# Investigating some Embedded Verification Techniques for Computer Algebra Systems

U. Martin    D. Shand*

Computer Science Division, School of Mathematical and Computational Sciences,
University of St Andrews, St Andrews, FIFE, KY16 9SS, SCOTLAND.
`E-mail: {um, ddshand}@dcs.st-and.ac.uk`

May 30, 1997

**Abstract**

This paper reports some preliminary ideas on a collaborative project between St Andrews University in the UK and NAG Ltd. The project aims to use embedded verification techniques to improve the reliability and mathematical soundness of computer algebra systems. We give some history of attempts to integrate computer algebra systems and automated theorem provers and discuss possible advantages and disadvantages of these approaches. We also discuss some possible case studies.

## 1 Introduction

A goal of much research in computational logic [CL] is to produce mathematical assistants. A plethora of advanced systems and significant theoretical results have been produced over the past 30 years. Computer algebra systems [CAS] have become standard tools, but few non-specialists understand or use automated theorem provers [ATP]. This paper will describe the early stages of a project in which we want to investigate how to enhance CAS using the techniques of CL. This is intended to give users of CAS the advantages of the methods of ATP while shielding them from their technicalities. This is in line with recent work on embedded verification techniques [EVT] in domains such as databases [9].

## 2 Computer algebra systems

After pioneering work in the 1960s CAS such as Mathematica and Maple have become mainstream commercial products: everyday tools not only for researchers but also for engineers and scientists. Their use in live applications is increasing: for example Aerospatiale

*Supported by EPSRC grant GR/K75767

[30] use a Maple-based system for motion planning in satellite control, and they are widely used in school and university science and mathematics teaching. Many mathematicians use CAS for elaborate computations [4] or for experimentation in formulating new results which they may then prove 'by hand'[24].

Advanced CAS incorporate modern language designs. AXIOM [21] is a strongly typed computer algebra system developed by NAG Ltd. User and system libraries are written in the Aldor language which supports a hierarchy of built-in parameterised types for mathematical objects such as rings, fields polynomials etc. Aldor is interpreted in the AXIOM kernel which provides basic routines such as simplification and evaluation: code developed in Aldor may be compiled to C for export to other products.

## 3 Automated theorem provers

Computational logic has a similar long history. Many advanced ATP (HOL, Isabelle, LP, PVS, NQ-THM,OTTER, NuPrl etc) have been developed, addressing both foundational logical issues and practical ones such as search and proof management. They have been particularly successful in specialised domains such as hardware verification [32], but despite initial optimism have failed to become widely used by non-specialists.

Notwithstanding, there have been a number of recent papers in which new theorems in various areas of mathematics have been proved with the aid of machine assisted theorem provers. For instance, McCune [28] shows that the Otter system can identify and verify new single axiomatisations for groups and in [29] proves new results about cubic curves. Fujita *et al* [15] show that model checking techniques can be used in an area of combinatorics. There has also been interest shown in formalising complex theorems in mathematics. Linton *et al* [26], formalise and verify a theorem about the wreath product of two infinite cyclic groups. These advances have been made possible by both the mathematical insight of the user and the ability of the systems to guide the search.

In [25, 26], ATP based on rewriting and induction, such as LP, have been shown to be particularly valuable to algebraists, and these ideas have been incorporated in standard group theory software. Recent significant advances include constrained rewriting [2] (based on [27]), induction [8, 3] and the advent of more flexible and powerful ATP such as LP[16] and PVS[31].

## 4 Combining CAS and ATP

A natural extension to CAS is to demand some theorem proving capacity of the CAS, for example support for simple induction schema. CAS have become more complicated, providing languages, graphics, programming environments and diverse sophisticated algorithms for integration, factorisation and so on, to meet the needs of a variety of users, many not expert in mathematics. All the usual software engineering issues arise: modularity, re-use, interworking, HCI etc. An issue peculiar to CAS that of soundness. A CAS may be able to compute an answer on a large class of inputs, be sound on only a subclass of those inputs and be able to check soundness easily on a smaller subclass still. Expert mathematicians may take this in their stride: others will be bemused or misled. For a simple example consider a deeply nested division by an expression which might be equal to zero. What value should the CAS return, what explanation might it provide to the user of what it has done and what warnings should it flag?

How can advanced CAS like AXIOM incorporate recent advances in verification so as to produce a more powerful, reliable and flexible mathematical environment and address the concerns raised above? One approach is the development of new CAS within theorem provers, following the pattern of, for example, analysis in AUTOMATH [12], computer algebra algorithms in NuPrl [20] and constructive reals in LEGO [22]. It can be seen that constructing moderately complex computer algebra algorithms in theorem provers is a difficult task. In [20] some first steps at a constructive verification of the Gröbner basis algorithm are made. Extending this to, for instance, a constructive proof of the termination of the Gröbner basis algorithm would seem to be a very demanding task.

Another approach to integrating ATP and CAS is to use the CAS as an oracle. Existing CAS may be used in such developments: their output provides statements of lemmas to guide ATP to sound proofs. Examples include incidence geometry using Mathematica [35], Maple as an oracle to HOL for the reals [17] and the Isabelle/Maple interface [18].

The *Theorema* project [5] takes a more integrated approach to combining systems. The aims of the project are to integrate a number of distinct decision procedures and provers with the Mathematica CAS. It is also intended to produce human readable proofs from the *Theorema* system. First order reasoning has been added to REDUCE [34] for the REDLOG system [13]. This focuses on methods which are particularly applicable to polynomials, such as simplification, parametric linear optimization and quantifier elimination.

However even if a full verification were feasible it is not what users usually want: they like the looseness of CAS and want to control what verification, if any, is done. Related work within CAS is spotty thus far: Maple has a restricted formulation of propositional logic [19]. Mathematica allows computation using rewrite rules, so for example elaborate power series expressions can be manipulated to derive impressive results [10], although there are no checks that application of the rules is sound. Other projects are developing frameworks to support 'mathematical systems' including CAS and ATP with emphasis on architectural issues [18] and common interchange formats [1]. The greatest successes have been in areas of geometric reasoning which can be directly translated into Gröbner basis problems [11]. There has also been a great deal of success with more traditional techniques such as Wu's method.

The above approaches are characterised by a master / slave relationship. An alternative to this can be described as ‘light formal methods for CAS’ [14], which aims to obtain consistency of typing, avoid degenerate cases, and provide precise analysis of conditions at the specification stage without the overhead of full verification, and in particular avoids ‘bolting’ an ATP to a CAS. They rely on annotations in a behavioural interface specification language [BISL] which can be manipulated using CL: the annotations form a partial specification which also aids reuse, debugging and checking for soundness. The parameterised domain structure of Axiom is particularly well-suited for this kind of approach because it forces the user to be explicit about all the assumptions he or she is making about the mathematical properties of the objects which are being manipulated. We will be using the Axiom system in the project.

# 5 Some EVT for CAS

In this section we will introduce possible case studies for EVT in CAS. The list here is certainly not exhaustive, but we wish to show that there are a number of different user communities who can be helped by the techniques we propose to use.

## 5.1 Flagging Degeneracies

Computer algebra systems have many different user communities. Expert users may be expected to know when there could be a division by an expression which might be equal to zero, or at what points a function is not continuous. Non experts in a particular field may be less adept at spotting these sorts of degeneracies. Thus for some users it may be useful to have these sorts of degeneracies flagged in some way.

This fits in with the work at St Andrews of Dunstan and Kelsey [14]. The code of Axiom is annotated using a BISL, in this case Larch / Aldor, to flag degeneracies in the style of a verification condition generator. If a degeneracy occurs, a proof assistant can be used to investigate properties of this degeneracy. This makes for safer and sounder use of computer algebra systems for non-expert users.

A simple example of a degeneracy can be seen in [33]. Suppose we have a function $integrate(x^n, x)$, where we wish to integrate $x^n$ with respect to $x$. Maple and Mathematica return the answer $\frac{x^{n+1}}{n+1}$. As there is no type associated with $n$, we cannot rule out the possibility that $n = -1$, in which case the integral returned by Maple and Mathematica is not correct. Of course, this is a straightforward example which mathematicians would recognise easily, although one could imagine more subtle degeneracies of this form which would not be so readily identified. It can be noted here that on the example given Axiom recognises that the case $n = -1$ can occur and returns the correct answer.

## 5.2 Induction

There has been a lot of recent interest in inductive theorem proving [23]. We intend to investigate extending CAS with some inductive theorem proving technology. This would enable a user to prove results about sequences, power series etc. One issue which must first be resolved is to whether it would be better to build an inductive theorem prover on top of a CAS or to use an existing inductive theorem prover.

The first option is used in systems such as THEOREMA [5], and has an advantage that it would shield the user from the complicated ATP technology, and fits in more with the light formal methods approach to theorem proving in which we are interested. Substantial development of the language of the CAS would also be needed to make use of the heuristics and techniques which are currently available in the CL world.

The second option allows the user to choose to work in his or her favourite inductive theorem prover. This means that it would be possible to use the technology and heuristics which have been built in the CL community for inductive theorem proving. For instance, *rippling* [8] has been developed as an effective heuristic for induction, and is used in mainstream theorem provers such as Oyster-Clam [7] and HOL. The major disadvantage to using these systems is that many people find them intimidating, and hard to learn. Historically, very few people from outside the CL world use theorem provers.

## 5.3 Proof or Calculation Planning

It would be advantageous to be able to automatically generate heuristics and be able to analyse the success (and failure) of particular techniques in CAS. This is closely related to ideas of method selection in CAS. Out of a multitude of functions which purport to do similar things, for a non-expert CAS user it is often difficult to choose the one which has the best chance of succeeding, and to know which heuristics would be best to tackle a particular problem. For example, knowing a suitable substitution for an integration problem is often helpful.

Automatic generation of these sorts of heuristics has been an area of particular recent interest in CL and artificial intelligence. Most notably, the DREAM group at Edinburgh University has developed a theory of 'proof plans'. Bundy [6] informally defines proof plans in the following way:

> *A proof plan captures the common patterns of reasoning in a family of similar proofs and is used to guide the search for new proofs in this family.*

The proof plans developed have been mainly in the area of inductive reasoning and their development involved looking at large numbers of proofs and identifying common patterns of reasoning. These are represented computationally. Proof plans introduce a notion of meta-level control into search. They are implemented through:

- *tactics*, programs from LCF-style theorem provers which automate tedious parts of the proof while guaranteeing soundness
- *methods* which specify tactics
- *critics* analyse failed proofs and suggest 'patches' to partial proofs

It would be interesting to find whether these ideas can be extended to 'calculation' planning in CAS. It seems likely that these sorts of methods would work: there are often standard 'tricks' which make certain problems computationally simpler, e.g. finding a 'good' substitution in an integration problem, so that the integral will then occur in a standard look-up table.

## 6 Acknowledgements

Thanks to Mike Dewar of Nag Ltd for several very helpful suggestions.

## References

[1] J Abbott et al, Objectives of OpenMath, RIACA TU Eindhoven,Technical Report 12 (1996).

[2] L Bachmair and H Ganzinger, Rewrite based equational theorem proving with selection and simplification, Journal of Logic and Computation 4 (1994) 1-31

[3] A Bouhoula, Spike: a system for sufficient completeness and parameterised inductive proofs, CADE 12, LNCS 814, Springer 1994

[4] R Brown and A Tonks, Calculations with simplicial and cubical groups in AXIOM, J. Symbolic Computation , 17 (1994) 159-179.

[5] B Buchberger *et al*, A Survey on the *Theorema* Project, to appear in Proc. Issac 97.

[6] A Bundy, The Use of Explicit Plans to Guide Inductive Proofs, in Ed. E. Lusk and R. Overbeek, Proc. CADE-9, LNCS 310, Springer-Verlag, 1988, pp. 111–120.

[7] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, 10th International Conference on Automated Deduction, Springer-Verlag, 1990, pp. 647–648.

[8] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. Artificial Intelligence, 62, 1993, pp. 185–223.

[9] M Cherniak, Form(er)s over function(s): The kola reference manual. June 1996.

[10] E Clarke and X Zhao, Combining symbolic computation and theorem proving: some problems of Ramanujan, CADE 12, LNCS 814 Springer 1994

[11] D Cox, J Little, and D O'Shea, Ideals Varieties and Algorithms, Undergraduate Texts in Mathematics, Springer-Verlag, 1993.

[12] N de Bruijn, The mathematical Language AUTOMATH, its usage, and some of its extensions, Symposium on Automatic Demonstration, Springer Lecture Notes in Mathematics 125, 1968.

[13] A Dolzmann and T Sturm. REDLOG: Computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universitt Passau, D-94030 Passau, Germany, February 1996.

[14] M Dunstan and T Kelsey, A BISL for AXIOM, Research Report, University of St Andrews, in preparation

[15] M Fujita, J Slaney, and F Bennett, Automatic Generation of Some Results in Finite Algebra, Proc. of IJCAI '93, Chambery, 1993, pp.52–57.

[16] J V Guttag and J J Horning, Larch: languages and tools for formal specification, Springer, New York, 1993.

[17] J Harrison and L Thery, Extending the HOL Theorem Prover with a Computer Algebra System to Reason About the Reals, in Higher Order Logic Theorem Proving and Its Applications, ed J Joyce and C Seger, LNCS 780, Springer 1993.

[18] K Homann and J Calmet, Combining theorem proving and symbolic mathematical computing, LNCS 958, Springer 1994

[19] A Heck, Introduction to Maple, Springer Verlag 1993

[20] P Jackson, Exploring Abstract Algebra in Constructive Type Theory, CADE 12, LNCS 814,Springer 1994

[21] R D Jenks and R S Sutor, axiom: The Scientific Computation System, Springer, 1992.

[22] C Jones, Completing the Rationals and Metric Spaces in LEGO, in Logical Environments, ed G Huet and G Plotkin, Cambridge University Press 1993.

[23] Journal of Automated Reasoning, Special Issue on Inductive Proof, vol 16 No 1/2, 1996.

[24] G Labelle, Some combinatorial results first found using computer algebra, J Symbolic Comp 20 (1995) 567-594 [volume contains several related papers]

[25] S Linton and D Shand, Group theoretic examples with completion theorem provers, J Automated Reasoning 17, 1996, pp. 145-169.

[26] S Linton, U.Martin, P Prohle and D. Shand, Algebra and automated reasoning, in Eds. M. McRobbie and J. Slaney, Proc. of CADE-13, LNAI 1104, Springer-Verlag, 1996, pp. 448-462.

[27] U Martin and T Nipkow, Ordered Rewriting and Confluence, CADE 10, LNCS , Springer 1990

[28] W M McCune, Single Axioms for Groups and Abelian Groups with Various Operations, J. Automated Reasoning 10(1), 1993, pp. 1–13

[29] W M McCune, and R Padmanabhan, Automated Deduction in Equational Logic and Cubic Curves, LNAI 1095 Springer-Verlag, 1996.

[30] P Rideau, Computer algebra and mechanics: the JAMES software, in Computer Algebra in Industry, (ed A M Cohen), Wiley 1993, pp. 143–158

[31] J Rushby et al, Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS, IEEETransactions on Software Engineering, 21(1995) 107-125,

[32] J Staunstrup et al, Mechanised Verification of Circuit Descriptions using the Larch Prover, Theorem Provers in Circuit Design, Nijmegen, June 1992. North-Holland pp. 277–299.

[33] D Stoutemyer, Crimes and Misdemeanors in the Computer Algebra Trade, Notices of the AMS, September 1991, volume 38 7.

[34] T Sturm, A REDUCE package for first-order logic, Universitat Passau, Technical Report 1995

[35] J Ueberberg, Interactive theorem proving and computer algebra, LNCS 958, Springer Verlag 1994