Journal of Automated Reasoning **16:** 79–111, 1996.

Productive Use of Failure in Inductive Proof*

ANDREW IRELAND** and ALAN BUNDY
Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, EH1 1HN Edinburgh, Scotland, UK
e-mail: air@cee.hw.ac.uk, a.bundy@ed.ac.uk

Abstract. Proof by mathematical induction gives rise to various kinds of eureka steps, e.g., missing lemmata and generalization. Most inductive theorem provers rely upon user intervention in supplying the required eureka steps. In contrast, we present a novel theorem-proving architecture for supporting the automatic discovery of eureka steps. We build upon *rippling*, a search control heuristic designed for inductive reasoning. We show how the failure if rippling can be used in bridging gaps in the search for inductive proofs.

Key words: Automated theorem proving, mathematical induction, proof patching.

1. Introduction

1.1. Motivation

G. H. Hardy [12] draws an analogy between a mathematician and a person observing "*a distant range of mountains.*" The key steps in a proof correspond to peaks in the mountain range. To see the proof, one must observe the complete mountain range, i.e., the ridges that link all the peaks. Hardy notes that sometimes the observer

> "*... can distinguish a ridge which vanishes in the distance, and conjectures that it leads to a peak in the clouds or below the horizon.*"

Conjecturing the unknown within a mathematical proof is often referred to as a *eureka step*. The discovery of eureka steps represents one of the major problems for automated theorem proving.

This is particularly true in the case of proof by mathematical induction. Reasoning about recursively defined structures or any form of repetition requires mathematical induction. Inductive proof is therefore crucial for reasoning about the correctness of computer systems. Consequently, techniques for automating inductive reasoning are more than just of academic interest; they have real practical significance to industry.

Inductive proof presents very challenging search control problems for automated reasoning, which give rise to various kinds of eureka steps:

* The research reported in this paper was supported by EPSRC grant GR/J/80702 and ARC grant 438.
** Current address: Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, EH14 4AS Edinburgh, Scotland, UK.

- **induction schemata selection:** the search for an inductive proof involves the selection of an appropriate induction schema. The induction schema is instantiated by the given conjecture and an induction variable in the conjecture. All universally quantified variables are candidate induction variables. While the set of possible induction variables is finite, the set of induction schemata is infinite. Consequently, the selection of the induction schema introduces an infinite branching point into the search space.
- **lemma discovery:** the word 'lemma' is used differently in inductive systems from the way it is used in non-inductive systems, e.g., predicate calculus provers. A lemma does not just mean an intermediate result which is generated as a side-effect of search. A lemma may be a separate theorem that is required to complete the original proof. The introduction of such a lemma requires the *cut rule* of inference, i.e.,

$$\frac{\Gamma, \alpha \vdash \beta \; \Gamma \vdash \alpha}{\Gamma \vdash \beta}$$

  The cut rule can be eliminated from predicate calculus [10] but not from inductive systems. Since the cut rule allows for the introduction of arbitrary new formulae, lemma discovery introduces an infinite branching point into the search space.
- **generalization:** paradoxically, it is sometimes necessary to generalize a conjecture in order for an inductive proof to succeed. Generalization is problematic, however, since it also requires the *cut rule* of inference. Like lemma discovery, generalization therefore introduces an infinite branching point into the search space. In choosing the cut formula we must guard against over-generalization, i.e., attempting to prove a non-theorem.
- **case analyses:** conditional proof is an important technique. Control is necessary, however, to guard against arbitrary case analysis leading to divergence in the search for a proof.

Most inductive theorem provers rely upon user intervention in spotting the need for and in constructing such eureka steps. In contrast, we present a theorem-proving architecture that supports the automatic discovery of eureka steps through the analysis of failed proof attempts. Our approach builds upon *rippling* [7, 4], a heuristic that plays a pivotal role in guiding the search for an inductive proof. We demonstrate how the constraints rippling places on the search space and its declarative nature enable us to automatically patch failed proof attempts through the construction of appropriate eureka steps. In particular, we show how the systematic analysis of the failure of rippling can be used in the selection of induction schemas and the conjecturing of lemmata, generalizations, and case analyses.

1.2. Background

It has been shown how the common structure that defines a family of proofs can be expressed as a *proof plan* [5]. This common structure can be exploited in the search for particular proofs. A proof plan has two complementary components: a proof *method* and a proof *tactic*. By prescribing the structure of a proof at the level of primitive inferences, a tactic [11] provides the guarantee part of the proof. In contrast, a method provides a more declarative explanation of the proof by means of *preconditions*. Each method has associated *effects*. The execution of the effects simulates the application of the corresponding tactic. Theorem proving in the proof planning framework is a two-phase process:

1. Tactic construction is by a process of method composition: Given a goal, an applicable method is selected. The applicability of a method is determined by evaluating the method's preconditions. The method effects are then used to calculate subgoals. This process is applied recursively until no more subgoals remain. Because of the one-to-one correspondence between methods and tactics, the output from this process is a composite tactic tailored to the given goal.
2. Tactic execution generates a proof in the object-level logic. Note that no search is involved in the execution of the tactic. All the search is taken care of during the planning process.

The real benefits of having separate planning and execution phases become apparent when a proof attempt fails. The declarative nature of method preconditions provides a basis for using failure productively. In [6] an extension to the proof planning framework is proposed in which proof *critics* are introduced in order to complement proof methods. The role of the proof critic is to capture patchable exceptions to the proof method. Since a proof method may fail in various ways, each method may be associated with a number of critics. A critic has preconditions and patches. The preconditions of a critic characterise an interesting failure, while the patch prescribes how the failure can be overcome. Critics are able to analyze partial proofs and have a global effect upon the proof process. As well as patching failed proof attempts, critics have also been applied to the problem of identifying and correcting faulty conjectures [17].

1.3. Overview

In this paper we present the use of the critics mechanism [16] in systematically analyzing the failure of the ripple heuristic. In Section 2 a proof plan for induction is outlined in which we emphasize the pivotal role played by rippling. Section 3 forms the core of the paper. It focuses on the ripple heuristic and how its failure can be interpreted productively in the search for inductive proofs. Search control issues relating to the selection and application of patches are discussed in Section 4. A comparison with related techniques is presented in Section 5. The implementation of the proof critics for rippling is outlined in Section 6 togeth-

er with a discussion of test results. Finally in Sections 7 and 8 we draw our conclusions and outline our plans for future work.

2. A Proof Plan for Induction

In the context of goal-directed proof, the application of a rule of induction generates base- and step-case subgoals. Example induction rules are presented in Figure 1. We will consider rule (3) in detail. The goal-directed application of (3) generates two subgoals, one base-case, i.e.,

$$P(nil)$$

and one step-case, i.e.,

$$P(t) \rightarrow P(h :: t)$$

In the step-case $P(h :: t)$ is called the *induction conclusion* and $P(t)$ is called the *induction hypothesis*. The method level structure of our proof plan for induction is given in Figure 2. The role of each method is outlined in the following sections. The heuristic that is embodied within the ripple method underpins the induction proof plan as a whole. That is, while the ripple heuristic controls the rewriting of step-case goals, it also constrains the search for induction rules and the application of induction hypotheses.

$$\frac{P(0)\ \forall n : nat.P(n) \rightarrow P(s(n))}{\forall x : nat.P(x)} \quad (1)$$

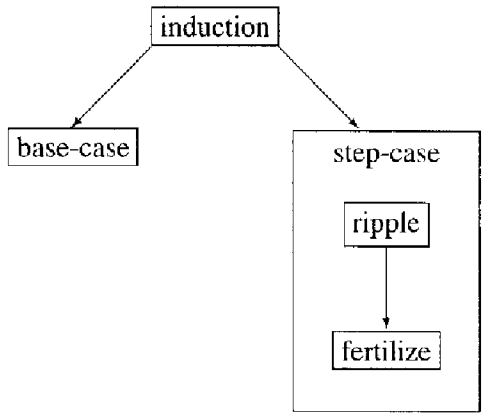
$$\frac{P(0)\ P(s(0))\ \forall n : nat.P(n) \rightarrow P(s(s(n)))}{\forall x : nat.P(x)} \quad (2)$$

$$\frac{P(nil)\ \forall h : T.\forall t : list(\tau).P(t) \rightarrow P(h :: t)}{\forall l : list(\tau).P(l)} \quad (3)$$

$$\frac{P(nil)\ \forall h : T.P(h :: nil)\ \forall h_1, h_2 : T.\forall t : list(\tau).\ P(t) \rightarrow P(h_1 :: h_2 :: t)}{\forall l : list(\tau).\ P(l)} \quad (4)$$

Four example rules of mathematical induction are shown. The first and the third are the structural inductions for natural numbers and lists respectively. Note that s *is the successor function for the natural numbers while* nil *and* :: *are the empty list and list constructor respectively. The second and fourth rules are examples of two step inductions for the naturals and lists respectively.*

Figure 1. Example rules of mathematical induction.



The proof plan for induction consists of the application of a rule of induction followed by one or more base- and step-cases. Only one of each is shown here. In the step-case, ripple controls the rewriting of the induction conclusion so that fertilize can apply the induction hypotheses.

Figure 2. A proof plan for mathematical induction.

The ripple heuristic is based upon the observation that a copy of the induction hypothesis is embedded within the induction conclusion. Following Hutter [15] we call this the *skeleton* term structure. The role of the ripple method is to eliminate the difference between the conclusion and hypothesis while preserving the skeleton term structure. Meta-level annotations are used to express this control information. To illustrate, the annotated version of (3) takes the form

$$\frac{\mathrm{P}(\mathtt{nil})\ \forall \mathtt{h} : \mathrm{T}.\forall \mathtt{t} : \mathtt{list}(\tau).\ \mathrm{P}(\mathtt{t}) \rightarrow \mathrm{P}(\boxed{\mathtt{h} :: \underline{\mathtt{t}}}^{\uparrow})}{\forall \mathtt{l} : \mathtt{list}(\tau).\ \mathrm{P}(\mathtt{l})}$$

The annotated term $\boxed{\mathtt{h} :: \underline{\mathtt{t}}}^{\uparrow}$ is called a *wave*. Note that the box and underlining are meta-level constructions. The underlined object-level term (i.e., t) is called the *wave-hole*. The object-level term structure within the box but excluding the wave-hole (i.e., $\mathtt{h} :: \ldots$) is called the *wave-front*. Wave-fronts highlight the difference between the conclusion and the hypothesis. The arrow is used to indicate the direction of movement of the wave-front within the expression tree of the induction conclusion. The need for directed wave-fronts will be explained in Section 2.3.1. In the following sections we outline the role each component of the proof plan plays in the search for inductive proof.

2.1. Induction Method

As mentioned above, the constraints that the ripple heuristic places upon step-case proof attempts also constrain the choice of induction rule. To illustrate, consider the conjecture

$$\forall t : list(\tau).\ rev(rev(t)) = t \tag{5}$$

where rev denotes list reversal. Note that rev is defined in terms of list concatenation, i.e., <>. The definitions of <> and rev give rise to the following rewrite rules:[1]

$$nil <> Z \Rightarrow Z \tag{6}$$

$$X :: Y <> Z \Rightarrow X :: (Y <> Z) \tag{7}$$

$$rev(nil) \Rightarrow nil \tag{8}$$

$$rev(X :: Y) \Rightarrow rev(Y) <> X :: nil \tag{9}$$

The manipulation of wave-fronts is performed by a syntactic class of rewrite rules called *wave-rules*. Wave-rules are guaranteed to preserve the skeleton term structure while making progress towards applying an induction hypothesis. Rewrite rules (7) and (9) provide a set of wave-rules that include

$$\boxed{X :: \underline{Y}^{\uparrow} <> Z} \Rightarrow \boxed{X :: \underline{(Y <> Z)}}^{\uparrow} \tag{10}$$

$$rev(\boxed{X :: \underline{Y}}^{\uparrow}) \Rightarrow \boxed{\underline{rev(Y)} <> X :: nil}^{\uparrow} \tag{11}$$

The general notion of a wave-rule is explained in Section 2.3.1. In terms of the induction method, wave-rules provide a mechanism for indexing appropriate rules of induction, i.e., induction rules which will enable the application of wave-rules. In the case of (5) induction rule (3) is suggested by the $rev(\boxed{X :: \underline{Y}}^{\uparrow})$ wave-term appearing on the *LHS* of (11).

2.2. Base-Case Method

The base-case method performs simplification through the use of definitional rewrite rules. For example, a proof of (5) by induction rule (3) generates a base-case subgoal of the form

$$rev(rev(nil)) = nil$$

Two applications of (8) reduces this subgoal to an identity, i.e.,

$$nil = nil$$

2.3. Step-Case Method

The ripple and fertilize methods form the step-case method. The role of the ripple method is to rewrite the induction conclusion so that the induction hypothesis can be applied by the fertilize method.[2]

2.3.1. *Ripple Method*

Wave-fronts and wave-rules greatly constrain the search for step-case proofs. The ripple method controls the application of wave-rules. To illustrate, consider the step-case proof obligation associated with the proof of (5). Induction rule (3) gives an induction hypothesis of the form

$$rev(rev(t)) = t \tag{12}$$

while the initial induction conclusion takes the form

$$rev(rev(\boxed{h :: \underline{t}}^{\uparrow})) = \boxed{h :: \underline{t}}^{\uparrow}$$

Note that the wave-fronts are associated with the induction rule so that the process of annotating an induction conclusion is automatic. The ripple method restricts the rewriting of an induction conclusion to wave-rules. The application of a wave-rule requires that both the object-level and meta-level term structure match. To illustrate, the application of wave-rule (11) to the initial induction conclusion gives rise to

$$rev(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow}) = \boxed{h :: \underline{t}}^{\uparrow}$$

Wave-rules are not restricted to recursive definitions; for instance,

$$rev(\boxed{\underline{Y} <> X :: nil}^{\uparrow}) \Rightarrow \boxed{X :: \underline{rev(Y)}}^{\uparrow} \tag{13}$$

$$\boxed{X :: \underline{Y}}^{\uparrow} = \boxed{X :: \underline{Z}}^{\uparrow} \Rightarrow Y = Z \tag{14}$$

are wave-rules that are derived from lemmata about rev, <>, and list equality. From (13), the induction conclusion becomes

$$\boxed{h :: \underline{rev(rev(t))}}^{\uparrow} = \boxed{h :: \underline{t}}^{\uparrow}$$

Finally, by (14) we obtain

$$rev(rev(t)) = t \tag{15}$$

All wave-fronts have been eliminated, so the rippling of the conclusion is complete. The conclusion is said to be *fully rippled* and ready to be fertilized.

The strategy as illustrated above is called *longitudinal* rippling. The aim of the strategy is to manipulate the wave-fronts so that they dominate the skeleton

term structure of the induction conclusion. Note that the above example is a degenerate case in which the wave-fronts are completely eliminated; that is, they are said to *peter out*.

An alternative strategy is called *transverse* rippling. This strategy exploits the fact that universally quantified variables in the induction hypothesis can be instantiated differently from the corresponding variables in the induction conclusion. To illustrate, consider the following conjecture:

$$\forall t,\ l : list(\tau).\ rev(t) <> l = qrev(t, l) \tag{16}$$

where qrev is the tail recursive version of rev. The definition of qrev provides the following wave-rule:

$$qrev(\boxed{X :: \underline{Y}}^{\uparrow}, Z) \Rightarrow qrev(Y, \boxed{X :: \underline{Z}}^{\downarrow}) \tag{17}$$

Note the change in direction of the wave-fronts. Rippling allows for upward directed wave-fronts to be turned downward but not vice versa. This restriction enables the use of bidirectional rewrite rules without the risk of looping. Other systems, such as Nqthm [2], rely upon the user to indicate which direction a bidirectional rewrite rule should be used. In proving (16), wave-rules (11) and (17) suggest induction on the variable t using rule (3). We concentrate here on the step-case that gives rise to an induction hypothesis of the form[3]

$$rev(t) <> L = qrev(t, L) \tag{18}$$

In order to exploit universally quantified variables, additional meta-level annotations are used. The meta-level construction $\lfloor \ldots \rfloor$ is used to indicate an object-level term within the conclusion, which corresponds to universally quantified variable in the hypothesis, e.g.,

$$rev(\boxed{h :: \underline{t}}^{\uparrow}) <> \lfloor l \rfloor = qrev(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor) \tag{19}$$

These meta-level terms are called *sinks*. The aim of the transverse strategy is to move wave-fronts into sinks. The rewriting of (19), the initial induction conclusion, using wave-rules (11) and (17) gives

$$\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow} <> \lfloor l \rfloor = qrev\Big(t, \Big\lfloor \boxed{h :: \underline{l}}^{\downarrow} \Big\rfloor\Big) \tag{20}$$

Now we need to use the associativity of <>, which gives rise to a wave-rule of the form

$$\boxed{\underline{X <> Y}}^{\uparrow} <> Z \Rightarrow X <> \boxed{Y <> \underline{Z}}^{\downarrow} \tag{21}$$

The application of (21) to the *LHS* of (20) gives rise to

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

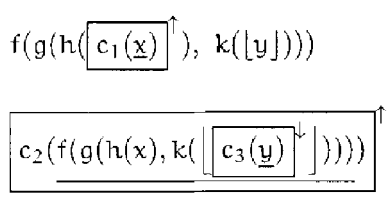


Figure 3 Rippling: *general pattern.*

Note that both wave-fronts now appear within the scope of sinks. In general, to exploit sinks, the application of a transverse wave-rule may need to be followed by further applications of longitudinal wave-rules. This is called *rippling-in* and typically corresponds to the folding of definitions. Finally, simplification[4] of the wave-front on the *LHS* using rewrite rules (6) and (7) gives

$$\mathrm{rev}(t) <> \left\lfloor \boxed{h :: l}^{\downarrow} \right\rfloor = \mathrm{qrev}\left(t, \left\lfloor \boxed{h :: l}^{\downarrow} \right\rfloor\right) \tag{22}$$

The rippling of the induction conclusion is now complete. The general pattern of the longitudinal and transverse strategies is presented in Figure 3. A detailed description of rippling appears in [4].

2.3.2. *Fertilize Method*

The fertilize method controls the application of induction hypotheses. In the case of conjecture (5) fertilization is trivial, since the hypothesis (12) and the fully rippled conclusion (15) are identical. At most, matching is required, as illustrated in conjecture (16) where the matching of the hypothesis (18) against the fully rippled conclusion (22) instantiates L to be $h :: l$.

3. Productive Use of Failure

We now consider how an inductive proof might fail. In particular we consider failure of the ripple method. To do this, however, we must consider the ripple method in more detail. The actual application of wave-rules is controlled by the *wave* method. The ripple method iterates over the wave method. The preconditions for the application of a longitudinal wave-rule are presented in Figure 4, while those for applying transverse wave-rules are presented in Figure 5.

In the following four sections we systematically analyze the ways in which preconditions of the wave method can fail, and we present the patches suggested in each case.

Preconditions:

1. wave_term(Conc, Poc, LHS)
2. wave_rule_match(Rn, long(D), Cond → LHS ⇒ RHS, Subs)
3. tautology(Hyps ⊢ Cond)

Definition of meta-logical terms:

- Hyps and Conc denote the current hypotheses and conclusion respectively;
- wave_term(T, P, W) means that W is the wave-term at position P within a term T;
- wave_rule_match(N, T(D), C → L ⇒ R, S) means that N is the name of a wave-rule of type T with rewrite direction D, which unifies with the term L. S is the set of substitutions for any higher-order meta variables instantiated by the unification.
- tautology(S) is true when the sequent S is a tautology.

Figure 4. Wave method: *longitudinal-rippling*.

Preconditions:

1. wave_term(Conc, Pos, LHS)
2. wave_rule_match(Rn, trans(D), Cond → LHS ⇒ RHS, Subs)
3. tautology(Hyps ⊢ Cond)
4. sinkable(RHS)

Definition of meta-logical terms:

- sinkable(T) term T contains a wave-front that is directed towards a sink.
- all other terms are as defined in Figure 4.

Figure 5. Wave method: *transverse-rippling*.

3.1. Induction Revision Critic

As mentioned in Section 2.1, rippling can be used in the selection of induction rules. We now examine how the ripple heuristic can select an inappropriate induction rule. Consider the conjecture

$$\forall l,\ l : list(\tau).\ even(length(t <> l)) \leftrightarrow even(length(l <> t)) \tag{23}$$

where the predicate *even* holds for the even natural numbers and *length* computes the length of a list. The definitions of *even* and *length* provide wave-rules that include

$$even(\boxed{s(s(\underline{X}))}^{\uparrow}) \Rightarrow even(X) \tag{24}$$

$$length(\boxed{X :: \underline{Y}}^{\uparrow}) \Rightarrow \boxed{s(\underline{length(Y)})}^{\uparrow} \tag{25}$$

We assume an additional lemma that prodives a wave-rule of the form

$$\mathrm{length}(X <> \boxed{Y :: Z}^{\uparrow}) \Rightarrow \boxed{s(\underline{\mathrm{length}(X <> Z)})}^{\uparrow} \tag{26}$$

In proving (23), wave-rule (10) suggests[5] an induction on t using rule (3). The initial induction conclusion takes the form

$$\mathrm{even}(\mathrm{length}(\boxed{h :: \underline{t}}^{\uparrow} <> l)) \leftrightarrow \mathrm{even}(\mathrm{length}(l <> \boxed{h :: \underline{t}}^{\uparrow}))$$

Using wave-rules (10), (25), and (26) the conclusion rewrites to give

$$\underbrace{\mathrm{even}(\boxed{s(\underline{\mathrm{length}(t <> l)})}^{\uparrow})}_{\text{blocked}} \leftrightarrow \underbrace{\mathrm{even}(\boxed{s(\underline{\mathrm{length}(l <> t)})}^{\uparrow})}_{\text{blocked}} \tag{27}$$

No more wave-rules are applicable, so the wave method fails to apply. This corresponds to the failure of precondition 2 of the wave method. However, the wave-fronts are preventing the fertilize method from applying. To patch this failure, we could attempt to introduce additional wave-fronts by refining our chosen induction rule. For this to be profitable, there must exist a wave-rule that partially matches with one of the blocked wave-terms, i.e., a wave-rule with an *LHS* of the following form:

$$\mathrm{even}(\boxed{s(\boxed{F_1(\underline{\mathrm{length}(Y <> \boxed{F_2(\underline{Z})}^{\uparrow})})}^{\uparrow})}^{\uparrow}) \Rightarrow \cdots \tag{28}$$

where F_1 and F_2 are second-order meta variables. Note that we use dotted boxes the denote *potential* wave-fronts, i.e., wave-fronts that may or may not exist. The *LHS* of wave-rule (24) unifies with the *LHS* of (28), instantiating F_1 and F_2 to be $\lambda x.s(x)$ and $\lambda x.x$, respectively. Discussion of higher-order unification is delayed until Section 4.3.

This success suggests the need for an additional wave-front in overcoming the blocked goal. This analysis is expressed as a critic in Figure 6. The application of the critic to goal (27) suggests that an additional wave-front of the form $\boxed{s(\ldots)}^{\uparrow}$ is required. The rippling-in of the composite wave-front, i.e., $\boxed{s(s(\ldots))}^{\downarrow}$, using wave-rules (25) and (10), suggests a two-step induction, i.e., the initial selection of rule (3) is replaced by rule (4). This gives rise to a revised induction conclusion of the form

$$\mathrm{even}(\mathrm{length}(\boxed{h_1 :: h_2 :: \underline{t}}^{\uparrow} <> l)) \leftrightarrow \mathrm{even}(\mathrm{length}(l <> \boxed{h_1 :: h_2 :: \underline{t}}^{\uparrow}))$$

This revision enables the induction conclusion to be fully-rippled, i.e.,

$$\mathrm{even}(\mathrm{length}(\boxed{h_1 :: (\underline{\boxed{h_2 :: \underline{t}}^{\uparrow} <> l)})}^{\uparrow})) \leftrightarrow \mathrm{even}(\boxed{s(\underline{\mathrm{length}(l <> \boxed{h_2 :: \underline{t}}^{\uparrow})})}^{\uparrow})$$

Preconditions:

1. Precondition 1 of wave method holds, i.e.,
   $wave\_term(\_, Pos, WaveTerm)$
2. Preconditions 2 and 3 of wave method fail, i.e.,
   no matching wave-rule
   no condition to check
3. $partial\_wave\_rule\_match(WaveTerm, RevisedWaveTerm)$

Patch:

1. $revise\_induction(RevisedWaveTerm, Pos, Conc, IndRule)$
2. $propagate\_induction(Plan, IndRule)$

Definition of meta-logical terms:

- Conc and Plan denote the current conclusion and plan structure, respectively;
- $partial\_wave\_rule\_match(T, W)$ holds when W is the wave-front suggested by a partial wave-rule match with the term T;
- $revise\_induction(W, P, G, R)$ R is the induction rule suggested by the wave-term W at position P within the goal G;
- $propagate\_induction(P, R)$ propagates R, the revised induction suggestion through the plan structure P.

Figure 6. Wave critic: *induction revision.*

$$even(\boxed{s(\underline{length}(\boxed{h_2 :: \underline{t}}^{\uparrow} <> l))}^{\uparrow}) \leftrightarrow even(\boxed{s(s(\underline{length}(l <> t)))}^{\uparrow})$$

$$even(\boxed{s(s(\underline{length}(t <> l)))}^{\uparrow}) \leftrightarrow even(length(l <> t))$$

$$even(length(t <> l)) \leftrightarrow even(length(l <> t))$$

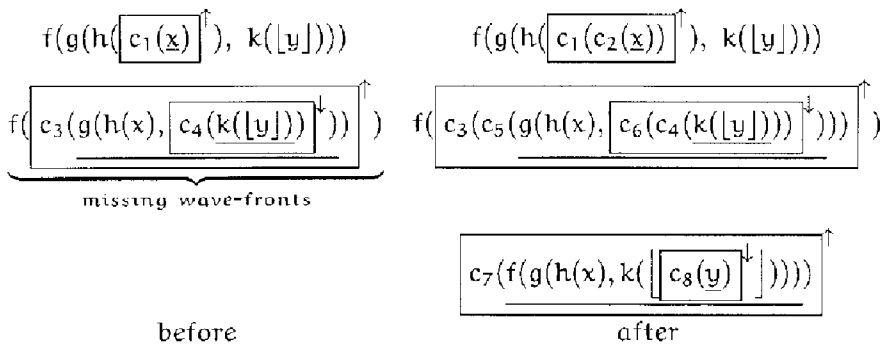
The exception to the general pattern of rippling, which leads to induction revision, is captured in Figure 7.

3.2. Lemma discovery critic

The proof of conjecture (23), presented in Section 3.1, relied upon a lemma that provided wave-rule (26). In this section we focus upon the problem of discovering such lemmata automatically. Consider the following term:

$$\underbrace{\ldots rev(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow})\ldots}_{\text{blocked}} \tag{29}$$

The rippling of this term blocked if we assume that wave-rule (13) is not available. This corresponds to the failure of precondition 2 of the wave method.



Conclusion schemata are shown before and after the application of the induction revision critic. The rippling of the initial conclusion becomes blocked. Partial wave-rule matching identifies missing wave-fronts, i.e., $\boxed{c_3(c_5(\ldots))}^{\uparrow}$ *and* $\boxed{c_6(c_4(\ldots))}^{\downarrow}$*, which are used to calculate a revised induction suggestion, i.e.,* $\boxed{c_1(c_2(\underline{x}))}^{\uparrow}$.

Figure 7. Exception: *missing wave-fronts.*

The induction revision critic, through precondition 2, will initiate a search for a wave-rule of the form

$$\mathrm{rev}\Big(\boxed{F_1(\underline{\mathrm{rev}(Y)}) <> X :: \mathrm{nil}}^{\uparrow}\Big) \Rightarrow \cdots$$

Assuming only the wave-rules that are provided by the definition of rev, then the search will fail. This failure means that any revision of the current induction will lead to a potentially infinite divergence in the proof, i.e.,

$$\ldots \mathrm{rev}\Big(\boxed{(\underline{\mathrm{rev}(t)} <> h_2 :: \mathrm{nil}) <> h_1 :: \mathrm{nil}}^{\uparrow}\Big)\ldots$$

$$\ldots \mathrm{rev}\Big(\boxed{((\underline{\mathrm{rev}(t)} <> h_3 :: \mathrm{nil}) <> h_2 :: \mathrm{nil}) <> h_1 :: \mathrm{nil}}^{\uparrow}\Big)\ldots$$

$$\ldots \mathrm{rev}\Big(\boxed{(((\underline{\mathrm{rev}(t)} <> h_4 :: \mathrm{nil}) <> h_3 :: \mathrm{nil}) <> h_2 :: \mathrm{nil}) <> h_1 :: \mathrm{nil}}^{\uparrow}\Big)\ldots$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

Note that Nqthm will diverge in this way. The failure of the partial wave-rule match (Precondition 3, Figure 6) suggests that we should instead be looking for a missing wave-rule, i.e., a lemma. This analysis is expressed as a critic in Figure 8.

Preconditions:

1. Precondition 1 of wave method holds, i.e.,
   $wave\_term(\_, \_, WaveTerm)$
2. Preconditions 2 and 3 of wave method fail, i.e.,
   no matching wave-rule
   no condition to check
3. not $partial\_wave\_rule\_match(WaveTerm, \_)$

Patch:

1. $construct\_lemma\_lhs(WaveTerm, Sequent, LHS)$
2. $construct\_lemma\_rhs(WaveTerm, Sequent, RHS)$
3. $generalize\_lemma(LHS \Rightarrow RHS, Rewrite)$
4. $validate\_lemma(Rewrite)$

Definition of meta-logical terms:

- Sequent is the current goal sequent;
- $construct\_lemma\_lhs(W, S, L)$ L is the left-hand-side of a wave-rule based upon the wave-term W and the current goal sequent S;
- $construct\_lemma\_rhs(W, S, R)$ R is the right-hand-side of a wave-rule based upon the wave-term W and the current goal sequent S;
- $generalize\_lemma(R, G)$ G is a generalization of the rewrite rule R;
- $validate\_lemma(R)$ records schematic wave-rules derived from R and verifies instantiations generated for R during subsequent proof planning.

Figure 8. Wave critic: *lemma discovery* (*speculation*).

The patch associated with the critic involves the search for a lemma and its proof. We view this as a four-step process, which, in outline, involves

1. construction of the left-hand-side of the lemma,
2. construction of the right-hand-side of the lemma,
3. generalization of the conjectured lemma, and
4. proof of the generalized lemma.

This process may involve backtracking. For example, step 3 may be resatisfied if an over-generalization is detected by the failure of step 4.

We have two strategies for implementing this process. The first is called *lemma calculation* and is the less general of the two strategies but involves little search. The second strategy, *lemma speculation*, is very general and as a consequence introduces additional search control problems. These alternative strategies are presented in the following two sections.

3.2.1. *Lemma Calculation*

Lemma calculation is applicable when we are in a position to partially apply an induction hypothesis. To illustrate, consider again the step-case proof of con-

jecture (5) presented in Section 2.3.1. The effect of not having wave-rule (13) means that the *LHS* of the conclusion becomes blocked, i.e.,

$$\underbrace{\mathrm{rev}(\boxed{\underline{\mathrm{rev}(\mathrm{t})} <> \mathrm{h} :: \mathrm{nil}}^{\uparrow})}_{\text{blocked}} = \boxed{\mathrm{h} :: \underline{\mathrm{t}}}^{\uparrow} \tag{30}$$

Fortunately the *RHS* is fully rippled. That is, the wave-hole on the *RHS* contains a copy of the *RHS* of (12), the induction hypothesis. Consequently, the induction hypothesis can be used to rewrite the conclusion. We now consider the construction of a wave-rule to unblock the *LHS* of the goal based upon this kind of partial use of the induction hypothesis:

1. The blocked wave-term defines the *LHS* of the missing wave-rule. In the case of goal (30) there is one blocked wave-term, which gives an *LHS* of the form

$$\mathrm{rev}(\boxed{\underline{\mathrm{rev}(\mathrm{t})} <> \mathrm{h} :: \mathrm{nil}}^{\uparrow}) \Rightarrow \cdots$$

2. The *RHS* of the missing wave-rule is constructed by rewriting the fully rippled subterm of the blocked conclusion using the induction hypothesis. In the current example this means rewriting the wave-hole associated with $\boxed{\mathrm{h} :: \underline{\mathrm{t}}}^{\uparrow}$ using (12), which gives

$$\cdots \Rightarrow \boxed{\mathrm{h} :: \underline{\mathrm{rev}(\mathrm{rev}(\mathrm{t}))}}^{\uparrow}$$

3. The generalization of the wave-rule resulting from steps 1 and 2 takes the form

$$\mathrm{rev}(\boxed{\underline{\mathrm{Y}} <> \mathrm{X} :: \mathrm{nil}}^{\uparrow}) \Rightarrow \boxed{\mathrm{X} :: \underline{\mathrm{rev}(\mathrm{Y})}}^{\uparrow}$$

   Generalization is desirable to maximize the applicability of wave-rules and to simplify the proof of the underlying lemma. Further discussion of generalization is delayed until Section 4.1.2.

4. The proof of the underlying lemma can be performed either before or after the wave-rule is applied in the main proof. In practice it is performed before.

In general lemmata may be conditional. Currently we use the context in which a ripple proof becomes blocked to suggest appropriate conditions.

3.2.2. *Lemma Speculation*

Lemma speculation is required where calculation is not applicable. For example, consider the conjecture

$$\forall \mathrm{t},\ \mathrm{l} : \mathrm{list}(\tau).\ \mathrm{rev}(\mathrm{rev}(\mathrm{t}) <> \mathrm{l}) = \mathrm{rev}(\mathrm{l}) <> \mathrm{t} \tag{31}$$

In proving (31), wave-rule (11) suggests[6] induction on t using rule (3). The initial induction conclusion takes the form

$$rev(rev(\boxed{h :: \underline{t}}^{\uparrow}) <> [l]) = rev([l]) <> \boxed{h :: \underline{t}}^{\uparrow}$$

By wave-rule (11) this rewrites to

$$\underbrace{rev(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow} <> [l])}_{\text{blocked}} = \underbrace{rev([l]) <> \boxed{h :: \underline{t}}^{\uparrow}}_{\text{blocked}} \tag{32}$$

Lemma calculation is not applicable because neither of the wave-fronts is fully rippled. Now we consider the construction of a wave-rule to unblock the *RHS* of (32). Note that we could equally have selected the *LHS*:

1. The *LHS* of the missing wave-rule is constructed as for lemma calculation, i.e.,

   $$rev(l) <> \boxed{h :: \underline{t}}^{\uparrow} \Rightarrow \cdots$$

2. From the skeleton-preserving property of rippling we know that the *RHS* of the missing wave-rule must fit the following general form

   $$\cdots \Rightarrow \cdots(\cdots rev(l) \cdots <> t) \cdots$$

   We exploit the skeleton-preserving property to construct a schematic *RHS*; that is, second-order meta variables are used to specify the missing wave-front structure. Transverse, longitudinal, and hybrid forms must be considered:

   $$\cdots \Rightarrow \boxed{F_1(\underline{rev(l)}, h, t, l)}^{\downarrow} <> t$$

   $$\cdots \Rightarrow \boxed{G_1(\underline{rev(l) <> t}, h, t, l)}^{\uparrow}$$

   $$\cdots \Rightarrow \boxed{G_1(\underline{\boxed{F_1(\underline{rev(l)}, h, t, l)}^{\downarrow} <> t}, h, t, l)}^{\uparrow}$$

3. The generalization of the schematic wave-rules resulting from steps 1 and 2 is

   $$W <> \boxed{X :: \underline{Y}}^{\uparrow} \Rightarrow \boxed{F_1(\underline{W}, X, Y)}^{\downarrow} <> Y \tag{33}$$

   $$W <> \boxed{X :: \underline{Y}}^{\uparrow} \Rightarrow \boxed{G_1(\underline{W <> Y}, X, Y)}^{\uparrow} \tag{34}$$

   $$W <> \boxed{X :: \underline{Y}}^{\uparrow} \Rightarrow \boxed{G_1(\underline{\boxed{F_1(\underline{W}, X, Y)}^{\downarrow}} <> Y, X, Y)}^{\uparrow} \tag{35}$$

   An application of a schematic wave-rule and its instantiation go hand in hand. Rippling constrains the process of instantiation; that is, any instantiations for meta variables must preserve the skeleton term structure. With the

addition of schematic wave-rules (33), (34), and (35), the wave method is now applicable to (32). Using wave-rule (33) the goal becomes

$$\underbrace{rev(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow} <> \lfloor l \rfloor)}_{\text{blocked}} = \boxed{F_1(\underline{rev(\lfloor l \rfloor)}, h, t)}^{\downarrow} <> t$$

The directionality of the potential wave-front and the associated wave-hole constrains the search for an applicable wave-rule. In this case wave-rule (11) can be applied in reverse, i.e.,

$$\boxed{\underline{rev(Y)} <> X :: nil}^{\downarrow} \Rightarrow rev(\boxed{X :: \underline{Y}}^{\downarrow})$$

The *RHS* of the induction conclusion becomes

$$\cdots = \boxed{F_3(\underline{rev(\boxed{F_2(rev(l), h, t) :: \underline{l}}^{\downarrow})}, h, t)}^{\downarrow} <> t \tag{36}$$

The associated unification instantiates F_1 to be

$$\lambda w.\lambda x.\lambda y.F_3(w <> F_2(w, x, y) :: nil, x, y)$$

Wave-rule schema (33) now becomes

$$W <> \boxed{X :: \underline{Y}}^{\uparrow} \Rightarrow \boxed{F_3(\boxed{\underline{W} <> F_2(W, X, Y) :: nil}^{\downarrow}, X, Y)}^{\downarrow} <> Y$$

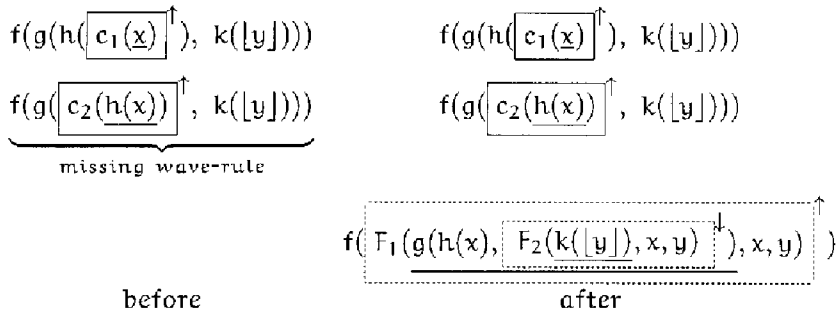
Each incremental instantiation of a wave-rule schema is coupled with an attempt to coerce the remaining meta variables. This coercion process involves exploring possible projections. Where a meta variable denotes a potential wave-front, only the projection of the wave-hole is considered. A conjecture disprover is employed to filter the candidates generated by the coercion process. The disprover is conservative; it may fail to identify invalid instantiations, but it will not reject results that are in fact correct. However, all instantiated wave-rule (lemma) schemas are submitted to the prover for verification. In the current example, coercion is successful and instantiates F_2 and F_3 to be $\lambda w.\lambda x.\lambda y.x$ and $\lambda w.\lambda x.\lambda y.w$, respectively. This completes the instantiation of wave-rule schema (33). The resulting wave-rule takes the form

$$W <> \boxed{X :: \underline{Y}}^{\uparrow} \Rightarrow \boxed{\underline{W} <> X :: nil}^{\downarrow} <> Y \tag{37}$$

Note that the process of constructing (37) completes the rippling of the *RHS* of the conclusion

$$\underbrace{rev(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow} <> \lfloor l \rfloor)}_{\text{blocked}} = rev(\boxed{h :: \underline{l}}^{\downarrow}) <> t$$

If we use (37) in reverse, i.e.,



Conclusion schemata are shown before and after the application of the lemma discovery (speculation) critic. The rippling of the initial conclusion becomes blocked by the term $g(\ldots, k(\lfloor y \rfloor))$. *The patch is to construct a schematic wave-rule, which will unblock the ripple. Further rippling will then instantiate the schema.*

Figure 9. Exception: *missing wave-rule.*

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

the rippling of the *LHS* of the conclusion can also be completed:

$$rev(rev(t) <> \lfloor \boxed{h :: \underline{l}}^{\downarrow} \rfloor) = rev(\lfloor \boxed{h :: \underline{l}}^{\downarrow} \rfloor) <> t$$

4. The proof of the underlying lemma has to be delayed until the wave-rule schema is fully instantiated.

The exception to the general pattern of rippling, which leads to the search for new wave-rules, is presented in Figure 9.

3.3. Generalization Critic

It is sometimes the case that a conjecture must be generalized before we can prove it by induction; that is, the generalization leads to a stronger induction hypothesis, which allows a proof to go through. Generalization through the introduction of accumulator variables is an example of such a phenomenon. To illustrate, consider the following specialization of conjecture (16):

$$\forall t : list(\tau).\ rev(t) = qrev(t, nil)$$

Wave-rules (11) and (17) suggest an induction on t using rule (3). In the step-case we get an induction hypothesis of the form

$$rev(t) = qrev(t, nil) \tag{38}$$

while the initial induction conclusion takes the form

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

Using wave-rule (11) the conclusion rewrites to give

$$\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow} = \underbrace{qrev(\boxed{h :: \underline{t}}^{\uparrow}, nil)}_{\text{blocked}} \tag{39}$$

Note that while wave-rule (17) matches the wave-term on the *RHS*, its applicability is ruled out because of the absence of a sink. In terms of the wave method this corresponds to the failure of Precondition 4 (see Figure 5). This failure is captured as a critic in Figure 10. The patch associated with this critic is the generalization of the goal through the introduction of an accumulator variable into the original conjecture. This is achieved with the use of second-order meta variables. The revised conjecture takes the form

$$\forall t,\ l : list(\tau).F_1(rev(t), l) = qrev(t, G_1(l)) \tag{40}$$

Note that the relationship between the accumulator variable l and the original term structure of the conjecture is partially specified using the meta variables F_1 and G_1. The position of $G_1(l)$ within the *RHS* is determined by the position of the wave-front on the *RHS* of wave-rule (17). The insertion of the meta variables

---

Preconditions:

1. Precondition 1 of wave method holds, i.e.,
   $wave\_term(\_, Pos, WaveTerm)$
2. Precondition 2 of wave method holds, i.e.,
   $wave\_rule\_match(Rn, trans(\_), Cond \rightarrow LHS \Rightarrow RHS, \_)$
3. Precondition 3 of wave method holds, i.e.,
   $tautology(Hyps \vdash Cond)$
4. Precondition 4 of wave method fails, i.e.,
   no sink.

Patch:

1. $generalize\_goal(Conc, Pos, Rn, GenConc)$
2. $propagate\_generalization(Plan, GenConc)$

Definition of meta-logical terms:

- $generalize\_goal(C, P, R, G)$ constructs G a generalization of C based upon the transverse wave-rule R and the position P within C where it is applicable;
- $propagate\_generalization(P, G)$ propagates the generalized goal G through the plan structure P.

Figure 10. Wave critic: *generalization.*

is automatic. With the revised conjecture an induction on t is again proposed. The induction hypotheses becomes

$$F_1(rev(t), L) = qrev(t, G_1(L)) \tag{41}$$

Note that (41) is stronger than (38), the original hypothesis. The induction conclusion associated with the new induction hypothesis takes the form

$$F_1(rev(\boxed{h :: \underline{t}}^{\uparrow}), [l]) = qrev(\boxed{h :: \underline{t}}^{\uparrow}, G_1([l]))$$

As before, wave-rule (11) is applicable to the *LHS* and refines the conclusion as follows:

$$F_1(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow}, [l]) = qrev(\boxed{h :: \underline{t}}^{\uparrow}, G_1([l]))$$

The presence of the sink on the *RHS* means that wave-rule (17) is now applicable, giving

$$F_1(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow}, [l]) = qrev(t, \boxed{h :: \underline{G_1([l])}}^{\downarrow}) \tag{42}$$

From wave-rule (21) the *LHS* can be further rippled to give

$$rev(t) <> (\boxed{h :: nil <> \underline{F_2(rev(t), [l])}}^{\downarrow}) = qrev(t, \boxed{h :: \underline{G_1([l])}}^{\downarrow})$$

The effect of this wave-rule application is to instantiate F_1 to be

$$\lambda x.\lambda y.x <> F_2(x, y)$$

As described in Section 3.2.2, after each incremental instantiation of a meta variable we attempt to coerce the remaining meta variables through projection. Here we have the added constraint that sinks corresponding to the same variable must be instantiated consistently. In this example the coercion process instantiates F_2 and G_1 to be $\lambda x.\lambda y.y$ and $\lambda x.x$, respectively. This completes the rippling of the induction conclusion:

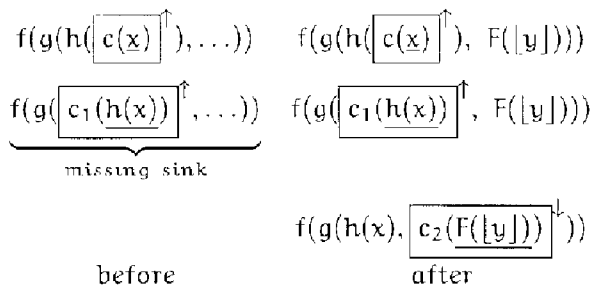
$$rev(t) <> (\left\lfloor \boxed{h :: \underline{l}}^{\downarrow} \right\rfloor) = qrev(t, \left\lfloor \boxed{h :: \underline{l}}^{\downarrow} \right\rfloor)$$

Note the instantiation of (40) corresponds to (16). The exception to the general pattern of rippling caused by missing sinks is presented in Figure 11.

3.4. Case Analyses Critic

The last critic deals with the failure of Precondition 3 of the wave method, i.e., where the condition attached to a wave-rule is not provable in the current context. To illustrate, consider the conjecture

$$\forall a : \tau.\forall t,\ l : list(\tau).\ a \in t \rightarrow a \in (t <> l) \tag{43}$$



Conclusion schemata are shown before and after the application of the generalization critic. The rippling of the initial conclusion becomes blocked because of the absence of a sink. A meta variable F *is used to introduce a new object-level variable* y. *The positioning within the goal structure of the annotated term* $F(\lfloor y \rfloor)$ *is determined by the RHS of an applicable transverse wave-rule.*

Figure 11. Exception: *missing sink.*

where $\in$ denotes list membership. The definition of $\in$ provides the following rewrite rules:

$$X \in \mathrm{nil} \Rightarrow \mathrm{false}$$

$$X \neq Y \rightarrow X \in \boxed{Y :: \underline{Z}}^{\uparrow} \Rightarrow X \in Z \tag{44}$$

$$X = Y \rightarrow X \in \boxed{Y :: \underline{Z}}^{\uparrow} \Rightarrow \mathrm{true} \tag{45}$$

Note that (45) is not a wave-rule because it is not skeleton preserving. It is instead classified as a *complementary* rewrite rule, since its condition forms part of a covering set that includes the condition associated with (44).

In proving (43), wave-rule (44) suggests an induction on t using rule (3). This provides an induction hypothesis of the form

$$a \in t \rightarrow a \in (t <> L)$$

and an induction conclusion of the form

$$a \in \boxed{h :: \underline{t}}^{\uparrow} \rightarrow a \in (\boxed{h :: \underline{t}}^{\uparrow} <> \lfloor l \rfloor)$$

By wave-rule (10) this rewrites to give

$$\underbrace{a \in \boxed{h :: \underline{t}}^{\uparrow}}_{\text{blocked}} \rightarrow \underbrace{a \in \boxed{h :: \underline{t <> \lfloor l \rfloor}}^{\uparrow}}_{\text{blocked}}$$

Although (44) matches both wave-terms, the associated condition, $a \neq h$, is not provable in the current context. This corresponds to the failure of Precondition 3

Preconditions:

1. Precondition 1 of wave method holds, i.e.,
   $wave\_term(\_, \_, LHS)$
2. Precondition 2 of wave method holds, i.e.,
   $wave\_rule\_match(Rn, \_, Cond \rightarrow LHS \Rightarrow \_, \_)$
3. Precondition 3 of wave method fails, i.e.,
   Cond not provable given Hyps

Patch:

1. $setof(Cond, (wave\_rule\_match(\_, \_, Cond \rightarrow LHS \Rightarrow \_, \_) \vee$
   $comp\_rule\_match(\_, Cond \rightarrow LHS \Rightarrow \_)),$
   $Conds)$
2. $case\_split(Plan, Conds)$

Definition of meta-logical terms:

- $comp\_rule\_match(N, R)$ means that N is the name of the complementary rewrite rule R;
- $case\_split(P, C)$ introduces a case-split on C at the current node in the plan structure P.

Figure 12. Wave critic: *missing condition.*

of the wave method. The critic associated with this failure is given in Figure 12. The patch is to perform a case analysis based upon the covering set of conditions defined the wave- and complementary-rules. In the current example this covering set gives rise to a case split on $(a \neq h) \vee (a = h)$. By (44), in the $a \neq h$ case, we get

$$a \in t \rightarrow a \in (t <> [l])$$

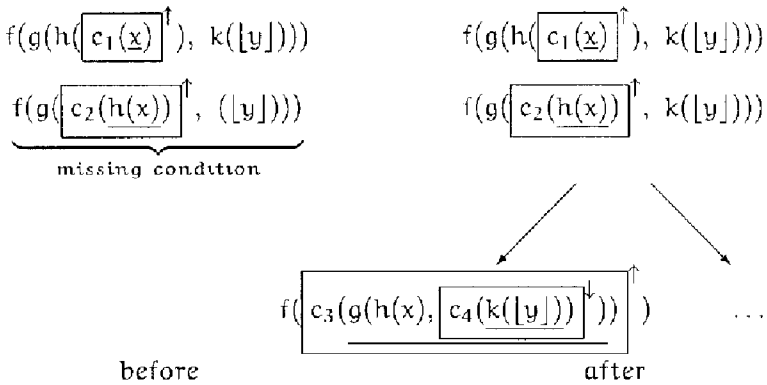
to which fertilize is applicable. In the $a = h$ case, using (45), we get

$$true \rightarrow true$$

which is trivially true. The exception to the general pattern of rippling caused by a missing condition is presented in Figure 13.

3.5. SUMMARY

We have argued that by having an explicit proof plan one is able to accurately pinpoint and interpret failures productively. In the case of the proof plan for induction, the breakdown of the preconditions for rippling provide a handle on the major eureka steps associated with inductive proof. The relationship between precondition failures and patches is summarized in Table I.



Conclusion schemata are shown before and after the application of the case analysis critic. The rippling of the initial conclusion becomes blocked by the absence of a condition. A set of conditional wave-rules and complementary rewrite rules suggests the missing case split.

Figure 13. Exception: *missing condition.*

Table I. Precondition failures and patches for rippling

Precondition	Generalization	Case Analysis	Induction Revision	Lemma Discovery
1	⋆	⋆	⋆	⋆
2	⋆	⋆	○	•
3	⋆	•		
4	•			

The association between precondition failure and patches for the ripple heuristic are shown. Note that ⋆, ○, and • denote success, partial success, and failure, respectively.

4. Search Control Issues

In this section we address a number of search control issues that arise in the critics mechanism and the use of higher-order meta variables in the search for proofs.

4.1. Wave-Term Selection

For a given goal the selection of wave-terms is determined by the *LHS* of an applicable wave-rule. In the absence of an applicable wave-rule we require a mechanism for selecting wave-terms. The search space associated with this selection process is discussed below.

4.1.1. *Most Nested Wave-Fronts*

A blocked goal may contain many blocked wave-fronts. In searching for a wave-front to unblock, we select a *most nested* wave-front. The motivation is that unblocking a more nested wave-front may in turn unblock a less-nested wave-front. To illustrate, consider the goal

$$\ldots even(\boxed{s(x + \boxed{s(\underline{x})}^{\uparrow})}^{\uparrow})\ldots$$

Both wave-fronts are blocked if we assume only the wave-rules provided by the definitions of *even* and $+$, i.e., wave-rules (24) and

$$\boxed{s(\underline{X})}^{\uparrow} + Y \Rightarrow \boxed{s(\underline{X + Y})}^{\uparrow}$$

If our search for a missing wave-rule is successful, i.e.,

$$X + \boxed{s(\underline{Y})}^{\uparrow} \Rightarrow \boxed{s(\underline{X + Y})}^{\uparrow}$$

then the unblocking of the most nested wave-front also enables the unblocking of the less nested wave-front, i.e.,

$$\ldots even(\boxed{s(x + \boxed{s(\underline{x})}^{\uparrow})}^{\uparrow})\ldots$$

$$\ldots even(\boxed{s(s(\underline{x + x}))}^{\uparrow})\ldots$$

$$\ldots even(x + x)\ldots$$

4.1.2. *Wave-Term Context*

The context of the wave-term is important when searching for a missing lemma. On the one hand, we wish to minimize the context associated with a wave-rule; on the other hand, we are interested only in wave-rules derived from theorems. For example, consider the following blocked goal:

$$\ldots even(length(l <> \boxed{h :: \underline{t}}^{\uparrow}))\ldots$$

Selecting the minimal wave-term, i.e.,

$$l <> \boxed{h :: \underline{t}}^{\uparrow}$$

will lead to the following invalid wave-rule

$$W <> \boxed{X :: \underline{Y}}^{\uparrow} \Rightarrow \boxed{X :: \underline{W <> Y}}^{\uparrow}$$

The required wave-term takes the form

$$\mathrm{length}(l <> \boxed{h :: \underline{t}}^{\uparrow})$$

and provides a valid wave-rule of the form

$$\mathrm{length}(W <> \boxed{X :: \underline{Y}}^{\uparrow}) \Rightarrow \boxed{s(\underline{\mathrm{length}(W <> Y)})}^{\uparrow}$$

Backtracking is therefore necessary to allow for the selection of progressively larger subterms of the skeleton term structure of the goal. In addition, we also consider the need to manipulate multiple wave-fronts simultaneously. Since the goal term structure is finite, the associated search space is also finite.

Finally, generalization is an important technique in minimizing context. We exploit two forms of generalization: *replacement of common terms* and *distinguishing variables apart*. In the context of automatic inductive theorem proving Aubin [1] was the first to investigate such techniques. We adopt similar heuristics to those developed by Aubin, e.g., exploiting *primary recursion paths*[7] is distinguishing between variables to generalization apart.

All of the above heuristic may lead to invalid wave-rule suggestions. In practice we rely upon a conjecture disprover to prune the search space of wave-rule suggestions.

4.2. Schematic Wave-Rule Construction

As observed in Section 3.2.2 there may be a choice as to whether to apply longitudinal, transverse, or hybrid wave-rule schemata. The strategy we employ is to pursue the direction that provides greatest constraints in the instantiation of higher-order meta variables. For example, in the case of goal (32), where three schematic wave-rule are generated, schema (32) was selected because it gave rise to further rippling.

In general, however, it may not be obvious which is the best schema to apply. We therefore allow backtracking over the selection process using a conjecture disprover to prune the search space.

4.3. Guiding Unification

As illustrated by the lemma discovery and generalization critics, the ability to apply wave-rules in the context of higher-order meta variables is crucial to our technique. The instantiation of such meta variables occurs as a side effect of applying wave-rules. The generality and controllability of such a technique is dependent upon the type of variables used. If we choose first-order variables, then control is not a problem, but the framework is not very general [15]. Alternatively, using higher-order variables we gain generality but at the cost of controllability.

Meta-level control information provides a basis for regaining controllability. Hesketh [13] used the meta-level control information embedded within the

wave-front annotations in controlling the search for generalizations. We adopt a similar aproach to Hesketh in using the wave-fronts and wave-holes to divide the higher-order unification task into a number of subtasks. We extend the approach, however, by exploiting the directionality of wave-fronts in focusing the unification process. That is, when rippling-in we match wave-holes first and when rippling-out we first match the superterm that contains the wave-front. This is necessary in order to support the incremental instantiation of meta variables.

Backtracking over alternative instantiations, however, is necessary. This is particularly the case for the generalization critic where we employ an iterative deepening planner. To illustrate the problem, consider again the *LHS* of goal (42), i.e.,

$$F_1(\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow}, [l]) = \cdots \tag{46}$$

In Section 3.3 the associativity of <> (wave-rule (21)) was used to obtain

$$rev(t) <> (\boxed{h :: nil <> \underline{F_2(rev(t), [l])}}^{\downarrow}) = \cdots$$

However, the associativity of <> provides another applicable wave-rule, i.e.,

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

which, when unified with the *LHS* of (46), gives

$$\boxed{\underline{F_2(rev(t), [l])} <> rev(t) <> h :: nil}^{\uparrow} = \cdots$$

This longitudinal ripple is rejected because it gives rise to a non-theorem that is detected by our conjecture disprover. Control is hard for generalization, since we are dealing with a schematic skeleton term structure. Consequently, the skeleton preservation property of rippling is less useful for restricting search than in the case of lemma discovery.

Higher-order annotated unification is a hard problem. We acknowledge that our approach is far from ideal. The interested reader is referred to [14] for some promising results we may be able to exploit in our future work.

4.4. Critic Selection

Multiple critics may be applicable to a particular proof failure. Preference is given to the critic with the most restrictive preconditions (see Table I). For example, the generalization critic was used to patch goal (39) although lemma calculation was also applicable. However, patches are not guaranteed to succeed so backtracking across critics is supported.

5. Comparison with Related Techniques

Most theorem provers rely upon user interaction to identify, interpret, and patch failures. In contrast, proof critics attempt to automate this process. In order to draw a comparison with other related techniques, we must consider the individual critics presented here.

First, induction revision appears to be unique in theorem-proving terms. In relation to other inductive theorem-proving systems, this kind of patching involves the user in supplying a hint in the form of a dummy recursive function that reflects the desired induction.

Second, lemma calculation is based upon the strategy of cross fertilization [2] and weak-fertilization [4]. A practical limitation of both these strategies is that they discover lemmas in-line. As a consequence the same lemma may be re-discovered and verified many times during the course of a proof. Lemma calculation factors out lemmas and verifies them as separate proof attempts. This eliminates the redundancy problem mentioned above and provides a degree of modularity in structuring proofs.

Third, lemma speculation was first presented in [16]. A missing lemma typically causes divergence in the rewriting of step-case proofs. Lemma speculation preempts such divergence. We are able to achieve this because of the constraints rippling imposes on the search space. In contrast, Thomas and Jantke [20] observe divergence patterns and use them to suggest generalizations in the context of inductive completion. A similar idea has been applied by Walsh [21] to the problem of proof divergence in the SPIKE [3] theorem prover. Divergence patterns generated by SPIKE are used to suggest lemmas. Central to this approach is the technique of *difference matching* [9]. Difference matching is used to speculate the structure of missing lemmas by generalizing over a sequence of diverging formulae. This process of overcoming divergence is largely independent of the SPIKE proof strategy. In contrast, the proof critics mechanism has direct access to the meta-level control information which is used to guide the search for proofs. This increases the sophistication by which failure can be interpreted. For example, without our meta-level notion of a 'sink' it is hard to see how one could distinguish between the need for generalization as opposed to a missing lemma. The diversity of patches associated with the ripple heuristic demonstrates the power of rippling and the proof critics mechanism. More generally the proof critics mechanism supports the global analysis of the proof process, since it has access to the whole proof structure (see Section 5 of [16]). As a consequence, we believe that the approach presented here provides a powerful framework for patching proofs.

Fourth, the use of higher-order meta variables in discovering generalizations through the introduction of accumulators was first achieved by Hesketh [13]. The generalization critic presented here represents a rational reconstruction and extension of this work.

Lastly, the case analyses critic is an alternative to having a method for supporting conditional rewriting [4]. The critic mechanism, however, allows for more sophisticated case analyses involving the composition of multiple partial proofs. This idea is illustrated in [16].

6. Implementation and Results

The proof critics presented here have been implemented and tested. Our implementation is an extension of the CLAM [6] proof planning system and exploits the higher-order features of λ-Prolog [18].

Our test results are presented in Tables II, III, and IV. The proofs of all the example conjectures[8] given in Table II are discovered completely automatically. These proofs are based only upon definitions supplied by the user. Except for the generalization examples, all additional lemmas are automatically discovered by the system.

The example conjectures in Table II are classified under the four critics. In the case of lemma discovery, conjectures T1 to T13, T22 to T26, and T48 to T50 required only the relatively weak strategy of lemma calculation. Even so, Nqthm failed[9] to prove T1, T3, T4, T6, T7, T8, T9, T10, T11, T13, and T22 through to T26. Examples T14 to T21 required lemma speculation while T27 to T35 required generalization. Note that different generalizations were obtained depending upon the available lemmata. All the examples that required induction revision, lemma speculation, or generalization fall into a class of inductive theorem that are not uncommon but are acutely difficult to prove. With the exception of T14, Nqthm failed to prove any of these examples. Finally, examples T22 to T26 and T48 to T50 illustrate the need for multiple critics in patching some conjectures.

7. Limitations and Further Work

We have focussed upon failure in the context of constructor style induction. The complementary destructor style induction is already incorporated within rippling. A destructor style induction introduces wave-fronts into the induction hypothesis, e.g.,

$$G[f(\boxed{d(\underline{x})}^{+}, y)] \vdash G[f(x, y)]$$

Rippling uses *creational* wave-rules [4] to set-up a ripple in the conclusion by *neutralizing* the wave-fronts in the hypothesis, e.g.,

$$G[f(\boxed{d(\underline{x})}^{+}, y)] \vdash G[\boxed{c(f(\boxed{d(\underline{x})}^{-}, y))}^{\uparrow}]$$

Creational wave-rules have the following general form:

$$Cond \rightarrow f(x, y) \Rightarrow \boxed{c(f(\boxed{d(\underline{x})}^{-}, y))}^{\uparrow}$$

Table II. Example conjectures

No.	Conjecture	(i)	(ii)	(iii)	(iv)
T1	double(X)=X+X		L1		
T2	length(X<>Y)=length(Y<>X)		L2		
T3	length(X<>Y)=length(Y)+length(X)		L1		
T4	length(X<>X)=double(length(X))		L2		
T5	length(rev(X))=length(X)		L3		
T6	length(rev(X<>Y))=length(X)+length(Y)		L2		
T7	length(qrev(X,Y))=length(X)+length(Y)		L1		
T8	nth(X,nth(Y,Z))=nth(Y,nth(X,Z))		L4&L5		
T9	nth(W,nth(X,nth(Y,Z)))=nth(Y,nth(X,nth(W,Z)))		L6&L7		
T10	rev(rev(X))=X		L8		
T11	rev(rev(X)<>rev(Y))=Y<>X		L9&10		
T12	qrev(X,Y)=rev(X)<>(Y)		L11		
T13	half(X+X)=X		L1		
T14	ordered(isort(X))		L12		
T15	X+s(X)=s(X+X)		L1		
T16	even(X+X)		L1		
T17	rev(rev(X<>Y))=rev(rev(X))<>rev(rev(Y))		L8		
T18	rev(rev(X)<>Y)=rev(Y)<>X		L11&L13		
T19	rev(rev(X))<>Y=rev(rev(X<>Y))		L8		
T20	even(length(X<>X))		L2		
T21	rotate(length(X),X<>Y)=Y<>X		L11&L13		
T22	even(length(X<>Y))↔even(length(Y<>X))	(3)↦(4)	L14		
T23	half(length(X<>Y))=half(length(Y<>X))	(3)↦(4)	L15		
T24	even(X+Y)↔even(Y+X)	(1)↦(2)	L16		
T25	even(length(X<>Y))↔even(length(Y)+length(X))	(3)↦(4)	L16		
T26	half(X+Y)=half(Y+X)	(1)↦(2)	L17		
T27	rev(X)=qrev(X,nil)			G1	
T28	revflat(X)=qrevflat(X,nil)			G2	
T29	rev(qrev(X,nil))=X			G3&G4	
T30	rev(rev(X)<>nil)=X			G5&G6	
T31	qrev(qrev(X,nil),nil)=X			G7&G8	
T32	rotate(length(X),X)=X			G9	
T33	fac(X)=qfac(X,1)			G10	
T34	X*Y=mult(X,Y,0)			G11	
T35	exp(X,Y)=qexp(X,Y,1)			G12	
T36	X∈Y→X∈(Y<>Z)				⋆
T37	X∈Z→X∈(Y<>Z)				⋆
T38	(X∈Y)∨(X∈Z)→X∈(Y<>Z)				⋆
T39	X∈nth(Y,Z)→X∈Z				⋆
T40	X⊂Y→(X∪Y=Y)				⋆
T41	X⊂Y→(X∩Y=X)				⋆
T42	X∈Y→X∈(Y∪Z)				⋆

Table II. Continued

No.	Conjecture	(i)	(ii)	(iii)	(iv)
T43	X∈Y→X∈(Z∪Y)				*
T44	(X∈Y)∧(X∈Z)→(X∈Y∩Z)				*
T45	X∈insert(X,Y)				*
T46	X=Y→(X∈insert(Y,Z)↔true)				*
T47	X≠Y→(X∈insert(Y,Z)↔X∈Z)				*
T48	length(isort(X))=length(X)		L18		*
T49	X∈isort(Y)→X∈Y		L19		*
T50	count(X,isort(Y))=count(X,Y)		L20&L21		*

The numbered columns denote (i) *induction revision,* (ii) *lemma discovery,* (iii) *generalization and* (iv) *case split. Note that* nth(X,Y) *denotes the list constructed by removing the first* Xth *elements from* Y. *Note also that* fac, exp *and* x *denote factorial, exponentiation and multiplication while* qfac, qexp *and* mult *denote tail recursive versions, respectively.*

Table III. Lemmata

No.	Lemma
L1	X+s(Y)=s(X+Y)
L2	length(X<>Y Z)=s(length(X<>Z))
L3	length(X<>Y nil)=s(length(X))
L4	nth(s(W),nth(X,Y·Z))=nth(W,nth(X,Z))
L5	nth(s(V),nth(s(W),X Y Z))=nth(s(V),nth(W,X Z))
L6	nth(s(V),nth(W,nth(X,Y Z)))=nth(V,nth(W,nth(X,Z)))
L7	nth(s(U),nth(V,nth(s(W),X Y Z)))=nth(s(U),nth(V,nth(W,X Z)))
L8	rev(X<>(Y nil))=Y rev(X)
L9	rev(X<>(Y<>Z nil))=Z rev(X<>Y)
L10	rev((X<>Y nil)<>nil)=Y rev(X<>nil)
L11	(X<>(Y nil))<>Z=X<>(Y Z)
L12	ordered(Y)→ordered(insert(X,Y))
L13	(X<>Y)<>Z nil=X<>(Y<>Z nil)
L14	even(length(W<>Z))↔even(length(W<>X Y Z))
L15	length(W<>X Y Z)=s(s(length(W<>Z)))
L16	even(X+Y)↔even(X+s(s(Y)))
L17	X+s(s(Y))=s(s(X+Y))
L18	length(insert(X,Y)=s(length(Y))
L19	(X≠Y→(X∈insert(Y,Z)→X∈Z)
L20	count(X,insert(X,Y))=s(count(X,Y))
L21	X≠Y→(count(X,insert(Y,Z))=count(X,Z))
L22	(X<>Y)<>Z=X<>(Y<>Z)
L23	(X*Y)*Z=X*(Y*Z)
L24	(X+Y)+Z=X+(Y+Z)

We are currently investigating the automatic discovery of creational wave-rules. This will involve generalizing our technique for generating conditional lemmata.

Table IV. Generalized conjectures

No.	Generalization	Lemmata
G1	rev(X)<>Y=qrev(X,Y)	L22
G2	revflat(X)<>Y=qrevflat(X,Y)	L22
G3	rev(qrev(X,Y))=rev(Y)<>X	L11
G4	rev(qrev(X,rev(Y)))=Y<>X	L8&L11
G5	rev(rev(X)<>Y)=rev(Y)<>X	L11
G6	rev(rev(X)<>rev(Y))=Y<>X	L8&L11
G7	qrev(qrev(X,Y),nil)=rev(Y)<>X	L11
G8	qrev(qrev(X,rev(Y)),nil)=Y<>X	L8&L11
G9	rotate(length(X),X<>Y)=Y<>X	L11&L22
G10	fac(X)*Y=qfac(X,Y)	L23
G11	(X*Y)+Z=mult(X,Y,Z)	L24
G12	exp(X,Y)*Z=qexp(X,Y,Z)	L23

The lemmata used to motivate each generalization are indicated in the right-hand column.

There is a strong connection between creational wave-rules and well-founded inductions. The discovery of missing creational wave-rules may provide a handle on the problem of generating new inductions dynamically.

In terms of the proof critics mechanism, the analysis presented here relies upon patterns of precondition failures that are constructed off-line. We are currently investigating the use of meta-level inference in deriving patches automatically. This will provide a more dynamic form of failure analysis.

8. Conclusions

The ripple method has proved to be a very successful search control heuristic within inductive proof. We have shown, through the use of proof critics, that rippling provides useful heuristic guidance even when a proof attempt fails.

More generally, it has been argued [8] that the separation of meta-level control information from the object-level logic brings clarity, flexibility, and modularity to reasoning systems as well as providing a more constrained search space. The proof critics technique demonstrates another advantage of this separation, namely, *robustness*, that is, the ability to exploit failure productively in the search for proofs.

Notes

[1] We use $\Rightarrow$ to denote rewrite rules and $\rightarrow$ to denote logical implication.

[2] The phrase *fertilization*, which describes the use of hypotheses, was introduces in [2].

[3] Note that we replace t in the hypothesis with L. We use upper-case letters to denote variables, while constants are denoted by lower-case letters.

[4] The simplification of wave-fronts is guaranteed to preserve the skeleton term structure.

[5] Actually t or l are equally good candidate induction variables. The analysis presented here works for either variable.

[6] Note that an induction on t would be equally suitable and that the technique being presented works for either suggestion.

[7] We exploit *ripple paths*, a generalization of the recursion path notion.

[8] The examples come from a number of sources that include [1, 2, 19], and [21].

[9] This is without the aid of the linear arithmetic decision procedure. With the decision procedure, Nqthm is able to prove T1, T3, T6, and T7. More generally, when we talk of 'failure' we are talking about the failure of Nqthm to find a proof without user intervention, i.e., additional lemmata, induction hints, or generalizations supplied by the user.

Acknowledgements

We are indebted to David Basin for his encouragement and detailed comments during the preparation of this paper. We also thank Ian Frank, Erica Melis, Raul Monroy, Julian Richardson, Toby Walsh, and three anonymous JARS referees for feedback on this paper.

References

1. Aubin, R.: Some generalization heuristics in proofs by induction, in G. Huet and G. Kahn (eds), *Actes du Colloque Construction: Amélioration et vérification de Programmes*, Institut de recherche d'informatique et d'automatique, 1975.
2. Boyer, R. S. and Moore, J. S.: *A Computational Logic*, Academic Press, New York, 1979. ACM monograph series.
3. Bouhoula, A. and Rusinowitch, M.: Automatic case analysis in proof by induction, in *Proc. 13th IJCAI, Int. Joint Conf. Artificial Intelligence*, 1993.
4. Bundy, A., Stevens, A., van Harmelen, F., Ireland, A., and Smaill, A.: Rippling: A heuristic for guiding inductive proofs, *Artificial Intelligence* **62** (1993), 185–253. Also available from Edinburgh as DAI Research Paper No. 567.
5. Bundy, A.: The use of explicit plans to guide inductive proofs, in R. Lusk and R. Overbeek (eds), *9th Conf. Automated Deduction*, Springer-Verlag, 1988, pp. 111–120. Longer version avilable from Edinburgh as DAI Research Paper No. 349.
6. Bundy, A., van Harmelen, F., Horn, C., and Smaill, A.: The Oyster-Clam system, in M. E. Stickel (ed.), *10th Int. Conf. Automated Deduction*, Lecture Notes in Artificial Intelligence 449, Springer-Verlag, 1990, pp. 647–648. Also available from Edinburgh as DAI Research Paper No. 507.
7. Bundy, A., van Harmelen, F., Smaill, A., and Ireland, A.: Extensions to the rippling-out tactic for guiding inductive proofs, in M. E. Stickel (ed.), *10th Int. Conf. Automated Deduction*, Lecture Notes in Artificial Intelligence 449, Springer-Verlag, 1990, pp. 132–146. Also available from Edinburgh as DAI Research Paper No. 459.
8. Bundy, A. and Welham, B.: Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation, *Artificial Intelligence* **16**(2) (1981), 189–212. Also available from Edinburgh as DAI Research Paper No. 121.
9. Basin, D. and Walsh, T.: Difference matching, in Deepak Kapur (ed.), *11th Conf. Automated Deduction*, Saratoga Springs, NY, USA, June 1992. Published as Springer Lecture Notes in Artificial Intelligence 607, pp. 295–309.
10. Gentzen, G.: *The Collected Papers of Gerhard Gentzen*, edited by M. E. Szabo, North-Holland, Amsterdam, 1969.
11. Gordon, M. J., Milner, A. J., and Wadsworth, C. P.: *Edinburgh LCF – A Mechanised Logic of Computation*, Lecture Notes in Computer Science 78, Springer-Verlag, 1979.
12. Hardy, G. H.: Mathematical proof, *MIND: A Quarterly Review of Psychology and Philosophy*, **38**(149) (January 1929), 1–25.
13. Hesketh, J. T.: Using middle-out reasoning to guide inductive theorem proving, PhD thesis, University of Edinburgh, 1991.

14. Hutter, D. and Kohlhase, M.: A Colored Version of the λ-Calculus, Seki-report sr-95-05, University of Saarland, 1995.
15. Hutter, D.: Guiding inductive proofs, in M. E. Stickel (ed.), *10th Int. Conf. Automated Deduction*, Lecture Notes in Artificial Intelligence 449, Springer-Verlag, 1990, pp. 147–161.
16. Ireland, A.: The use of planning critics in mechanizing inductive proofs, in A. Voronkov (ed.), *Int. Conf. Logic Programming and Automated Reasoning, LPAR 92, St. Petersburg*, Lecture Notes in Artificial Intelligence 624, Springer-Verlag, 1992, pp. 178–189. Also available from Edinburgh as DAI Research Paper No. 592.
17. Monroy, R., Bundy, A., and Ireland, A.: Proof plans for the correction of false conjectures, in F. Pfenning (ed.), *5th Int. Conf. Logic Programming and Automated Reasoning, LPAR 94*, Lecture Notes in Artificial Intelligence 822, Springer-Verlag, 1994, pp. 54–68. Also available from Edinburgh as Research Paper No. 681.
18. Miller, D. and Nadathur, G.: An overview of λProlog, in R. Bowen and K. Kowalski (eds), *Proc. 5th Int. Logic Programming Conf./5th Symp. on Logic Programming*, MIT Press, 1988.
19. Manna, Z. and Waldinger, R.: *The Logical Basis for Computer Programming*, Vol. 1. *Deductive Reasoning*, Addison/Wesley, Reading, MA, 1985.
20. Thomas, M. and Jantke, K. P.: Inductive inference for solving divergence in Knuth–Bendix, in K. P. Jantke (ed.), *Analogical and Inductive Inference, Proc. AII'89*, Springer-Verlag, 1989, pp. 288–303.
21. Walsh, T.: A divergence critic, in Alan Bundy (ed.), *12th Conf. Automated Deduction*, Lecture Notes in Artificial Intelligence 814, Springer-Verlag, Nancy, France, 1994.