Extensions to a Generalization Critic for Inductive Proof *

Andrew Ireland

Computing & Electrical Engineering
Heriot-Watt University
Riccarton
Edinburgh EH14 4AS
Scotland, U.K.
Email: air@cee.hw.ac.uk
Tel: +44-131-451-3409

Alan Bundy

Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland, U.K.
Email: bundy@ed.ac.uk
Tel: +44-131-650-2716

Abstract

In earlier papers a critic for automatically generalizing conjectures in the context of failed inductive proofs was presented. The critic exploits the partial success of the search control heuristic known as rippling. Through empirical testing a natural generalization and extension of the basic critic emerged. Here we describe our extended generalization critic together with some promising experimental results.

1 Introduction

A major obstacle to the automation of proof by mathematical induction is the need for generalization. A generalization is underpinned by the cut-rule of inference. In a goal-directed framework, therefore, a generalization introduces an infinite branching point into the search space. It is known [13] that the cut-elimination theorem does not hold for inductive theories. Consequently heuristics for controlling generalization play an important role in the automation of inductive proof.

There are a number of different kinds of generalization. In this paper we present a technique for generalization which involves the introduction of accumulator variables. This technique relies upon the analysis of failed proof attempts. We illustrate the problem of accumulator generalization in the context of program verification using an example taken from list processing. The example is based upon the functions defined in figure 1. Rewrite rules derived from these definitions are among those given in appendix A. Using these definitions we can specify an equivalence between a single and a distributed application of the $\mathtt{map}$ function by a conjecture of the form[1]:

$$\forall t : \mathtt{list}(A).\forall f : A \to B.\forall n : \mathbb{N}.$$
$$\mathtt{map}(f, t) = \mathtt{reduce}(\lambda x.\lambda y.\mathtt{app}(x, y), \mathtt{map}(\lambda x.\mathtt{map}(f, x), \mathtt{split}(n, t))) \quad (1)$$

*The research reported in this paper was supported by EPSRC grant GR/J/80702 and ARC grant 438.

[1] Note that we use λ and @ to denote function abstraction and application respectively.

```
fun atend x nil    = (x::nil) |
    atend x (y::z) = y::(atend x z)

fun split1 v w x nil  = (x::nil) |
    split1 v w x (y::z) =
       if (v > w) then
          x::(split1 2 w (y::nil) z)
       else
          (split1 v+1 w (atend y x) z)
```

```
fun split x y = split1 1 x nil y

fun app nil    z = z |
    app (x::y) z = x::(app y z)

fun map x nil    = nil |
    map x (y::z) = (x y)::(map x z)

fun reduce x nil    = nil |
    reduce x (y::z) = (x y (reduce x z))
```

Figure 1: Example list processing functions

This conjecture was provided[2] by an independent research group working on the development parallel systems from functional prototypes [14]. Their development process involves formal proof. Currently proofs are constructed by hand and represent a time consuming hurdle to the research project. Having failed to prove conjecture (1) by hand it was passed to us as a challenge theorem.

In order to prove (1) we must first unfold the definition of split. An application of rewrite rule (12) gives rise to a refined goal of the form:

$$\forall t : \mathsf{list}(A).\forall f : A \to B.\forall n : \mathbb{N}.$$
$$\mathsf{map}(f, t) = \mathsf{reduce}(\lambda x.\lambda y.\mathsf{app}(x, y), \mathsf{map}(\lambda x.\mathsf{map}(f, x), \mathsf{split}_1(1, n, \mathsf{nil}, t))) \quad (2)$$

A proof of (2) requires induction. However, (2) must first be generalized in order for an inductive proof attempt to succeed. An accumulator generalization is required. The generalized conjecture takes the form:

$$\forall t : \mathsf{list}(A).\forall f : A \to B.\forall n : \mathbb{N}.\forall l_1 : \mathbb{N}.\forall l_2 : \mathsf{list}(A).$$
$$\mathsf{map}(f, \mathsf{app}(l_2, t)) = \mathsf{reduce}(\lambda x.\lambda y.\mathsf{app}(x, y), \mathsf{map}(\lambda x.\mathsf{map}(f, x), \mathsf{split}_1(l_1, n, l_2, t))) \quad (3)$$

Note the two new universally quantified variables l_1 and l_2. These act as accumulators in a subsequent inductive proof. We return to this example in §7. This paper addresses two questions: Firstly, how the need for such a generalization can be identified and secondly, how the construction of the required generalized conjecture can be automated.

2 Background

2.1 Proof Methods and Critics

We build upon the notion of a proof plan [3] and tactic-based theorem proving [7]. While a *tactic* encodes the low-level structure of a family of proofs a *proof plan* expressions the high-level structure. In terms of automated deduction, a proof plan guides the search for a proof. That is, given a collection of general purpose tactics the associated proof plan can be used to automatically tailor a special purpose tactic to prove a particular conjecture.

[2] With thanks to Greg Michaelson.

The basic building blocks of proof plans are *methods*. Using a meta-logic, methods express the preconditions for tactic application. The benefits of proof plans can be seen when a proof attempt goes wrong. Experienced users of theorem provers, such as NQTHM, are used to intervening when they observe the failure of a proof attempt. Such interventions typically result in the user generalizing their conjecture or supplying additional lemmata to the prover. Through the notion of a proof *critic* [10] we have attempted to automate this process. Critics provide the proof planning framework with an exception handling mechanism which enables the partial success of a proof plan to be exploited in search for a proof. The mechanism works by allowing proof patches to be associated with different patterns of precondition failure. We previously reported [11] various ways of patching of inductive proofs based upon the partial success of the ripple method described below.

2.2 A Method for Guiding Inductive Proof

In the context of mathematical induction the *ripple* method plays a pivotal role in guiding the search for a proof. The ripple method controls the selective application of rewrite rule in order to prove step case goals.

Schematically a step case goal can be represented as follows:

$$\cdots \underbrace{\forall b'.\ P[a, b']}_{\text{hypothesis}} \cdots \vdash \underbrace{P[c_1(a), b]}_{\text{conclusion}}$$

where $c_1(a)$ denotes the induction term. To achieve a step case goal the conclusion must be rewritten so as to allow the hypothesis to be applied:

$$\cdots \forall b'.\ P[a, b'] \cdots \vdash c_2(P[a, c_3(b)])$$

Note that in order to apply the induction hypothesis we must first instantiate b' to be $c_3(b)$, *i.e.*

$$\cdots P[a, c_3(b)] \cdots \vdash c_2(P[a, c_3(b)])$$

Induction and recursion are closely related. The application of an induction hypothesis corresponds to a recursive call while the instantiation of an induction hypothesis corresponds to the modification of an accumulator variable. The need to instantiate induction hypotheses is commonplace in inductive proof. Our technique, as will be explained below, exploits this fact.

Syntactically an induction hypothesis and conclusion are very similar. More formally, the hypothesis can be expressed as an embedding within the conclusion. Restricting the rewriting of the conclusion so as to preserve this embedding maximizes the chances of applying an induction hypothesis. This is the basic idea behind the ripple method. The application of the ripple method, or *rippling*, makes use of meta-level annotations called *wave-fronts* to distinguish the term structures which cause the mismatch between the hypothesis and conclusion. Using shading to represent wave-fronts then the schematic step case goal takes the form:

$$\cdots \forall b'.\ P[a, b'] \cdots \vdash P[\,c_1(a)^{\uparrow}, \lfloor b \rfloor]$$

The arrows are used to indicate the direction in which wave-fronts can be moved through the term structure. The unshaded term structure is called the *skeleton* and corresponds to the embedding of the hypothesis within the conclusion. In order to distinguish terms within the conclusion which can be matched by accumulator variables in the hypothesis we use annotations

called *sinks*, *i.e.* $\lfloor \ldots \rfloor$. As will be explained below sinks play an important role in identifying the need for accumulator generalization.

A successful application of the ripple method can be characterized as follows:

$$\cdots \forall b'.\, P[a, b'] \cdots \vdash \; c_2(P[a, \lfloor c_3(b)^{\downarrow} \rfloor])^{\uparrow}$$

Note that the term $c_3(b)$, *i.e.* the instantiation for b', occurs within a sink. Rippling restricts rewriting to a syntactic class of rules called *wave-rules*. Wave-rules make progress towards eliminating wave-fronts while preserving skeleton term structure. A wave-rule which achieves the ripple given above takes the form[3]:

$$P[c_1(X)^{\uparrow}, Y] \Rightarrow \; c_2(P[X, c_3(Y)^{\downarrow}])^{\uparrow} \tag{4}$$

Wave-rules are derived automatically from definitions and logical properties like substitution, associativity and distributivity *etc.* In general, a successful ripple will require multiple wave-rule applications as will be illustrated in §7. There are three elementary forms a ripple can take:

outwards: the movement of wave-fronts into less nested term tree positions.

sideways: the movement of wave-fronts between distinct branches in the term tree.

inwards: the movement of wave-fronts into more nested term tree positions.

Note that a sideways ripple is only performed if progress is made towards a sink. In general, a wave-rule may combine all three forms. For a complete description of rippling see [1, 4].

2.3 A Critic for Discovering Generalizations

In terms of the ripple method, the need for an accumulator generalization can be explained in terms of the failure of a sideways ripple due to the absence of sinks. Schematically this failure pattern can be represented as follows:

$$\cdots P[a, d] \cdots \vdash P[c_1(a)^{\uparrow}, d]$$

where d denotes a term which does not contain any sinks. We call the occurrence of d a *blockage term* because it blocks the sideways ripple, in this case the application of wave-rule (4).

The identification of a blockage term triggers the generalization critic. The associated proof patch introduces schematic terms into the goal in order to partially specify the occurrences of an accumulator variable. In the example presented above this leads to a patched goal of the form:

$$\cdots \forall l'.P[a, \mathcal{M}(l')] \cdots \vdash \forall l.P[c_1(a)^{\uparrow}, \mathcal{M}(\lfloor l \rfloor)]$$

where $\mathcal{M}$ denotes a second-order meta-variable. Note that wave-rule (4) is now applicable, giving rise to a refined goal of the form:

$$\cdots \forall l'.P[a, \mathcal{M}(l')] \cdots \vdash \forall l.\, c_2(P[a, c_3(\mathcal{M}(\lfloor l \rfloor))^{\downarrow}])^{\uparrow}$$

The expectation is that an inward ripple will determine the identity of $\mathcal{M}$. Our approach to the problem of constraining the instantiation of schematic terms will be detailed in §5. We will refer to the above generalization as the *basic critic*.

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

3 Limitations of the Basic Critic

The basic critic described in §2.3 has proved very successful [11]. Through our empirical testing, however, a number of limitations have been observed:

1. Certain classes of example require the introduction of multiple accumulator variables. The basic critic only deals with single accumulators.
2. The basic critic was designed in the context of equational proofs. An accumulator variable is assumed to occur on both sides of an equation. On the side opposite to the blockage term it is assumed that in the resulting generalized term structure the accumulator (auxiliary) will occur as an argument of the outermost functor.
3. Accumulator term occurrences which are motivated by blockage terms are more constrained than those which are not. This is not exploited by the basic critic during the search for a generalization.

From these observations a number of natural extensions to the basic critic emerged. These extensions are described in the following sections.

4 Specifying Accumulator Terms

In order to exploit the distinction between different accumulator term occurrences hinted at above we extend the meta-level annotations to include the notions of *primary* and *secondary* wave-fronts. A wave-front which provides the basis for a sideways ripple but which is not applicable because of the presence of a blockage term is designated to be *primary*. All other wave-fronts are designated to be *secondary*. To illustrate, consider the following schematic conclusion:

$$g(f(c_1(a,b)^{\uparrow},d),c_1(a,b)^{\uparrow}) \tag{5}$$

and the following wave-rules:

$$f(c_1(X,Y)^{\uparrow},Z) \Rightarrow f(X,c_2(Z,Y)^{\downarrow}) \tag{6}$$

$$g(X,c_1(Y,Z)^{\uparrow}) \Rightarrow c_3(g(X,Y),Z)^{\uparrow} \tag{7}$$

Assuming that the occurrence of d in (5) denotes a blockage term then wave-rule (6) is not applicable. Wave-rule (7) is applicable and enables an outwards ripple, *i.e.*

$$c_3(g(f(c_1(a,b)^{\uparrow},d),a),b)^{\uparrow}$$

Using subscripts[4] to denote primary and secondary wave-fronts then the analysis presented above gives rise to the following classification of the wave-fronts appearing in (5):

$$g(f(c_1(a,b)^{\uparrow}_1,d),c_1(a,b)^{\uparrow}_2) \tag{8}$$

[4]Note that wave-rules must also take account of the extension to the wave-front annotations.

4.1 Primary Accumulator Terms

For each primary wave-front an associated accumulator term is introduced. We refer to these as *primary accumulator terms*. The position of a primary accumulator term corresponds to the position of the blockage term within the conclusion. The structure of a primary accumulator term is a function of the blockage term and is computed as follows:

$$\mathtt{pri}(X) = \begin{cases} \mathcal{M}_i(\lfloor l_i \rfloor) & \text{if } X \text{ is a constant} \\ \mathcal{M}_i(X, \lfloor l_i \rfloor) & \text{if } X \text{ is a wave-front} \\ F(\mathtt{pri}(Y_1), \ldots, \mathtt{pri}(Y_n)) & \text{otherwise} \\ \quad \text{where } X \equiv F(Y_1, \ldots, Y_n) & \end{cases}$$

Note that $\mathcal{M}_i$ denotes a higher-order meta-variable while l_i denotes a new object-level variable. Assuming d denotes a constant then $\mathtt{pri}(d)$ evaluates to $\mathcal{M}_1(\lfloor l_1 \rfloor)$. Substituting this accumulator term for d in (8) gives a schematic conclusion of the form:

$$g(f(\boxed{c_1(a,b)}^{\uparrow}_1, \mathcal{M}_1(\lfloor l_1 \rfloor)), \boxed{c_1(a,b)}^{\uparrow}_2) \tag{9}$$

4.2 Secondary Accumulator Terms

For each secondary wave-front we eagerly attempt to apply a sideways ripple by introducing occurrences of the variables associated with the primary accumulator terms. These occurrences are specified again using schematic term structures and are called *secondary accumulator terms*. The construction of secondary accumulator terms are as follows. For each subterm, X, of the conclusion which contains a secondary wave-front, we compute a secondary accumulator term as follows:

$$\mathtt{sec}(X) = \mathcal{M}_i(X, \lfloor l_1 \rfloor, \ldots, \lfloor l_m \rfloor)$$

where $l_1, \ldots, l_m$ denote the vector of variables generated by the construction of the primary accumulator terms. To illustrate, consider again the schematic conclusion (9). Taking X to be $\boxed{c_1(a,b)}^{\uparrow}_2$ then the process of introducing secondary accumulator terms gives rise to a new schematic conclusion of the form:

$$g(f(\boxed{c_1(a,b)}^{\uparrow}_1, \mathcal{M}_1(\lfloor l_1 \rfloor)), \mathcal{M}_2(\boxed{c_1(a,b)}^{\uparrow}_2, \lfloor l_1 \rfloor)) \tag{10}$$

The selection of X represents a choice point which we delay discussion of until §6.

5 Instantiating Accumulator Terms

The process of instantiating the accumulator terms introduced by the generalization critic is guided by the application of wave-rules. In general, the application of wave-rules in the presence of schematic term structure requires higher-order unification. In our application we only require second-order unification. Below we show in detail how the meta-level annotations of a sideways ripple can be used to constrain the unification process.

Consider a schematic term of the form:

$$\mathcal{M}_1(\boxed{c_1(a,b)}^{\uparrow}_2, \lfloor l_1 \rfloor)$$

and the wave-rule:

$$f(\,c_1(X,Y)^{\uparrow}_{N}, Z) \Rightarrow f(X,\, c_2(Z,Y)^{\downarrow}_{N})$$

In order to apply the wave-rule we must unify the schematic term with the left-hand-side of the wave-rule. The process of unification is constrained by firstly performing a first-order match on the wave-fronts and the wave-holes[5]. This partially instantiates the wave-rule as follows:

$$f(\,c_1(a,b)^{\uparrow}_{2}, Z) \Rightarrow f(a,\, c_2(Z,b)^{\downarrow}_{2})$$

Secondly we higher-order unify the skeleton of the schematic term and the skeleton of the left-hand-side of the wave-rule. This further instantiates the wave-rule to give:

$$f(\,c_1(a,b)^{\uparrow}_{2}, \mathcal{M}_2(c_1(a,b), \lfloor l_1 \rfloor)) \Rightarrow f(a,\, c_2(\mathcal{M}_2(c_1(a,b), \lfloor l_1 \rfloor), b)^{\downarrow}_{2})$$

where $\mathcal{M}_1$ is instantiated to be $\lambda x.\lambda y.f(x, \mathcal{M}_2(x,y))$. The application of the wave-rule gives rise to a refined schematic term of the form:

$$f(a,\, c_2(\mathcal{M}_2(c_1(a,b), \lfloor l_1 \rfloor), b)^{\downarrow}_{2})$$

This should be compared with the proliferation of meta-variables introduced by unification if the constraints of rippling are not exploited, *i.e.*

$$f(\mathcal{M}_2(c_1(a,b), l_1), c_2(\mathcal{M}_4(c_1(a,b), l_1), \mathcal{M}_3(c_1(a,b), l_1)))$$

The application of an outwards ripple follows a similar pattern. In the case of an inwards ripple the first-order match is only performed on the wave-fronts and not the wave-holes. To illustrate, consider the following schematic term:

$$c_2(\mathcal{M}_1(a, \lfloor l_1 \rfloor), b)^{\downarrow}_{1}$$

and the application of a wave-rule of the form:

$$c_2(f(X,Z),Y)^{\downarrow}_{N} \Rightarrow f(\,c_1(X,Y)^{\downarrow}_{N}, Z)$$

The schematic term resulting from the inwards ripple takes the form:

$$f(\,c_1(\mathcal{M}_2(a, \lfloor l_1 \rfloor), b)^{\downarrow}_{1}, \mathcal{M}_3(a, \lfloor l_1 \rfloor)) \tag{11}$$

where $\mathcal{M}_1$ is instantiated to be $\lambda x.\lambda y.f(\mathcal{M}_2(x,y), \mathcal{M}_3(x,y))$. Note that in this case rippling does not reduce the number of meta-variables introduced by the unification process. However, by maintaining the sink annotations rippling does constrain the selection of subsequent projections. Projections are used to eagerly terminate inward ripples. A projection is applied whenever the immediate superterm of an accumulator term is an inward directed wave-front. To illustrate, in the case of (11) the sink annotation results in $\mathcal{M}_2$ being instantiated to be a projection onto its second argument, *i.e.*

$$f(\lfloor\, c_1(l_1,b)^{\downarrow}_{1} \rfloor, \mathcal{M}_3(a, \lfloor l_1 \rfloor))$$

Note that while rippling is complete a meta-variable still remains. There are a number of ways in which one might attempt to instantiate such a meta-variable. We shall delay discussion, however, until §9. The strategy of eager instantiation of meta-variables may of course give rise to an over-generalization, *i.e.* a non-theorem. A conjecture disprover, therefore, is used to filter candidate instantiations of the schematic conjecture. On detecting an non-theorem the critic mechanism backtracks and attempts further rippling.

[5] The wave-hole is the subterm of the skeleton term structure which occurs immediately beneath the wave-front.

6 Organizing the Search Space

In controlling the search for a generalization we place a number of constraints on the proof planning process:

- Planning in the context of schematic term structures requires a bounded search strategy. We use an iterative deepening strategy based upon the length of ripple paths[6].
- Backtracking over the construction of secondary accumulator terms deals with the choice point issue raised in §4. To illustrate, consider again schematic conclusion (10). Failure to find a valid instantiation of (10), for a given ripple path depth, results in an incremental increase in the size of the secondary accumulator term, *i.e.*

  $$\mathcal{M}_2(g(f(c_1(a,b)^{\uparrow}_1, \mathcal{M}_1(\lfloor l_1 \rfloor)), c_1(a,b)^{\uparrow}_2), \lfloor l_1 \rfloor)$$

  By this process of revision all possible secondary accumulator term positions can be systematcally explored. Note that no revision of primary accumulator terms is required.
- Since primary accumulator terms are more constrained than secondary accumulator terms priority is given to the rippling of primary wave-fronts.

7 Implementation and Testing

The extensions to the basic critic described above directly address the limitations highlighted in §3:

1. The linkage of blockage terms with the introduction of primary accumulator terms within the schematic conjecture addresses the issue of multiple accumulator variables.
2. The issue of positioning auxiliary accumulator variables is dealt with by the ability to revise the construction of secondary accumulator terms.
3. By extending the meta-logic to include the notions of primary and secondary wave-fronts we are able to exploit the observation that certain accumulator occurrences are more constrained than others during the search for generalizations.

Our extended critic has been implemented and integrated within the CLAM proof planner [5]. The implementation makes use of the higher-order features of λ-Prolog [15]. Below we document the testing of our implementation.

7.1 Experimental Results

The results presented in [11] for the basic critic were replicated by the extended critic. The extended critic, however, discovered generalizations which the basic critic missed. Moreover, a number of new examples were generalized by the extended critic for which the application of the basic critic resulted in failure. Our results are documented in the tables given in appendix C. The example conjectures for which the extended critic improves upon the performance of the

[6] Given a wave-front, its associated *ripple paths* are defined to be the sequence(s) of term tree positions which can be reached by the application of wave-rules. The length of a particular ripple path is defined to be the number of wave-rule applications used in its construction.

basic critic are presented in table I. All the examples require accumulator generalization and therefore cannot be proved automatically by other inductive theorem provers such as NQTHM [2]. The relative performance of the basic and extended critics on the example conjectures is recorded in table II. The lemmata used in motivating the generalizations are presented in table III while the actual generalized conjectures are given in table IV. All these generalizations are discovered automatically, *i.e.* no user intervention.

7.2 A Case Study

To illustrate more fully the mechanism presented above consider again verification conjecture (2) given in §1. We focus upon the role our extended critic plays in automating the proof. In particular, how it generates (3), the required generalization. The wave-rules required for this proof are given in appendix B. With the exception of wave-rules (17) and (18) all the wave-rules are derived from definitions.

7.2.1 First proof attempt

An inductive proof of (2) requires induction on the structure of the list t. The base case goal is trivial. We focus here on the step case goal which gives rise to an induction hypothesis of the form:

$$\forall f' : A \to B.\forall n' : \mathbb{N}.$$
$$map(f', t) = reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(f', x), split_1(1, n', nil, t)))$$

and an induction conclusion of the form:

$$map(\lfloor f \rfloor, \boxed{h :: t}^{\uparrow}) = reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(\lfloor f \rfloor, x), split_1(1, \lfloor n \rfloor, nil, \boxed{h :: t}^{\uparrow})))$$

Wave-rule (15) is applicable and gives rise to a conclusion of the form:

$$\boxed{(f@h) :: map(\lfloor f \rfloor, t)}^{\uparrow} = reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(\lfloor f \rfloor, x), split_1(1, \lfloor n \rfloor, nil, \boxed{h :: t}^{\uparrow})))$$

However, wave-rules (13) and (14) are not applicable because of the blockage terms 1 and nil which occur in the first and third argument positions of $split_1$. Triggered by these blockage terms the extended generalization critic generates a schematic hypothesis of the form:

$$\forall f' : A \to B.\forall n' : \forall l_1' : \mathbb{N}.\forall l_2' : list(A)$$
$$map(f', \mathcal{M}_3(t, l_1', l_2')) =$$
$$reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(f', x), split_1(\mathcal{M}_1(l_1'), n', \mathcal{M}_2(l_2'), t)))$$

while the schematic conclusion takes the form:

$$map(\lfloor f \rfloor, \mathcal{M}_3(\boxed{h :: t}_2^{\uparrow}, \lfloor l_1 \rfloor, \lfloor l_2 \rfloor)) =$$
$$reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(\lfloor f \rfloor, x), split_1(\mathcal{M}_1(\lfloor l_1 \rfloor), \lfloor n \rfloor, \mathcal{M}_2(\lfloor l_2 \rfloor), \boxed{h :: t}_1^{\uparrow})))$$

Note that the blockage terms 1 and nil have been replaced by primary accumulator terms $\mathcal{M}_1(\lfloor l_1 \rfloor)$ and $\mathcal{M}_2(\lfloor l_2 \rfloor)$ respectively. Note also that the wave-front on the left-hand-side of the goal equation is classified as secondary and consequently it is associated with a secondary accumulator term which contains occurrences of l_1 and l_2.

7.2.2 Second proof attempt

The ripple method is now applied to the schematic goal. Priority is given to the rippling of primary wave-fronts so there is no choice as to which wave-rules should be initially applied. The introduction of accumulator terms $\mathcal{M}_1(\lfloor l_1 \rfloor)$ and $\mathcal{M}_2(\lfloor l_2 \rfloor)$ enable wave-rules (13) and (14) to be applied. Jointly they motivate a case split on $\mathcal{M}_1(l_1)$ and n.

Case: $\mathcal{M}_1(l_1) \leq n$

Using wave-rule (13) a sideways ripple can be applied to the right-hand-side of the conclusion:

$$\ldots = \mathtt{reduce}(\lambda x.\lambda y.\mathtt{app}(x,y), \mathtt{map}(\lambda x.\mathtt{map}(\lfloor f \rfloor, x), \mathtt{split}_1(\lfloor l_1 + 1^{\downarrow}_1 \rfloor, \lfloor n \rfloor, \lfloor \mathtt{atend}(h, l_2)^{\downarrow}_1 \rfloor, t)))$$

Note that $\mathcal{M}_1$ and $\mathcal{M}_2$ have been eagerly instantiated to be projections, *i.e.* $\lambda x.x$. The left-hand-side of the conclusion contains a secondary accumulator term so rippling involves search. The sink instantiations, however, on the right-hand-side can be exploited in constraining this search, *i.e.* wave-rule (18) gives rise to:

$$\mathtt{map}(\lfloor f \rfloor, \mathtt{app}(\mathtt{atend}(h, \mathcal{M}_4(h :: t, \lfloor l_1 \rfloor, \lfloor l_2 \rfloor))^{\downarrow}_2), t) = \ldots$$

which instantiates $\mathcal{M}_3$ to be $\lambda x.\lambda y.\lambda z.\mathtt{app}(\mathcal{M}_4(x,y,z),x)$. Note that to be consistent with the sink instantiations on the right-hand-side of the conclusion, $\mathcal{M}_4$ must be instantiated to be a projection of the form $\lambda x.\lambda y.\lambda z.z$. The rippling in this branch of the case split is complete:

$$\begin{aligned}&\mathtt{map}(\lfloor f \rfloor, \mathtt{app}(\lfloor \mathtt{atend}(h, l_2)^{\downarrow}_2 \rfloor)) = \\ &\quad \mathtt{reduce}(\lambda x.\lambda y.\mathtt{app}(x,y), \mathtt{map}(\lambda x.\mathtt{map}(\lfloor f \rfloor, x), \mathtt{split}_1(\lfloor l_1 + 1^{\downarrow}_1 \rfloor, \lfloor n \rfloor, \lfloor \mathtt{atend}(h, l_2)^{\downarrow}_1 \rfloor, t)))\end{aligned}$$

The induction hypothesis can be applied by instantiating l'_1 to be $l_1 + 1$ and l'_2 to be $\mathtt{atend}(h, l_2)$. The instantiations for $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ are propagated through the remaining branch of the case split.

Case: $l_1 > n$

Using wave-rule (14) the right-hand-side of the conclusion ripples to give:

$$\ldots = \mathtt{reduce}(\lambda x.\lambda y.\mathtt{app}(x,y), \mathtt{map}(\lambda x.\mathtt{map}(\lfloor f \rfloor, x), l_2 :: \mathtt{split}_1(\lfloor 2^{\downarrow}_1 \rfloor, \lfloor n \rfloor, \lfloor h :: \mathtt{nil}^{\downarrow}_1 \rfloor, t)^{\uparrow}_1))$$

By wave-rule (15) the conclusion ripples further to give[7]:

$$\ldots = \mathtt{reduce}(\lambda x.\lambda y.\mathtt{app}(x,y), \mathtt{map}(f, l_2) :: \mathtt{map}(\lambda x.\mathtt{map}(\lfloor f \rfloor, x), \mathtt{split}_1(\lfloor 2^{\downarrow}_1 \rfloor, \lfloor n \rfloor, \lfloor h :: \mathtt{nil}^{\downarrow}_1 \rfloor, t))^{\uparrow}_1)$$

A further outward ripple using wave-rule (16) gives[8]:

$$\ldots = \mathtt{app}(\mathtt{map}(f, l_2), \mathtt{reduce}(\lambda x.\lambda y.\mathtt{app}(x,y), \mathtt{map}(\lambda x.\mathtt{map}(\lfloor f \rfloor, x), \mathtt{split}_1(\lfloor 2^{\downarrow}_1 \rfloor, \lfloor n \rfloor, \lfloor h :: \mathtt{nil}^{\downarrow}_1 \rfloor, t))))^{\uparrow}_1$$

[7] Note that the conclusion has been β-reduced automatically.

[8] Again the conclusion has been β-reduced automatically.

Using wave-rule (17) the left-hand-side of the conclusion becomes:

$$app(map(f, l_2), map(\lfloor f \rfloor, app(\lfloor h :: nil \downarrow_2 \rfloor, t)))^{\uparrow}_2 = \ldots$$

Finally, by wave-rule (19) the rippling of the conclusion is complete:

$$map(\lfloor f \rfloor, app(\lfloor h :: nil \downarrow_2 \rfloor, t)) =$$
$$reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(\lfloor f \rfloor, x), split_1(\lfloor 2 \downarrow_1 \rfloor, \lfloor n \rfloor, \lfloor h :: nil \downarrow_1 \rfloor, t)))$$

The induction hypothesis can be applied by instantiating l'_1 to be 2 and l'_2 to be $h :: nil$. To summarize, the ripple method in conjunction with the extended critic have automatically generated (3), the required generalization of (2). A proof of (3) can be constructed by CLAM completely automatically.

8 Related Work

Jane Hesketh in her thesis work [9] also tackled the problem of accumulator generalization in the context of proof planning. Her approach, however, did not deal with multiple accumulators. By introducing the primary and secondary classification of wave-fronts we believe that our approach provides greater control in the search for generalizations. This becomes crucial as the complexity[9] of examples increases. In addition, we use sink annotations explicitly in selecting potential projections for higher-order meta-variables.

Jane's work, however, was much broader than ours in that she unified a number of different kinds of generalization. Moreover, she was also able to synthesize tail-recursive functions given equivalent naive recursive definitions [8].

9 Future Work

Our results for the extended critic have been promising. More testing is planned. We believe that our technique is not restricted to reasoning about functional programs. This will be reflected in future testing. Below we outline the key areas where we are looking to develop this work.

9.1 Automatic Discovery of Loop Invariants

We believe that our technique transfers directly to imperative programs. Discovering a loop invariant is typically seen as a eureka step in the process of verifying an imperative program. This is reflected in the fact that some of the major contributions in this area rely to a large extent upon user interaction, *e.g.* in the GYPSY verification environment [6] all loop invariants are supplied by the user. A common strategy for discovering invariants is to start with a desired post-condition from which the invariant is derived by a process of weakening. The notion of a *tail invariant* [12] represents one such way of deriving an invariant. The search for a tail invariant is appropriate when the desired post-condition takes the form:

$$r = f(X, Y)$$

[9] That is, as the number of definitions and lemmata available to the prover increases.

where r denotes a program variable while f denotes a tail recursive function and X and Y denote constants. Given a post-condition of this form then the required (tail) invariant takes the form:

$$f(x, r) = f(X, Y)$$

where the initial value of the program variable x is X. A special case of this scheme occurs when the post-condition takes the form

$$r = g(X)$$

where g is not tail recursive. In such situations the tail invariant can be specified by the following schema:

$$\mathcal{M}(g(x), r) = g(X)$$

where $\mathcal{M}$ denotes a second-order meta-variable. The problem of discovering the invariant is reduced to finding the identity of $\mathcal{M}$. There are strong similarities between step case and invariant proofs. The technique we have developed, therefore, can be used to guide the construction and instantiation of such schematic invariants.

9.2 Hardware Verification

We also believe that our technique is applicable in the context of hardware verification. For instance, we believe that it subsumes the procedure described in [16] for generalizing hardware specifications.

9.3 User Interaction

The critic mechanism was motivated by a desire to build an automatic theorem prover which was more robust than the conventional provers. The high-level representation provided by a proof plan enabled us to achieve this goal. We believe, however, that the critic mechanism also provides a basis for developing effective user interaction. To illustrate, consider conjecture C5 from table I (appendix C). Based purely upon the definitions arising from the statement of the conjecture the extended critic, as currently implemented, automatically generates the following partial generalization:

$$app(partition(evenel(X), Y, \mathcal{M}_1(X, Y, Z)), partition(oddel(X), \mathcal{M}_2(X, Y, Z), Z)) = partition(X, Y, Z)$$

We are currently implementing an interactive version of the critic mechanism which will invite the user to complete the instantiation of such partial generalizations. An obvious candidate here is $\lambda x.\lambda y.\lambda z.nil$ which gives rise to the following generalized conjecture:

$$app(partition(evenel(X), Y, nil), partition(oddel(X), nil, Z)) = partition(X, Y, Z)$$

Note that this generalization of C5 is easily proved by CLAM.

10 Conclusion

The search for inductive proofs cannot avoid the problem of generalization. In this paper we describe extensions to a proof critic for automatically generalizing inductive conjectures. The ideas presented here build upon a proof patch mechanism documented in [11]. These extensions have significantly improved the performance of the technique while preserving the spirit of original proof patch. Our implementation of the extended critic has been tested on the verification of functional programs with some promising results. More generally, we believe that our technique has wider application in terms of both software and hardware verification.

Appendix A: definitional rewrite rules

$$
\begin{aligned}
atend(X, nil) &\Rightarrow X :: nil \\
atend(X, Y :: Z) &\Rightarrow Y :: atend(X, Z) \\
split_1(V, W, X, nil) &\Rightarrow X :: nil \\
V \leq W \rightarrow split_1(V, W, X, Y :: Z) &\Rightarrow split_1(V + 1, W, atend(Y, X), Z) \\
V > W \rightarrow split_1(V, W, X, Y :: Z) &\Rightarrow X :: split_1(2, W, Y :: nil, Z) \\
split(X, Y) &\Rightarrow split_1(1, X, nil, Y) \\
app(nil, Z) &\Rightarrow Z \\
app(X :: Y, Z) &\Rightarrow X :: app(Y, Z) \\
map(X, nil) &\Rightarrow nil \\
map(X, Y :: Z) &\Rightarrow (X@Y) :: map(X, Z) \\
reduce(X, nil) &\Rightarrow nil \\
reduce(X, Y :: Z) &\Rightarrow ((X@Y)@reduce(X, Z)) \\
rev(nil) &\Rightarrow nil \\
rev(X :: Y) &\Rightarrow app(rev(Y), X :: nil) \\
qrev(nil, Z) &\Rightarrow Z \\
qrev(X :: Y, Z) &\Rightarrow qrev(Y, X :: Z) \\
revflat(nil) &\Rightarrow nil \\
revflat(X :: Y) &\Rightarrow app(revflat(Y), X) \\
qrevflat(nil, Z) &\Rightarrow Z \\
qrevflat(X :: Y, Z) &\Rightarrow qrevflat(Y, app(X, Z)) \\
perm(nil, nil) &\Rightarrow true \\
perm(X :: Y, Z) &\Rightarrow perm(Y, delete(X, Z)) \wedge member(X, Z)) \\
evenel(nil) &\Rightarrow nil \\
even(X) \rightarrow evenel(X :: Y) &\Rightarrow X :: evenel(Y) \\
odd(X) \rightarrow evenel(X :: Y) &\Rightarrow evenel(Y) \\
oddel(nil) &\Rightarrow nil \\
odd(X) \rightarrow oddel(X :: Y) &\Rightarrow X :: oddel(Y) \\
even(X) \rightarrow oddel(X :: Y) &\Rightarrow oddel(Y) \\
partition(nil, Y, Z) &\Rightarrow app(Y, Z) \\
even(W) \rightarrow partition(W :: X, Y, Z) &\Rightarrow partition(X, atend(W, Y), Z) \\
odd(W) \rightarrow partition(W :: X, Y, Z) &\Rightarrow partition(X, Y, atend(W, Z))
\end{aligned}
\tag{12}
$$

Appendix B: selection of example wave-rules

$$V \leq W \rightarrow split_1(V, W, X, Y :: Z\,\boxed{}^{\uparrow}_{N}) \Rightarrow split_1(\,V + 1\,\boxed{}^{\downarrow}_{N}, W, atend(Y, X)\,\boxed{}^{\downarrow}_{N}, Z) \tag{13}$$

$$V > W \rightarrow split_1(\lfloor V \rfloor, W, \lfloor X \rfloor, Y :: Z\,\boxed{}^{\uparrow}_{N}) \Rightarrow X :: split_1(\left\lfloor 2\,\boxed{}^{\downarrow}_{N} \right\rfloor, W, \left\lfloor Y :: nil\,\boxed{}^{\downarrow}_{N} \right\rfloor, Z)\,\boxed{}^{\uparrow}_{N} \tag{14}$$

$$map(X, Y :: Z\,\boxed{}^{\uparrow}_{N}) \Rightarrow (X@Y) :: map(X, Z)\,\boxed{}^{\uparrow}_{N} \tag{15}$$

$$reduce(X, Y :: Z\,\boxed{}^{\uparrow}_{N}) \Rightarrow ((X@Y)@reduce(X, Z))\,\boxed{}^{\uparrow}_{N} \tag{16}$$

$$map(W, app(\lfloor X \rfloor, Y :: Z\,\boxed{}^{\uparrow}_{N})) \Rightarrow app(map(W, X), map(W, app(\left\lfloor Y :: nil\,\boxed{}^{\downarrow}_{N} \right\rfloor, Z)))\,\boxed{}^{\uparrow}_{N} \tag{17}$$

$$app(X, Y :: Z\,\boxed{}^{\uparrow}_{N}) \Rightarrow app(\,atend(Y, X)\,\boxed{}^{\downarrow}_{N}, Z) \tag{18}$$

$$app(X, Y)\,\boxed{}^{\uparrow}_{M} = app(X, Z)\,\boxed{}^{\uparrow}_{N} \Rightarrow Y = Z \tag{19}$$

Appendix C: experimental results

No	Conjecture
C1	$rev(X) = qrev(X, nil)$
C2	$revflat(X) = qrevflat(X, nil)$
C3	$qrev(qrev(X, nil), nil) = rev(rev(X))$
C4	$permute(rev(X), qrev(X, nil))$
C5	$app(evenel(X), oddel(X)) = partition(X, nil, nil)$
C6	$map(F, X) = reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(F, x), split_1(1, W, nil, X)))$

Table I: conjectures

No	Basic Critic	Extended Critic
C1	G1	G1, G2
C2	G3	G3, G4
C3	G5,G6,G7,G8,G9	G5,G6,G7,G8,G9,G10,G11
C4	FAILURE	G12,G13,G14
C5	FAILURE	G15
C6	FAILURE	G16

The improved performance of the extended critic on conjectures C1, C2 and C3 can be attributed to its ability to revise the construction of secondary accumulator terms. The failure of the basic critic on conjecture C4 is due to its "artificial" restrictions on the placement of secondary accumulator terms. The same is true for C5 and C6 but in addition both these conjectures require multiple accumulators.

Table II: performance of generalization critics

No	Lemma
L1	$app(app(X, Y), Z) = app(X, app(Y, Z))$
L2	$app(app(X, Y :: nil), Z) = app(X, Y :: Z)$
L3	$rev(app(X, Y :: nil)) = Y :: rev(X)$
L4	$app(X, Y :: Z) = app(atend(Y, X), Z)$
L5	$map(W, app(X, Y :: Z)) = app(map(W, X), map(W, app(Y :: nil, Z)))$

Table III: lemmata used to motivate generalizations

No	Generalization	Lemmata
G1	$rev(qrev(Y, X)) = qrev(X, Y)$	
G2	$app(rev(X), Y) = qrev(X, Y)$	L1
G3	$revflat(qrevflat(Y, X))$	
G4	$app(revflat(X), Y) = qrevflat(X, Y)$	L1
G5	$qrev(qrev(X, Y), nil) = app(rev(Y), rev(rev(X)))$	L2 & L3
G6	$qrev(qrev(X, Y), nil) = qrev(Y, rev(rev(X)))$	L3
G7	$qrev(qrev(X, Y), nil) = qrev(rev(rev(Y)), rev(rev(X)))$	L3
G8	$qrev(qrev(X, rev(Y)), nil) = app(Y, rev(rev(X)))$	L2 & L3
G9	$qrev(qrev(X, rev(rev(Y))), nil) = qrev(Y, rev(rev(X)))$	L3
G10	$qrev(qrev(X, Y), nil) = rev(app(rev(X), Y))$	L1
G11	$qrev(qrev(X, Y), nil) = rev(rev(qrev(Y, X)))$	
G12	$perm(rev(qrev(X, Y)), qrev(X, Y))$	
G13	$perm(rev(qrev(Y, X)), qrev(X, Y))$	
G14	$perm(app(rev(X), Y), qrev(X, Y))$	L1
G15	$app(app(Y, evenel(X)), app(Z, oddel(X))) = partition(X, Y, Z)$	L4
G16	$map(F, app(Y, X)) = reduce(\lambda x.\lambda y.app(x, y), map(\lambda x.map(F, x), split_1(Z, W, Y, X)))$	L4 & L5

Note that different combinations of lemmata give rise to different generalizations. These are indicated by the multiple references given in the third column. No entry appears if the generalization was discovered using purely definitional rewrite rules.

Table IV: generalized conjectures

References

[1] D. Basin and T. Walsh. Difference unification. In *Proceedings of the 13th IJCAI*. International Joint Conference on Artificial Intelligence, 1993. Also available as Technical Report MPI-I-92-247, Max-Planck-Institute für Informatik.

[2] R.S. Boyer and J.S. Moore. *A Computational Logic*. Academic Press, 1979. ACM monograph series.

[3] 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.

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

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

[6] D.I. Good. Mechanical proofs about computer programs. In C.A.R. Hoare and J.C. Shepherdson, editors, *Mathematical Logic and Programming Languages*, chapter 3, pages 55–75. Prentice-Hall, 1985.

[7] 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.

[8] J. Hesketh, A. Bundy, and A. Smaill. Using middle-out reasoning to control the synthesis of tail-recursive programs. In Deepak Kapur, editor, *11th Conference on Automated Deduction*, pages 310–324, Saratoga Springs, NY, USA, June 1992. Published as Springer Lecture Notes in Artificial Intelligence, No 607.

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

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

[11] A. Ireland and A. Bundy. Productive use of failure in inductive proof. Research Paper 716, Dept. of Artificial Intelligence, Edinburgh, 1995. To appear in the special issue of JAR on inductive proof.

[12] A. Kaldewaij. *Programming: The Derivation of Algorithms*. Prentice Hall, London, 1990.

[13] G. Kreisel. Mathematical logic. In T.L. Saaty, editor, *Lectures on Modern Mathematics*, volume III, pages 95–195. John Wiley and Sons, 1965.

[14] G. Michaelson and N. Scaife. Prototyping a parallel vision system in Standard ML. *Journal of Functional Programming*, 5:345–382, 1995.

[15] D. Miller and G. Nadathur. An overview of λProlog. In R. Bowen, K. & Kowalski, editor, *Proceedings of the Fifth International Logic Programming Conference/ Fifth Symposium on Logic Programming*. MIT Press, 1988.

[16] L Pierre. An automatic generalization method for the inductive proof of replicated and parallel architectures. In R.Kumar & T.kropf, editor, *Theorem Provers in Circuit Design*. LNCS 901, Springer-Verlag, 1995.