On the Automatic Discovery of Loop Invariants*

Andrew Ireland† and Jamie Stark‡
Department of Computing & Electrical Engineering
Heriot-Watt University
Edinburgh, Scotland, U.K. EH14 4AS

Abstract

We present a technique for automating the discovery of loop invariants based upon the analysis of failed proof attempts. Previously we have shown how failure analysis may be used productively in the search for inductive proofs. This work had direct application to the verification of functional programs. Here we show how these ideas can also play an important role in the formal verification of imperative programs. While presented as an automatic technique we believe that our approach may be easily integrated within an interactive proof environment.

1 Introduction

The notion of an invariant is a well established technique for specifying the behaviour of computer programs. The discovery and verification of suitable invariants remains, however, a significant challenge to the formal verification community. While some believe that the challenge is too great [9] we feel there is still significant progress to be made in terms of mechanizing the wealth of heuristic knowledge which exists within the programming methodology literature [1, 8, 16, 22].

We focus here on the problems of reasoning about imperative programs, and in particular, the search control issues associated with the discovery and verification of loop invariants. In terms of formal verification the discovery of a loop invariant is typically seen as a eureka step. This is reflected in the fact that some of the foremost verification environments, *e.g.* GYPSY [12] and PENELOPE [17], rely on the user to supply all loop invariants.

We build upon *proof plans* [3], a meta-level reasoning technique for automating proof search. The technique is implemented in the CLAM theorem proving system [5]. Given a conjecture, CLAM attempts to automatically generate a LCF style tactic [15] guided by its available proof plans. If successful then the tactic is used to control proof construction in an object-level theorem prover. Currently CLAM is coupled to a tactic based theorem prover called OYSTER[1]. By explicitly separating the meta- and object-levels, CLAM provides an effective framework for developing and sharing proof strategies. That is, CLAM can potentially be used to support proof construction in any theorem proving environment which supports a tactic or proof script mechanism, *e.g.* LARCH [11] and HOL[2] [13].

Our approach to discovering loop invariants is based upon the analysis of failed proof attempts. Proof *critics* [19], an extension to the basic idea of a proof plan, support the analysis of proof failures. Previously this approach to automating formal reasoning has been applied very successfully within the context of inductive proof [20, 21]. This success exploited the constraints of *rippling* [4], a proof plan designed to guide step case proofs. The systematic analysis of partial success of rippling enabled us to

*The research reported in this paper was supported by EPSRC grant GR/L/11724 and ARC grant 438.

†A.Ireland@hw.ac.uk
‡jamie@cee.hw.ac.uk

[1] A Prolog implementation of NUPRL [7].

[2] Currently CLAM is being coupled to HOL through a collaborative research project between the Artificial Intelligence Department in Edinburgh and the Computer Laboratory in Cambridge funded by EPSRC grant GR/L/14381.

bridge gaps within the formal reasoning process, *e.g.* to automate the discovery of missing lemmata and generalizations.

Our contribution here is two fold. Firstly, we demonstrate that rippling can also play a role in automating the verification of loop invariants. Secondly, we show that the critics developed for inductive proof provide a basis for automating the discovery of loop invariants through the analysis of failed verification proofs.

Background material on reasoning about the behaviour of imperative programs is covered in §2 where we focus in particular on the while-loop construct. Building on the close connection between induction and iteration we show in §3 how a generalization of the ripple method enables its application to loop invariant proofs. Exploiting this connection further, we demonstrate in §4 how an existing generalization critic developed for inductive proof can be used as a basis for guiding the discovery of loop invariants automatically. Our implementation and experimental results are described in §5. Related work and future extensions are discussed in §6 and §7 respectively.

2 Reasoning about programs

We consider the problem of reasoning about the behaviour of imperative programs in the context of a Floyd-Hoare style logic [10, 18]. Specifications in this logic take the form of a triple, *i.e.* $\{P\}C\{Q\}$, where P and Q denote first order assertions which express the pre- and post-conditions with respect to C, an imperative program. To illustrate, consider `exp1` given in figure 1 for computing exponentiation[3]. Note that we adopt the convention of using calligraphic letters to denote values while lower case letters are used to denote program variables. The process of verification is typically divided into three steps as described below:

1. **Assertion propagation:** using the given assertions and the axiomatic definition of the language constructs, intermediate program assertions are calculated. To illustrate, consider the rule defining the while-loop:

$$\frac{\vdash \{P \wedge S\}\ C\ \{P\}}{\vdash \{P\}\ \textbf{while}\ S\ \textbf{do}\ C\ \{P \wedge \neg S\}}$$

Note however that in order to apply this rule we must discover some property P which is invariant with respect to the execution of C, the body of the loop. In the case of the exponentiation algorithms an appropriate *loop invariant* takes the form:

$$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y}) \tag{1}$$

This choice of P, in the case of `exp1`, gives rise to the following intermediate assertions:

$\{r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y}) \wedge y > 0\}$
begin
  $r := r * x;$
  $y := y - 1$
end
$\{r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})\}$

Propagation of assertions is complete once all atomic program statements are assigned pre- and post-conditions.

2. **Verification condition generation:** given the pre- and post-conditions for an atomic program statement a purely mathematical statement of partial correctness is extracted. Such statements are called *verification conditions*. The verification condition generated by the assignment command, *i.e.*

$$\{P\}V := E\{Q\}$$

takes the form $P \rightarrow Q[E/V]$. Based upon this schema the verification condition generated for the loop-body given above takes the form:

$$\begin{aligned} r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y}) \wedge (y > 0) \rightarrow \\ (r * x) * exp(x, (y - 1)) = exp(\mathcal{X}, \mathcal{Y}) \end{aligned} \tag{2}$$

3. **Theorem proving:** verification of the overall program is based upon the verification of the atomic program statements. The verification task requires a theorem prover which can deal with implications such as (2).

In terms of calculating intermediate assertions a loop invariant represents a major bottle-neck to the formal verification process. Once the propagation of assertions is complete the generation of verification conditions is mechanical. Our overall approach is to use failure productively at step 3 in order to refine the choice of intermediate assertions, *i.e.* loop invariants, at step 1.

[3] Definitions are given in appendix A.

```
exp1:   {x = 𝒳 ∧ y = 𝒴}
        r := 1;
        while (y > 0) do
        begin
            r := r * x;
            y := y - 1
        end
        {r = exp(𝒳, 𝒴)}
```

```
exp2:   {x = 𝒳 ∧ y = 𝒴}
        r := 1;
        while (y > 0) do
        begin
            if odd(y) then r := r * x;
            y := y div 2;
            x := x * x
        end
        {r = exp(𝒳, 𝒴)}
```

Figure 1: Algorithms for computing exponentiation

3 Verifying loop invariants

Proofs which share commonality of structure will also share common methods of proof. This is the premise on which the notion of a proof plan is based. Loop invariant proofs and the proofs of inductive conjectures share commonality of structure. Schematically, proving a loop invariant gives rise to a goal of the form

$$P \rightarrow P[E/V]$$

where E denotes the term assigned to the variable V. Now compare this with the schematic structure of a step case goal, *i.e.*

$$\frac{\ldots \quad P \rightarrow P[E/V]}{\forall V.\ P(V)}$$

where E denotes the induction term and V the induction variable. Note that in both goals there exists a strong syntactic similarity between antecedent and consequent. The method of proof known as rippling was designed for step case goals. The commonality between step case and loop invariant proofs observed above led us to investigate the role in which rippling may play in guiding loop invariant proofs. Below we outline rippling and illustrate its applicability to step case and loop invariant proofs.

3.1 Rippling: the general pattern

Rewriting often involves the manipulation of a goal formula so that a target formula, *e.g.* a hypothesis or previously established result, can be applied. Rippling exploits syntactic similarities between the goal and target in guiding the selective application of rewrite rules. In particular it identifies term structure within the goal which prevents a match with the target. Such term structures are called *wave-fronts*. Conversely any term structure within the goal which corresponds to the target is called *skeleton*. In general, embedded within each wave-front will be parts of the skeleton term structure, these are known as *wave-holes*.

Wave-fronts, wave-holes and skeletons are meta-level notions. Given a schematic goal of the form $f(g(c(x), y), z)$ and a target of the form $f(g(x, y), z)$ then $c(\ldots)$ denotes a wave-front. We use a box and an underline to represent wave-fronts and wave-holes respectively, *e.g.* an annotated version of the goal given above takes the form:

$$f(g(\boxed{c(\underline{x})}^{\uparrow}, y), z)$$

The arrow is used to indicate the direction in which a wave-front can be moved with respect to the skeleton term structure. Directed wave-fronts enable the termination of rippling to be guaranteed [2]. The movement of wave-fronts is performed by *wave-rules*, a syntactic class of rewrite rule which preserves the skeleton while making progress to eliminating wave-fronts. Wave-rules are derived automatically from definitions and lemmata. Example wave-rules are provided in the following sections. Proofs guided by rippling can be classified in terms of the direction in which wave-fronts are moved with respect to the skeleton term structure. The three basic strategies for rippling are summarised in figure 2. For a completely formal account of rippling see [2, 4]. In general a proof may require a *hybrid* form of rippling. The

rippling-out:

$$f_1(\ldots(f_n(\boxed{c_1(\underline{\ldots})}^{\uparrow}))\ldots) \qquad \boxed{c_n(\underline{f_1(\ldots(f_n(\ldots))\ldots)})}^{\uparrow}$$

before after

rippling-sideways:

$$f_1(\boxed{c_1(\underline{\ldots})}^{\uparrow},\ldots,f_i(\ldots),\ldots) \qquad f_1(\ldots,\ldots,\boxed{c_i(\underline{f_i(\ldots)})}^{\downarrow},\ldots)$$

before after

rippling-in:

$$\boxed{c_n(\underline{f_1(\ldots f_n(\ldots)\ldots)})}^{\downarrow} \qquad f_1(\ldots f_n(\boxed{c_1(\underline{\ldots})}^{\downarrow})\ldots)$$

before after

Figure 2: The three basic rippling strategies

notion of sideways and hybrid rippling is crucial to our work on invariant discovery. We begin by illustrating how rippling can guide both step case and loop invariant proofs in the following sections.

3.2 Rippling for step cases

Consider the following inductive conjecture:

$$\forall A, B, C : nat.\ mult(A, B, C) = (A * B) + C \qquad (3)$$

where $*$ and *mult* are defined in appendix A. Note that *mult* is tail-recursive. A proof of (3) may be constructed by structural induction on A giving rise to a step case goal in which we have an induction hypothesis of the form:

$$\forall B, C : nat.\ mult(a, B, C) = (a * B) + C \qquad (4)$$

The annotated version of the associated conclusion takes the form:

$$mult(\boxed{\underline{a}+1}^{\uparrow}, \lfloor b \rfloor, \lfloor c \rfloor) = (\boxed{\underline{a}+1}^{\uparrow} * \lfloor b \rfloor) + \lfloor c \rfloor \qquad (5)$$

A winning strategy in this context involves rippling-sideways. The motivation for rippling-sideways in this context is to direct wave-fronts towards term structures within the conclusion which correspond to universally quantified variables in the hypothesis. We call such term structures *sinks*, since they may "absorb" wave-fronts through the specialization of the induction hypothesis. We delimit sinks by the meta-level annotation $\lfloor \ldots \rfloor$. Note that the occurrences of b and c within (5) denote sinks since they correspond to the universal variables B and C within (4). The proof of (5) guided by rippling is given below:

$$\begin{aligned}
mult(\boxed{\underline{a}+1}^{\uparrow}, \lfloor b \rfloor, \lfloor c \rfloor) &= (\boxed{\underline{a}+1}^{\uparrow} * \lfloor b \rfloor) + \lfloor c \rfloor \\
mult(a, \lfloor b \rfloor, \left\lfloor \boxed{b+\underline{c}}^{\downarrow} \right\rfloor) &= (\boxed{\underline{a * \lfloor b \rfloor} + b}^{\uparrow}) + \lfloor c \rfloor \\
mult(a, \lfloor b \rfloor, \left\lfloor \boxed{b+\underline{c}}^{\downarrow} \right\rfloor) &= (a * \lfloor b \rfloor) + \left\lfloor \boxed{b+\underline{c}}^{\downarrow} \right\rfloor \qquad (6)
\end{aligned}$$

Note that by instantiating B and C to b and $b + c$ respectively then (4) and (6) match which completes the proof. The wave-rules which are required[4] for this proof are as follows[5]:

$$\begin{aligned}
\boxed{\underline{X}+1}^{\uparrow} * Y &\Rightarrow \boxed{\underline{(X * Y)} + Y}^{\uparrow} \qquad (7) \\
mult(\boxed{\underline{X}+1}^{\uparrow}, Y, Z) &\Rightarrow mult(X, Y, \boxed{Y+\underline{Z}}^{\downarrow}) \qquad (8) \\
\boxed{\underline{X}+Y}^{\uparrow} + Z &\Rightarrow X + \boxed{Y+\underline{Z}}^{\downarrow} \qquad (9)
\end{aligned}$$

Wave-rules (7) and (9) ripple the right-hand-side and wave-rule (8) is used on the left-hand-side. Here wave-rules (7), (8) and (9) are derived from equations (23), (25) and (26) respectively.

[4] Note that for a given theory **all possible** wave-rules are automatically derived and stored by CLAM.

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

3.3 Rippling for loop invariants

We demonstrate the applicability of rippling to loop invariant proofs by considering verification condition (2) which gives rise to an invariant hypothesis of the form:

$$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y}) \tag{10}$$

The associated annotated version of the conclusion takes the form:

$$\boxed{\underline{r} * x}^{\uparrow} * exp(x, \boxed{\underline{y} - 1}^{\uparrow}) = exp(\mathcal{X}, \mathcal{Y}) \tag{11}$$

Again the winning strategy involves rippling-sideways. However, in the context of loop invariant proofs there are no universally quantified variables within the invariant hypothesis to exploit. The absence of such quantification precludes the use of the meta-level notion of a sink but not sideways rippling. Here the motivation to ripple-sideways is to achieve a match with the invariant hypothesis through wave-front cancellation, *i.e.* destructive interference. The proof of (11) guided by rippling is as follows:

$$\begin{aligned}
\boxed{\underline{r} * x}^{\uparrow} * exp(x, \boxed{\underline{y} - 1}^{\uparrow}) &= exp(\mathcal{X}, \mathcal{Y}) \\
r * \boxed{x * \underline{exp(x, \boxed{\underline{y} - 1}^{\uparrow})}}^{\downarrow} &= exp(\mathcal{X}, \mathcal{Y}) \\
r * exp(x, y) &= exp(\mathcal{X}, \mathcal{Y})
\end{aligned} \tag{12}$$

Note that (10) matches (12) and completes the proof. The wave-rules required for the proof of (11) are as follows:

$$\boxed{\underline{X} * Y}^{\uparrow} * Z \Rightarrow X * \boxed{Y * \underline{Z}}^{\downarrow} \tag{13}$$

$$Y > 0 \rightarrow \boxed{X * \underline{exp(X, \boxed{\underline{Y} - 1}^{\uparrow})}}^{\downarrow} \Rightarrow exp(X, Y) \tag{14}$$

Wave-rule (13) comes from (27) while (14) is derived from (24), the definition of *exp*. Note that (14) provides the destructive interference of wave-fronts mentioned above and is an example of a hybrid wave-rule.

3.4 Generalizing rippling

Two alternative preconditions for rippling-sideways were demonstrated above. While sinks do not play a role within loop invariant proofs the notion of wave-front cancellation is applicable to both domains. For instance when proving an inductive conjecture in which both constructor and destructor style definitions are intermixed then wave-front cancellation will be crucial. Consequently, a more general precondition, than described in [4], is required which takes into account the potential for term cancellation and hypothesis specialization. This is reflected in the generalized version of the rippling preconditions given in figure 3.

4 Discovering loop invariants

The full benefits of a proof plan become apparent when a proof attempt fails. The declarative nature of method preconditions provides a basis for using failure productively. As mentioned in §1 we analyse failure by means of proof critics. Experimental evidence demonstrating the merits of this approach within the context of inductive proof can be found in [21]. This work documented a family of critics associated with the ripple method. Here we focus on one of these critics, a critic for generalizing inductive conjectures. In the same way that rippling has been shown to be applicable to loop invariant proofs we now show how the generalization critic can be transferred to this new domain.

4.1 A generalization critic

The generalization critic is triggered by the failure of precondition 4 of rippling (see figure 3). In the context of a step case proof this corresponds to a *missing sink*. To illustrate consider the following modified version of conjecture (3):

$$\forall A, B : nat.\ mult(A, B, 0) = A * B \tag{15}$$

Consider an attempt to prove (15) by structural induction on A. Using rippling to guide the associated step case proof we obtain an induction hypothesis of the form:

$$\forall B : nat.\ mult(a, B, 0) = a * B$$

Using wave-rule (7) the annotated conclusion ripples to give:

$$mult(\underbrace{\boxed{\underline{a} + 1}^{\uparrow}}_{\text{blocked}}, b, 0) = \underbrace{\boxed{\underline{a * b} + b}^{\uparrow}}_{\text{blocked}} \tag{16}$$

No further rippling is possible on either side of the equation. Wave-rule (8) fails to apply on the left-hand side because of a missing sink. This suggests

Input sequent:

$$H \vdash G[f_1(\boxed{c_1(\underline{\ldots})}^{\uparrow}, f_2(\lfloor \ldots \rfloor), f_3(\boxed{c_2(\underline{\ldots})}^{\uparrow}))]$$

Method preconditions:

1. there exists a subterm T of G which contains wave-front(s), *e.g.*

   $$f_1(\boxed{c_1(\underline{\ldots})}^{\uparrow}, f_2(\lfloor \ldots \rfloor), f_3(\boxed{c_2(\underline{\ldots})}^{\uparrow}))$$

2. there exists a wave-rule which matches T, *e.g.*

   $$C \to f_1(\boxed{c_1(\underline{X})}^{\uparrow}, Y, Z) \Rightarrow \boxed{c_5(\underline{f_1(X, \boxed{c_3(\underline{Y})}^{\downarrow}, \boxed{c_4(\underline{Z})}^{\downarrow})})}^{\uparrow}$$

3. the wave-rule condition follows from the context, *e.g.*

   $H \vdash C$

4. resulting inward directed wave-fronts are potentially removable, *e.g.*

   $$\ldots \boxed{c_3(\underline{f_2(\lfloor \ldots \rfloor)})}^{\downarrow} \ldots \quad \text{(sinkable)} \qquad \text{or}$$

   $$\ldots \boxed{c_4(\underline{f_3(\boxed{c_2(\underline{\ldots})}^{\uparrow})})}^{\downarrow} \ldots \quad \text{(cancellable)}$$

Output sequent:

$$H \vdash G[\boxed{c_5(\underline{f_1(\ldots, \boxed{c_3(\underline{f_2(\lfloor \ldots \rfloor)})}^{\downarrow}, \boxed{c_4(\underline{f_3(\boxed{c_2(\underline{\ldots})}^{\uparrow})})}^{\downarrow})})}^{\uparrow}]$$

Figure 3: Generalized preconditions for rippling

Blockage:

$$mult(\boxed{\underline{a}+1}^{\uparrow}, \lfloor b \rfloor, 0) = \ldots$$

Critic preconditions:

- preconditions 1, 2 and 3 of rippling succeed, *i.e.*
  1. there exists a subterm which contains wave-front(s), *e.g.*

     $mult(\boxed{\underline{a}+1}^{\uparrow}, \lfloor b \rfloor, 0)$
  2. there exists a wave-rule which matches, *e.g.*

     $mult(\boxed{\underline{X}+1}^{\uparrow}, Y, Z) \Rightarrow mult(X, Y, \boxed{Y+\underline{Z}}^{\downarrow})$
  3. the wave-rule condition follows from the context, *e.g.*

     no condition
- precondition 4 fails, *i.e.*
  4. resulting inward directed wave-fronts are potentially removable, *e.g.*

     $mult(a, \lfloor b \rfloor, \underbrace{\boxed{b+\underline{0}}^{\downarrow}}_{\text{no sink}})$

Patch specification:
Coerce precondition 4 by introducing a new universally quantified variable into the conjecture, *e.g.*

$$mult(\boxed{\underline{a}+1}^{\uparrow}, \lfloor b \rfloor, F_1(\lfloor c \rfloor)) = \ldots$$

where F_1 denotes a second-order meta-variable.

Figure 4: Patch: introduction of sinks (primary)

Blockage:

$$\ldots = \boxed{\underline{a * \lfloor b \rfloor} + b}^{\uparrow}$$

Critic preconditions:

- precondition 1 of rippling succeeds, *i.e.*
    1. there exists a subterm which contains wave-front(s), *e.g.*

       $$\boxed{\underline{a * \lfloor b \rfloor} + b}^{\uparrow}$$
- precondition 2 of rippling partially succeeds, *i.e.*
    2. there exists a partial match with a sideways wave-rule, *e.g.*

       $$\boxed{\underline{a * \lfloor b \rfloor} + b}^{\uparrow}$$

       $$\boxed{\underline{X} + Y}^{\uparrow} + Z \Rightarrow X + \boxed{Y + \underline{Z}}^{\downarrow}$$

Patch specification:
Coerce precondition 2 by introducing additional skeleton term structure into the conjecture such that preconditions 3 and 4 will also potentially succeed, *i.e.*

$$\ldots = F_2(\boxed{\underline{a} + 1}^{\uparrow} * \lfloor b \rfloor \,, \lfloor c \rfloor)$$

where F_2 denotes a second-order meta-variable.

Figure 5: Patch: introduction of sinks (secondary)

patching the proof by introducing a sink occurrence within the third argument position of *mult*. On the right-hand side no wave-rules match. The blocked wave-front, however, does match with the wave-front associated with (9). This partial match with a sideways wave-rule suggests patching the proof by introducing additional skeleton term structure which contains a sink occurrence. We previously referred to these as *primary* and *secondary* ripples [20]. The associated critics are presented in figures 4 and 5 respectively. The resulting patches give rise to the following schematic induction hypothesis:

$$\forall B, C : nat.\ mult(a, B, F_1(C)) = F_2(a * B, C)$$

and a schematic conclusion of the form:

$$mult(\boxed{\underline{a} + 1}^{\uparrow}, \lfloor b \rfloor, F_1(\lfloor c \rfloor)) = F_2(\boxed{\underline{a} + 1}^{\uparrow} * \lfloor b \rfloor, \lfloor c \rfloor)$$

where F_1 and F_2 denote second-order meta-variables. Space prevents us from providing the complete proof, however, the constraints of rippling instantiate F_1 to be $\lambda X.X$ and F_2 to be $\lambda X.\lambda Y.X + Y$. For a detailed description of the constraint mechanism see [20]. The result of this patched proof attempt is to generalize (15) where the generalized conjecture corresponds to (3).

4.2 A loop invariant critic

The basic ideas underlying our generalization critic can be used to suggest loop invariants. We show how each of the failure patterns presented above maps onto a heuristic for discovering loop invariants.

Consider again `exp1` given in figure 1. Previously we verified that (1) is invariant with respect to the associated while-loop. We now consider how this property can be discovered automatically.

4.2.1 Replacing constants by variables

As our starting point we use the following equation as a first approximation[6] to the loop invariant:

$$r * exp(x, \mathcal{Y}) = exp(\mathcal{X}, \mathcal{Y})$$

[6]This kind of invalid invariant might be supplied by the user of a development environment. The ability for a theorem prover to correct invalid invariants automatically would be of great benefit in such a context.

The corresponding verification condition gives rise to an annotated conclusion of the form:

$$\underbrace{\boxed{\underline{r} * x}^{\uparrow}}_{\text{blocked}} * exp(x, \mathcal{Y}) = exp(\mathcal{X}, \mathcal{Y})$$

No rippling is possible. The pattern of this failure is almost identical to the generalization critic given in figure 4. The patch differs in that we introduce an opposing wave-front rather a sink occurrence. The full critic is presented in figure 6 where the patch specified gives rise to a schematic invariant hypothesis of the form:

$$r * exp(x, F_1(r, x, y)) = exp(\mathcal{X}, \mathcal{Y}) \qquad (17)$$

The rippling of the associated schematic conclusion proceeds as follows:

$$\boxed{\underline{r} * x}^{\uparrow} * exp(x, F_1(\boxed{\underline{r} * x}^{\uparrow}, x, \boxed{\underline{y} - 1}^{\uparrow})) = exp(\mathcal{X}, \mathcal{Y}) \qquad (18)$$

$$r * \boxed{x * \underline{exp(x, F_1(\boxed{\underline{r} * x}^{\uparrow}, x, \boxed{\underline{y} - 1}^{\uparrow}))}}^{\downarrow} = exp(\mathcal{X}, \mathcal{Y})$$

$$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})$$

As before the proof is achieved by wave-rules (13) and (14). However, as a side-effect of this ripple, F_1 is instantiated to be $\lambda X.\lambda Y.\lambda Z.Z$. Note that the wave-front annotations are used to constrain the instantiation of the second-order meta-variables. Propagating this instantiation through (17) gives rise to a revised invariant hypothesis which is identical to (1), the required invariant. Note that this form of invariant patching corresponds to the *replacement of constants by variables* heuristic which appears in the literature [16, 22].

4.2.2 Tail-invariants

We now use the given post-condition for `exp1` (see figure 1) as our first approximation to the loop invariant, *i.e.*

$$r = exp(\mathcal{X}, \mathcal{Y})$$

The corresponding verification condition gives rise to an annotated conclusion of the form:

$$\underbrace{\boxed{\underline{r} * x}^{\uparrow}}_{\text{blocked}} = exp(\mathcal{X}, \mathcal{Y})$$

Blockage:

$$\boxed{\underline{r} * x}^{\uparrow} * exp(x, \mathcal{Y}) = \ldots$$

Critic preconditions:

- preconditions 1, 2 and 3 of rippling succeed, *i.e.*
  1. there exists a subterm which contains wave-front(s), *e.g.*

     $\boxed{\underline{r} * x}^{\uparrow} * exp(x, \mathcal{Y})$
  2. there exists a wave-rule which matches, *e.g.*

     $\boxed{\underline{X} * Y}^{\uparrow} * Z \Rightarrow X * \boxed{Y * \underline{Z}}^{\downarrow}$
  3. the wave-rule condition follows from the context, *e.g.*

     no condition
- precondition 4 fails, *i.e.*
  4. resulting inward wave-fronts are potentially removable, *e.g.*

     $\underbrace{\boxed{x * \underline{exp(x, \mathcal{Y})}}^{\downarrow}}_{\text{no opposing wave-front}}$

Patch specification:
Coerce precondition 4 by introducing new structure into the conjecture which contains an opposing wave-front, *e.g.*

$$\boxed{\underline{r} * x}^{\uparrow} * exp(x, F_1(\boxed{\underline{r} * x}^{\uparrow}, x, \boxed{\underline{y} - 1}^{\uparrow})) = exp(\mathcal{X}, \mathcal{Y})$$

where F_1 denotes a second-order meta-variable.

Figure 6: Patch: replacement of constants by variables

Blockage:

$$\boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, \mathcal{Y})$$

Critic preconditions:

- precondition 1 of rippling succeeds, *i.e.*
  1. there exists a subterm which contains wave-front(s), *e.g.*

     $\boxed{\underline{r} * x}^{\uparrow}$
- precondition 2 of rippling partially succeeds, *i.e.*
  2. there exists a partial match with a sideways wave-rule, *e.g.*

     $\boxed{\underline{r} * x}^{\uparrow}$

     $\boxed{\underline{X} * Y}^{\uparrow} * Z \Rightarrow X * \boxed{Y * \underline{Z}}^{\downarrow}$

Patch specification:
Coerce precondition 2 by introducing additional skeleton term structure into the conjecture such that preconditions 3 and 4 will also potentially succeed, *i.e.*

$$F_1(\boxed{\underline{r} * x}^{\uparrow}, x, \boxed{\underline{y} - 1}^{\uparrow}) = exp(\mathcal{X}, \mathcal{Y})$$

where F_1 denotes a second-order meta-variable.

Figure 7: Patch: tail-invariant

Again no rippling is possible. This pattern of failure corresponds to the generalization critic given in figure 5. As previously the patch differs in that we introduce additional skeleton term structure in which an opposing wave-front rather a sink occurs. The complete critic is presented in figure 7 where the specified patch gives rise to a schematic invariant of the form:

$$F_1(r, x, y) = exp(\mathcal{X}, \mathcal{Y}) \tag{19}$$

The rippling of the associated schematic conclusion takes the form:

$$\begin{aligned} F_1(\boxed{\underline{r} * x}^{\uparrow}, x, \boxed{\underline{y} - 1}^{\uparrow}) &= exp(\mathcal{X}, \mathcal{Y}) \quad (20) \\ r * \boxed{x * \underline{F_2(\boxed{\underline{r} * x}^{\uparrow}, x, \boxed{\underline{y} - 1}^{\uparrow})}}^{\downarrow} &= exp(\mathcal{X}, \mathcal{Y}) \\ r * exp(x, y) &= exp(\mathcal{X}, \mathcal{Y}) \end{aligned}$$

As before this proof exploits wave-rules (13) and (14). This time, however, rippling instantiates F_1 to be $\lambda X.\lambda Y.\lambda Z.X * F_2(X, Y, Z)$ while F_2 is instantiated to be $\lambda X.\lambda Y.\lambda Z.exp(Y, Z)$. Propagating these instantiations through (19) gives rise to a refined invariant hypothesis which again is identical to (1), the required invariant. This property is a special case of what is known as a *tail-invariant*. Tail-invariants exploit the strong connection which exists between tail-recursive functions and while-loops. Although exp is not tail-recursive, as Kaldewaij [22] observes, any function definition which fits the following pattern strongly suggests the need for a tail-invariant:

$$\begin{aligned} b(x) \to g(x) &= a \\ \neg b(x) \to g(x) &= h(x) \oplus g(d(x)) \end{aligned} \tag{21}$$

where $\oplus$ is associative. Note that (21) gives rise to a hybrid wave-rule, *i.e.*

$$\neg b(x) \to \boxed{h(x) \oplus \underline{g(\boxed{d(\underline{x})}^{\uparrow})}}^{\downarrow} \Rightarrow g(x) \tag{22}$$

which is exactly the form of wave-rule required in order to achieve a successful sideways ripple in the context of a loop invariant proof.

5 Implementation and results

The work presented here has been implemented within the CLAM proof planner [5]. The implementation makes use of the higher-order features of λ-Prolog [24]. For our application we require only second-order unification. We build upon the technique described in [20] which uses rippling to constrain the unification process. Here we have generalized the technique so that it can deal with hybrid wave-rules. Previously it could deal with only single wave-fronts.

Our initial testing of the proof critics for loop invariant discovery have provided some promising results which are documented in table 1. We draw the reader's attention to the branching rate statistics. These statistics clearly show the significance of rippling in constraining the search space when rewriting in the presence of second-order meta-variables.

6 Related work

Early research into Automatic Programming investigated heuristic based methods for discovering loop invariants. Wegbreit [25] developed an approach in which a candidate invariant was incrementally refined using both domain-independent and domain-specific heuristics. The choice of refinement was guided by the satisfiability and validity of the current candidate invariant. The theorem proving and heuristic components were only loosely coupled. This was reflected in other heuristic approaches at the time [23]. Wegbreit hinted, however, that a closer relationship between the heuristic guidance and the theorem prover would be desirable. The proof planning framework in which our heuristics are expressed enables this close relationship to be exploited.

Our approach, like all heuristic based techniques, is not complete. A complete approach has been designed [6] based upon a novel unskolemization technique for deriving logical consequences of first-order formulae. Completeness, however, comes with a price. In practice this means that any inductive lemmas required for a particular verification task must be provided by hand. A strength of our approach is that we can attempt to automate the discovery of such inductive lemmas. I return to this point in §7.

As noted above the patch specified in figure 7 corresponds to the tail-invariant heuristic found in the literature. The heuristic is based upon observing recursive definitions which match the form of equation (21). Such definitions provide exactly the hybrid form of wave-rule which is required to achieve a successful sideways ripple. Wave-rules are not restricted,

ex	*loop invariant*		*critic*		*lemma*	*branching rate*			
	initial	*final*	*i*	*ii*		*MW*	*TW*	*MR*	*TR*
`exp1`	$r * exp(x, \mathcal{Y}) = exp(\mathcal{X}, \mathcal{Y})$	$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})$	•		27	1	133	4	22
`exp1`	$r = exp(\mathcal{X}, \mathcal{Y})$	$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})$		•	27	2	133	28	22
`sum`	$r + sum(\mathcal{X}) = sum(\mathcal{X})$	$r + sum(x) = sum(\mathcal{X})$	•		26	1	82	6	21
`sum`	$r = sum(\mathcal{X})$	$r + sum(x) = sum(\mathcal{X})$		•	26	3	82	25	21
`fac`	$r * fac(\mathcal{X}) = fac(\mathcal{X})$	$r * fac(x) = fac(\mathcal{X})$	•		27	1	124	6	28
`fac`	$r = fac(\mathcal{X})$	$r * fac(x) = fac(\mathcal{X})$		•	27	2	124	32	28
`div`	$r + div(\mathcal{X}, y) = div(\mathcal{X}, \mathcal{Y})$	$r + div(x, y) = div(\mathcal{X}, \mathcal{Y})$	•		28	2	106	5	19
`div`	$r = div(\mathcal{X}, \mathcal{Y})$	$r + div(x, y) = div(\mathcal{X}, \mathcal{Y})$		•	28	4	106	28	19
`sumd`	$r + sumd(\mathcal{X}) = sumd(\mathcal{X})$	$r + sumd(x) = sumd(\mathcal{X})$	•		26	1	135	4	27
`sumd`	$r = sumd(\mathcal{X})$	$r + sumd(x) = sumd(\mathcal{X})$		•	26	3	135	30	27
`exp2`	$r * exp(\mathcal{X}, \mathcal{Y}) = exp(\mathcal{X}, \mathcal{Y})$	$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})$	•		27,29	1	264	14	27
`exp2`	$r = exp(\mathcal{X}, \mathcal{Y})$	$r * exp(x, y) = exp(\mathcal{X}, \mathcal{Y})$		•	27,29	4	264	37	27

Algorithms `exp1` and `exp2` are given in figure 1. The `sum` algorithm computes the summation of a natural number while `sumd` computes the sum of the digits of a natural number using the quotient and remainder operators for integer division. Repeated subtraction is used as the basis for `div` which computes integer division while `fac` uses repeated multiplication to compute factorials. The *loop invariant* columns show the candidate invariant as supplied to CLAM and the invariant resulting from the proof patching process. The candidates either correspond to the overall program post-condition or are potentially the result of user interaction. The *critic* columns refer to the *i) replacement of constants by variables* and *ii) tail-invariant* patches respectively. Non-definitional equations used in discovering invariants are indicated in the *lemma* column. The *branching rate* columns compare the applicability of wave-rules and rewrite rules with respect to the schematic conclusion generated by an invariant critic (see (18) and (20) in the case of `exp1`). Note that the branching rates relate to the first step in each proof attempt. The *MW* and *TW* columns give the number of wave-rule matches and the total number of wave-rules respectively. The *MR* and *TR* columns give the same information for rewrite rules. Note that in some cases the *MR* value is greater than the *TR* value. This occurs because some matches give rise to multiple unifiers. This is not the case with rippling since it provides greater constraints on the matching process.

Table 1: Results for Loop Invariant Critics

however, to recursive definitions. Properties such as associativity and distributivity are also a rich source of wave-rules. Consequently our rippling critics can be seen as a generalization of the tail-invariant heuristic described by Kaldewaij. To illustrate this point more precisely consider `exp2` given in figure 1. For this algorithm the definition of exp, which provides wave-rule (14), does not suggest an appropriate tail-invariant. The suggestion comes instead from (29), a property of exp which gives rise to the following hybrid wave-rule:

$$odd(Y) \rightarrow \boxed{X * \underline{exp(\boxed{\underline{X} * X}^{\uparrow}, \boxed{\underline{Y}\ div\ 2}^{\uparrow})}}^{\downarrow} \Rightarrow exp(X, Y)$$

Using this rule our technique discovers the required invariant automatically[7].

7 Future work

Building on the results presented here we are planning to undertake larger scale verification case studies. To achieve this goal we have begun to generalize our strategies to deal with nested loops and arrays.

As discussed in §4, the work described here is based upon the generalization critic developed for inductive proof. Other critics developed for inductive proof may also prove useful in guiding the discovery of loop invariants. For instance there exists a strong connection between the induction revision critic [21] and the *replacement of constants by terms* heuristic [16, 22] which we plan to investigate.

As hinted in §6 and demonstrated in [21] our heuristic approach provides the potential for automating the discovery of inductive lemmas such as (29). We aim to exploit this potential in future work.

Our long term aim is to develop new proof methods and critics for guiding the synthesis of imperative programs building upon the wealth of heuristic knowledge which exists within the literature.

8 Conclusion

A loop invariant represents a key eureka step in the verification of an imperative program. By generalizing the preconditions of rippling we have shown how it may be used to guide the verification of loop invariants.

Building upon critics developed for induction we have demonstrated how rippling can also be used to guide the discovery of loop invariants automatically. Second-order meta-variables play an important role in a critic's attempt to patch a proof. This new application has led to a generalization of the mechanism we use for constraining the instantiation of second-order meta-variables.

More generally, a major benefit of developing strategies within the proof planning framework is that our work will readily transfer to any formal development environment which uses a tactic or proof script based theorem prover.

Acknowledgements

We would like to thank Lincoln Wallen for drawing our attention to the connection between tail-invariants and our generalization critic. Thanks also go to Alan Bundy for his encouragement with regard to this work.

Appendix: definitions

$$
\begin{aligned}
0 * Y &= 0 \\
(X+1) * Y &= (X * Y) + Y && (23) \\
X = 0 \rightarrow sum(X) &= 0 \\
X > 0 \rightarrow sum(X) &= X + sum(X-1) \\
X = 0 \rightarrow fac(X) &= 0 \\
X > 0 \rightarrow fac(X) &= X * fac(X-1) \\
Y = 0 \rightarrow exp(X,Y) &= 1 \\
Y > 0 \rightarrow exp(X,Y) &= X * exp(X, Y-1) && (24) \\
mult(0, Y, Z) &= Z \\
mult(X+1, Y, Z) &= mult(X, Y, Y+Z) && (25) \\
X < Y \rightarrow div(X,Y) &= 0 \\
X >= Y \rightarrow div(X,Y) &= 1 + div(X-Y, Y) \\
(X+Y)+Z &= X+(Y+Z) && (26) \\
(X*Y)*Z &= X*(Y*Z) && (27) \\
s(X)+Y &= X+s(Y) && (28) \\
X = 0 \rightarrow sumd(X) &= 0 \\
X > 0 \rightarrow sumd(X) &= (X\ mod\ 10) + sumd(X\ div\ 10) \\
odd(B) \rightarrow exp(A,B) &= A * exp(A*A, B\ div\ 2) && (29)
\end{aligned}
$$

[7] This example is based upon an exercise presented in [14] where the loop invariant is provided as a hint to the reader.

References

[1] R.C. Backhouse. *Program Construction and Verification.* Prentice Hall, 1986.

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

[3] Alan 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.

[4] Alan Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993.

[5] Alan 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.

[6] Chadha and Plaisted. On the mechanical derivation of loop invariants. *JSL*, 15:705–744, 1993.

[7] R. L. Constable, S. F. Allen, H. M. Bromley, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

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

[9] E. W. Dijkstra. Invariance and non-determinacy. In C.A.R. Hoare and J.C. Shepherdson, editors, *Mathematical Logic and Programming Languages*, chapter 9, pages 157–165. Prentice-Hall, 1985.

[10] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics 19*, pages 19–32. American Mathematical Society, 1967.

[11] S. Garland, J. Stephen, and John V. Guttag. *A Guide to LP, The Larch Prover*, November 1991.

[12] 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.

[13] M. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis.* Kluwer, 1988.

[14] M. J. Gordon. *Programming Language Theory and its Implementation.* International Series in Computer Science. Prentice-Hall, 1988.

[15] 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.

[16] David Gries. *The Science of Programming.* Springer-Verlag, New York, 1981.

[17] D. Guaspari, C. Marceau, and W. Polak. Formal verification of Ada programs. *IEEE Trans. on Software Engineering*, 16(9):1058–1075, September 1990.

[18] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[19] 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.

[20] A. Ireland and A. Bundy. Extensions to a Generalization Critic for Inductive Proof. In M. A. McRobbie and J. K. Slaney, editors, *13th Conference on Automated Deduction*, pages 47–61. Springer-Verlag, 1996. Springer Lecture Notes in Artificial Intelligence No. 1104.

[21] A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996.

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

[23] S.M. Katz and Z. Manna. A heuristic approach to program verification. In *Proceedings of IJCAI-73.*

[24] 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.

[25] Wegbreit. Heuristic methods for mechanically deriving inductive assertions. In *Proceedings of IJCAI-73.*