The Use of Planning Critics in Mechanizing Inductive Proofs*

Andrew Ireland

Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
EH1 1HN

Abstract

Proof plans provide a technique for guiding the search for a proof in the context of tactical style reasoning. We propose an extension to this technique in which failure may be exploited in the search for a proof. This extension is based upon the concept of planning critics. In particular we illustrate how proof critics may be used to patch proof plans in the domain of inductive proofs.

1 Introduction

Proof plans [Bundy 88] guide the search for a proof in the context of tactical style reasoning [Gordon *et al* 79]. A proof plan contains a *tactic* together with a *proof rationale*. The tactic component specifies the low-level structure of a proof in terms of the object-level logic inference rules and is used to control the theorem prover. In contrast, the proof rationale, which is expressed in a meta-logic, captures the high-level structure of a proof. Proof plans are constructed from tactic specifications called *methods*. Using the meta-logic, a method expresses the preconditions under which a tactic is applicable and the effects of applying the tactic. The proof plan ideas have been implemented in a plan formation system called

*This paper was published in: *Logic Programming and Automated Reasoning LPAR'92*, St.Petersburg, Lecture Notes in Artificial Intelligence 624, Springer-Verlag 1992. The research reported in this paper was supported by SERC grant GR/F/71799.

CLAM [van Harmelen 89] and applied very successfully to the domain of inductive proofs [Bundy *et al* 91]. However, experience in automated inductive theorem proving [Boyer & Moore 88] has shown that failed proof attempts often hold the key to discovering a complete proof. Currently CLAM provides no mechanism for interpreting failure or even partial success in the search for a proof. In this paper we propose the use of explicit planning critics in providing the means of exploiting such negative information. Proof critics represent an extension to the hierarchy of inference rules, tactics and methods.

We outline a proof plan for induction in §2 and as motivation for our proposal we demonstrate the potential for exploiting partial success of this proof plan in §3. We show in §4 how this potential may be captured through the use of planning critics. The use of critics is extended is §5 where the need for global analysis is demonstrated.

2 A proof plan for inductive proofs

The overall proof plan for induction is composed of a hierarchy of subplans. Based upon a process called *recursion analysis*[Bundy *et al* 89], the preconditions of the induction plan determine the most promising form of induction. The planning of the resulting proof obligations are carried out by the *base_case* and *step_case* subplans. We concentrate here on the proof plan for the step case proofs.

The step_case plan decomposes into *ripple* and *fertilize*. The ripple plan has been identified as the central component of the induction plan and is described in detail in [Bundy *et al* 90b]. In essence, the ripple plan guides the rewriting of the induction conclusion so that it may be rewritten using the induction hypothesis. The fertilize plan controls the use of induction hypotheses. In planning step cases, CLAM has access to meta-level information in addition to the object-level goal structure. This meta-level information is represented by annotations of the object-level term structures. Annotations provide a means of describing the role particular terms and formulae play within a proof plan. To illustrate, consider the theorem:

$$\forall l : list(\tau).\, a :\ list(\tau).\, rotate(length(l), l <> a) = a <> l \qquad (1)$$

The function *rotate* takes a number n, and a list l, and returns a list constructed by concatenating the first n elements onto the remaining elements of l. List concatenation is denoted by the infix operator $<>$ and the function *length* returns the number of elements in a list. A proof of (1) follows by structural induction on the list l giving rise to one step case. The associated induction hypothesis takes the form:

$$\forall a' : list(\tau).\, rotate(length(v), v <> a') = a' <> v$$

while the induction conclusion is:

$$rotate(length(\boxed{u :: \underline{v}}^{\uparrow}), \boxed{u :: \underline{v}}^{\uparrow} <> \lfloor a \rfloor) = \lfloor a \rfloor <> \boxed{u :: \underline{v}}^{\uparrow} \qquad (2)$$

The annotated terms $\boxed{u :: \underline{v}}^{\uparrow}$ and $\lfloor a \rfloor$ are called *wave* and *sink* terms respectively. The role these annotations play in controlling the rewriting of the induction conclusion will be described shortly.

In (2) the wave terms correspond to the induction terms introduced by applying list induction, where :: is the infix list constructor. A wave term is composed of a *wave-front* and one or more *wave-holes*. Wave-fronts mark the mismatch between the induction conclusion and the induction hypothesis. Wave-holes are the subterms of the wave term which match against an induction hypothesis. In $\boxed{u :: \underline{v}}^{\uparrow}$ the wave-front is of the form $u :: \ldots$ while the subterm v occupies the wave-hole. The arrow indicates the direction in which the wave-front is to be moved within the expression tree.

Sinks are used to indicate term structure in the induction conclusion which may be matched against an universally quantified variable in the induction hypothesis.

The ripple plan embodies two strategies each of which rewrites the induction conclusion to the point where fertilization is applicable: The first strategy attempts to dominate the induction conclusion with a wave-front. If this is achieved then the associated wave-hole will match against the induction hypothesis. The second strategy attempts to move wave-fronts into sinks. Wave-fronts are directed either *upwards* or *downwards* depending which strategy is being employed.

The ripple plan restricts the rewriting of the induction conclusion to a syntactic class of rewrites known as *wave-rules* which also contain wave-fronts. Recursive definitions provide a rich source of wave-rules. For example, the definitions of the functions which appear in (1) give rise to the following wave-rules[1]:

$$length(\boxed{X :: \underline{Y}}^{\uparrow}) \;\Rightarrow\; \boxed{s(\underline{length(Y)})}^{\uparrow} \tag{3}$$

$$\boxed{X :: \underline{Y}}^{\uparrow} <> Z \;\Rightarrow\; \boxed{X :: \underline{Y <> Z}}^{\uparrow} \tag{4}$$

$$Y \neq nil \rightarrow rotate(\boxed{s(\underline{X})}^{\uparrow}, Y) \;\Rightarrow\; rotate(X, \boxed{tl(\underline{Y}) <> hd(\underline{Y}) :: nil}^{\downarrow}) \tag{5}$$

The functions *hd* and *tl* return the head and tail of a list respectively while *nil* is the empty list. The application of a wave-rule is guaranteed to make progress towards achieving the preconditions of fertilization. An important property of wave-rules is that they preserve what is known as the *skeleton* term structure. This invariant term structure corresponds to the "reflection" of the induction hypothesis in the conclusion and is essential for the success of fertilization. Wave-rules can be classified as being either *longitudinal* (3 and 4) or *transverse* (5). This classification reflects the rewriting strategies outlined above: Repeated longitudinal rippling is required in order to achieve the first strategy while the second requires transverse wave-rules.

As mentioned in §1, plans are constructed from tactic specifications called methods. The schema for methods is given in figure 1. The ripple plan is built up from

[1] We adopt the convention of using $\Rightarrow$ to denote rewriting and $\rightarrow$ to denote implication.

the *wave* method. The preconditions of the wave method determine the applicability of the wave-rules. The method which specifies the applicability of longitudinal

Name: method name and arguments.

Input: input slot is matched against the current goal sequent.

Preconditions: the preconditions determine the applicability of the method.

Effects: if the preconditions succeed then the effects slot is used to calculate the outputs.

Output: the output slot contains a possibly empty list of subgoal sequents.

Tactic: the program which controls the object-level theorem prover.

Figure 1: The Format of Methods

wave-rules is given in figure 2. In words, the preconditions to the wave method state:

> *There exists a wave-rule Rn with left-hand-side L which matches a sub-expression at position Pos within the goal G and the side condition C is provable given the hypothesis list H.*

Rewriting (2) using longitudinal wave-rules (3) and (4) gives rise to:

$$rotate(\boxed{s(\underline{length(v)})}^{\uparrow}, \boxed{u :: \underline{v <> \lfloor a \rfloor}}^{\uparrow}) = \lfloor a \rfloor <> \boxed{u :: \underline{v}}^{\uparrow} \tag{6}$$

Transverse rippling introduces an extra precondition to the wave method:

> *The least nested wave-front associated with the sub-expression at position Pos lies within the scope of a sink at position SPos in the goal G.*

A wave-front lies within the scope of a sink if the sink occurs in a distinct branch of the expression tree. Expressing this precondition by the meta-logical term $sinkable(Pos, G, SPos)$, then the wave method for transverse rippling may be represented as shown in figure 3. Since the wave term $\boxed{s(\underline{length(v)})}^{\uparrow}$ in (6) satisfies precondition $P4$ we are able to use transverse wave-rule (5) to rewrite the induction conclusion.

$$
\begin{array}{l}
method\ (\ wave(Pos, [Rn, D]), \\
\quad H \vdash G, \\
\quad [\ P1 : wave\_rule(Rn, long(D), C \rightarrow L \Rightarrow R) \wedge \\
\quad\ \ P2 : exp\_at(G, Pos, L) \wedge \\
\quad\ \ P3 : tautology(H \vdash C)], \\
\quad [replace(Pos, R, G, NewG)], \\
\quad [H \vdash NewG], \\
\quad wave(Pos, [Rn, D]) \\
\ \ ).
\end{array}
$$

Meanings of the meta-logical terms:

- $wave\_rule(Rn, T(D), Rule)$ means that Rn is the name of the wave-rule $Rule$ of type T and rewrite orientation D;
- $exp\_at(Exp, Pos, SubExp)$ means that $SubExp$ is the sub-expression in Exp at position Pos;
- $tautology(H \vdash C)$ is true when the condition C is provable given the hypothesis list H;
- $replace(Pos, R, G, NewG)$ means that $NewG$ is the result of replacing the sub-expression at position Pos in the goal G by R.

Figure 2: Wave method: *longitudinal–rippling*

3 Exploiting partial success

The applicability of a method depends upon all its associated preconditions succeeding. Here we consider the potential for exploiting partial success where previously the planning process would fail.

Currently the ripple plan is able to rewrite (6) into:

$$rotate(length(v), v <> \left\lfloor \boxed{\underline{a} <> u :: nil}^{\downarrow} \right\rfloor) = \lfloor a \rfloor <> \boxed{u :: \underline{v}}^{\uparrow}$$

While the wave-fronts on the left-hand-side have been *sunk* the wave-front on the right-hand-side prevents a complete match with the induction hypothesis from being achieved. By analysing the failure of the transverse wave method we find that it is due to preconditions $P1$ and $P2$: That is, there are no wave-rules to ripple $\boxed{u :: \underline{v}}^{\uparrow}$ into $\lfloor a \rfloor$ on the right-hand-side. The crucial point in the analysis is that $P4$ succeeds because the *blocked* wave-front lies within the scope of a sink. In

$$
\begin{array}{l}
method\ (\ wave(Pos, [Rn, D]), \\
\quad H \vdash G, \\
\quad [\ P1 : wave\_rule(Rn, trans(D), C \rightarrow L \Rightarrow R) \wedge \\
\quad\ \ P2 : exp\_at(G, Pos, L) \wedge \\
\quad\ \ P3 : tautology(H \vdash C) \wedge \\
\quad\ \ P4 : sinkable(Pos, G, \_)], \\
\quad [replace(Pos, R, G, NewG)], \\
\quad [H \vdash NewG], \\
\quad wave(Pos, [Rn, D]) \\
\ \ ).
\end{array}
$$

Figure 3: Wave method: *transverse–rippling*

§4 we examine the possibility of using this information to speculate and initiate the search for the lemma corresponding to the missing wave-rule.

In addition to the generation of lemmas, partial success may also motivate generalizations online. Note that (1) is a generalization of the theorem:

$$\forall l : list(\tau).\ rotate(length(l), l) = l \tag{7}$$

Applying the induction plan to (7) gives rise to a induction conclusion of the form:

$$rotate(\boxed{s(\underline{length(v)})}^{\uparrow}, \boxed{u :: \underline{v}}^{\uparrow}) = \boxed{u :: \underline{v}}^{\uparrow} \tag{8}$$

Now consider the rewriting of (8) using wave-rule (5). The transverse wave method fails because of precondition $P4$: That is, there are no sinks within the scope of the wave term $\boxed{s(\underline{length(v)})}^{\uparrow}$. The important point in the analysis, however, is that preconditions $P1$, $P2$ and $P3$ succeed. This analysis of partial success may be used to motivate generalization through the introduction of sinks, a possibility which will be explored further in §4.

In both the above examples we have identified the need for term structure without knowing its exact form. In the first case the term structure corresponds to the right-hand-side of a missing wave-rule. In the second case we know that the term must introduce a sink but we do not know the form the relationship between the sink and the existing term structure will take. These are particular instances of a general phenomenon which has given rise to a least commitment strategy called *middle-out reasoning* [Bundy *et al* 90a]. In essence, this strategy involves the use of meta variables (potentially higher-order) in speculating missing term structure. Such *speculation terms* get instantiated at a later point in the proof

planning process. Jane Hesketh in her thesis [Hesketh 91] developed the technique of middle-out reasoning in the context of generalizations such as the one outlined above. In §4 we present our extension to CLAM in which critics are used to trigger middle-out reasoning given the partial success of methods.

4 Capturing exceptions to proof plans

We now turn to representational issues and our proposal for using constructive planning critics in mechanizing the kind of meta-level analysis outlined in §3. A critic is a small program which analyses plan structures. Critics are "constructive" in the sense that as well as identifying problems in the plan structure they also take corrective action. The notion of constructive critics was first introduced in the NOAH system[Sacerdoti 77] and has had lasting effect on the classical AI planning paradigm [Wilkins 88]. While a method specifies precisely the conditions under which a tactic is applicable our proposed proof critics may be used to capture patchable exceptions to a tactic. The schema for critics is given in figure 4. Each

Name: the name of a critic corresponds to the name of its associated method.

Input: a pointer to the set of partial plans generated by the critic's associated method.

Preconditions: the preconditions determine the applicability of the critic.

Effects: the effects slot may be used to modify the goal and plan structure.

Figure 4: The Format of Critics

proof critic is associated with a method. The failure or partial success of the method activates its associated critics. The applicability of a critic is determined by the preconditions. A critic may perform the kind of local analysis of partial success outlined in §3. Alternatively, the preconditions of a critic can relate to the space of partial proof plans constructed by its associated method. Such global analysis is illustrated in §5. Running the effects of a critic may modify the goal and the plan structure.

We now present the critics for the transverse wave method based upon our precondition analysis. Consider again the example of the missing wave-rule in which only precondition $P4$ succeeded. The patch is achieved by speculating the right-hand-side of the missing wave-rule. Such a speculation requires the use of a higher-order meta variable. The speculation term is built up from the sink at position *SPos*

and the contents of the least nested wave-front associated with the sub-expression at position Pos in the goal G. The application of the wave-rule speculation gives rise to a goal of the form:

$$rotate(length(v), v <> \left\lfloor \boxed{\underline{a} <> u :: nil}^{\downarrow} \right\rfloor) = \left\lfloor \boxed{F(\underline{a}, u)}^{\downarrow} \right\rfloor <> v$$

where F is a higher-order meta variable. Constrained by the contents of the sink on the left-hand-side the fertilize plan instantiates F to be $\lambda x.\lambda y.x <> y :: nil$. Consequently, the missing wave-rule takes the form:

$$X <> \boxed{Y :: \underline{Z}}^{\uparrow} \Rightarrow \boxed{\underline{X} <> Y :: nil}^{\downarrow} <> Z$$

This partial success is captured in the critic given in figure 5.

$$\begin{array}{l} critic\ (\ wave, \\ \quad Plans, \\ \quad [\ preconds(Plans, [], [P4 : sinkable(Pos, G, SPos)])], \\ \quad [\ speculate\_lemma(Pos, SPos, G, Rn : Lemma), \\ \quad\ add\_wave\_rules(Lemma), \\ \quad\ insert\_method(Plans, [], wave(Pos, [Rn, \_]))] \\ \quad). \end{array}$$

Meanings of the meta-logical terms:

- $preconds(Plans, Pos, Preconds)$ is used to access the preconditions $Preconds$ recorded at position Pos within the partial plans referenced by $Plans$ (Note that [] denotes the root node associated with $Plans$);
- $speculate\_lemma(Pos, SPos, G, Rn : Lemma)$ means that Rn is the name of the lemma speculation $Lemma$ which is constructed from the goal G and the relative positions of the source and target wave-fronts given by Pos and $SPos$ respectively;
- $add\_wave\_rules(Lemma)$ records all possible wave-rules which the lemma $Lemma$ provides;
- $insert\_method(Plans, Pos, Method)$ splices in the method $Method$ at position Pos in the plan referenced by $Plans$.

Figure 5: Transverse wave critic: *missing wave-rule*

In the second example it is the lack of a sink which caused the failure of the transverse wave method. The patching of the goal requires the speculation of terms

which introduce sinks. Such speculations require the use of higher-order meta variables to bind new object-level variables (the sinks) to existing term structures. The patched goal takes the form:

$$rotate(\boxed{s(\underline{length(v)})}^{\uparrow}, F(\lfloor a \rfloor, \boxed{u :: \underline{v}}^{\uparrow})) = G(\lfloor a \rfloor, \boxed{u :: \underline{v}}^{\uparrow})$$

Note that the speculation of a sink on the left-hand-side is balanced with the speculation of a sink on the right-hand-side. Reactivating the ripple plan will instantiate F and G to be $\lambda x.\lambda y.y <> x$ and $\lambda x.\lambda y.x <> y$ respectively. Figure 6 illustrates how this kind of partial success of the transverse wave method can be captured by the preconditions of a critic.

$$\begin{array}{l}
critic\ (\ wave, \\
\quad Plans, \\
\quad [\ preconds(Plans, [], [\ P1 : wave_rule(Rn, _, _), \\
\qquad\qquad\qquad\qquad\qquad\qquad P2 : exp_at(G, Pos, _), \\
\qquad\qquad\qquad\qquad\qquad\qquad P3 : _])], \\
\quad [\ speculate_sink_terms(Rn, Pos, G, NewG), \\
\quad\ \ replace_goal(Plans, [], NewG), \\
\quad\ \ insert_method(Plans, [], wave(Pos, [Rn, D]))] \\
\quad).
\end{array}$$

Meanings of the meta-logical terms:

- $speculate\_sink\_terms(Rn, Pos, G, NewG)$ means that $NewG$ is a transformation of G in which speculation sink terms are introduced based upon the application of wave-rule Rn at position Pos in G;
- $replace\_goal(Plans, Pos, NewG)$ replaces the goal slot associated with the plan node at position Pos within $Plans$ (Note if the speculation introduces term structure which is not part of a wave-front then this term structure must be propagated back through the plan structure).

Figure 6: Transverse wave critic: *missing sinks*

5 Productive use of failed proof attempts

The framework outlined in §4 for patching proof plans is extended here to support global analysis of the planning process. In particular, we demonstrate how critics

may be used to suggest a case analysis by recognizing complementary failed partial proofs. We illustrate the idea by examining the proof of a synthesis theorem. A synthesis theorem may be expressed by the schema:

$$\forall inputs.\ \exists output.\ spec(inputs, output) \tag{9}$$

If we use a constructive logic [Martin-Löf 79] to prove a theorem of this form then the associated proof will yield a function, *prog*, which satisfies the logical relation $spec(inputs, prog(inputs))$. Proving synthesis theorems introduces the problem of calculating the value of existential witnesses. To be true to the synthesis paradigm we must incrementally construct the witnessing term hand–in–hand with the proof. Middle-out reasoning provides one approach to achieving this objective. We begin by replacing *output* in (9) by the meta-term $F(inputs)$, our initial speculation for the existential witness. This gives a new goal of the form:

$$\forall inputs.\ spec(inputs, F(inputs))$$

The application of the induction plan to this goal will incrementally build up the required definition of F.

Here we consider the synthesis of a partitioning function based upon Dijkstra's Dutch national flag problem [Dijkstra 76]. Informally we require a program to partition a list of red and blue objects into a pair of lists containing the component colours. A program to achieve this task may be specified formally by the theorem:

$$\begin{aligned}&\forall l : list(objects).\ \exists x : list(objects) \times list(objects).\\ &\qquad perm(l, fst(x) <> snd(x)) \wedge reds(fst(x)) \wedge blues(snd(x))\end{aligned}$$

The predicate $perm(p, q)$ is true when p and q are lists such that p is a permutation of q while the predicates $reds(p)$ and $blues(p)$ hold when p is a list of *red* and *blue* objects, respectively. In addition we assume the following closure property for elements of type *objects*:

$$\forall x : objects.\ red(x) \vee blue(x) \tag{10}$$

These predicates give rise to the following wave-rules:

$$perm(\boxed{W :: \underline{X}}^{\uparrow}, \boxed{W :: \underline{Y}}^{\uparrow} <> Z) \ \Rightarrow\ perm(X, Y <> Z) \tag{11}$$

$$perm(\boxed{W :: \underline{X}}^{\uparrow}, Y <> \boxed{W :: \underline{Z}}^{\uparrow}) \ \Rightarrow\ perm(X, Y <> Z) \tag{12}$$

$$reds(\boxed{X :: \underline{Y}}^{\uparrow}) \ \Rightarrow\ \boxed{red(X) \wedge \underline{reds(Y)}}^{\uparrow} \tag{13}$$

$$blues(\boxed{X :: \underline{Y}}^{\uparrow}) \ \Rightarrow\ \boxed{blue(X) \wedge \underline{blues(Y)}}^{\uparrow} \tag{14}$$

Introducing a speculation term $F(l)$ for x the initial goal is reduced to:

$$perm(l, fst(F(l)) <> snd(F(l))) \wedge reds(fst(F(l))) \wedge blues(snd(F(l)))$$

The proof plan for induction selects list induction. This has the effect of partially instantiating F to the primitive recursion schema which is the dual of structural induction on lists:

$$F(l) \equiv \textbf{if } l = nil \textbf{ then } G \\ \textbf{else let } u :: v = l \textbf{ in } H(u, v, F(v))$$

This recursion schema provides a wave-rule of the form:

$$F(\boxed{X :: \underline{Y}}^{\uparrow}) \Rightarrow \boxed{H(X, Y, \underline{F(Y)})}^{\uparrow} \tag{15}$$

The base_case proof plan instantiates G to be $\langle nil, nil \rangle$. As before we focus on the proof plan for the step case where the induction conclusion takes the form:

$$\begin{aligned} perm(\boxed{u :: \underline{v}}^{\uparrow}, fst(F(\boxed{u :: \underline{v}}^{\uparrow})) <> snd(F(\boxed{u :: \underline{v}}^{\uparrow}))) \quad \wedge \\ reds(fst(F(\boxed{u :: \underline{v}}^{\uparrow}))) \quad \wedge \\ blues(snd(F(\boxed{u :: \underline{v}}^{\uparrow}))) \end{aligned}$$

Using (15) the induction conclusion rewrites to:

$$\begin{aligned} perm(\boxed{u :: \underline{v}}^{\uparrow}, fst(\boxed{H(u, v, \underline{F(v)})}^{\uparrow}) <> snd(\boxed{H(u, v, \underline{F(v)})}^{\uparrow})) \quad \wedge \\ reds(fst(\boxed{H(u, v, \underline{F(v)})}^{\uparrow})) \quad \wedge \\ blues(snd(\boxed{H(u, v, \underline{F(v)})}^{\uparrow})) \end{aligned}$$

General properties of the projections *fst* and *snd* provide the wave-rules:

$$fst(\boxed{\langle J(X, Y, fst(\underline{Z})), L \rangle}^{\uparrow}) \quad \Rightarrow \quad \boxed{J(X, Y, \underline{fst(Z)})}^{\uparrow} \tag{16}$$

$$snd(\boxed{\langle L, K(X, Y, snd(\underline{Z})) \rangle}^{\uparrow}) \quad \Rightarrow \quad \boxed{K(X, Y, \underline{snd(Z)})}^{\uparrow} \tag{17}$$

Using (16) and (17) to rewrite the induction conclusion we get:

$$\begin{aligned} perm(\boxed{u :: \underline{v}}^{\uparrow}, \boxed{J(u, v, \underline{fst(F(v))})}^{\uparrow} <> \boxed{K(u, v, \underline{snd(F(v))})}^{\uparrow}) \quad \wedge \\ reds(\boxed{J(u, v, \underline{fst(F(v))})}^{\uparrow}) \quad \wedge \\ blues(\boxed{K(u, v, \underline{snd(F(v))})}^{\uparrow}) \end{aligned} \tag{18}$$

H is instantiated to $\lambda x.\lambda y.\lambda z.\langle J(x, y, fst(z)), K(x, y, snd(z)) \rangle$ as a side effect of this rewriting. At this point in the proof the partial definition of the program takes the form:

$$F(l) \equiv \textbf{if } l = nil \textbf{ then } \langle nil, nil \rangle$$
$$\textbf{else let } u :: v = l \textbf{ in } \langle J(u, v, \mathit{fst}(F(v))), K(u, v, \mathit{snd}(F(v))) \rangle$$

A choice point now arises within the ripple plan. Either wave-rule (11) or (12) may be applied to (18) but not both. If we choose (11) J is instantiated to $\lambda x.\lambda y.\lambda z.x :: z$ and the induction conclusion becomes:

$$\begin{aligned} perm(v, \mathit{fst}(F(v)) <> \boxed{K(u, v, \underline{\mathit{snd}(F(v))})}^{\uparrow}) \quad \wedge \\ reds(\boxed{u :: \underline{\mathit{fst}(F(v))}}^{\uparrow}) \quad \wedge \\ blues(\boxed{K(u, v, \underline{\mathit{snd}(F(v))})}^{\uparrow}) \end{aligned}$$

and using (13) we get:

$$\begin{aligned} perm(v, \mathit{fst}(F(v)) <> \boxed{K(u, v, \underline{\mathit{snd}(F(v))})}^{\uparrow}) \quad \wedge \\ \boxed{red(u) \wedge \underline{reds(\mathit{fst}(F(v)))}}^{\uparrow} \quad \wedge \\ blues(\boxed{K(u, v, \underline{\mathit{snd}(F(v))})}^{\uparrow}) \end{aligned}$$

Finally the propositional wave-rules:

$$\boxed{P \wedge \underline{Q}}^{\uparrow} \wedge R \Rightarrow \boxed{P \wedge \underline{(Q \wedge R)}}^{\uparrow}$$
$$P \wedge \boxed{Q \wedge \underline{R}}^{\uparrow} \Rightarrow \boxed{Q \wedge \underline{(P \wedge R)}}^{\uparrow}$$

enable us to rewrite the conclusion to the point where the fertilize plan is applicable. Fertilization instantiates K to be $\lambda x.\lambda y.\lambda z.z$ leaving the subgoal $red(u)$. The definition of the program at this point takes the form:

$$F(l) \equiv \textbf{if } l = nil \textbf{ then } \langle nil, nil \rangle$$
$$\textbf{else let } u :: v = l \textbf{ in } \langle u :: \mathit{fst}(F(v)), \mathit{snd}(F(v)) \rangle$$

Note that this program will only produce correct results when given a list of red objects. A further invocation of the induction plan to prove the remaining subgoal is ruled out by recursion analysis because *red* is not inductively defined. By backtracking to (18) and selecting wave-rule (12) the ripple and fertilize plans succeed but again leave an unproven subgoal. This time the subgoal is $blue(u)$ and the resulting program definition is:

$$F(l) \equiv \textbf{if } l = nil \textbf{ then } \langle nil, nil \rangle$$
$$\textbf{else let } u :: v = l \textbf{ in } \langle \mathit{fst}(F(v)), u :: \mathit{snd}(F(v)) \rangle$$

$$
\begin{array}{l}
critic\ (\ step\_case, \\
\quad Plans, \\
\quad [\ \exists\ C \in closures. \\
\quad\quad \forall\ D \in disjuncts(C). \\
\quad\quad\quad \exists\ G \in \{X : forms \mid failure(Plans, \_, X)\}.G = D], \\
\quad [\ insert\_method(Plans, [], case\_split(C))] \\
\quad).
\end{array}
$$

Meanings of meta-logical sorts and terms:

- *closures* is the set of known closure properties;
- *forms* is the set of all formulae;
- $disjuncts(Form)$ is the set of disjuncts of the formula *Form*;
- $failure(Plans, Pos, G)$ means that G is the failed goal at position Pos in the plan referenced by $Plans$.

Figure 7: Step_case critic: *missing case split*

The step_case plan fails at this point. Neither of the failed proof attempts alone suggests how the proof plan could be patched. Taken together, however, the failed subgoals suggest a case split based upon (10). This global analysis can be captured by the critic given in figure 7. The patched proof plan gives rise to the following complete definition for the program:

$$
\begin{array}{l}
F(l) \equiv \textbf{if}\ l = nil\ \textbf{then}\ \langle nil, nil\rangle \\
\qquad\qquad\qquad \textbf{else}\ \ \textbf{let}\ u :: v = l\ \textbf{in} \\
\qquad\qquad\qquad\qquad \textbf{case}\ colour(u)\ \textbf{of} \\
\qquad\qquad\qquad\qquad\qquad red : \langle u :: fst(F(v)), snd(F(v))\rangle \\
\qquad\qquad\qquad\qquad\qquad blue : \langle fst(F(v)), u :: snd(F(v))\rangle
\end{array}
$$

where *colour* is the decision procedure associated with (10). This kind of retrospective discovery of case splits appears to be a common feature of synthesis proofs, for instance, it arises in the synthesis of insertion sort among other list manipulation functions.

6 Conclusion

We have presented an extended framework for constructing proof plans. Our notion of proof critics allows us to reason about partial success and failure of plans. We have illustrated how both local and global analysis of failure may be used to patch partial plans. The uses of proof critics, however, are not limited to the analysis of explicit failures as presented here. For example, the use of *pre-emptive* critics in detecting potentially infinitely nested inductions seems a useful direction to explore. Proof critics complement the original concept of proof plans: While a proof plan characterizes a family of proofs a proof critic captures the patchable exceptions to the basic proof plan. The goal of patchability was emphasized in the original proof plans proposal [Bundy 88]. We believe that once implemented the proposal presented here will realize this goal.

Acknowledgements

I would like to thank Alan Bundy for his encouragement in the development of my ideas. I am grateful to Jane Hesketh for her ideas on generalization upon which I have drawn. My thanks also go to David Basin and Alan Smaill for their constructive comments on earlier versions of this paper.

References

[Boyer & Moore 88] R.S. Boyer and J.S. Moore. *A Computational Logic Handbook*. Academic Press, 1988. Perspectives in Computing, Vol 23.

[Bundy 88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.

[Bundy *et al* 89] A. Bundy, F. van Harmelen, J. Hesketh, A. Smaill, and A. Stevens. A rational reconstruction and extension of recursion analysis. In N.S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 359–365. Morgan Kaufmann, 1989. Also available from Edinburgh as DAI Research Paper 419.

[Bundy *et al* 90a] A. Bundy, A. Smaill, and J. Hesketh. Turning eureka steps into calculations in automatic program synthesis. In S.L.H.

Clarke, editor, *Proceedings of UK IT 90*, pages 221–6, 1990. Also available from Edinburgh as DAI Research Paper 448.

[Bundy *et al* 90b] A. Bundy, F. van Harmelen, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 132–146. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 459.

[Bundy *et al* 91] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.

[Dijkstra 76] E. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[Gordon *et al* 79] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer Verlag, 1979.

[Hesketh 91] J.T. Hesketh. *Using Middle-Out Reasoning to Guide Inductive Theorem Proving.* Unpublished PhD thesis, University of Edinburgh, 1991.

[Martin-Löf 79] Per Martin-Löf. Constructive mathematics and computer programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, Hanover, August 1979. Published by North Holland, Amsterdam. 1982.

[Sacerdoti 77] E.D. Sacerdoti. *A Structure for Plans and Behaviour.* Artificial Intelligence Series. North Holland, 1977. Also as SRI AI Technical note number 109, August 1975.

[van Harmelen 89] F. van Harmelen. The CLAM proof planner, user manual and programmer manual: version 1.4. Technical Paper TP-4, DAI, 1989.

[Wilkins 88] D.E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.