# A Critique of Proof Planning *

Alan Bundy

Division of Informatics,
University of Edinburgh

**Abstract.** Proof planning is an approach to the automation of theorem proving in which search is conducted, not at the object-level, but among a set of proof methods. This approach dramatically reduces the amount of search but at the cost of completeness. We critically examine proof planning, identifying both its strengths and weaknesses. We use this analysis to explore ways of enhancing proof planning to overcome its current weaknesses.

### Preamble

*This paper consists of two parts:*

1. *a brief 'bluffer's guide' to proof planning*[1]*; and*
2. *a critique of proof planning organised as a 4x3 array.*

*Those already familiar with proof planning may want to skip straight to the critique which starts at §2, p4.*

## 1 Background

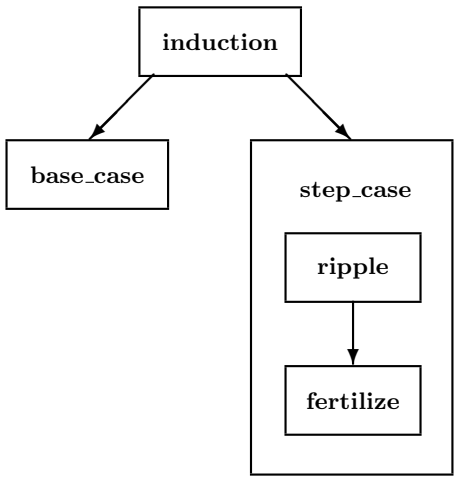
*Proof planning* is a technique for guiding the search for a proof in automated theorem proving, [Bundy, 1988,Bundy, 1991,Kerber, 1998,Benzmüller *et al*, 1997]. The main idea is to identify common patterns of reasoning in families of similar proofs, to represent them in a computational fashion and to use them to guide the search for a proof of conjectures from the same family. For instance, proofs by mathematical induction share the common pattern depicted in figure 1. This common pattern has been represented in the proof planners *Clam* and $\lambda$*Clam* and used to guide a wide variety of inductive proofs [Bundy *et al*, 1990b,Bundy *et al*, 1991,Richardson *et al*, 1998].

### 1.1 Proof Plans and Critics

The common patterns of reasoning are represented using *tactics*: computer programs which control proof search by applying rules of inference [Gordon *et al*, 1979]. These tactics are specified by *methods*. These methods give both the preconditions under which the tactics are applicable and the effects of their successful application. Meta-level reasoning is used to combine the tactics into a customised proof plan for the current conjecture. This meta-level reasoning matches the preconditions of later tactics to the effects of earlier ones. Examples of such customised proof plans are given in figure 2.

* The research reported in this paper was supported by EPSRC grant GR/M/45030. I would like to thank Andrew Ireland, Helen Lowe, Raul Monroy and two anonymous referees for helpful comments on this paper. I would also like to thank other members of the Mathematical Reasoning Group and the audiences at CIAO and Scottish Theorem Provers for helpful feedback on talks from which this paper arose.

[1] Pointers to more detail can be found at http://dream.dai.ed.ac.uk/projects/proof_planning.html




*Inductive proofs start with the application of an* induction rule*, which reduces the conjecture to some* base *and* step cases*. One of each is shown above. In the step case* rippling *reduces the difference between the* induction conclusion *and the* induction hypothesis *(see §1.2, p3 for more detail).* Fertilization *applies the induction hypothesis to simplify the rippled induction conclusion.*

**Fig. 1.** *ind_strat*: A Strategy for Inductive Proof

Proof planning has been extended to capture common causes of proof failure and ways to patch them [Ireland, 1992,Ireland & Bundy, 1996b]. With each proof method are associated some *proof critics*. Critics have a similar format to methods, but their preconditions specify situations in which the method's associated tactic will fail and instead of tactics they have instructions on patching a failed proof. Each of the critics associated with a method has a different precondition. These are used to decide on an appropriate patch. Most of the critics built to date have been associated with the ripple method, or rather with its principle sub-method, *wave*, which applies one ripple step (see §1.2, p3). Among the patches these critics suggest are: a generalisation of the current conjecture, the use of an intermediate lemma, a case split and using an alternative induction rule. The use of a critic to generalise a conjecture is illustrated in figure 8.

Proof planning has been tested successfully on a wide range of inductive and other theorems. These include conjectures arising from formal methods, *i.e.* from the verification, synthesis and transformation of both software and hardware. They include, for instance: the transformation of naive into tail recursive programs [Hesketh *et al*, 1992], the verification of a microprocessor, [Cantu *et al*, 1996], the synthesis of logic programs [Kraan *et al*, 1996], decision procedures [Armando *et al*, 1996] and the rippling tactic [Gallagher, 1993], resolution completeness proofs [Kerber & C. Sehn, 1997], proofs of limit theorems [Melis, 1998] and diagonalization proofs [Huang *et al*, 1995,Gow, 1997]. Critics are especially useful at coming up with, so called, 'eureka' steps, *i.e.* those proof steps that usually seem to require human intervention, for instance constructing appropriate induction rules, intermediate lemmas and generalisations [Lowe *et al*, 1998] and loop invariants [Ireland & Stark, 1997].

Proof planning has also been applied outwith mathematics to the computer games of bridge [Frank *et al*, 1992] and Go [Willmott *et al*, 1999] and also to problems of configuring systems from parts, [Lowe, 1991,Lowe *et al*, 1996].

| | |
|---|---|
| $ind\_strat(\boxed{x+1}^{\uparrow}, x)$ | $ind\_strat(\boxed{x+1}^{\uparrow}, x)$ *then* |
| | [ $ind\_strat(\boxed{y+1}^{\uparrow}, y)$ |
| | $ind\_strat(\boxed{y+1}^{\uparrow}, y)$ |
| | ] |
| Associativity of $+$ | Commutativity of $+$ |
| $x + (y + z) = (x + y) + z$ | $x + y = y + x$ |

*The associativity of + is an especially simple theorem, which can be proved with a single application of ind_strat from figure 1, using a one step induction rule on induction variable $x$. The commutativity of + is a bit more complicated. ind_strat is first applied using induction variable $x$ then in both the base and step cases there is a nested application of ind_strat using $y$. The first argument of ind_strat indexes the induction rule using the rippling concept of* wave-fronts *(see §1.2, p3). The second argument specifies the induction variable.*

**Fig. 2.** Special-Purpose Proof Plans

### 1.2 Rippling

Rippling is the key method in proof plans for inductive proof. Not only does it guide the manipulation of the induction conclusion to prepare it for the application of the induction hypothesis, but preparation for rippling suggests an appropriate induction rule and variable and different patterns of rippling failure suggest new lemmas and generalisations. Since it is also cited several times in the critique, we have included a brief introduction to rippling here.

Rippling is useful whenever there is a goal to be proved in the context of one or more 'givens'. Givens may be axioms, previously proved theorems, assumptions or hypotheses. It works by calculating the difference between the goal and the given(s) and then systematically reducing it. The similarities and differences between the goal and given(s) are marked with meta-level annotations. These annotations are shown graphically in figure 5, where the notation of rippling is explained. An example of rippling is given in figure 6.

$$rev(nil) = nil$$
$$rev(H :: T) = rev(T) <> (H :: nil)$$

$$qrev(nil, L) = L$$
$$qrev(H :: T, L) = qrev(T, H :: L)$$

*rev and qrev are alternative recursive functions for reversing a list. Each is defined by a one-step list recursion using a base and step case. :: is an infix list cons and <> an infix list append. rev is a naive reverse function and qrev a more efficient, tail-recursive function. The second argument of qrev is called an* accumulator*. This accumulator should be set to nil when qrev is first applied to reverse a list. Figure 4 states two theorems that relate these two functions.*

**Fig. 3.** Recursive Definitions of Two Reverse Functions

$$\forall k.\ rev(k) = qrev(k, nil) \quad (1)$$

$$\forall k, l.\ rev(k) <> l = qrev(k, l) \quad (2)$$

*Theorem (1) shows that rev and qrev output the same result from the same input when the accumulator of qrev is initialised to nil. Theorem (2) generalises theorem (1) for all values of this accumulator. Paradoxically, the more specialised theorem (1) is harder to prove. One way to prove it is first to generalise it to theorem (2).*

**Fig. 4.** Two Theorems about List Reversing Functions

## 2 Critique

Our critique of proof planning is organised along two dimensions. On the first dimension we consider four different aspects of proof planning: (1) its potential for advance formation, (2) its theorem proving power, (3) its support for interaction and (4) its methodology. On the second dimension, for each aspect of the first dimension we present: (a) the original dream, (b) the reality of current implementations and (c) the options available for overcoming obstacles and realising part of that original dream.

### 2.1 The Advance Formation of Plans

**The Dream:** In the original proposal for proof planning [Bundy, 1988] it was envisaged that the formation of a proof plan for a conjecture would precede its use to guide the search for a proof. Meta-level reasoning would be used to join general proof plans together by matching the preconditions of later ones to the effects of earlier ones. A tactic would then be extracted from the customised proof plan thus constructed. A complete proof plan would be sent to a tactic-based theorem prover where it would be unpacked into a formal proof with little or no search.

**The Reality:** Unfortunately, in practice, this dream proved impossible to realise. The problem is due to the frequent impossibility of checking the preconditions of methods against purely abstract formulae. For instance, the preconditions of rippling include checking for the presence of wave-fronts in the current goal formula, that a wave-rule matches a sub-expression of this goal and that any new inwards wave-fronts have a wave-hole containing a sink. These preconditions cannot be checked unless the structure of the goal is known in some detail. To know this structure requires anticipating the effects of the previous methods in the current plan. The simplest way to implement this is to apply each of the tactics of the previous methods in order.

Similar arguments hold for most of the other proof methods used by proof planners. This is especially true in applications to game playing where the different counter actions of the opposing players must be explored before a response can be planned, [Willmott *et al*, 1999]. So the reality is an interleaving of proof planning and proof execution. Moreover, the proof is planned in a *consecutive* fashion, *i.e.* the proof steps are developed starting at one end of the proof then proceeding in order. At any stage of the planning process only an initial or final segment of the object-level proof is known.

**The Options:** One response to this reality is to admit defeat, abandon proof planning and instead recycle the preconditions of proof methods as preconditions for the

**Given:** $rev(t) <> L = qrev(t, L)$

**Goal:** $rev(\boxed{h :: t}^{\uparrow}) <> \lfloor l \rfloor = qrev(\boxed{h :: t}^{\uparrow}, \lfloor l \rfloor)$

**Wave-Rules:**

$$rev(\boxed{H :: T}^{\uparrow}) \Rightarrow \boxed{rev(T) <> H :: nil}^{\uparrow} \quad (3)$$

$$qrev(\boxed{H :: T}^{\uparrow}, L) \Rightarrow qrev(T, \boxed{H :: L}^{\downarrow}) \quad (4)$$

$$(\boxed{X <> Y}^{\uparrow}) <> Z \Rightarrow X <> (\boxed{Y <> Z}^{\downarrow}) \quad (5)$$

*The example is drawn from the inductive proof of theorem (2) in figure 4. The given and the goal are the induction hypothesis and induction conclusion, respectively, of this theorem. Wave-rules (3) and (4) are annotated versions of the step cases of the recursive definitions of the two list reversing functions in figure 3. Wave-rule (5) is from the associativity of <>.*
*The grey boxes are called* wave-fronts *and the holes in them are called* wave-holes*. The wave-fronts in the goal indicate those places where the goal differs from the given. Those in the wave-rules indicate the differences between the left and right hand sides of the rules. The arrows on the wave-fronts indicate the direction in which rippling will move them: either outwards (↑) or inwards (↓). The corners, $\lfloor \ldots \rfloor$, around the l in the goal indicate a* sink*. A sink is one of rippling's target locations for wave-fronts; the other target is to surround an instance of the whole given with a wave-front.*
*The wave-rules are used to rewrite each side of the goal. The effect is to move the wave-fronts either to surround an instance of the given or to be absorbed into a sink. An example of this process is given in figure 6*

**Fig. 5.** The Notation of Rippling

application of tactics. Search can then be conducted in a space of condition/action production rules in which the conditions are the method preconditions and the actions are the corresponding tactics. Satisfaction of a precondition will cause the tactic to be applied thus realising the preconditions of subsequent tactics. Essentially, this strategy was implemented by Horn in the Oyster2 system [Horn, 1992]. The experimental results were comparable to earlier versions of *Clam*, *i.e.* if tactics are applied as soon as they are found to be applicable then proof planning conveys no advantage over Horn's production rule approach.

However, in subsequent developments some limited abstraction has been introduced into proof planning, in particular, the use of (usually second-order) meta-variables. In many cases the method preconditions can be checked on such partially abstract formulae. This allows choices in early stages of the proof to be delayed then made subsequently, *e.g.* as a side effect of unification of the meta-variables. We call this *middle-out reasoning* because it permits the non-consecutive development of a proof, *i.e.* instead of having to develop a proof from the top down or the bottom up we can start in the middle and work outwards. Middle-out reasoning can significantly reduce search by postponing a choice with a high branching factor until the correct branch can be determined. Figure 8 provides an example of middle-out reasoning.

Among the choices that can be successfully delayed in this way are: the witness of an existential variable, the induction rule, [Bundy *et al*, 1990a], an intermediate lemma and generalisation of a goal [Ireland & Bundy, 1996b,Ireland & Bundy, 1996a]. Each of these has a high branching factor – infinite in some cases. A single abstract

**Given:** $rev(t) <> L = qrev(t, L)$
**Goal:**

$$rev(\boxed{h :: t}^{\uparrow}) <> \lfloor l \rfloor = qrev(\boxed{h :: t}^{\uparrow}, \lfloor l \rfloor)$$

$$(\boxed{rev(t) <> h :: nil}^{\uparrow}) <> \lfloor l \rfloor = qrev(t, \lfloor h :: l \rfloor)$$

$$rev(t) <> \lfloor (h :: nil) <> l \rfloor = qrev(t, \lfloor h :: l \rfloor)$$

$$rev(t) <> \lfloor h :: l \rfloor = qrev(t, \lfloor h :: l \rfloor)$$

*The example comes from the step case of the inductive proof of theorem (2) from figure 4. Note that the induction variable $k$ becomes the constant $t$ in the given and the wave-front* $\boxed{h :: t}^{\uparrow}$ *in the goal. However, the other universal variable, $l$, becomes a first-order meta-variable, $L$, in the given, but a sink, $\lfloor l \rfloor$, in the goal. We use uppercase to indicate meta-variables and lowercase for object-level variables and constants.*
*The left-hand wave-front is rippled-out using wave-rule (3) from figure 5, but then rippled-sideways using wave-rule (5), where it is absorbed into the left-hand sink. The right-hand wave-front is rippled-sideways using wave-rule (4) and absorbed into the right-hand sink. After the left-hand sink is simplified, using the recursive definition of <>, the contents of the two sinks are identical and the goal can be fertilized with the given, completing the proof. Note that fertilization unifies the meta-variable $L$ with the sink $h :: l$.*
*Note that there is no point in rippling sideways* unless *this absorbs wave-fronts into sinks. Sinks mark the potential to unify wave-fronts with meta-variables during fertilization. Without sinks to absorb the wave-fronts, fertilization will* fail. *Such a failure is illustrated in figure 7*

**Fig. 6.** An Example of Rippling

branch containing meta-variables can simultaneously represent all the alternative branches. Incremental instantiation of the meta-variables as a side effect of subsequent proof steps will implicitly exclude some of these branches until only one remains. Even though the higher-order[2] unification required to whittle down these choices is computationally expensive the cost is far less than the separate exploration of each branch. Moreover, the wave annotation can be exploited to control higher-order unification by requiring wave-fronts to unify with wave-fronts and wave-holes to unify with wave-holes. We have exploited this middle-out technique to especially good effect in our use of critics, [Ireland & Bundy, 1996b].

Constraints have also been used as a least commitment mechanism in the $\Omega$mega proof planner [Benzmüller *et al*, 1997]. Suppose a proof requires an object with certain properties. The existence of such an object can be assumed and the properties posted as constraints. Such constraints can be propagated as the proof develops and their satisfaction interleaved with that proof in an opportunistic way [Melis *et al*, 2000b,Melis *et al*, 2000a].

Middle-out reasoning recovers a small part of the original dream of advance proof planning and provides some significant search control advantage over the mere use of method preconditions in tactic-based production rules.

[2] Only second-order unification is required for the examples tackled so far, but higher-order unification is required in the general case.

**Given:** $rev(t) = qrev(t, nil)$
**Goal:**

$$rev(\ h :: t\ ^{\uparrow}\ ) = qrev(\ h :: t\ ^{\uparrow}, nil)$$

$$(\ rev(t) <> h :: nil\ ^{\uparrow}\ ) = \underbrace{qrev(\ h :: t\ ^{\uparrow}, nil)}_{\textbf{blocked}}$$

*The example comes from the failed step case of the inductive proof of theorem (1) from figure 4. A particular kind of ripple failure is illustrated.*
*The left-hand wave-front can be rippled-out using wave-rule (3) and is then completely rippled. However, the right-hand wave-front* cannot *be rippled-sideways even though wave-rule (4) matches it. This is because there is no sink to absorb the resulting inwards directed wave-front. If the wave-rule was nevertheless applied then any subsequent fertilization attempt would fail.*
*Figure 8 shows how to patch the proof by a generalisation aimed to introduce a sink into the appropriate place in the theorem and thus allow the ripple to succeed.*

**Fig. 7.** A Failed Ripple

### 2.2 The Theorem Proving Power of Proof Planning

**The Dream:** One of the main aims of proof planning was to enable automatic theorem provers to prove much harder theorems than conventional theorem provers were capable of. The argument was that the meta-level planning search space was considerably smaller than the object-level proof search space. This reduction was partly due to the fact that proof methods only capture common patterns of reasoning, excluding many unsuccessful parts of the space. It was also because the higher-level methods, *e.g. ind_strat*, each cover many object-level proof steps. Moreover, the use of abstraction devices, like meta-variables, enables more than one proof branch to be explored simultaneously. Such search space reductions should bring much harder proofs into the scope of exhaustive search techniques.

**The Reality:** This dream has been partially realised. The reduced search space does allow the discovery of proofs that would be beyond the reach of purely object-level, automatic provers: for instance, many of the proofs listed in §1.1, p1.

Unfortunately, these very search reduction measures can also *exclude* the proofs of hard theorems from the search space, making them impossible to find. The reduced plan space is *incomplete*. Hard theorems may require uncommon or even brand new patterns of reasoning, which have not been previously captured in proof methods. Or they may require existing tactics to be used in unusual ways that are excluded by their current heuristic preconditions. Indeed, it is often a characteristic of a breakthrough in mathematical proof that the proof incorporates some new kind of proof method, *cf* Gödel's Incompleteness Theorems. Such proofs will not be found by proof planning using only already known proof methods, but could potentially be stumbled upon by exhaustive search at the object-level.

**The Options:** Firstly, we consider ways of reducing the incompleteness of proof planning, then ways of removing it.

We should strive to ensure that the preconditions of methods are as general as possible, for instance, minimising the use of heuristic preconditions, as opposed to preconditions that are *required* for the legal application of the method's tactic.

This will help ensure that the tactic is applied whenever it is appropriate and not excluded due to a failure to anticipate an unusual usage. A balance is required here since the absence of *all* heuristic preconditions may increase the search space to an infeasible size. Rather diligence is needed to design both tactics and their preconditions which generalise away from the particular examples that may have suggested the reasoning pattern in the first place.

The use of critics expands the search space by providing a proof patch when the preconditions of a method fail. In practice, critics have been shown to facilitate the proof of hard theorems by providing the 'eureka' steps, *e.g.* missing lemmas, goal generalisations, unusual induction rules, *etc*, that hard theorems often require [Ireland & Bundy, 1996b]. However, even with these additions, the plan space is still incomplete; so the problem is only postponed.

One way to *restore* completeness would be to allow arbitrary object-level proof steps, *e.g.* the application of an individual rule of inference such as rewriting, generalisation, induction, *etc*, with no heuristic limits on its application. Since such a facility is at odds with the philosophy of proof planning, its use would need to be carefully restricted. For instance, a proof method could be provided that made a single object-level proof step at random, but only when all other possibilities had been exhausted. Provided that the rest of the plan space was finite, *i.e.* all other proof methods were terminating, then this random method would occasionally be called and would have the same potential for stumbling upon new lines of proof that a purely object-level exhaustive prover does, *i.e.* we would not expect it to happen very often – if at all.

It is interesting to speculate about whether it would be possible to draw a more permanent benefit from such serendipity by learning a new proof method from the example proof. Note that this might require the invention of new meta-level concepts: consider, for instance, the learning of rippling from example object-level proofs, which would require the invention of the meta-level concepts of wave-front, wave-hole, *etc*.

Note that a first-order object-level proof step might be applied to a formula containing meta-variables. This would require the first-order step to be applied using higher-order unification, – potentially creating a larger search space than would otherwise occur. Also, some object-level proof steps require the specification of an expression, *e.g.* the witness of an existential quantifier, an induction variable and term, the generalisation of an expression. If these expressions are not provided via user interaction then infinite branching could be avoided by the use of meta-variables. So object-level rule application can introduce meta-variables even if they are not already present. These considerations further underline the need to use such object-level steps only as a last resort.

### 2.3 The Support for Interaction of Proof Planning

**The Dream:** Proof planning is not just useful for the automation of proof, it can also assist its interactive development. The language of proof planning describes the high-level structure of a proof and, hence, provides a high-level channel of communication between machine and user. This can be especially useful in a very large proof whose description at the object-level is unwieldy. The different proof methods chunk the proof into manageable pieces at a hierarchy of levels. The method preconditions and effects describe the relationships between and within each chunk and at each level. For instance, the language of rippling enables a proof state to be described in terms of differences between goals and givens, why it is important to reduce those differences and of ways to do so.

The preconditions and effects of methods and critics support the automatic analysis and patching of failed proof attempts. Thus the user can be directed to the

reasons for a failed proof and the kind of steps required to remedy the situation. This orients the user within a large and complex search space and gives useful hints as to how to proceed.

**The Reality:** The work of Lowe, Jackson and others in the *XBarnacle* system [Lowe & Duncan, 1997] shows that proof planning can be of considerable assistance in interactive proof. For instance, in Jackson's PhD work, [Jackson, 1999,Ireland *et al*, 1999], the user assists in the provision of goal generalisations, missing lemmas, *etc.* by instantiating meta-variables. However, each of the advantages listed in the previous section brings corresponding disadvantages.

Firstly, proof planning provides an enriched language of human/computer communication but at the price of introducing new jargon for the user to understand. The user of *XBarnacle* must learn the meaning of wave-fronts, flawed inductions, fertilization, *etc.*

Secondly, and more importantly, the new channel of communication assists users at the cost of restricting them to the proof planning search space; *cf* the discussion of incompleteness in §2.2, p7. For instance, *XBarnacle* users can get an explanation of why a method or critic did or did not apply in terms of successful or failed preconditions. They can over-ride those preconditions to force or prevent a method or critic applying. But their actions are restricted to the search space of tactics and critics. If the proof lies *outside* that space then they are unable to direct *XBarnacle* to find it.

**The Options:** The first problem can be ameliorated in a number of ways. Jargon can be avoided, translated or explained according to the expertise and preferences of the user. For instance, "fertilization" can be avoided in favour of, or translated into, the "use of the induction hypothesis". "Wave-front", on the other hand, has no such ready translation into standard terminology and must be explained within the context of rippling. Thus, although this problem can be irritating, it can be mitigated with varying amounts of effort.

The second problem is more fundamental. Since it is essentially the same as the problem of the incompleteness of the plan space, discussed in §2.2, p7, then one solution is essentially that discussed at the end of §2.2. New methods can be provided which apply object-level proof steps under user control. As well as providing an escape mechanism for a frustrated user this might also be a valuable device for system developers. It would enable them to concentrate on the parts of a proof they were interested in automating while using interaction to 'fake' the other parts.

The challenge is to integrate such object-level steps into the rest of the proof planning account. For instance, what story can we now tell about how such object-level steps exploit the effects of previous methods and enable the preconditions of subsequent ones?

### 2.4 The Methodology of Proof Planning

**The Dream:** Proof planning aims to capture common patterns of reasoning and repair in methods and critics. In [Bundy, 1991] we provide a number of criteria by which these methods and critics are to be assessed. These include expectancy[3], generality, prescriptiveness[4], simplicity, efficiency and parsimony. In particular, each method and critic should apply successfully in a wide range of situations (generality)

[3] Some degree of assurance that the proof plan will succeed.
[4] The less search required the better.

and a few methods and critics should generate a large number of proofs (parsimony). Moreover, the linking of effects of earlier methods and critics to the preconditions of later ones should enable a good 'story' to be told about how and why the proof plan works. This 'story' enables the expectancy criterion to be met.

**The Reality:** It is hard work to ensure that these criteria are met. A new method or critic may originally be inspired by only a handful of examples. There is a constant danger of producing methods and critics that are too fine tuned to these initial examples. This can arise both from a lack of imagination in generalising from the specific situation and from the temptation to get quick results in automation. Such over-specificity leads to a proliferation of methods and critics with limited applicability. Worse still, the declarative nature of methods may be lost as methods evolve into arbitrary code tuned to a particular problem set. The resulting proof planner will be brittle, *i.e.* will frequently fail when confronted with new problems. It will become increasing hard to tell an intelligible story about its reasoning. Critical reviewers will view the empirical results with suspicion, suspecting that the system has been hand-tuned to reproduce impressive results on only a handful of hard problems.

As the consequences of over-specificity manifest themselves in failed proof attempts so the methods and critics can be incrementally generalised to cope with the new situations. One can hope that this process of incremental generalisation will converge on a few robust methods and critics, so realising the original dream. However, a reviewer may suspect that this process is both infinite and non-deterministic, with each incremental improvement only increasing the range of the methods and critics by a small amount.

The opposite problem is caused by an over-general or missing precondition, permitting a method to apply in an inappropriate situation. This may occur, for instance, where a method is developed in a context in which a precondition is implicit, but then applied in a situation in which it is absent. This problem is analogous to feature interaction in telecomms or of predicting the behaviour of a society of agents.

**The Options:** The challenge is not only to adopt a development methodology that meets the criteria in [Bundy, 1991] but also *to be seen to do so*. This requires both diligence in the development of proof plans and the explicit demonstration of this diligence. Both aims can be achieved by experimental or theoretical investigations designed to test explicit hypotheses.

For instance, to test the criterion of generality, systematic and thorough application of proof planning systems should be conducted. This testing requires a large and diverse set of examples obtained from independent sources. The diversity should encompass the form, source and difficulty level of the examples. However, the generality of the whole system should not be obtained at the cost of parsimony, *i.e.* by providing lots of methods and critics 'hand crafted' to cope with each problematic example; so *each* of the methods and critics must be shown to be general-purpose. Unfortunately, it is not possible to test each one in isolation, since the methods and critics are designed to work as a family. However, it is possible to record how frequently each method and critic is used during the course of a large test run.

To meet the criterion of expectancy the specifications of the methods and critics should be declarative statements in a meta-logic. It should be demonstrated that the effects of earlier methods enable the preconditions of later ones and that the patches of critics invert the failed preconditions of the methods to which they are attached. Such demonstrations will deal both with the situation in which method preconditions/effects are too-specific (they will not be strong enough hypotheses)

and in which they are too general (they will not be provable). The work of Gallagher [Gallagher, 1993] already shows that this kind of reasoning about method preconditions and effects can be automated.

To meet the criterion of prescriptiveness the search space generated by rival methods needs to be compared either theoretically or experimentally; the method with the smaller search space is to be preferred. However, reductions in search space should not be obtained at the cost of unacceptable reductions in success rate. So it might be shown experimentally and/or via expectancy arguments that acceptable success rates are maintained. Reduced search spaces will usually contribute to increased efficiency, but it is possible that precondition testing is computationally expensive and that this cost more than offsets the benefits of the increased prescriptiveness, so overall efficiency should also be addressed.

## 3 Conclusion

In this paper we have seen that some of the original dreams of proof planning have not been fully realised in practice. We have shown that in some cases it has not been possible to deliver the dream in the form in which it was originally envisaged, for instance, because of the impossibility of testing method preconditions on abstract formulae or the inherent incompleteness of the planning search space. In each case we have investigated whether and how a lesser version of the original dream *can* be realised. This investigation both identifies the important benefits of the proof planning approach and points to the most promising directions for future research. In particular, there seem to be three important lessons that have permeated the analysis.

Firstly, the main benefits of proof planning are in facilitating a non-consecutive exploration of the search space, *e.g.* by 'middle-out' reasoning. This allows the postponement of highly branching choice points using least commitment mechanisms, such as meta-variables or constraints. Parts of the search space with low branching rates are explored first and the results of this search determine the postponed choices by side-effect, *e.g.* using higher-order unification or constraint solving. This can result in dramatic search space reductions. In particular, 'eureka' steps can be made in which witnesses, generalisations, intermediate lemmas, customised induction rules, *etc*, are incrementally constructed. The main vehicle for such non-consecutive exploration is critics. *Our analysis points to the further development of critics as the highest priority in proof planning research.*

Secondly, in order to increase the coverage of proof planners in both automatic and interactive theorem proving it is necessary to combine it with more brute force approaches. For instance, *it may be necessary to have default methods in which arbitrary object-level proof steps are conducted either at random or under user control.* One might draw an analogy with simulated annealing in which it is sometimes necessary to make a random move in order to escape from a local minimum.

Thirdly, frequent and systematic rational reconstruction is necessary to off-set the tendency to develop over-specialised methods and critics. This tendency is a natural by-product of the experimental development of proof planning as specifications are tweaked and tuned to deal with challenging examples. It is necessary to clean-up non-declarative specifications, merge and generalise methods and critics and to test proof planners in a systematic and thorough way. *The assessment criteria of [Bundy, 1991] must be regularly restated and reapplied.*

Despite the limitations exposed by the analysis of this paper, proof planning has been shown to have a real potential for efficient and powerful, automatic and interactive theorem proving. Much of this potential still lies untapped and our analysis has identified the priorities and directions for its more effective realisation.

**Afterword**

*I first met Bob Kowalski in June 1971, when I joined Bernard Meltzer's Metamathematics Unit as a research fellow. Bernard had assembled a world class centre in automatic theorem proving. In addition to Bob, the other research fellows in the Unit were: Pat Hayes, J Moore, Bob Boyer and Donald Kuehner; Donald was the co-author, with Bob, of SL-Resolution, which became the theoretical basis for Prolog.*

*Bob's first words to me were "Do you like computers? I don't!". This sentiment was understandable given the primitive computer facilities then available to us: one teletype with a 110 baud link to a shared ICL 4130 with 64k of memory. Bob went on to forsake the automation of mathematical reasoning as the main domain for theorem proving and instead pioneered logic programming: the application of theorem proving to programming. I stuck with mathematical reasoning and focussed on the problem of proof search control. However, I was one of the earliest adopters of Prolog and have been a major beneficiary of Bob's work, using logic programming both as a practical programming methodology and as a domain for formal verification and synthesis. I am also delighted to say that Bob has remained a close family friend for 30 years.*

*Happy 60th birthday Bob!*

## References


[Armando *et al*, 1996] Armando, A., Gallagher, J., Smaill, A. and Bundy, A. (3-5 January 1996). Automating the synthesis of decision procedures in a constructive metatheory. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics*, pages 5–8, Florida. Also in the Annals of Mathematics and Artificial Intelligence, 22, pp 259–79, 1998.

[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). $\Omega$mega: Towards a mathematical assistant. In McCune, W., (ed.), *14th International Conference on Automated Deduction*, pages 252–255. Springer-Verlag.

[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, A. (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.

[Bundy *et al*, 1990a] Bundy, A., Smaill, A. and Hesketh, J. (1990a). Turning eureka steps into calculations in automatic program synthesis. In Clarke, S. L.H., (ed.), *Proceedings of UK IT 90*, pages 221–6. IEE. Also available from Edinburgh as DAI Research Paper 448.

[Bundy *et al*, 1990b] Bundy, A., van Harmelen, F., Horn, C. and Smaill, A. (1990b). 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*, 1991] Bundy, A., van Harmelen, F., Hesketh, J. and Smaill, A. (1991). Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324. Earlier version available from Edinburgh as DAI Research Paper No 413.

[Cantu *et al*, 1996] Cantu, Francisco, Bundy, Alan, Smaill, Alan and Basin, David. (1996). Experiments in automating hardware verification using inductive proof planning. In Srivas, M. and Camilleri, A., (eds.), *Proceedings of the Formal Methods for Computer-Aided Design Conference*, number 1166 in Lecture Notes in Computer Science, pages 94–108. Springer-Verlag.

[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.

[Gallagher, 1993] Gallagher, J. K. (1993). *The Use of Proof Plans in Tactic Synthesis*. Unpublished Ph.D. thesis, University of Edinburgh.

[Gordon *et al*, 1979] Gordon, M. J., Milner, A. J. and Wadsworth, C. P. (1979). *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag.

[Gow, 1997] Gow, J. (1997). *The Diagonalization Method in Automatic Proof*. Undergraduate project dissertation, Dept of Artificial Intelligence, University of Edinburgh.

[Hesketh *et al*, 1992] Hesketh, J., Bundy, A. and Smaill, A. (June 1992). Using middle-out reasoning to control the synthesis of tail-recursive programs. In Kapur, Deepak, (ed.), *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 310–324, Saratoga Springs, NY, USA.

[Horn, 1992] Horn, Ch. (1992). Oyster-2: Bringing type theory into practice. *Information Processing*, 1:49–52.

[Huang *et al*, 1995] Huang, X., Kerber, M. and Cheikhrouhou, L. (1995). Adapting the diagonalization method by reformulations. In Levy, A. and Nayak, P., (eds.), *Proc. of the Symposium on Abstraction, Reformulation and Approximation (SARA-95)*, pages 78–85. Ville d'Esterel, Canada.

[Ireland & Bundy, 1996a] Ireland, A. and Bundy, A. (1996a). Extensions to a Generalization Critic for Inductive Proof. In McRobbie, M. A. and Slaney, J. K., (eds.), *13th International Conference on Automated Deduction*, pages 47–61. Springer-Verlag. Springer Lecture Notes in Artificial Intelligence No. 1104. Also available from Edinburgh as DAI Research Paper 786.

[Ireland & Bundy, 1996b] Ireland, A. and Bundy, A. (1996b). 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 & Stark, 1997] Ireland, A. and Stark, J. (1997). On the automatic discovery of loop invariants. In *Proceedings of the Fourth NASA Langley Formal Methods Workshop*. NASA Conference Publication 3356. Also available as Research Memo RM/97/1 from Dept of Computing and Electrical Engineering, Heriot-Watt University.

[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.

[Ireland *et al*, 1999] Ireland, A., Jackson, M. and Reid, G. (1999). Interactive Proof Critics. *Formal Aspects of Computing: The International Journal of Formal Methods*, 11(3):302–325. A longer version is available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/98/15.

[Jackson, 1999] Jackson, M. (1999). *Interacting with Semi-automated Theorem Provers via Interactive Proof Critics*. Unpublished Ph.D. thesis, School of Computing, Napier University.

[Kerber & C. Sehn, 1997] Kerber, Manfred and C. Sehn, Arthur. (1997). Proving ground completeness of resolution by proof planning. In Dankel II, Douglas D., (ed.), *FLAIRS-97, Proceedings of the 10th International Florida Artificial Intelligence Research Symposium*, pages 372–376, Daytona, Florida, USA. Florida AI Research Society, St. Petersburg, Florida, USA.

[Kerber, 1998] Kerber, Manfred. (1998). Proof planning: A practical approach to mechanized reasoning in mathematics. In Bibel, Wolfgang and H. Schmitt, Peter, (eds.), *Automated Deduction, a Basis for Application – Handbook of the German Focus Programme on Automated Deduction*, chapter III.4, pages 77–95. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[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, 1991] Lowe, Helen. (1991). Extending the proof plan methodology to computer configuration problems. *Artificial Intelligence Applications Journal*, 5(3). Also available from Edinburgh as DAI Research Paper 537.

[Lowe *et al*, 1996] Lowe, H., Pechoucek, M. and Bundy, A. (October 1996). Proof planning and configuration. In *Proceedings of the Ninth Exhibition and Symposium on Industrial Applications of Prolog.* Also available from Edinburgh as DAI Research Paper 859.

[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.

[Melis, 1998] Melis, E. (1998). The "limit" domain. In Simmons, R., Veloso, M. and Smith, S., (eds.), *Proceedings of the Fourth International Conference on Artificial Intelligence in Planning Systems*, pages 199–206.

[Melis *et al*, 2000a] Melis, E., Zimmer, J. and Müller, T. (2000a). Extensions of constraint solving for proof planning. In Horn, W., (ed.), *European Conference on Artificial Intelligence*, pages 229–233.

[Melis *et al*, 2000b] Melis, E., Zimmer, J. and Müller, T. (2000b). Integrating constraint solving into proof planning. In Ringeissen, Ch., (ed.), *Frontiers of Combining Systems, Third International Workshop, FroCoS'2000*, number 1794 in Lecture Notes on Artificial Intelligence, pages 32–46. Springer.

[Richardson *et al*, 1998] Richardson, J. D. C, Smaill, A. and Green, I. (July 1998). System description: proof planning in higher-order logic with Lambda-Clam. In Kirchner, Claude and Kirchner, Hélène, (eds.), *15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 129–133, Lindau, Germany.

[Willmott *et al*, 1999] Willmott, S., Richardson, J., Bundy, A. and Levine, J. (1999). An adversarial planning approach to Go. In Jaap van den Herik, H. and Iida, H., (eds.), *Computers and Games*, pages 93–112. 1st Int. Conference, CG98, Springer. Lecture Notes in Computer Science No. 1558.

**Schematic Conjecture:** $\forall k, l.\ F(rev(k), l) = qrev(k, G(l))$
**Given:** $F(rev(t), L) = qrev(t, G(L))$
**Goal:**

$$F(rev(\ h :: t\ ^{\uparrow}), \lfloor l \rfloor) = qrev(\ h :: t\ ^{\uparrow}, G(\lfloor l \rfloor))$$

$$F(\ rev(t) <> h :: nil\ ^{\uparrow}, \lfloor l \rfloor) = qrev(t,\ h :: G(\lfloor l \rfloor)\ ^{\downarrow})$$

$$rev(t) <> (\ h :: nil <> F'(\ rev(t) <> h :: nil\ ^{\uparrow}, \lfloor l \rfloor)\ ^{\downarrow}) = qrev(t,\ h :: G(\lfloor l \rfloor)\ ^{\downarrow})$$

$$rev(t) <> (\ h :: F'(\ rev(t) <> h :: nil\ ^{\uparrow}, \lfloor l \rfloor)\ ^{\downarrow}) = qrev(t,\ h :: G(\lfloor l \rfloor)\ ^{\downarrow})$$

$$rev(t) <> (\lfloor h :: l \rfloor) = qrev(t, \lfloor h :: l \rfloor)$$

**Meta-Variable Bindings:**

$$\lambda u, v.\ u <> F'(u, v)/F$$
$$\lambda u, v.\ v./F'$$
$$\lambda u.\ u./G$$

**Generalised Conjecture:** $\forall k, l.\ rev(k) <> l = qrev(k, l)$

*The example shows how the failed proof attempt in figure 7 can be analysed using a critic and patched in order to get a successful proof. The patch generalises the theorem to be proved by introducing an additional universal variable and hence a sink. Middle-out reasoning is used to delay determining the exact form of the generalisation. This form is determined later as a side effect of higher-order unification during rippling.*
*First a schematic conjecture is introduced. A new universal variable $l$ is introduced, in the right-hand side, at the point where a sink was required in the failed proof in figure 7. Since we are not sure exactly how $l$ relates to the rest of the right-hand side a second-order meta-variable $G$ is wrapped around it. On the left-hand side a balancing occurrence of $l$ is introduced using the meta-variable $F$. Note that $l$ becomes a first-order meta-variable $L$ in the given, but a sink $\lfloor l \rfloor$ in the goal.*
*Induction on $k$, rippling, simplification and fertilization are now applied, but higher-order unification is used to instantiate $F$ and $G$. If the schematic conjecture is now instantiated we see that the generalised conjecture is, in fact, theorem (2) from figure 4.*

**Fig. 8.** Patching a Failed Proof using Middle-Out Reasoning