $E_4 = T^4(E_0) =$

{ $p_1 = c.d.$, $s_1 = a + b$, $p_2 = b.p_1$, $s_2 = f + g$, $s_3 = a + p_2$, $p_3 = s_2.s_3$,

$e_1 = s_1 + p_1 + p_3$, $e_2 = a + f + p_1.p_3$, $e_3 = a + g.p_1.(g + p_1).s_1$ },

implying that $AC_4 = (9, 7)$. This result shows the power of "noise" conquering.

Applications of Breuer searches remain limited to simple incidence matrices, which however reflect relations between structures of continuously increasing complexity, as visualized in figure 2. Therefore, this example also serves to illustrate the dynamic behaviour of our "data base", imposed by "noise" conquering.

i	Σ_i	"Noise" conquering	Π_i	AC_i
0	$s_1 = a + b \rightarrow$		$\leftarrow p_1 = c.d$	(12, 14)
1	$s_2 = f + g \rightarrow$		$\leftarrow p_2 = b.c.d$	(11, 9)
2	$s_3 = a + b.c.d \rightarrow$			(10, 8)
3		$\leftarrow p_3 = (f + g)(a + b.c.d)$		(9, 8)
4				(9, 7)

Figure 2

The CPU-times for this example are given in figure 3, column 1. Let us now try to give some indications for the time (and also implicity for the space) complexity.

R is obviously an $O(n)$ algorithm, where $n=\text{size}(E_0)=c \cdot \sum_{i=1}^{v_0} 2\{\delta\sigma_0\}_i + \{\delta\pi_0\}_i\}$ if $v_0=|V_0|$. To allow a real optimal choise from the initially $2(\frac{v_0}{2})=v_0(v_0-1)$ cse-candidates,

all alternatives have to be computed. This causes combinatorial problems, leading to exponential computing times, as also indicated by Allen [4], when discussing Breuer's algorithm. However, Breuer's heuristic approach is fast, albeit certainly not always optimal. In addition it ought to be stated that operating in an input structure preserving way can imply that equivalent but not identical representations not necessarily lead to identical AC-reductions. However, demanding for a canonical internal representation, if possible at all when also including elementary transcendental functions [7], can lead to a considerable increase of the inputsize n, and thus of AC_0 . It also might require a restructuring based on factorization, hoping to retrieve the original structure. All we request is a reasonable balance between efficiency and "optimality". Hence a time complexity estimation ought to be given in terms of profit gained, as also argued in [18]. Assume s denotes the average search time needed for finding one of the $m \geq l$ cse's. The average length of a cse is $\lambda=\lceil(AC_0 - AC_l)/m\rceil$. Such a search consists of repeatedly finding maximal elements from sets of atmost v_i short integers and of computing weightfactors by adding, in

general less than, $n/4v_i$ short integers. The repetitions are bounded by the cse-length. This suggests that the total time required is $O(\frac{n}{4} \cdot \lambda \cdot m) = O(n \cdot m)$. This is merely a somewhat pessimistic indication. In practice Ew_k will reduce the value of v_i considerable. The estimation $n/4v_i$ is, certainly after some steps, much too pessimistic. But the figures 3.2. and 3.3 indicate that a large v_0 -value has its price. For example 2, a problem in microwave technology, many indeterminates were

Example	1	2	3	
R	230	3762	4072	CPU-times
$T_{\beta^{-1}}$	75	6145	4692	in millisec.'s
R	10	251	592	on a DEC KL10 using Reduce-2
References	[10]	[10,18]	[21]	

Figure 3

required to symbolize minors, allowing to obtain large determinants in factored form [18]. Example 3, Hearn's problem of substitution, deals with an expression, which requires many powers and constants for its description. These objects are, as an ad hoc solution in the original algorithm, replaced by indeterminates, thus making the algorithm deaf for interesting structure information. It has the negative side-effect of increasing v_0 considerable, as also indicated in [21]. These experiments show why dag-models used in code optimization are questionable [2,3,9]. These dags are based on an intermediate 3-address code representation of E_0 , thus requiring associativity. It forces to consider constants to act as if they were indeterminates and powers as objects requiring function calls. This influences the time complexity negatively and forces to think of $2a+3b$ and $4a+6b$ or of a^2, a^4 and a^6 as being different entities. Another disadvantage is of course that a mathematical context, like provided by a computer algebra system, is completely missing. Kedem's approach [12] has a similar disadvantage, as clearly indicated by Rall's discussion of it [16]. The R^{-1} -part of T is in general fast, this in spite of the fact that a reordering of the cse's seems to be NP-complete on various machine models [1,6]. However ordering or minimization of the total number of temporary storage cells (or registers) required to hold the results delivered by T , can hardly be done realistically when only producing code for high level languages.

2. Code optimization of multivariate polynomial schemes.

When further extending this algorithm, we must avoid an explosive growth of the matrices and we have to limit the cses to fast heuristics. It is possible to achieve

this in a relatively simple way. But, before being able to argue this we need

Definition 2.1: Let $\omega_0 = \dagger, \omega_1 = *, \omega_2 = +, \Omega_1 = \Pi, \Omega_2 = \Sigma$. Let $v_i = (x_1, \dots, x_{M_i})$ be an M_i -vector, consisting of the elements of $V_i = \{x_1, \dots, x_{M_i}\}$. Let $b\omega_k(i, m)$ be an M_i -vector denoting the m -th row of $(B\omega_k)_i$. Then the generic inner product $G\omega_k(i, m)$ is defined by: $G\omega_k(i, m) = v_i \circ b\omega_k(i, m) = [\Omega_k]_{j=1}^{M_i} x_j \omega_{k-1}(b\omega_k(i, m))_j$. \square

Example 2.1: Figure 1 shows that

$$G\Pi(0, 3) = v_0 \circ b\Pi(0, 3) = (a, b, c, d, g, f) \circ (0, 1, 1, 1, 0, 0) = \prod_{j=1}^6 x_j \circ b\Pi(0, 3)_j = b.c.d \text{ and}$$

$$G\sigma(0, 1) = v_0 \circ b\sigma(0, 1) = (a, b, c, d, g, f) \circ (1, 1, 0, 0, 0, 0) = \sum_{j=1}^6 x_j \circ b\sigma(0, 1)_j = a + b. \quad \square$$

Let us now replace the $B\omega_k$ by integer matrices for a while. Assume, for instance, that $b\Pi(0, 3) = (0, 3, 2, 5, 0, 0)$ and $b\sigma(0, 1) = (2, 3, 0, 0, 0, 0)$. Then $G\Pi(0, 3) = b^3 c^2 d^5$ and $G\sigma(0, 1) = 2a + 3b$. Hence when allowing primitive sums to be linear combinations of elements of V_0 with constant coefficients from \mathbb{Z} and primitive products to be arbitrary monomials, the description D_0 of the original set E_0 will hardly differ from a similar description for a set E_0 of multivariate polynomials in $\mathbb{Z}_2[V_0]$. Hardly, since allowing arbitrary integer coefficients and powers requires an extension of $H\sigma_0$ with an exponent column and of $H\Pi_0$ with a coefficient column. So, in principle, we "only" have to modify the functions $C\omega_k$, $U\omega_k$ and $M\omega_k$. The overall strategy however remains intact.

Example 2.2. To illustrate these modifications we reconsider example 1.1, with inclusion of coefficients and exponents. So, let

$$q_1 = \{2a + 3b + 3c^2d^2 + (2g + 5f)^2(a + bc^2d^5)\}^3$$

$$q_2 = a + 3f + c^2d^3(4g + 10f)(a + bc^2d^5)^2$$

$$q_3 = 9a + 2gc^2d^5(g + c^2d)(4a + 6b)^2$$

Then D_0 , the description of E_0 , is given in figure 4. The column weights are omit-

Σ	a	b	c	d	g	f	$\alpha\sigma$	$\psi\sigma$	$\beta\sigma$	$\ell\sigma$	$\varepsilon\sigma$	Π	a	b	c	d	g	f	$\alpha\Pi$	$\psi\Pi$	$\beta\Pi$	$\ell\Pi$	$\gamma\Pi$
	1	2	3	4	5	6	1	2	3	4	5		1	2	3	4	5	6	7	8	9	10	
q_1	1	2	3				2	0	2	(1, 2)	3		1	2	2		2	1	0			3	
	2		2	5			2	2	0				2	2		0	1	2	(2, 3)	1			
	3	1					1	2	1	(3)	1		3	1	2	5	3	3	0			1	
q_2	4	1		3			2	0	1	(4)	1		4	2	3		2	4	2	(5, 6)	1		
	5		4	10			2	4	0				5	1	2	5	3	6	0			1	
	6	1					1	4	1	(5)	2		6	2	5	1	3	7	2	(8, 9)	2		
q_3	7	9					1	0	1	(6)	1		7	2	1		2	8	0			1	
	8		1				1	6	1	(7)	1												
	9	4	6				2	6	0														

$$D_0 = (D\sigma_0, H\sigma_0, D\Pi_0, H\Pi_0) = ((B\sigma_0, \alpha\sigma_0, \delta\sigma_0), (\psi\sigma_0, \beta\sigma_0, \varepsilon\sigma_0), (B\Pi_0, \alpha\Pi_0, \delta\Pi_0), (\psi\Pi_0, \beta\Pi_0, \gamma\Pi_0))$$

Figure 4

ted, since they need further consideration, in view of the modifications required for $C\sigma$ and $C\pi$. \square

2.1 A strategy for optimizing sets of linear expressions.

We now discuss how to reduce the $AC_i = (n_i^+, n_i^*)$ of the set of N_{σ_i} linear expressions defined by the triple $D\sigma_i = (B\sigma_i, \alpha\sigma_i, \delta\sigma_i)$, $i \geq 0$. The modifications in the function $T\sigma = U\sigma \cdot C\sigma \cdot E\sigma$ are related to the required reconsideration of the weights.

Let $v_+(B\sigma_i)$ and $v_*(B\sigma_i)$ denote the number of nonzero elements of the integer matrix $B\sigma_i$ and the number of its elements not equal to -1, 0 or 1, respectively. Then $n_i^+ = v_+(B\sigma_i) - N_{\sigma_i}$ and $n_i^* = v_*(B\sigma_i)$. Again $v_+(B\sigma_i)$ is the sum of the previously introduced rowweights (columnweights), now called additive weights and denoted by $\alpha_k(\delta a_k)$. These weights again define the lengths of primitive sums and the indeterminate densities in E_i , respectively. As before they can be used for removal of redundancy by $E\sigma$. But when trying to locate cse's by reducing AC_i requires consideration of n_i^* too. Let us therefore call the number of elements of the k -th row (or column) of $B\sigma_i$, not equal to -1, 0 or 1, the multiplicative weights $\alpha_m(\delta m_k)$. We finally consider $\alpha\sigma_k = \alpha a_k + \phi \cdot \alpha m_k (\delta a_k = \delta a_k + \phi \cdot \delta m_k)$ as total weights. Here ϕ is an expense factor, reflecting the number of binary operations which can be performed during one binary operation. The weightfactor mechanism will now be based on total weights, i.e. w_k is the sum of the total weights of those rows which have a nonzero element in common with the k -th column. The function $C\sigma$, defining a css, will again be based on Breuer's algorithm. All we require, in principle, are the following simple observations:

Let $a = (a_1, \dots, a_m) \neq (0, \dots, 0)$ and $b = (b_1, \dots, b_m) = (\lambda_1 a_1, \dots, \lambda_m a_m)$ be two rows of an (n, m) -integer matrix. Let $q_k(a, b) = b_k / a_k$ and $p_{k,m}(a, b) = q_k(a, b) / q_m(a, b) = \lambda_k / \lambda_m$. Then $\sum_{i=1}^l b_{k_i} x_{k_i} = \lambda \sum_{i=1}^l a_{k_i} x_{k_i}$, where $\{k_1, \dots, k_l\} \subseteq \{1, \dots, m\}$ and if $\lambda = \lambda_{k_1} = \dots = \lambda_{k_m}$. Alternatively holds $p_{k_1, k_2}(a, b) = \dots = p_{k_l, k_l}(a, b) = 1$ or $q_{k_1}(a, b) = \dots = q_{k_l}(a, b) = \lambda$.

Example 2.3 : Assume $a = (2, 3, 4, 5)$ and $b = (1, 6, 8, 10)$. Then $q_2(a, b) = q_3(a, b) = q_4(a, b) = 2$ and $p_{2,3}(a, b) = p_{2,4}(a, b) = p_{3,4}(a, b) = 1$. In fact $p_{2,3}(a, b) = 1$, for instance, expresses that $\det [\begin{smallmatrix} a_2 & a_3 \\ b_2 & b_3 \end{smallmatrix}] = 0$, since $p_{2,3}(a, b) = b_2 a_3 / (a_2 b_3) = 1$. Let, according to definition 2.1, a and b represent the primitive sums $psa = 2x_1 + 3x_2 + 4x_3 + 5x_4$ and $psb = x_1 + 6x_2 + 8x_3 + 10x_4$, respectively. Then $s = 3x_1 + 4x_2 + 5x_3$ is a cse, i.e. $psa = 2x_1 + s$ and $psb = x_1 + q_2(a, b) \cdot s = x_1 + \lambda \cdot s$. The AC of $\{psa, psb\}$ is thus reduced from $(6, 7)$ to $(4, 5)$. \square

To make these observations operational we use:

Lemma 2.1: Let $B = ||b_{ij}||$ be an (N, M) -integer matrix. Let $\rho_i(\gamma_i)$ denote its i -th row (column). Let ρ_m and γ_k contain only nonzero elements. Let $A = ||a_{ij}||$ be an (N, N) -diagonal matrix such that $a_{ii} = q_k(\rho_m, \rho_i)$. Let $C = ||c_{ij}||$ be an (M, M) -diagonal matrix such that $c_{jj} = b_{mj}$. Let $T = ||t_{ij}||$ be an (N, M) -matrix such that $t_{ij} = p_{j,k}(\rho_m, \rho_i)$.

Then $B = A \cdot T \cdot C$. □

Proof: Let $Z = A \cdot T \cdot C$.

$$\begin{aligned} \text{Then } z_{ij} &= \sum_{n=1}^N a_{in} \cdot \sum_{\ell=1}^M t_{nl} \cdot c_{\ell j} = \sum_{n=1}^N a_{in} \cdot (t_{nj} \cdot c_{jj}) = a_{ii} \cdot t_{ij} \cdot c_{jj} \\ &= q_k(p_m, p_i) \cdot p_{j,k}(p_m, p_i) \cdot b_{mj} = b_{ij}. \end{aligned}$$
□

Hence lemma 2.1 expresses a correspondence between the b_{ij} and the t_{ij} . Obviously $t_{rc} = 1$ iff:

1- $r=m$, $c \in \{1, \dots, M\}$, since $t_{mc} = p_{c,k}(p_m, p_m) = 1$.

2- $c=k$, $r \in \{1, \dots, N\}$, since $t_{rk} = p_{k,k}(p_m, p_r) = 1$.

3- $p_{c,k}(p_m, p_r) = 1$, i.e. $\det \begin{bmatrix} b_{mk} & b_{mc} \\ b_{rk} & b_{rc} \end{bmatrix} = 0$.

$$B = \begin{bmatrix} & & & & & \\ & \cdot & \cdot & & & \\ & \cdot & b_{mk} & b_{mc} & \cdots & p_m \\ & & \cdot & \cdot & & \\ & \cdot & b_{rk} & b_{rc} & \cdots & p_r \\ & & \cdot & \cdot & & \\ & & \cdot & \cdot & & \end{bmatrix}$$

This suggests to associate an incidence matrix I with B , by considering T as a sum matrix $T = I + U$, such that

1- $i_{rc} = 1$, $u_{rc} = 0$ if $t_{rc} = 1$.

$$I = \begin{bmatrix} & \gamma_k & \gamma_c & & & & \\ & 1 & . & & & & \\ & 1 & . & & & & \\ & 1 & 1 & 1 & 1 & 1 & 1 & 1 & p_m \\ & 1 & . & & & & & & \\ & \cdot & 1 & . & 1 & \cdots & & & p_r \\ & 1 & . & & & & & & \end{bmatrix}$$

I defines where fragments of p_i 's, $i \neq m$, and γ_i 's, $i \neq k$, are located which are proportional to corresponding fragments of the nonzero row p_m and the nonzero column γ_k , respectively. Once such an I is found Breuer's algorithm can again be applied on I . If a cse s is found it is a primitive sum defined by (a subset of) p_m . Its multiplicity mf_s in the primitive sum, defined by p_i , is given by $q_k(p_m, p_i) = b_{ik}/b_{mk}$. However $B\sigma_i$ cannot be identified with B , since we have no guarantee that $B\sigma_i$ has a row and column, exclusively formed by nonzero integers. But since we employ possible sparseness, by chaining only nonzero elements together in interconnected row- and columnlists, this problem is superficial. The total weights are used to select a (sub)-matrix B of $B\sigma_i$, and thus also to obtain I , in the following way:

1- γ_k is formed by the nonzero elements of that column of $B\sigma_i$, for which holds that:

1.1. It belongs to the set of columns with a maximal total weight.

1.2. It is the left most column in this set with a maximal weightfactor.

2- p_m is formed by the nonzero elements of that row of $B\sigma_i$, for which holds that

2.1. It belongs to the set of rows sharing a nonzero elements with γ_k .

2.2. The additive weights of the other elements of this set are at most equal to its additive weight.

2.3. Its index is minimal with respect to these constraints.

Example 2.3. To illustrate the overall selection mechanism we employ an example given

by Morgenstern ([14], p. 187):

$$\text{Given } B\sigma_i = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 2 & 2 & 3 \\ 2 & 3 & 4 & 5 \\ 4 & 6 & 0 & 0 \\ 1 & 6 & 8 & 10 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 3 + 2\phi \\ 3 + 3\phi \\ 4 + 4\phi \\ 2 \\ 3 + 3\phi \end{bmatrix} = \alpha\sigma_i$$

$$\delta\sigma_i = [3 + 2\phi, 5 + 4\phi, 5 + 4\phi, 5 + 4\phi]$$

We find $k=2, m=4$ leading to

$$B = \begin{bmatrix} 0 & 1 & 2 & 2 \\ 0 & 2 & 2 & 3 \\ 2 & 3 & 4 & 5 \\ 4 & 6 & 0 & 0 \\ 1 & 6 & 8 & 10 \end{bmatrix}, q_2(p_4, p_i) = \begin{bmatrix} 1/3 \\ 2/3 \\ 1 \\ 2 \\ 2 \end{bmatrix} \text{ and } I = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

This results in the cse $3x_2 + 4x_3 + 5x_4$. \square

Although a detailed description of the required modifications of the function $C\sigma$ is inappropriate in this context, some further indications are worth mentioning:

- 1- One of the parents of a cse is defined by p_m . All its parent descriptions share a nonzero element with γ_k . Hence we can slightly modify Breuer's algorithm to allow operating on a reduced version IR of I , obtained by deleting its " m -th" row and " k -th"-column. All rows and columns of IR, exclusively formed by zero's can also be neglected. So in practice, when referring to example 2.3, we use $IR = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ instead of I .
- 2- If a cse s is found, the function $U\sigma$ is applied to produce $B\sigma_{i+1}$ from $B\sigma_i$. The latter matrix is extended with a column, to store the mf_i , and with a row, defining s . The choice of p_m, γ_k can be unlucky because the mf_i might be rational fractions, as can be verified by interchanging p_4 and p_6 in example 2.3. However this can easily be repaired, as indicated by:

Lemma 2.2: Let the integer matrix $C = \begin{bmatrix} a_1 \dots a_m \\ b_1 \dots b_m \end{bmatrix}$ have rank 1 and let $g = \gcd(b_1, \dots, b_m)$. If $t/n = a_1/b_1 \notin \mathbb{Z}$ then $n|g$. \square

Proof: Straightforward. \square

Corollary: Let g_m be the gcd of the $|J|$ elements of p_m needed to generate s .

If $g_m > 1$ then the cofactors of g_m can be used to define s and the mf_i are replaced by the integers $g_m \cdot mf_i$. \square

- 3- Morgenstern [14] discussed similar problems, albeit in a nondeterministic way. His analysis - leading to a theoretical complexity measure for an optimal algorithm to compute a given set of primitive sums - is based on reducing the number of additions, with emphasis on discrete Fourier transforms. He showed that a less restrictive

tive use of linear dependencies can be profitable. His example 2.3, for instance, shows the relation $(1,2,2) + (2,2,3) = (3,4,5)$. However we accept eventually less optimal results, since we want to employ fast heuristics. Our strategy embodies potential failures, as also indicated by example 2.3. Once $B\sigma_{i+1}$ is made, the selection of ρ_m, γ_k again results in ρ_4, γ_2 . To avoid such disappointments such a column is disregarded during a second try.

2.2. A pragmatic treatment of a set of specific addition chain problems.

Let us now turn our attention to the question how to modify $T\pi = U\pi.C\pi.E\pi$. We use $D\pi_i = (B\pi_i, \alpha\pi_i, \delta\pi_i)$, $i \geq 0$. According to definition 2.1 the $(N\pi_i, M_i)$ -nonnegative integer matrix $B\pi_i = ||b_{ij}||$ defines a set of $N\pi_i$ monomials in M_i indeterminates x_1, \dots, x_{M_i} . Again the interpretation of the weights, $\alpha\pi_i$ and $\delta\pi_i$, is important. Similar problems received considerable attention in literature. Recently Pippenger [15] discussed bounds for $L(p, q, N)$, being the number of multiplications required to compute p monomials in q indeterminates, if N is the maximum of the integer powers occurring in the monomials, i.e. in terms of our problem $\max_{l,j} (b_{lj})$. Yao [22] gave an algorithm for the $L(p, 1, N)$ problem, which is a special case of $L(1, 1, N)$, the wellknown addition chain problem (see Knuth [13], 441-466). All these algorithms have one common aspect: p, q and N are constants. Therefore our problem is only seemingly identical to the $L(p, q, N)$ question, studied by Pippenger. We operate on a $B\pi_i$ matrix, resulting from an iterative CSS, using previous versions of both $B\sigma_i$ and $B\pi_i$, $i \geq 0$. This enforced, dynamic behaviour of our "database", imposed by "noise" conquering, demands for an extendable approach. A column γ_k of $B\pi_i$ defines a set of integer powers of x_k , the corresponding indeterminate. This set can contain identical powers. Fig.5.a, for instance, shows the d -column of $B\pi_0$, fig.4. As shown in fig.5.b this set of powers can be viewed as a set of products, such that the factors d, d^2 are used to construct d, d^2, d^3 and d^5 . This product matrix can easily be interpreted as an incidence matrix B_k , associated with γ_k . Its construction is defined by algorithm 2.1. Then the column weight of γ_k have to be replaced by a vector v_k . In the previous subsection total weights were used, being pairs of an additive and a multiplicative weight. The former reflects the x_k -density in $B\sigma_i$; the latter its arithmetical impact. Similarly each element of v_k is defined as a pair $v_{kl} = (d_{kl}, p_{kl})$, if d_{kl} denotes the number of occurrences of $x_k^{p_{kl}}$ in the given $N\pi_i$ monomials. The weights associated with the rows of B_k , denote the number of factors produced with algorithm 2.1. The sparse representation of γ_k allows to consider it as a not yet partly ordered set of d positive integers. Let us denote them by c_m , $m = 1, \dots, d$. Obviously $m_k \leq c_m \leq M_k$. This suffices to formulate

Algorithm 2.1 [Construction of a $(d, \min(M_k - m_k + 1, d))$ incidence matrix $B_k = ||b_{m,l}||$.

$$\gamma_k = \begin{bmatrix} 2 \\ 0 \\ 5 \\ 3 \\ 5 \\ 5 \\ 5 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} d^2 \\ d^5 \\ d^3 \\ d^5 \\ d^5 \\ d^5 \\ d \\ d \end{bmatrix} \approx \begin{bmatrix} d & d \\ d & d & d & d^2 \\ d & d & d \\ d & d & d & d^2 \\ d & d & d & d^2 \\ d & d & d & d^2 \\ d \end{bmatrix} \rightarrow B_k = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 \end{bmatrix} = \alpha \pi_k$$

(a)

(b)

(c)

$$v_k = \{(6,1), (5,1), (4,1), (3,2)\}$$

```

d2 = d*d
d3 = d*d2
d5 = d2*d3

```

`[(1, 1), (1, 2), (1, 3), (3, 5)]`

Figure 5

from γ_k , such that $b_{ml} = 1$ if $x_k \uparrow p_{kl} \mid x_k \uparrow c_m$ and $b_{ml} = 0$ otherwise].
 B1. [Initialize Set $\{f_{kl}=0 : \max_{l \in M} f_{kl} < \min_{l \in m} f_{kl}\}$.]

B1 [Initialize] Set $\{l \leftarrow 0; \max \leftarrow M_k; \min \leftarrow m_k\}$.

B2 [Termination] If $\max=0$ then return.

B3 [Extend B_k] Set $\{l \leftarrow l+1; p_{kl} \leftarrow \min\}$. For all $m \in \{1, \dots, d\}$ set $b_{ml} \leftarrow 0$;

For all $m \in \{1, \dots, d\}$ such that $c_m \neq 0$ set $\{c_m \leftarrow c_m - \min; b_m \leftarrow 1\}$.

B4 [Recycle] Set $\max \leftarrow \max - \min$; If $\max > 0$ then set $\min \leftarrow \min - \min(c_n > 0)$; Goto B2.

These "small" B-matrices allow two obvious approaches:

- Start with a Breuer search on each B_k separately. Fig.5.d shows the result of such an operation, leading to cse's d2, d3 and d5, which can be found by applying a Yao-like algorithm on the set $\{p_{k_1}, \dots, p_{k_4}\} = \{1, 2, 3, 5\}$. After these M_i operations the remaining non redundant columns can be combined together in one "big" incidence-matrix for further cse-searches. But then the d-, d2- and d3-columns, given in fig 5.d are neglected.
 - Combine the "small" B_k -matrices directly in one "big"-matrix before a Breuer search is started. This allows to combine intermediate steps, required to compute d2, d3 and d5 for instance, with intermediate steps needed to compute other power configurations.

Although both approaches are extendable, the second has obvious advantages. To illustrate this, the result of both techniques, applied on the $B\pi_0$ -matrix, given in example 2.1, is shown in fig.6.

$c2 = c*c$	$c2d = c2*d$	$gc2d5 = g*c2d5$	$c2 = c*c$
$d2 = d*d$	$c2d2 = c2*d2$		$d2 = d*d$
$d3 = d*d2$	$c2d3 = c2*d3$		$c2d = c2*d$
$d5 = d2*d3$	$gc2d5 = g*c2d5$		$c2d2 = c2d*d$
$c2d5 = c2*d5$			$c2d3 = c2d2*d$
$bc2d5 = b*c2d5$			$c2d5 = c2d3*d2$
			$bc2d5 = b*c2d5$
cse's	No cse's		cse's
Approach(1)	Figure 6		Approach(2)

The Breuer searches in the "big" matrix do not require special weight factor techniques and are based on reduction of multiplications, since powering is considered as repeated multiplication. However, once $D\pi_l$ is found the powersets of the v_k -vectors can be subjected to a Yao-like technique.

2.3 "Noise"-conquering aspects

The functions $M\omega_k$, regulating the "noise"-conquering aspects of $T_\beta = M\pi \cdot T_0 \cdot M\sigma \cdot T\pi$, also require some modifications. If a σ -type cse $s\sigma$ can be transferred to the product matrix its integer power, given in the $\sigma\sigma$ -part of $H\sigma_i$, $i \geq 0$, and its multiplicity factor have to accompany $s\sigma$. Likewise the coefficient of a π -type cse $s\pi$, given in the $\gamma\pi$ -part of $H\pi_i$ have to be taken into account when migrating $s\pi$ to the sum matrix. The overall result of applying the modified algorithm T on the set E_0 , given in example 2.2, is $E_6 = T^6(E_0) = \{c2=c.c, d2=d.d, p1=c2.d, s1=2a + 3b, p2=p1.d, s2=2g + 5f, p3=p2.d, p4=p2.p3, p5=b.p4, s3=a + p5, p6=s2.s3, e1=(s1 + 3p2 + s2.p6)^3, e2=a + 3f + 2p3.p6.s3, e3=9a + 4g.p4.(g + p1).s1\}$. In figure 7 the process is shortly visualized.

3. Some conclusions

The application of various modifications of Breuer's grow factor algorithm, as indicated in the previous sections, is quite instrumental for optimizing the description of a set of multivariate polynomials with integer coefficients. Essentially these techniques allow to operate in an expression structure preserving way, because we only assume the validity of the commutative law for addition and multiplication.

Σ_i	"Noise" conquering	Π_i	AC_i
0	$s1 = 2a + 3b \rightarrow \{\epsilon\sigma_9=2, mf_9=4\} \rightarrow$	$c2 = c^2, d2 = d^2$	(12, 55)
		$\leftarrow \{\gamma\pi_7=1\} \leftarrow p1 = c^2d$	
1	$s2 = 2g + 5f \rightarrow \{\epsilon\sigma_2=2, mf_2=1,$ $\epsilon\sigma_5=1, mf_5=2\} \rightarrow$	$\leftarrow \{\gamma\pi_1=3\} \leftarrow p2 = c^2d^2$	(11, 40)
2		$p3 = c^2d^3$	(10, 35)
3		$p4 = c^2d^5$	(10, 31)
4	$s3 = a + p5 \rightarrow \{\epsilon\sigma_3=1, mf_3=1,$ $\epsilon\sigma_6=2, mf_6=1\} \rightarrow$	$\leftarrow \{\gamma\pi_3=\gamma\pi_5=1\} \leftarrow p5 = b c^2d^5$	(10, 28)
5		$p6 = (2g + 5f)(a + b c^2d^5)$	(9, 26)
6			(9, 25)

Figure 7.

The given examples emphasize this aspect. Consequently they might be misleading. Dropping the wish to maintain the problem structure completely invites to apply distributivity too. This is easily verified by adding for instance $q_4 = a.c.d + b.c.d + e.c.d$ to the set E_0 of example 1.1. Cook [8], and also Brenner, experimented with the distributive law in the context of computational physics, and using MACSYMA. And indeed, expansion with respect to indeterminates with a certain physical meaning, i.e. playing a weighted role, followed by locally factoring out these indeterminates, can increase compactness. Experiments of Hulshof [11], based on a combined use of both elementary laws and the first version of the code optimizer [10], indicate that such an approach is promising, albeit that further research is required before deciding if one general command, a set of facilities, to be used interactively as part of an expression analysis package [21], or both have to be recommended. Future extensions of the presented algorithms will certainly cover elementary transcendental functions. Such functions can easily be introduced in REDUCE via the kernel concept, which itself generalizes the notion "indeterminate". Once such extensions are available one can start thinking of partial derivative generation and automated error analysis.

References

- [1] Aho, A.V., Johnson, S.C., Ullmann, J.D.: Code generation for expressions with common subexpressions. J. ACM 24, 1, 146-160 (1977).
- [2] Aho, A.V., Ullmann, J.D.: The theory of parsing, translating and compiling, Vol II: Compiling. Englewood Cliffs, N.J.: Prentice Hall (1973).
- [3] Aho, A.V., Ullmann, J.D.: Principles of compiler design. Reading, Mass: Addison Wesley (1979).

- [4] Allen,F.E.: Annotated bibliography of selected papers on program optimization.
Yorktown Heights, N.Y.: IBM Research Report RC 5889(1976).
- [5] Breuer,M.A.: Generation of optimal code for expressions via factorization.
C.ACM 12,6, 333-340 (1969).
- [6] Bruno,J., Sethi,R.: Code generation for a one-register machine. J.ACM 23,3,
502-510 (1976).
- [7] Buchberger,B. Loos,R.: Algebraic simplification. Computer Algebra
(B.Buchberger, G.E.Collins, R.Loos, ed's), Computing Supplementum 4 (1982).
- [8] Cook,Jr.,G.O.: Development of a magnetohydrodynamic code for axisymmetric,
high- β plasmas with complex magnetic fields. Lawrence Livermore Nat. Lab.Cal.:
Ph.D. Thesis (December 1982).
- [9] Gonzales,T., Ja'Ja',J.: Evaluation of arithmetic expressions with algebraic
identities. SIAM J. Comp. 11, 4, 633-662 (1982).
- [10] Hulshof,B.J.A., van Hulzen,J.A., Smit,J.: Code optimization facilities applied
in the NETFORM context.Twente University of Technology,TW-memorandum 368 (1981).
- [11] Hulshof,B.J.A.: private communication.
- [12] Kedem,G.: Automatic differentiation of computer programs. ACM TOMS 6,2,150-165
(1980).
- [13] Knuth,D.E.: The art of computer programming, Vol.2 (second edition).
Reading, Mass.: Addison-Wesley Publ. Comp. (1980).
- [14] Morgenstern,J.: The linear complexity of computation. J.ACM 22,2, 184-194 (1975).
- [15] Pippenger,N.: On the evaluation of powers and monomials. SIAM J. Comp. 9,2,
230-250(1980).
- [16] Rall,L.B.: Automatic differentiation techniques and applications. Springer LNCS
series nr 120. Berlin-Heidelberg-New-York: Springer Verlag (1981).
- [17] Smit,J.: A cancellation free algorithm, with factoring capabilities for the
efficient solution of large, sparse sets of equations. Proceedings SYMSAC '81
(P.S. Wang, ed.), 146-154. New York: ACM (1981).
- [18] Smit,J., van Hulzen,J.A., Hulshof,B.J.A.: NETFORM and code optimizer manual
SIGSAM Bulletin 15,4, 23-32. New York: ACM (1981).
- [19] Smit,J., van Hulzen,J.A.: Symbolic-numeric methods in microwave technology.
Proceedings EUROCAM '82 (J. Calmet, ed.). Springer LNCS series nr. 144, 281-238.
Berlin-Heidelberg-New York: Springer Verlag (1982).
- [20] van Hulzen,J.A.: Breuer's grow factor algorithm in computer algebra.
Proceedings SYMSAC '81 (P.S. Wang, ed.), 100-104. New York: ACM (1981).
- [21] van Hulzen,J.A., Hulshof,B.J.A.: An expression analysis package for REDUCE.
SIGSAM Bulletin 16,4. New York: ACM (1982).
- [22] Yao,A.C.C.: On the evaluation of powers. SIAM J. Comp. 5,1, 100-103 (1976).

EUROCAL '83

CONFERENCE PROGRAM

MONDAY, MARCH 28

9.00 - 10.40 SESSION 1: Algorithms I - Miscellaneous

Chaired by J.A. van Hulzen, Technische Hogeschool Twente, the Netherlands

INTEGRATION - WHAT DO WE WANT FROM THE THEORY? (Invited Paper)

J.H. Davenport, IMAG, Grenoble, France

THE EUCLIDEAN ALGORITHM FOR GAUSSIAN INTEGERS

H. Rolletschek, University of Delaware, U.S.A.

MULTI POLYNOMIAL REMAINDER SEQUENCE AND ITS APPLICATION TO LINEAR DIOPHANTINE EQUATIONS

A. Furukawa, Tokyo Metr. University, Japan, & T. Sasaki, IPCR, Saitama, Japan

11.10 - 12.50 SESSION 2: Applications - Miscellaneous

Chaired by J.P. Fitch, University of Bath, England

TOWARDS MECHANICAL SOLUTION OF THE KAHAN ELLIPSE PROBLEM I

D.S. Arnon & S.F. Smith, Purdue University, U.S.A.

AUTOMATICALLY DETERMINING SYMMETRIES OF ORDINARY DIFFERENTIAL EQUATIONS

F. Schwarz, Universität Kaiserslautern, Germany-West

ALGEBRAIC COMPUTATION OF THE STATISTICS OF THE SOLUTION OF SOME NONLINEAR STOCHASTIC DIFFERENTIAL EQUATIONS

F. Lamnabhi-Lagarrigue, LSE, Gif sur Yvette, France & M. Lamnabhi, LEPA, Limeil-Brevannes, France

CHARACTERIZATION OF A LINEAR DIFFERENTIAL SYSTEM WITH A REGULAR SINGULARITY

A. Hilali, IMAG, Grenoble, France

14.00 - 15.10 SESSION 3: Systems and Language Features I

Chaired by K.A. Bahr, GMD, Darmstadt, Germany-West

THE BATH CONCURRENT LISP MACHINE

J. Marti, Rand Corporation, Santa Monica, U.S.A. & J.P. Fitch, University of Bath, England

L-NETWORKS: MULTI PROCESSOR SYSTEMS FOR PARALLEL ALGORITHMS IN COMPUTER ALGEBRA

B. Buchberger, Johannes Kepler Universität, Linz, Austria

THE ECOLOGY OF LISP OR THE CASE FOR THE PRESERVATION OF THE ENVIRONMENT

J.A. Padget, University of Bath, England

15.45 - 16.45 SESSION 4: Differential Equations I

Chaired by J. Åman, Universitet Stockholm, Sweden

DOCUMENTATION OF SMALL PROGRAMS, KNOW-HOW OF REDUCE AND THE COMPUTER

P.K.H. Gragert, Technische Hogeschool Twente, the Netherlands

FLAVOURS OF NON COMMUTATIVITY IN STENSOR

L. Hörnfeldt, Universitet Stockholm, Sweden

INTERACTIVE SOLUTION OF A NONLINEAR OSCILLATOR PROBLEM BY PERTURBATION METHODS

R. Caboz, Université de Pau, France, P. Kupsch & D. Brune, Universität Karlsruhe, Germany-West

TUESDAY, MARCH 29

9.00 - 10.30 SESSION 5 : Systems and Language Features II

Chaired by A.C. Hearn, Rand Corporation, Santa Monica, U.S.A.

THE DESIGN OF MAPLE : A COMPACT, PORTABLE AND POWERFUL COMPUTER ALGEBRA SYSTEM
(Invited Paper)

B.W. Char, K.D. Geddes, W.M. Gentleman & G.H. Gonnet, University of Waterloo, Canada

LISP COMPILED AS PROVABLE SEMANTICS PRESERVING PROGRAM
TRANSFORMATION

H. Stoyan, Universität Erlangen, Germany-West

IMPLEMENTING REDUCE ON A MICRO COMPUTER

J.P. Fitch, University of Bath, England

11.00 - 12.40 SESSION 6: Algorithms II - Polynomial Ideal Bases

Chaired by M. Pohst, Universität Düsseldorf, Germany-West

A NOTE ON THE COMPLEXITY OF CONSTRUCTING GRÖBNER-BASES

B. Buchberger, Johannes Kepler Universität, Linz, Austria

GRÖBNER-BASES, GAUSSIAN ELIMINATION AND RESOLUTION OF SYSTEMS OF ALGEBRAIC EQUATIONS

D. Lazard, Université de Poitiers, France

THE COMPUTATION OF THE HILBERT FUNCTION

F. Mora, Università di Genova, Italy & H.M. Möller, FernUniversität, Hagen, Germany-West

AN ALGORITHM FOR CONSTRUCTING DETACHING BASES IN THE RING OF POLYNOMIALS OVER A FIELD

F. Winkler, Johannes Kepler Universität, Linz, Austria

14.00 - 15.15 SESSION 7 : Future Trends in Computer Algebra (Panel Discussion)

Chaired by R. Loos, Universität Karlsruhe, Germany-West

Secretary : J.H. Davenport, IMAG, Grenoble, France

Panellists: B. Buchberger, Johannes Kepler Universität, Linz, Austria

I. Cohen, Universitet Stockholm, Sweden

A.C. Norman, University of Cambridge, England

15.45 - 17.00 SESSION 8 : Business Session (including Status Reports)

Chaired by J.A. van Hulzen, Technische Hogeschool Twente, the Netherlands

Agenda : - SAME, Its Organization and Management

- Future Meetings and Publication Policy

- Status Reports:

PSL and REDUCE

A.C. Hearn, Rand Corporation, Santa Monica, U.S.A.

VAXIMA

R.D. Fateman, University of California, Berkeley, U.S.A.

20.00 Banquet

WELCOME ADDRESS

K. Barker, Kingston Polytechnic, London, England

BANQUET ADDRESS : A TRANSITIONAL PERIOD FOR SAM

P.S. Wang, Kent State University, U.S.A.

WEDNESDAY, MARCH 30

9.00 - 10.50 SESSION 9 : Algorithms III - Computational Number Theory

Chaired by J.H. Davenport, IMAG, Grenoble, France

ON THE PROBLEM OF BEHÂ EDDÎN 'AMÛLÎ AND THE COMPUTATION OF HEIGHT FUNCTIONS
(Invited Paper)

H.G. Zimmer, Universität des Saarlandes, Germany-West

A PROCEDURE FOR DETERMINING ALGEBRAIC INTEGERS OF GIVEN NORM

U. Fincke & M. Pohst, Universität Düsseldorf, Germany-West

COMPUTATION OF INTEGRAL SOLUTIONS OF A SPECIAL TYPE OF SYSTEMS OF QUADRATIC
EQUATIONS

M. Pohst, Universität Düsseldorf, Germany-West

A GENERALISATION OF VORONOI'S ALGORITHM TO ALL ALGEBRAIC NUMBERFIELDS OF
UNIT RANK 1 AND 2

J. Buchmann, Universität Köln, Germany-West

11.20 - 13.00 SESSION 10 : Algorithms IV - Factorization

Chaired by M. Mignotte, Université de Strasbourg, France

FACTORIZATION OF SPARSE POLYNOMIALS

J.H. Davenport, IMAG, Grenoble, France

EARLY DETECTION OF TRUE FACTORS IN UNIVARIATE POLYNOMIAL FACTORIZATION

P.S. Wang, Kent State University, U.S.A.

ON THE COMPLEXITY OF FINDING SHORT VECTORS IN INTEGER LATTICES

E. Kaltofen, University of Toronto, Canada

FACTORING POLYNOMIALS OVER ALGEBRAIC NUMBER FIELDS

A.K. Lenstra, Mathematisch Centrum, Amsterdam, The Netherlands

14.00 - 15.00 SESSION 11 : Differential Equations II

Chaired by I. Frick, Universitet Stockholm, Sweden

A DIFFERENTIAL FORM PROGRAM IN REDUCE

E. Schrufer, Universität Bonn, Germany-West

INTEGRATION OF OVERRDETERMINED SYSTEMS OF LINEAR PARTIAL DIFFERENTIAL EQUATIONS

P.H.M. Kersten, Technische Hogeschool Twente, the Netherlands

THE RICCIER-JANET THEORY AND ITS APPLICATION TO NONLINEAR EVOLUTION EQUATIONS

F. Schwarz, Universität Kaiserslautern, Germany-West

15.30 - 17.20 SESSION 12 : System Oriented Applications

Chaired by B. Buchberger, Johannes Kepler Universität, Linz, Austria

THE CONSTRUCTION OF A COMPLETE MINIMAL SET OF CONTEXTUAL NORMAL FORMS

M. Rice, CRI, Nancy, France

A KNOWLEDGE-BASED APPROACH TO USER-FRIENDLINESS IN SYMBOLIC COMPUTING

F. Gardin & J.A. Campbell, University of Exeter, England

COMPUTER ALGEBRA AND VLSI, PROSPECTS FOR CROSS FERTILIZATION

J. Smit, Technische Hogeschool Twente, the Netherlands

CODE OPTIMIZATION OF MULTIVARIATE POLYNOMIAL SCHEMES: A PRAGMATIC APPROACH

J.A. van Hulzen, Technische Hogeschool Twente, the Netherlands