THE SYMBOLIC-NUMERIC INTERFACE

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

Introduction

Since I believe the symbolic-numeric interface to be important for many potential application fields, I decided to limit my response to Bruno Buchberger's invitation to express my views to this aspect of computer algebra. Not only to try to avoid duplication, but also to emphasize that it only seemingly suggests a routine use of computer algebra. Another reason is that it allows to stress the importance of education, documentation and management.

Management and Education

The possible impact of symbolic and algebraic computation on research in other disciplines is only partly related to the present level of sophistication. At least equally important are management and education, or stating it slightly different, mutual interest.

The strength of FORTRAN is not the language itself, but its standardization and portability, which have lead to extensive libraries and impressive application packages. There exist a rich diversity of symbolic and algebraic manipulation systems. However, the level of acceptance FORTRAN reached, is missing. Only a few of these systems have been or are "widely" used. Examples are MACSYMA, via the ARPA net, and REDUCE, due to its portability. But let us not confuse these systems with their user-level top language. Libraries are either built in or system dependent. More recent systems, like SMP and MAPLE, explicitly employ a library concept, i.e. special purpose packages grouped around a fast kernel. Most manuals are somewhat misleading. They please innocent noval users with simple commands, which however hide algorithms with a possible exponential behaviour. Explicit information about techniques and strategies for extending systems, based on specific expertise of users, is often missing. And making such extensions (or library templates) do not demand experience with FORTRAN, but requires knowledge of (some dialect of) LISP or C.

Conclusions are obvious. The nature of our software creates the need to find a communis opinio about implementation directed publication standards for algorithms and their behaviour. To allow other disciplines to enjoy the results of our efforts demands other manuals and additonal education. Transparent documentation of system concepts and structure is our obligation. A working knowledge of LISP like languages is a minimal need for them.

The symbolic-numeric interface

Systems like TRIGMAN, SCHOONSCHIP and SHEEP largely contributed to "our" successes in celestial mechanics, high energy physics and general relativity, respectively. It is obvious why. These systems are tailored for a specific use and the underlying mathematics is well understood and fairly straightforward. Here computer algebra is accepted and routinely used. But when one wants to try different things with SCHOONSCHIP user defined simplification rules, as to manage the new structures, have to be made. This is in a sense also the concept behind SMP. Polynomial algebra is well

understood. Recursive representations are often preferred, although experiments of A.C. Norman showed that an unordered internal representation does not dramatically change the overall performance of a system. Using non rational labels in such recursive representations easily allows to extend the class of mathematical expressions to be handled. But it can impose some problems. Brown's ALTRAN, not using this recursive approach, is known to be very efficient, due to his adagium "Factors Factors Everywhere with Opportunities to Share". Similar arguments can hold for Engeli's SYMBAL. Why recalling these well known facts?

Among the mathematical structures and transformations which are characteristic for systems theory, control theory, optimization, robotics, finite element analysis, etc, certainly ought to be mentioned matrices and their inverses, such as Jacobians and Hessians, demanding differentiation. Repeated integration is also worth stipulating. So Brown's adagium is again applicable. But how well is that possible with recursive representations. When thinking of attempts to use FORTRAN to generate FORTRAN code Warner's partial derivative generator and the Augment compiler of Kedem, e.a., are worth mentioning. Certainly the performance of the latter program is largely influenced by simplification problems (see L.B. Rall, LNCS series nr 120, Springer-Verlag (1981)). Both examples suggest not only a hesitation to employ computer algebra, but also that adequate tools for numerical code generation are important. Such tools ought to consist of code optimization, program structure generation and a priori error analysis leading to guaranteed precision. These three aspects are strongly interrelated. Code optimization largely depends on simplification viewed as pre-optimization. However automatic simplification is often too powerful, because it hinders lazy evaluation, i.e. a delay of algebraic processing to avoid all the factors, which can be shared, to be routinely produced too often. Expressions are too easily considered as entities, unrelated to each other. My own experience with post-optimization, i.e. with rearranging a set of output expressions as to avoid duplication of common subexpressions, learned me that the performance of such compression techniques can be largely improved when trying to employ Brown's adagium. The generation of code alone is certainly not sufficient. The code blocks ought to be logically connected to get programs. In view of the possible structure of such programs, nested loops and the like, partial evaluation techniques ought to be combined with code optimization. Last but not least the thus produced programs ought to be subjected to an a priori error analysis. I expect such facilities to become available during the coming years, assuming we are willing to reconsider simplification by concentrating on lazy evaluation. These short remarks are as well applicable for parallel computations. Then code optimization has to be considered as a distribution technique instead of a reduction strategy.

Conclusions

I tried to argue that a less traditional education in computer science in other disciplines is vital for a recognition of the potentials of computer algebra, assuming we are improving the symbolic-numeric interface. I limited myself to this aspect, since Fateman's view's, certainly on workstations, graphics and expertice, cover my opinions as well.