An Architecture for Proof Planning Systems

Louise A. Dennis
School of Computer Science and Information Technology,
University of Nottingham, Nottingham, NG8 1BB
lad@cs.nott.ac.uk

Abstract

This paper presents a generic architecture for proof planning systems in terms of an interaction between a customisable proof module and search module. These refer to both global and local information contained in reasoning states.

1 Introduction

Proof planning systems attempt to automate the process of mathematical proof by searching for a high-level representation of a proof rather than one at the level of logical inference rules. This makes the resulting proofs more human-like; aiding comprehension, cognitive modeling and higher-level reasoning about failed proof attempts. Proof planning represents an important methodology in automated reasoning and has been proposed as a generic framework for understanding reasoning in general [Bundy, 1991]. Unfortunately over 15 years of research and exploration of the concept has so altered the initial descriptions that it is no longer trivial to describe exactly what proof planning *is*. This paper presents a generic architecture for proof planning systems as a first step towards a unified description.

2 Proof Planning

Proof planning was first introduced in [Bundy, 1988] where the term *proof plan* was used to describe both a way of going about a particular sort of proof and a high-level representation of the proof itself. In order to avoid confusion, this papers refers to these two concepts as the *proof strategy* and the *proof representation* respectively. A common example of a proof strategy is mathematical induction. Here the strategy is to split the problem into a base case and a step case and the step case can be further broken down into a sequence of rewrites followed by appeal to the induction hypothesis.

The proof representation is a graph/tree structure. The nodes are labelled by proof goals and a justification associated with a *proof method*. The intended semantics is that a goal formula can be derived from its children and that this derivation is justified by the method. These proof representations are formed using the proof methods as plan operators within a framework inspired by STRIPS [Fikes and Nilsson, 1971]. The edges in a proof representation can be expanded to a proof at the level of logical inference rules using planning operators based on LCF-style tactics. These tactics are associated with the method that justifies the parent node.

Methods are generally represented as a frame structure with slots for preconditions which must be satisfied before the method can apply. A proof strategy is either represented by some form of *method hierarchy* which restricts consideration of particular methods to particular points in a proof or by *control rules* [Siekmann *et al.*, 2003] which analyse the current state of the plan representation and the history of the planning attempt to decide on the methods to be considered at any point. The proof strategy is an important technique for limiting search and has been represented in a variety of ways.

The proof planning methodology also includes failure-triggered plan operators (eg. [Ireland and Bundy, 1996]) which propose major changes to the current proof representation (as opposed to methods which simply extend the graph). Selection and application of these operators is determined by preconditions and the proof strategy.

At present there is no general description of proof planning which which addresses the varied ideas of proof strategy implemented in the proof planning systems of the last decade. This paper seeks to answer the question of what proof planning is by presenting a generic architecture that can encompass existing implementations.

3 A General Architecture for Proof Planning Systems

Figure 1 shows a general architecture for proof planning systems as a data/control flow diagram. In this diagram circles indicate functions or operations and rectangles stores of data. Solid arrows into operators represent information used by the operator and arrows from it show outputs or side-effects of its operation. Dashed arrows show the flow of control between operations.

Most descriptions of proof planning have described the proof planner as operating on proof representations however an important insight in [Dixon and Fleuriot, 2003]

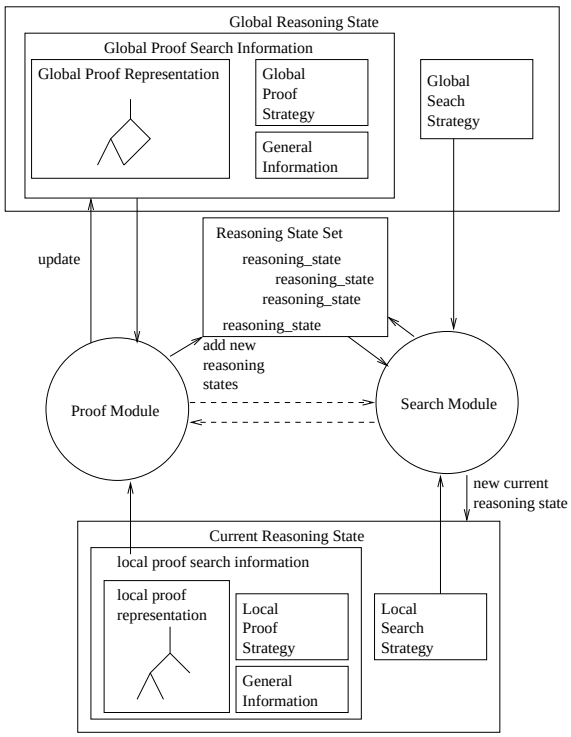

Figure 1: A General Architecture for Proof Planning Systems

is to notice that proof planners operate on *reasoning states*. Reasoning states contain a search strategy (such as depth-first search) and proof search information which includes general information (such as constraints) as well as a proof representation and a proof strategy.

There are two reasoning states consulted by the system: a global reasoning state which is in existence for the lifetime of the program's execution and which can be dynamically updated and a current reasoning state which is generated by the proof module at one point in time, placed in a global set and later selected by the search module to become current after which it is deleted. This reasoning state is not altered once it is generated.

Global information is used in all systems for pragmatic reasons. However there is also a need to store a *history* (eg. [Siekmann *et al.*, 2003]) of the planning attempt, including abandoned branches of the search space. This needs to be global and dynamically updatable.

At the start of a proof attempt a user presents a global reasoning state and a set (usually a singleton) of local reasoning states to the system. The search module then selects one of the local reasoning states to become current and hands control to the proof module.

Control passes back and forth between the proof module and the search module. The proof module uses proof search information to generate new reasoning states and to update the global reasoning state. Control is then handed to the search module which uses the search strategies to propose a new current reasoning state and to update the reasoning state set.

A key idea is that the proof module should be customisable by the user via the proof strategy. In this way a proof planner can be adapted for individual proof families without the user needing a firm grip on the internal code of the system. User support for expressing a proof strategy varies widely and appears to be an area ripe for further research.

4 Conclusion and Further Work

I have presented here a general architecture for proof planning as an interaction between a proof module and a search module operating on two reasoning states (one global and one local) and a reasoning state set.

Dixon et al. [2004] have recently started work on a comparison of the major proof planning systems. We are in the process of combining our work and hope this will generate further insights into the definition of proof planning and the construction of proof planning systems.

Acknowledgements

This work was supported by EPSRC Platform Grant GR/S01771/01. Thanks are due to Lucas Dixon and Martin Pollet for valuable discussions.

References

[Bundy, 1988] Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *CADE 9*, volume 310 of *LNCS*, pages 111–120. Springer, 1988.

[Bundy, 1991] 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.

[Dixon and Fleuriot, 2003] L. Dixon and J. D. Fleuriot. IsaPlanner: A prototype proof planner in Isabelle. In F. Baader, editor, *CADE 19*, volume 2741 of *LNCS*, pages 279–283. Springer, 2003.

[Dixon *et al.*, 2004] L. Dixon, M. Jamnik, and M. Pollet. Proof planning: Comparing ΩMEGA, *λClam* and IsaPlanner. In B. Bennett, editor, *ARW 11*, pages 50–52. University of Leeds, School of Computing, 2004. Held in Association with the AISB'04 Convention.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *AI*, (2):189–208, 1971.

[Ireland and Bundy, 1996] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *JAR*, 16(1–2):79–111, 1996.

[Siekmann *et al.*, 2003] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann, and M. Pollet. Proof development in OMEGA: The irrationality of square root of 2. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, Kluwer Applied Logic series. Kluwer Academic Publishers, 2003.