School of Informatics, University of Edinburgh

Centre for Intelligent Systems and their Applications

Planning and Patching Proof

by

Alan Bundy

Informatics Research Report EDI-INF-RR-0229

School of Informatics **July 2004**
http://www.informatics.ed.ac.uk/

Planning and Patching Proof

Alan Bundy

Informatics Research Report EDI-INF-RR-0229

SCHOOL *of* INFORMATICS
Centre for Intelligent Systems and their Applications

July 2004

submitted to AISC-04

Abstract :

We describe proof planning: a technique for both describing the hierarchical structure of proofs and then using this structure to guide proof attempts. When such a proof attempt fails, these failures can be analyzed and a patch formulated and applied. We also describe rippling: a powerful proof method used in proof planning. We pose and answer a number of common questions about proof planning and rippling.

Keywords :

Planning and Patching Proof *

Alan Bundy

School of Informatics, University of Edinburgh,
3.09 Appleton Tower, 11 Crichton Street, Edinburgh, EH8 9LE, United Kingdom.
A.Bundy@ed.ac.uk

Abstract. We describe proof planning: a technique for both describing the hierarchical structure of proofs and then using this structure to guide proof attempts. When such a proof attempt fails, these failures can be analyzed and a patch formulated and applied. We also describe rippling: a powerful proof method used in proof planning. We pose and answer a number of common questions about proof planning and rippling.

1 Introduction

The Program Committee Chair of AISC-04 suggested that the published version of my talk might be:

> " ... a short paper telling our audience what are the highlights of the publication landscape for the subject of your presentation, what they should read if they want to become better informed by systematic reading, and why they should read the cited material (i.e. why it represents the highlights), could have both immediate use and longer-term educational use for people who don't attend the conference but buy or read the proceedings later."

Below I have attempted to fulfill this brief. I have organized the paper as a 'Frequently Asked Questions' about proof planning, in general, and rippling, in particular.

2 Proof Planning

2.1 Introduction

What is proof planning? Proof planning is a technique for guiding the search for a proof in automated theorem proving. A proof plan is an outline or plan of a proof. To prove a conjecture, proof planning constructs a proof plan for a proof and uses it to guide the construction of the proof itself. Proof planning reduces the amount of search and curbs the combinatorial explosion. It also helps pinpoint the cause of any proof attempt failure, suggesting a patch to facilitate a renewed attempt.

Common patterns in proofs are identified and represented in computational form as general-purpose tactics, i.e. programs for directing the proof search process. These tactics are then formally specified with methods using a meta-language. Standard patterns of proof failure and appropriate patches to the failed proofs attempts are represented as critics. To form a proof plan for a conjecture the proof planner reasons with these methods and critics. The proof plan consists of a customized tactic for the conjecture, whose primitive actions are the general-purpose tactics. This customized tactic directs the search of a tactic-based theorem prover.

For a general, informal introduction to proof planning see [Bundy, 1991]. Proof planning was first introduced in [Bundy, 1988]. An earlier piece of work that led to the development of proof planning was the use of meta-level inference to guide equation solving, implemented in the Press system (see [Sterling *et al*, 1989]).

* The research reported in this paper was supported by EPSRC grant GR/S01771.

Has proof planning been implemented? Yes, in the *Oyster/Clam* system [Bundy *et al*, 1990] and λ*Clam* system at Edinburgh and the Omega system at Saarbrücken [Benzmüller *et al*, 1997]. *Clam* and λ*Clam* are the proof planners. They constructs a customized tactic for a conjecture and then a proof checker, such as *Oyster*, executes the tactic.

In principle, *Clam* could be interfaced to any tactic-based theorem prover. To test this assertion, we interfaced *Clam* to the Cambridge HOL theorem prover [Boulton *et al*, 1998]. We are currently building a proof planner, called IsaPlanner, in Isabelle [Dixon & Fleuriot, 2003].

How has proof planning been evaluated? One of the main domains of application has been in inductive reasoning [Bundy, 2001], with applications to software and hardware verification, synthesis and transformation, but it has also been applied to co-induction [Dennis *et al*, 2000], limit theorems, diagonalization arguments, transfinite ordinals, summing series, equational reasoning, meta-logical reasoning, algebra, *etc.* A survey of such applications can be found in chapter 5 of [Bundy *et al*, 2005].

Can proof planning be applied to non-mathematical domains? Yes. We have had some success applying proof planning to game playing (Bridge [Frank *et al*, 1992,Frank & Basin, 1998] and Go [Willmott *et al*, 2001]) and to configuration problems [Lowe *et al*, 1998]. It is potentially applicable wherever there are common patterns of reasoning. Proof planning can be used to match the problem to the reasoning method in a process of meta-level reasoning. Proof planning gives a clean separation between the factual and search control information, which facilitates their independent modification.

What is the relation between proof planning and rippling? Rippling is a key method in our proof plans for induction. It is also useful in non-inductive domains. However, you can certainly envisage a proof planning system which did not contain a rippling method (the Saarbrücken Omega system, for instance) and you can envisage using rippling, e.g. as a tactic, in a non-proof planning system. So there is no necessary connection. For more on rippling see §3.

What are the scope and limitations of proof planning? A critical evaluation of proof planning can be found in [Bundy, 2002].

2.2 Discovery and Learning

Is it possible to automate the learning of proof plans? Proof plans can be learnt from example proofs. In the case of equation-solving methods, this was demonstrated in [Silver, 1985]; in the case of inductive proof methods it was demonstrated in [Desimone, 1989]. Both projects used forms of explanation-based generalization. We also have a current project on the use of data-mining techniques (both probabilistic reasoning and genetic programming) to construct tactics from large corpora of proofs, [Duncan *et al*, 2004].

The hardest aspect of learning proof plans is coming up with the key meta-level concepts to describe the preconditions of the methods. An example of such a meta-level concept is that of 'wave-front' idea used in rippling. We have not made much progress on automating the learning of these.

How can humans discover proof plans? This is an art similar to the skill used by a good mathematics teacher when analyzing a student's proof or explaining a new method of proof to a class. The key is identifying the appropriate meta-level concepts to generalize from particular examples. Armed with the right concepts, standard inductive learning techniques can form the right generalization (see §2.2).

2.3 Drawbacks and Limitations

What happens if the attempt to find a proof plan fails? In certain circumstances proof critics can suggest an appropriate patch to a partial proof plan. Suppose the preconditions of a method succeed, but this method is unpacked into a series of sub-methods one of which fails, i.e. the preconditions of the sub-method fail. Critics are associated with some of these patterns of failure. For instance, one critic may fire if the first two preconditions of a method succeed, but the last one fails. It will then suggest an appropriate patch for this kind of failure, e.g. suggest the form of a missing lemma, or suggest generalizing the conjecture. The patch is instituted and proof planning continues.

The original critics paper is [Ireland, 1992]. A more recent paper is [Ireland & Bundy, 1996]. Two important application of critics are: discovering loop invariants in the verification of imperative programs [Stark & Ireland, 1998]; and the correction of false conjectures [Monroy *et al*, 1994].

In other circumstances, a subgoal may be reached to which no method or critic is applicable. It may be possible to back-up to a choice point in the search, i.e. a place where two or more methods or critics were applicable. However, the search space defined by the methods and critics is typically much smaller than the search space defined by the object-level rules and axioms; that is both the strength and the weakness of proof planning. The total search space is cropped to the portion where the proof is most likely to occur – reducing the combinatorial explosion, but losing completeness. It is always possible to regain completeness by supplementing the methods with a default, general-purpose exhaustive search method, but some would regard this as a violation of the spirit of proof planning. For more discussion of these points see [Bundy, 2002].

Is it possible to discover new kinds of proof in a proof planning system? Since general-purpose proof plans represent common patterns in proofs, then, by definition, they cannot discover new kinds of proof. This limitation could be overcome in several ways. One would be to include a default method which invoked some general search technique. This might find a new kind of proof by accident. Another might be to have meta-methods which constructed new methods. For instance, a method for one domain might be applied to another by generalizing its preconditions[1]. Or a method might be learnt from an example proof (see §2.2). Proof plans might, for instance, be learnt from proofs constructed by general search. For more discussion of these points see [Bundy, 2002].

Isn't totally automated theorem proving infeasible? For the foreseeable future theorem provers will require human interaction to guide the search for non-trivial proofs. Fortunately, proof planning is also useful in interactive theorem provers. Proof plans facilitate the hierarchical organization of a partial proof, assisting the user to navigate around it and understand its structure. They also provide a language for chunking the proof and for describing the interrelation between the chunks. Interaction with a semi-automated theorem prover can be based on this language. For instance, the user can: ask why a proof method failed to apply; demand that a heuristic precondition is overridden; use the analysis from proof critics to patch a proof; etc.

The XBarnacle system is an semi-automated theorem prover based on proof planning [Lowe & Duncan, 1997]. There is also a version of XBarnacle with interaction critics, where the user assists the prover to find lemmas and generalizations [Jackson & Lowe, 2000].

Doesn't proof planning promote cheating by permitting ad hoc adjustments to enable a prover to 'discover' particular proofs? Not if the recommended methodology is adopted. [Bundy, 1991] specifies a set of criteria for assessing proof plans. These include generality and parsimony, which discourage the creation of ad hoc methods designed to guide particular theorems. Rather, they encourage the design of a few, general-purpose methods which guide a wide range

[1] Often new departures come in mathematics when mathematicians switch from one area to another, bringing their proof methods with them.

of theorems. The expectancy criterion promotes the association of a method with an explanation of why it works. This discourages the design of methods which often succeed empirically, but for poorly understood reasons. Of course, these criteria are a matter of degree, so poor judgment may produce methods which other researchers regard as ad hoc. The criteria of proof planning then provide a basis for other researchers to criticize such poor judgment.

2.4 Is all this of relevance to me?

Would a proof planning approach be appropriate for my application? The properties of a problem that indicate that proof planning might be a good solution are: 1. A search space which causes a combinatorial explosion when searched without heuristic guidance; 2. The existence of heuristic tactics which enable expert problem solvers to search a much smaller search space defined by these tactics; 3. The existence of specifications for each tactic to determine when it is appropriate to apply it and what effect it will have if it succeeds.

How would I go about developing proof plans for my domain? The key problem is to identify the tactics and their specifications. This is usually done by studying successful human problem solving and extracting the tactics. Sometimes there are texts describing the tactics, e.g. in bridge and similar games. Sometimes knowledge acquisition techniques, like those used in expert systems, are needed, e.g. analysis of problem solving protocols, exploratory interviews with human experts.

3 Rippling

3.1 Introduction

$$
\begin{aligned}
& t <> (Y <> Z) = (t <> Y) <> Z \\
\vdash\ & \boxed{h :: \boxed{t}^{\uparrow}} <> (y <> z) = (\boxed{h :: \boxed{t}^{\uparrow}} <> y) <> z \\
\vdash\ & \boxed{h :: \boxed{t <> (y <> z)}^{\uparrow}} = \boxed{h :: \boxed{t <> y}^{\uparrow}} <> z \\
\vdash\ & \boxed{h :: \boxed{t <> (y <> z)}^{\uparrow}} = \boxed{h :: \boxed{(t <> y) <> z}^{\uparrow}}^{\uparrow} \\
\vdash\ & \boxed{h = h \wedge \boxed{t <> (y <> z) = (t <> y) <> z}^{\uparrow}}^{\uparrow}
\end{aligned}
$$

Fig. 1. This example is taken from the step case of the induction proof of the associativity of append, where <> is infix list append and :: is infix list cons. The hollow grey boxes in the induction conclusion represent wave-fronts. Orange boxes are used when colour is available. Notice how these grow in size until a copy of the induction hypothesis appears inside them.

What is rippling? A technique for controlling term rewriting using annotations to restrict application and to ensure termination, see Fig. 1. A goal expression is rippled with respect to one or more given expressions. Each given expression embeds in the goal expression. Annotations in the goal mark those subexpressions which correspond to bits of the given (the skeleton) and those which do not (the wave-fronts). The goal is rewritten so that the embeddings are preserved, i.e. rewriting can only move the wave-fronts around within the skeleton - wave-fronts can change but the skeleton cannot. Furthermore, wave-fronts are given a direction (outwards or inwards) and movement can only be in that direction. Outward wave-fronts can mutate to inward, but not vice

versa. This ensures termination. Rippling can be implemented by putting wave annotation into the rewrite rules to turn them into wave-rules, see Fig. 2. The successful application of wave-rule requires that any wave-front in the wave-rule must match a wave-front in the goal. Rippling was originally developed for guiding the step cases of inductive proofs, in which the givens are the induction hypotheses and the goal is the induction conclusion.

For an informal introduction to rippling with a large selection of examples see [Bundy *et al*, 1993]. For a more formal account see: [Basin & Walsh, 1996]. A thorough account will shortly be available in [Bundy *et al*, 2005].

$$H :: T^{\uparrow} <> L \Rightarrow H :: T <> L^{\uparrow}$$

$$rev(H :: T^{\uparrow}) \Rightarrow rev(T) <> (H :: nil)^{\uparrow}$$

$$X_1 :: X_2^{\uparrow} = Y_1 :: Y_2^{\uparrow} \Rightarrow X_1 = Y_1 \wedge X_2 = Y_2^{\uparrow}$$

$$X <> (Y <> Z)^{\uparrow} \Rightarrow (X <> Y) <> Z^{\uparrow}$$

Fig. 2. Rewrite rules from the recursive definition of <> and rev, the replacement rule for equality (backwards) and the associativity of <> are annotated as wave rules. The bits not in wave-fronts are called the skeleton. Note that the skeleton is the same on each side of the wave rule, but that more of it is surrounded by the wave-front on the right hand side compared to the left hand side.

Why is it called rippling? Raymond Aubin coined the term 'rippling-out', in his 1976 Edinburgh PhD thesis, to describe the pattern of movement of what we now call wave-fronts, during conventional rewriting with constructor-style recursive definitions. In [Bundy, 1988], we turned this on its head by taking such a movement of wave-fronts as the definition of rippling rather than the effect of rewriting. This enabled the idea to be considerably generalized. Later we invented 'rippling-sideways', 'rippling-in', etc and so generalized the combined technique to 'rippling'.

3.2 Relation to Standard Rewriting Techniques

Are wave-rules just the step cases of recursive definitions? No. Many lemmas and other axioms can also be annotated as wave-rules. Examples include: associative laws; distributive laws; replacement axioms for equality; many logical axioms; etc. Equations that cannot be expressed as wave-rules include commutative laws and (usually) the step cases from mutually recursive definitions. The latter can be expressed as wave-rules in an abstraction in which the mutually defined functions are regarded as indistinguishable. Lots of example wave-rules can be found in [Bundy *et al*, 2005].

How does rippling differ from the standard application of rewrite rules? Rippling differs from standard rewriting in two ways. Firstly, the wave annotation may prevent the application of a wave-rule which, viewed only as a rewrite rule, would otherwise apply. This will happen if the left-hand side of the wave-rule contains a wave-front which does not match a wave-front in the expression being rewritten. Secondly, equations can usually be oriented as wave-rules in both directions, but without loss of termination. The wave annotations prevent looping. An empirical comparison of rippling and rewriting can be found in [Bundy & Green, 1996].

Since rippling is terminating, is it restricted to terminating sets of rewrite rules? No. If all the rewrite rules in a non-terminating set can be annotated as wave-rules then the additional conditions of wave annotation matching, imposed by rippling, will ensure that rippling still terminates. Examples are provided by the many rewrite rules that can be annotated as wave-rules in both directions, and may even both be used in the same proof, without loss of termination.

Couldn't we simply perform rippling using a suitable order, e.g. recursive path ordering, without the need for annotations? No, each skeleton gives (in essence) a different termination ordering which guides the proof towards fertilization with that skeleton. Different annotations on the same term can result in completely different rewritings.

Is rippling restricted to first-order, equational rewriting? No, there are at least two approaches to higher-order rippling. One is based on viewing wave annotation as representing an embedding of the given in the goal [Smaill & Green, 1996]. The other is based on a general theory of colouring λcalculus terms in different ways [Hutter & Kohlhase, 1997].

Rippling can also be extended to support reasoning about logic programs — and other situations where values are passed between conjoined relations via shared existential variables, as opposed to being passed between nested functions. Relational rippling adapts rippling to this environment [Bundy & Lombart, 1995].

3.3 Wave Annotations

Is the concept of wave-rule a formal or informal one? 'Wave-rule' can be defined formally. It is a rewrite rule containing wave annotation in which the skeletons are preserved and the wave-front measure of the right-hand side is less than that of the left-hand side. Informally, the skeleton consists of those bits of the expression outside of wave-fronts or inside the wave-holes. The measure records the position of the wave-fronts in the skeleton. It decreases when outwards directed wave-fronts move out or downwards wave-fronts move in. A formal definition of skeleton and of the wave-front measure can be found in [Basin & Walsh, 1996] and in chapter 4 of [Bundy *et al*, 2005].

Where do the wave annotations in wave-rules and in induction rules come from? Wave annotation can be inserted in expressions by a family of difference unification algorithms invented by Basin and Walsh (see [Basin & Walsh, 1993]). These algorithms are like unification but with the additional ability to hide non-matching structure in wave-fronts. Ground difference matching can be used to insert wave annotation into induction rules and ground difference unification for wave-rules. 'Ground' means that no instantiation of variables occurs. 'Matching' means that wave-fronts are inserted only into the induction conclusion and not the induction hypothesis. 'Unification' means that wave-fronts are inserted into both sides of wave-rules. Note that the process of inserting wave annotations can be entirely automated.

3.4 Performance

Has rippling been used to prove any hard theorems? Yes. Rippling has been used successfully in the verification of the Gordon microprocessor and the synthesis of a decision procedure and of the rippling tactic itself. It has also been used outwith inductive proofs for the summing of series and the Lim+ theorem. A survey of some of these successes can be found in chapter 5 of [Bundy *et al*, 2005].

Can rippling fail? Yes, if there is no wave-rule available to move a wave-front. In this case we apply critics to try to patch the partial proof. For inductive proofs, for instance, these may: generalize the induction formula; revise the induction rule; introduce a case split; or introduce and prove an intermediate lemma, according to the precise circumstances of the breakdown. The

fact that a failed ripple provides so much information to focus the attempt to patch the proof is one of the major advantages of rippling. More details about critics, including some hard examples which have been proved with their use, can be found in [Ireland & Bundy, 1996] and in chapter 3 of [Bundy *et al*, 2005].

3.5 Miscellaneous

How is rippling used to choose induction rules? There is a one-level look-ahead into the rippling process to see what induction rules would permit rippling to take place. In particular, which wave-fronts placed around which induction variables would match with corresponding wave-fronts in induction rules. We call this ripple analysis. It is similar to the use of recursive definitions to suggest induction rules, as pioneered by Boyer and Moore, but differs in that all available wave-rules are used in ripple analysis, and not just recursive definitions. More detail of ripple analysis can be found in [Bundy *et al*, 1989], although that paper is rather old now and is not a completely accurate description of current rippling implementations. In particular, the term 'ripple analysis' was not in use when that paper was written and it is misleadingly called 'recursion analysis' there. A recent alternative approach is to postpone the choice of induction rule by using meta-variables as place holders for the induction term and then instantiating these meta-variables during rippling: thus tailoring the choice of induction rule to fit the needs of rippling [Kraan *et al*, 1996,Gow, 2004].

3.6 And Finally ...

Why do you use orange boxes to represent wave-fronts? [2]

The boxes form hollow squares, which help to display the outwards or inwards movement of wave-fronts. In the days of hand-written transparencies, orange was used because it is one of the few transparent overhead pen colours, allowing the expression in the wave-front to show through[3].

For more information about the research work outlined above, electronic versions of some of the papers and information about down loading software, see the web site of my research group at http://dream.dai.ed.ac.uk/.

References

[Basin & Walsh, 1993] Basin, David A. and Walsh, Toby. (1993). Difference unification. In Bajcsy, R., (ed.), *Proc. 13th Intern. Joint Conference on Artificial Intelligence (IJCAI '93)*, volume 1, pages 116–122, San Mateo, CA. Morgan Kaufmann. Also available as Technical Report MPI-I-92-247, Max-Planck-Institut für Informatik.

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

[Benzmüller *et al*, 1997] Benzmüller, C., Cheikhrouhou, L., Fehrer, D., Fiedler, A., Huang, X., Kerber, M., Kohlhase, K., Meier, A, Melis, E., Schaarschmidt, W., Siekmann, J. and Sorge, V. (1997). Ωmega: Towards a mathematical assistant. In McCune, W., (ed.), *14th International Conference on Automated Deduction*, pages 252–255. Springer-Verlag.

[Boulton *et al*, 1998] Boulton, R., Slind, K., Bundy, A. and Gordon, M. (September/October 1998). An interface between CLAM and HOL. In Grundy, J. and Newey, M., (eds.), *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98)*, volume 1479 of *Lecture Notes in Computer Science*, pages 87–104, Canberra, Australia. Springer.

[Bundy & Green, 1996] Bundy, Alan and Green, Ian. (December 1996). An experimental comparison of rippling and exhaustive rewriting. Research paper 836, Dept. of Artificial Intelligence, University of Edinburgh.

[2] The boxes in Figs 1 and 2 are grey rather than orange because colour representation is not possible in this book.

[3] Famously, Pete Madden coloured his wave-fronts red for a conference talk. The underlying expressions were made invisible, ruining his presentation.

[Bundy & Lombart, 1995] Bundy, A. and Lombart, V. (1995). Relational rippling: a general approach. In Mellish, C., (ed.), *Proceedings of IJCAI-95*, pages 175–181. IJCAI.

[Bundy, 1988] Bundy, A. (1988). The use of explicit plans to guide inductive proofs. In Lusk, R. and Overbeek, R., (eds.), *9th International Conference on Automated Deduction*, pages 111–120. Springer-Verlag. Longer version available from Edinburgh as DAI Research Paper No. 349.

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

[Bundy, 2001] Bundy, Alan. (2001). The automation of proof by mathematical induction. In Robinson, A. and Voronkov, A., (eds.), *Handbook of Automated Reasoning, Volume 1.* Elsevier.

[Bundy, 2002] Bundy, A. (2002). *A Critique of Proof Planning*, pages 160–177. Springer.

[Bundy *et al*, 1989] Bundy, A., van Harmelen, F., Hesketh, J., Smaill, A. and Stevens, A. (1989). A rational reconstruction and extension of recursion analysis. In Sridharan, N. S., (ed.), *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 359–365. Morgan Kaufmann. Also available from Edinburgh as DAI Research Paper 419.

[Bundy *et al*, 1990] Bundy, A., van Harmelen, F., Horn, C. and Smaill, A. (1990). The Oyster-Clam system. In Stickel, M. E., (ed.), *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

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

[Bundy *et al*, 2005] Bundy, A., Basin, D., Hutter, D. and Ireland, A. (2005). *Rippling: Meta-level Guidance for Mathematical Reasoning.* Cambridge University Press.

[Dennis *et al*, 2000] Dennis, L., Bundy, A. and Green, I. (2000). Making a productive use of failure to generate witness for coinduction from divergent proof attempts. *Annals of Mathematics and Artificial Intelligence*, 29:99–138. Also available as paper No. RR0004 in the Informatics Report Series.

[Desimone, 1989] Desimone, R. V. (1989). Explanation-Based Learning of Proof Plans. In Kodratoff, Y. and Hutchinson, A., (eds.), *Machine and Human Learning.* Kogan Page. Also available as DAI Research Paper 304. Previous version in proceedings of EWSL-86.

[Dixon & Fleuriot, 2003] Dixon, L. and Fleuriot, J. D. (2003). IsaPlanner: A prototype proof planner in Isabelle. In *Proceedings of CADE'03*, Lecture Notes in Computer Science.

[Duncan *et al*, 2004] Duncan, H., Bundy, A., Levine, J., Storkey, A. and Pollet, M. (2004). The use of data-mining for the automatic formation of tactics. In *Workshop on Computer-Supported Mathematical Theory Development.* IJCAR-04.

[Frank & Basin, 1998] Frank, I. and Basin, D. (1998). Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1–2):87–123.

[Frank *et al*, 1992] Frank, I., Basin, D. and Bundy, A. (1992). An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, pages 72–76, Vienna, Austria. Longer Version available from Edinburgh as DAI Research Paper No. 575.

[Gow, 2004] Gow, Jeremy. (2004). *The Dynamic Creation of Induction Rules Using Proof Planning.* Unpublished Ph.D. thesis, Division of Informatics, University of Edinburgh.

[Hutter & Kohlhase, 1997] Hutter, D. and Kohlhase, M. (1997). A colored version of the λ-Calculus. In McCune, W., (ed.), *14th International Conference on Automated Deduction*, pages 291–305. Springer-Verlag. Also available as SEKI-Report SR-95-05.

[Ireland & Bundy, 1996] Ireland, A. and Bundy, A. (1996). Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111. Also available from Edinburgh as DAI Research Paper No 716.

[Ireland, 1992] Ireland, A. (1992). The Use of Planning Critics in Mechanizing Inductive Proofs. In Voronkov, A., (ed.), *International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence No. 624, pages 178–189. Springer-Verlag. Also available from Edinburgh as DAI Research Paper 592.

[Jackson & Lowe, 2000] Jackson, M. and Lowe, H. (June 2000). XBarnacle: Making theorem provers more accessible. In McAllester, D., (ed.), *CADE17*, number 1831 in Lecture Notes in Computer Science, Pittsburg. Springer.

[Kraan *et al*, 1996] Kraan, I., Basin, D. and Bundy, A. (1996). Middle-out reasoning for synthesis and induction. *Journal of Automated Reasoning*, 16(1–2):113–145. Also available from Edinburgh as DAI Research Paper 729.

[Lowe & Duncan, 1997] Lowe, H. and Duncan, D. (1997). XBarnacle: Making theorem provers more accessible. In McCune, William, (ed.), *14th International Conference on Automated Deduction*, pages 404–408. Springer-Verlag.

[Lowe *et al*, 1998] Lowe, H., Pechoucek, M. and Bundy, A. (1998). Proof planning for maintainable configuration systems. *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, 12:345–356. Special issue on configuration.

[Monroy *et al*, 1994] Monroy, R., Bundy, A. and Ireland, A. (1994). Proof Plans for the Correction of False Conjectures. In Pfenning, F., (ed.), *5th International Conference on Logic Programming and Automated Reasoning, LPAR'94*, Lecture Notes in Artificial Intelligence, v. 822, pages 54–68, Kiev, Ukraine. Springer-Verlag. Also available from Edinburgh as DAI Research Paper 681.

[Silver, 1985] Silver, B. (1985). *Meta-level inference: Representing and Learning Control Information in Artificial Intelligence*. North Holland, Revised version of the author's PhD thesis, Department of Artificial Intelligence, U. of Edinburgh, 1984.

[Smaill & Green, 1996] Smaill, Alan and Green, Ian. (1996). Higher-order annotated terms for proof search. In von Wright, Joakim, Grundy, Jim and Harrison, John, (eds.), *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1275 of *Lecture Notes in Computer Science*, pages 399–414, Turku, Finland. Springer-Verlag. Also available as DAI Research Paper 799.

[Stark & Ireland, 1998] Stark, J. and Ireland, A. (1998). Invariant discovery via failed proof attempts. In Flener, P., (ed.), *Logic-based Program Synthesis and Transformation*, number 1559 in LNCS, pages 271–288. Springer-Verlag.

[Sterling *et al*, 1989] Sterling, L., Bundy, Alan, Byrd, L., O'Keefe, R. and Silver, B. (1989). Solving symbolic equations with PRESS. *J. Symbolic Computation*, 7:71–84. Also available from Edinburgh as DAI Research Paper 171.

[Willmott *et al*, 2001] Willmott, S., Richardson, J. D. C., Bundy, A. and Levine, J. M. (2001). Applying adversarial planning techniques to Go. *Journal of Theoretical Computer Science*, 252(1-2):45–82. Special issue on algorithms, Automata, complexity and Games.