Proof Planning in Maple

Toby Walsh[1]

University of York, York, England. tw@cs.york.ac.uk

Abstract. We are building a system that helps us mix proof with computation. On the one hand, theorem proving tools are often good at helping us construct proofs but poor at doing algebraic computations within these proofs. On the other hand, computer algebra systems are good at doing algebraic computations but usually lack the ability to construct proofs. To tackle this problem, we are adding theorem proving capabilities to the computer algebra system Maple. To make the system easy to extend and to apply to new domains, we have built a simple "proof planning" shell, called CLAM-Lite in which the proof methods are declaratively and explicitly represented. To test the system, we have developed some simple methods for reasoning inductively about summation.

1 Introduction

A variety of tools have been developed to assist mathematicians, scientists and engineers "do" mathematics. For example, computer algebra systems perform complex and sometimes tedious computations, whilst theorem provers find proofs to theorems. Surprisingly, there are very few systems that allow you to do both proof and computation, even though mathematics in practice often involves both activities. For instance, we might prove that a series is absolutely convergent before computing its limit by rearranging the order of terms. Alternatively, our failure to compute a limit might suggest attempting to prove that the series diverges.

Three possible architectures have been proposed to build tools that do both proof and computation. First, we might add proving capabilities to an existing computer algebra system. Second, we might add computer algebra capabilities to an existing theorem prover. Or third, we might provide an interface to link an existing computer algebra system with an existing theorem prover. There are technical arguments in favour and against the three approaches. For example, if we are worried about correctness, we might have a preference for the second approach. There is, however, a forceful non-technical argument. There are many users of computer algebra systems but relatively few users of theorem provers. We are therefore adding proving capabilities to an existing computer algebra system.

2 Proof planning

One of our aims is to see if current theorem proving techniques, specifically Bundy's "proof planning" approach [Bun91], can be successfully *embedded* within another system. Proof planning has been under development for several years now. It is therefore timely to consider how well it works outside of its initial application domain, inductive proof. Proof planning has often been associated with "rippling" [BSvH+93], a powerful heuristic for guiding search in inductive proof. However, proof planning is a technology that can in theory be applied to any mathematical domain (and not just that of reasoning about recursive functions using inductive proof). How well does proof planning work in other domains? Can we invent proof methods like rippling which provide strong search control in other mathematical domains?

Proof planning offers several potential advantages over other theorem proving techniques. First, the search control is cleanly separated from the logic. It is therefore often relatively easy to extend proof methods without compromising soundness. Indeed, the typical way of working with a system like CLAM is to modify and extend the existing proof methods. This is not possible in systems like the Boyer-Moore prover. Users can only propose intermediate lemmas which they hope will guide the prover in the right direction. At times, this can be a frustrating experience as the inbuilt heuristics may resist attempts to be lead in a different direction. Second, proof planning offers a high level meta-logic in which proof strategies can often be easily expressed. For example, we are able to discuss such concepts as "recursive argument position" and "blocked wave-front". We can even reason with this meta-logic to derive properties of our proof methods.

A proof planner builds a proof plan, and the execution of this proof plan constructs a "proof object". In the long term, we aim to offer a variety of different proof objects including proofs written in plain text, in LaTeX, and in HTML. However, our initial implementation currently only supports plain text proofs. An even more ambitious longer term project would be to support tactics which expand proofs down to the object level. At present, we aim to write out a sufficiently detailed proof that the user is convinced of its correctness. As some of the proof "steps" involve calls to Maple's simplification routines, it would be a considerable challenge to expand these proofs out to the level of the individual inference steps.

3 Computer algebra

Just as we allowed non-technical reasons to guide the design of the system architecture (i.e. a proof planner built on top of an existing computer algebra system), we allowed non-technical reasons to guide the choice of the particular computer algebra system used. Maple has a large user base within our university and indeed within universities as a whole. We therefore chose to build our proof planner on top of Maple. However, we could easily have chosen one of the other rival

computer algebra systems like Mathematica or AXIOM. Indeed, a system like Mathematica might have been an easier choice since it arguably provides better support for pattern matching and rewriting than Maple. However, many of the lessons learnt in adding proof planning capabilities to Maple will map across onto other computer algebra systems. To enable a proof planner to be rapidly built on top of Maple, we decided to prototype the proof planning shell in Prolog. It was a simple matter to provide an interface so Maple could call Prolog (and vice versa). In the longer term, it may prove better to have the proof planning shell implemented directly in Maple's procedural language. However, one advantage of writing the proof planner in Prolog is that we should be able to interface it to other computer algebra systems. A second advantage is that we can easily write the method pre- and post-conditions in a declarative and logical form that can then be immediately executed (see Table 2 for an example). So far, we have added three new commands to Maple: `prove`, `assumption`, `noassumptions`. The first command tries to prove the given theorem from a set of assumptions, the second command adds a new assumption to the set of assumptions, and the third creates an empty set of assumptions. These commands are discussed in further detail in the following sections.

4 CLAM-Lite

We interfaced Maple to a cut-down version of the CLAM proof planner called CLAM-Lite. A proof planner takes a goal to prove, and selects a method from a database of methods which matches this goal. The proof planner checks that the pre-conditions of the method (which are a sequence of statements in a meta-logic) hold. If the pre-conditions hold then the proof planner executes the post-conditions (which are also a sequence of statements in the meta-logic). This constructs the output goal or goals. Table 1 gives the code for the simple depth-first proof planner in CLAM-Lite. We are in the process of adding other proof planners which explore the search space using alternative strategies like breadth-first, best-first, and limited-discrepancy search. Methods are stored as 5-tuples[1]: a name slot (which is also the tactic name), the input goal (which is pattern matched against), the list of pre-conditions, the list of post-conditions, and the list of output goals. An example method is given in Table 2.

One positive side-effect of this project is that we now have a simple enough proof planner that we can start to reason about it formally. For example, we have proved that determining if a valid proof plan exists can be undecidable even if the meta-theory in which pre- and post-conditions are checked is itself decidable. On a more positive note, we have proved that an iterative deepening proof planner is sound (only finds valid proof plans) and complete (if it terminates signaling failure or fails to terminate then no valid proof plan exists). We have also shown that the depth-first proof planner is sound but not complete. A valid proof plan

[1] In CLAM, they are 6-tuples. However, since one slot contains the method name and another contains the tactic name which is always identical, we avoid this redundancy.

```
proof_plan(Goal,[Method|Plan]):-
   apply(Method,Goal,Outputs),
   map_proof_plan(Outputs,Plan).

apply(Method,Goal,Outputs):-
   method(Method,Goal,Pre,Post,Outputs),
   call_goals(Pre),
   call_goals(Post).

map_proof_plan([],[]).
map_proof_plan([Output|Outputs],[Plan|Plans]):-
   proof_plan(Output,Plan),
   map_proof_plan(Outputs,Plans).
```

Table 1. The simple depth-first proof planner used within CLAM-Lite. Other search strategies (e.g. breadth-first, and iterative deepening) are defined in a similar fashion.

```
method(equality(poly_expand,A1,B1),
   H==>A=B,
   [polynomial(A),
    polynomial(B)]
   [normal_form(A,A1),
    normal_form(B,B1)]
   [H==>A1=B1]).
```

Table 2. An example method used within CLAM-Lite. A method has five slots: the method (and tactic) name, the input goal, the pre-conditions, the post-conditions, and the list of output goals. The method above proves equality of two polynomial terms by expanding them out into a normal form. The input goal is a sequent with the set of hypotheses on the left and the conclusion on the right.

exists that the depth-first proof planner will fail to find. This proof plan will be found by the iterative deepening or the breadth-first proof planners.

5 Some examples

To illustrate the potential applications of proof planning within Maple, we implemented a small number of simple methods. In our future work, we aim to extend greatly the number and the functionality of these methods. The speed and ease with which these methods were implemented offers promise in this direction. The first domain we considered was summation. Computer algebra systems like Maple containing complex routines like Gosper's algorithm for finding closed forms for summations. It would be difficult to prove that (the implementations of) such algorithms was correct. By comparison, we may only need a simple inductive proof to verify that a given closed form is indeed correct. In previous

work, we had used proof planning to find close form sums [WNB92]. Here, we focus instead on proving such sums correct.

We wrote two methods: `summation` which sets up the base and step cases when given a sum to prove and `equality` which proves the resulting equality problems using Maple's `simplify` and `expand` functions. The associated tactics write out a natural language proof. Table 3 is a script of an example session with Maple. This example (the sum of the first n integers) was used as the development set for writing the two proof methods. We then tested these methods on the dozen problems listed in [WNB92]. The results are summarized in Table 4. Maple found a closed form sum for all but one of these examples. Despite their simplicity, our proof methods were able to prove that eight out of the twelve closed form sums were correct. Interestingly, all the failures appear to be on problems on which rippling [BSvH$^+$93,BW96] is likely to complete the proof. With a little more effort, we anticipate that the 75% success rate on this test set could be raised to 100%. Let us repeat, however, that there is still a considerable distance to obtain proofs at the level of individual inference steps.

6 Related work

The Theorema project [BJK$^+$97] is extending the Mathematica computer algebra system with theorem proving capabilities. The system consists of a collection of special purpose provers. These include a prover for induction over the natural numbers, and another for induction over lists. The Analytica prover [BCZ96] also adds theorem proving capabilities to the Mathematica computer algebra system. The system is able to prove some complex theorems in analysis about sums and limits, as well as some simple inductive theorems. In both the Theorema and Analytica projects, the prover's heuristics are hard-wired into the code. This has several disadvantages. For example, it is difficult to separate the search control from other issues like logical correctness. It would also be difficult to extend the provers or to apply them to new domains. One of the main novelties of this project is that we explicitly represent the prover's heuristics by means of the proof planning methodology. Such an approach gives a system that we believe will be easy to extend and to apply to new domains.

Proof planning has been used in two different theorem proving systems. The CLAM proof planner developed in Edinburgh controls search in the Oyster proof checker [BvHHS90]. CLAM has also been linked to the HOL theorem prover [BSBG98]. Whilst much of the development of CLAM has been for inductive proof [BSvH$^+$93], several other domains have been explored including the summation of finite series [WNB92]. The difficulty of manipulating algebraic expressions within a theorem prover was identified as one of the main limitations in CLAM's ability to sum series [Fre92]. In this project, CLAM-Lite has the full benefit of Maple's abilities to perform algebraic manipulation. Prior to CLAM, the PRESS system used a meta-level representation of proof methods to solve algebraic equations [BW81]. Despite the lack of a planner to put the methods together, PRESS was competitive with computer algebra systems of its era.

```
    |\^/|     Maple V Release 5 (WMI Campus Wide License)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.

> Thm:= Sum(i,i=0..n)=sum(i,i=0..n);
                           n
                         -----
                          \                     2
                 Thm :=    )   i = 1/2 (n + 1)  - 1/2 n - 1/2
                          /
                         -----
                         i = 0

> prove(Thm);

The proof uses induction on i.

In the base case, n = 0 and we need to prove:
   sum(i,i=0..0) = 1/2*(0+1)^2-1/2*0-1/2
This simplifies to:
   0 = 0
In the step case, we assume:
   sum(i,i=0..n) = 1/2*(n+1)^2-1/2*n-1/2
And try to prove:
   sum(i,i=0..n+1) = 1/2*(n+2)^2-1/2*n-1
                   = sum(i,i=0..n) + n+1
Using the induction hypothesis, we must prove:
   1/2*(n+2)^2-1/2*n-1 = 1/2*(n+1)^2-1/2*n-1/2 + n+1

Consider the first subgoal.
This is an equality axiom.

Consider the second subgoal.
On the lhs, 1/2*(n+2)^2-1/2*n-1 simplifies to 1/2*n^2+3/2*n+1 leaving:
   1/2*n^2+3/2*n+1 = 1/2*(n+1)^2-1/2*n-1/2+(n+1)
On the rhs, 1/2*(n+1)^2-1/2*n-1/2+(n+1) simplifies to 1/2*n^2+3/2*n+1 leaving:
   1/2*n^2+3/2*n+1 = 1/2*n^2+3/2*n+1
This is an equality axiom.
                                      QED

>
```

Table 3. An example of proof planning within Maple. This is the first problem from [WNB92]. `Sum` is Maple's unevaluated summation operator, `sum` is the evaluated summation operator, and `0..n` is the (integer) interval from 0 to n.

No	Problem	Closed form found by Maple	Proved correct by CLAM-Lite
1	$\sum i$	$\surd$	$\surd$
2	$\sum i^2$	$\surd$	$\surd$
3	$\sum i + i^2$	$\surd$	$\surd$
4	$\sum a^i$	$\surd$	$\surd$
5	$\sum i.a^i$	$\surd$	$\surd$
6	$\sum (i+1).a^i$	$\surd$	$\surd$
7	$\sum \frac{1}{i.(i+1)}$	$\surd$	$\surd$
8	$\sum F_i$	$\times$	$\times$
9	$\sum \sin(i.\theta)$	$\surd$	$\surd$
10	$\sum \cos(i.\theta)$	$\surd$	$\times$
11	$\sum \binom{m+i}{i}$	$\surd$	$\times$
12	$\sum \binom{s(i)}{s(m)}$	$\surd$	$\times$

Table 4. Some example results. All sums are for i from 0 to n (except problem 7 which is from 1 to n), F_i is the ith Fibonacci number and $\binom{n}{m}$ is the binomial coefficient. $\surd$ indicates success, whilst $\times$ denotes failure.

The Ω system developed in Saarbrücken also implements proof planning, but in this case for a a higher order natural deduction style logic [HKK+94]. The SAPPER system provides an interface between this theorem prover and a simple home-grown computer algebra system for polynomial manipulation and differentiation [KKS96]. The computer algebra system outputs a verbose derivation which is used by SAPPER to construct a proof plan for the theorem prover. Rather than provide an interface between a proof planner and a specially written computer algebra system, we are embedding a proof planner on top of an existing computer algebra system. We also use the computer algebra system to communicate with the user. Many people have learnt to use the interface to computer algebra systems like Maple. It is therefore desirable to tap into this knowledge and expertise.

NAG Ltd (who market the AXIOM computer algebra system) and St Andrews and Bath Universities are using formal specification and theorem proving techniques in the development of computer algebra systems [DKLM98]. Their aim is to improve the robustness and reliability of the computer algebra system. We largely ignore issues about the correctness of the computer algebra system, and focus instead on extending the system's abilities to reason about mathematical objects.

7 Conclusions

We have described a simple proof planning shell, CLAM-Lite which has been built on top of the Maple computer algebra system. This system allows us to explore how proof and computation can be mixed together. To test the prototype system, we developed some simple methods for reasoning inductively about

summation. Tactics associated with these methods output a natural language proof. The system was quick to prototype, and the methods even quicker to write.

There are several directions that our future work will follow. For example, we intend to develop more powerful proof methods for reasoning about sums. We also intend to look at some other domains like reasoning about limits, continuity, etc. Many questions remain for the longer term. For example, can we expand these "proofs" out to the level of individual inference rules? Can we develop better natural language proofs? At present, the system has little knowledge about the narrative structure of proofs and merely outputs isolated sentences. Finally, are methods useful for planning computations as well as for planning proofs?

References

[BCZ96] A. Bauer, E. Clarke, and X. Zhao. Analytica: an experiment in combining theorem proving and symbolic computation. In *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation, AISMC-3*, 1996.

[BJK+97] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A survey of the Theorema project. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC'97*, 1997.

[BSBG98] R. Boulton, K. Slind, A. Bundy, and M. Gordon. An interface between Clam and HOL. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*. Springer Verlag, Lecture Notes in Computer Science, 1998.

[BSvH+93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[Bun91] A. Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.

[BvHHS90] 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*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.

[BW81] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981. Also available as DAI Research Paper 121, Dept. Artificial Intelligence, Edinburgh.

[BW96] D. Basin and T. Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(1–2):147–180, 1996.

[DKLM98] M. Dunstan, T. Kelsey, S. Linton, and U. Martin. Lightweight formal methods for computer algebra systems. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC'98*, 1998.

[Fre92] N. Free. Summing series using proof plans. Master's thesis, Dept. of Artificial Intelligence, University of Edinburgh, 1992.

[HKK+94] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. Ω-MKRP: A proof development environment. In Alan Bundy, editor, *Automated Deduction — CADE-12*, Proceedings of the 12th International Conference on Automated Deduction, pages 788–792, Nancy, France, 1994. Springer-Verlag, Berlin, Germany. LNAI 814.

[KKS96] Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra with proof planning. In Jacques Calmet and Carla Limongelli, editors, *Design and Implementation of Symbolic Computation Systems, DISCO'96*, number 1128 in LNCS, pages 204–215, Karlsruhe, Germany, 1996. Springer Verlag.

[WNB92] T. Walsh, A. Nunes, and A. Bundy. The use of proof plans to sum series. In D. Kapur, editor, *11th Conference on Automated Deduction*, pages 325–339. Springer Verlag, 1992. Lecture Notes in Computer Science No. 607. Also available from Edinburgh as DAI Research Paper 563.