Invariant Discovery via Failed Proof Attempts*

Jamie Stark[†]**and Andrew Ireland**[‡]

Department of Computing & Electrical Engineering,
Heriot-Watt University, Riccarton,
Edinburgh EH14 4AS Scotland, UK
Email: jamie@cee.hw.ac.uk, air@cee.hw.ac.uk

1 Introduction

Loop invariants are a well understood technique for specifying the behaviour of imperative loop programs. The discovery of suitable invariants, however, is a major bottle neck for automatic verification of imperative programs. Early research into heuristic based methods [17, 21] focused on the heuristic and the theorem proving components separately. The proof planning framework, in which we reconstruct the standard heuristics, couples the heuristic and theorem proving components tightly, allowing us to focus on both of these components together. This enables us to exploit the relationship between failed proof attempts and invariant discovery. The framework described here builds upon previous work [15] and shows how successive approximations are used to find a suitable invariant by a system automatically. This work was motivated by previous work on discovering generalisations for inductive proofs. We exploit the relationship between while loops and tail recursive functions, but rather than turn our loop programs into tail recursive functions and discover a generalisation we choose to move the heuristics across. This enables us to communicate proof attempts, and specifications in the same language. i.e. we choose to talk about invariants rather than generalisations.

2 Background

Hoare [11] first introduced his axiomatic proof rules for imperative programs involving loops in 1969. The aim was to assist the understanding of programs using rigourous logical techniques. The proof rules allow the transformation of a program with suitable assertions, into a set of purely logical implications called verification conditions. These assertions are relationships between the program variables and program constants[1]. Program constants are important since they are used to capture initial inputs to a program and are used in the post condition to relate a desired property in terms of these initial values.

The verification of simple imperative programs involving loops falls short of full automation since Hoare's rule for while loops involves synthesising an assertion known as the invariant. The while rule takes the form:

$$\frac{P \rightarrow I, \quad \{I \wedge B\}S\{I\}, \quad (I \wedge \neg B) \rightarrow Q}{\{P\}\textbf{while} \;\; B \;\textbf{begin}\; S \;\textbf{end}\; \{Q\}}$$

The second premise is used to generate a verification condition using Hoare's other proof rules. It is this verification condition which establishes I to be an invariant of the loop body S and is the focus of our work.

The literature [1, 7, 9, 16] contains many heuristics for constructing invariants in a systematic way, which involves making successive approximations at the invariant. These heuristics involve replacing a constant by a variable or term, strengthening an invariant by adding a conjunct and

*Extended Abstracts of LOPSTR'98, Eighth International Workshop on Logic-based Program Synthesis and Transformation, 15-19 June 1998, Manchester, UK. Technical Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-98-6-1. http://www.cs.man.ac.uk/cstechrep/titles98.html

†This work is supported by an EPSRC student ship award 96307451

‡This work is supported by EPSRC grant GL/L/11724

[1]Program constants are denoted as calligraphic letters throughout (e.g. $\mathcal{X}$).

tail invariants. Previously [15] we showed how proof planing was able to successfully discover tail invariants. Here this work is extended by adding heuristics to replace program constants and strengthen invariants. A general strategy is presented that enables the successive approximation of invariants using these heuristics.

Proof planning [3] consists of tactics, methods and critics. Tactics are compounded basic rules of inference. Methods partial specify a tactic and allow the use of heuristics. A method has preconditions and effects. Proof planning involves the search for a plan at the meta level by using the methods. A proof at the object level is extracted from a plan and run for completeness. Critics [12] are used to handle the failure of a methods precondition. They patch the failed proof attempt and provide a powerful mechanism for dealing with failure.

The CLAM proof planner [5] has been used, amongst other applications, for inductive proof [4]. The success of the proof plan for induction is due to the ripple heuristic. 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*. Wave-fronts 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 boxes to represent wave-fronts, *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 skeleton while making progress to eliminating wave-fronts. Wave-rules are generated automatically by CLAM for all definitions and properties loaded into the system. For any equation there are usually a number of corresponding wave-rules. 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. For a full account of rippling see [2, 4].

Middle-out reasoning is a proof planning strategy where meta terms are used to delay choice within the proof. The motivation is that the middle of a proof is typically more constrained than the start. The aim of middle-out reasoning is to use the proof to add structure where we do not know what it is. The implementation makes use if the higher-order features of λ-prolog [20]. Rippling is used to constrain the matching of wave-rules, by taking the annotations into account. Middle-out techniques have been used for generalisation [10, 13], logic program synthesis [18], lemma discovery and induction revision [14].

3 A Proof Plan for Invariant Verification

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

Figure 1: Algorithm for computing exponentiation

Consider the program in figure (1). To verify this program we need a loop invariant. This is a *eureka* step and an invariant is not immediately obvious. An invariant for this program is

$r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X}$. The aim is to prove the following verification condition generated by the loop body:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \quad \rightarrow \quad (r * x = exp(\mathcal{X}, \mathcal{Y} - (y-1)) \wedge x = \mathcal{X}) \tag{1}$$

The differences in the conclusion arises from the assignment statements in the loop body. To show that this is invariant we can rewrite the conclusion and move these differences up through the term structure until we are able to appeal to our hypothesis. Rippling provides us with the power to do this if we are able to annotate the conclusion with respect to the hypothesis. This suggests the following proof plan: Either a goal can be simplified or annotate the conclusion with respect to the hypothesis then apply the wave method until we can fertilize with the hypothesis. The power of the proof plan is not that it can do the verification, but that it can be used to encode the heuristics we need to use to discover a suitable invariant from an initial guess. In more detail the proof plan consists of:

- **simplify:**

  preconditions: If a conjunct C in the conclusion is trivial i.e. it is a tautology e.g. $r = r$, or it follows from some simple data type properties e.g. $0 \neq 0 + 1$.

- **invariant:**

  preconditions: If annotation is possible. i.e. there is difference in the conclusion which prevents a match with a hypothesis. And we can apply wave repeatedly until we can apply fertilize. Where wave and fertilize are:

  - **wave:**

    preconditions: If there is a sub-term W in the conclusion containing a wave occurrence and there is a wave-rule $C \rightarrow L \Rightarrow R$ such that L matches W and any condition C of the wave-rule are satisfied.

  - **fertilize:**

    preconditions: The conclusion contains a copy of a hypothesis H.

A Verification Proof

Consider the verification condition (1) above:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow (r * x = exp(\mathcal{X}, \mathcal{Y} - (y-1)) \wedge x = \mathcal{X})$$

and the following wave-rules [2]:

$$exp(A, \boxed{\underline{B} + 1}^{\uparrow}) \quad \Rightarrow \quad \boxed{\underline{exp(A, B)} * A}^{\uparrow} \tag{2}$$

$$\boxed{\underline{A} * C}^{\uparrow} = \boxed{\underline{B} * D}^{\uparrow} \quad \Rightarrow \quad \boxed{\underline{A = B} \wedge C = D}^{\uparrow} \tag{3}$$

$$A - \boxed{(\underline{B} - C)}^{\uparrow} \quad \Rightarrow \quad \boxed{(\underline{A - B}) + C}^{\uparrow} \tag{4}$$

The simplify method applies here since $x = \mathcal{X}$ appears in both the conclusion and the hypothesis simplifying the verification condition to:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow r * x = exp(\mathcal{X}, \mathcal{Y} - (y-1))$$

The invariant method is applicable and applied to give the sub-goal:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, \mathcal{Y} - \boxed{(\underline{y} - 1)}^{\uparrow})$$

[2] Note that $\Rightarrow$ describes the rewrite relation and $\rightarrow$ describes first order implication throughout.

We can use the wave method with the sub-term $\mathcal{Y} - \boxed{(\underline{y} - 1)}^{\uparrow}$ and the wave-rule (4) giving the new subgoal:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, \boxed{(\underline{\mathcal{Y} - y}) + 1}^{\uparrow})$$

Now we use the sub-term $exp(\mathcal{X}, \boxed{(\underline{\mathcal{Y} - y}) + 1}^{\uparrow})$ and the wave-rule (2) giving the new subgoal:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = \boxed{\underline{exp(\mathcal{X}, \mathcal{Y} - y)} * \mathcal{X}}^{\uparrow}$$

We are now in a position to apply wave-rule (3) to the whole conclusion giving:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow \boxed{\underline{r = exp(\mathcal{X}, \mathcal{Y} - y)} \wedge x = \mathcal{X}}^{\uparrow}$$

Since both of the conjuncts in the conclusion appear in the hypothesis we can discharge them with the fertilize method finishing the proof.

4 Analysis of Failure

The merits of the rippling heuristic become apparent when the proof attempt fails. Since we expect to systematically refine invariants from an initial guess, we are aware that we will be trying to prove that a property is invariant when it is not. If this is the case, the ripple will soon get blocked. One way a ripple can get blocked is that there is some missing wave structure. By relating this failure to a standard heuristic, namely replacing constants by terms, for developing loop invariants, we are able to suggest a patch which will give a new candidate invariant.

Replacement of Constants

A standard heuristic for developing loop invariants is replacing constants by terms containing program variables. Constants are replaced with term structure containing a program variable in a relation to increase the state space that the relation describes. In this state space this relation may be more easily established.

Rippling can fail because it fails to match the left hand side of a wave-rule with part of the goal. Wave-rule matching is defined as:

Wave-rule Match: Given a goal T and a wave-rule term L, then T and L match iff:

- The object level term structures of L and T match.
- For all wave-fronts in L there exists a matching wave-front in T.

Constants can block a ripple because they can prevent a match between the left hand side of a wave-rule and a sub-term of the goal. This happens when the wave-rule expects a wave but there is a term containing a constant in the sub-term. However we can define a potential match which suggests we have the potential to match if we could introduce some difference by replacing constants:

Potential Match: Given a goal T and a wave-rule term L, then T and L match iff:

- For all wave-fronts in L there exists either a matching wave-front in T or a term containing a constant.
- The object level term structures of L and T match everywhere expect at the positions where a wave-front in L matches a term containing a constant.

For example attempting to match $\boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, \mathcal{Y})$ and the left hand side of the wave-rule (3):

$$\boxed{\underline{A} * C}^{\uparrow} = \boxed{\underline{B} * D}^{\uparrow}$$

would be a potential match because there is a term containing a constant in the right hand side of the equality in the sub-term, e.g. $exp(\mathcal{X}, \mathcal{Y})$, where the wave-rule expects a wave, e.g. $\boxed{\underline{B} * D}^{\uparrow}$ and all other structure matches.

Having a potential match tells us that we need a wave-front where we have a term containing a constant. The replacement of constants heuristic tells us that this wave-front can come from a constant. To coerce this wave-rule match we need to discover a term that will replace that term containing a constant. We use middle-out reasoning to find such a term. To find the patch we first need to discover where this difference came from. Since the wave-rule we want to coerce may be later in the proof, i.e. we have to move any new difference before this wave-rule is applicable, we have to look back across this proof gap to find from which constant this difference came from. This is done by applying wave-rules backwards until all the difference surrounds only one constant.

Strengthening invariants

Gries [8] explains a simple, but powerful, heuristic for dealing with stuck proof obligations. The idea is to assume the very thing that we are trying to prove. This has the effect of instantly making the proof obligation trivial. However we now have to show that this proof obligation is itself invariant.

At the end of a ripple proof we are sometimes left with some proof residue. This is a sub-term that requires some further proof. This residue can sometimes be proved using the simplify method, but when it cannot, we have to start the invariant proof process again. The invariant method tries to annotate a subgoal. If we are unable to annotate it then we are stuck since we will not be able to ripple.

Using this failure to annotate a subgoal we can refine our candidate invariant using the heuristic above. i.e. the conclusion we are unable to annotate is added to our candidate invariant and we start the proof again. For example suppose we where left with the failed proof attempt:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge y \neq 0) \rightarrow x = \mathcal{X}$$

The patch is to add $x = \mathcal{X}$ to the invariant. Thus the new candidate invariant becomes:

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

5 Refining Invariants

The above analysis of failure has been encoded as critics within the CLAM proof planning system. Critics enable the automatic construction of such patches from failed proof goals. CLAM is allowed to exhaustively search before attempting to fire any critics.

These critics together allow us to successively refine approximation to our invariant. The idea is to use the post condition as our initial guess and successively refine our invariant. If we have proved that our new property is invariant we then have to show that actually implies the post condition i.e. it is not too weak. For example suppose we have shown I to be invariant over successive refinements of our post condition Q we have to show that $(I \wedge \neg B) \rightarrow Q$ where B is the loop guard.

For example consider the program in figure (1), and the wave-rules (2), (3) and (4) introduced above.

First Approximation (Post-condition)

We take the post-condition $r = exp(\mathcal{X}, \mathcal{Y})$ as our first approximation to the invariant. This gives the verification condition:

$$(r = exp(\mathcal{X}, \mathcal{Y}) \wedge y \neq 0) \rightarrow r * x = exp(\mathcal{X}, \mathcal{Y})$$

This goal cannot be simplified so we apply the invariant method giving:

$$(r = exp(\mathcal{X}, \mathcal{Y}) \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, \mathcal{Y})$$

Now we are stuck. We are unable to apply any of the wave rules. We are left resorting to using the critics to try and patch our goal. Firstly we notice that we have a potential match between the goal:

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

and the left hand side of the wave-rule (3):

$$\boxed{\underline{A} * C}^{\uparrow} = \boxed{\underline{B} * D}^{\uparrow}$$

We apply a look back to decide which constant to replace. The right hand side of the wave-rule (2):

$$\boxed{\underline{exp(A, B)} * A}^{\uparrow} \text{ matches with: } \boxed{\underline{exp(\mathcal{X}, \mathcal{Y})} * D}^{\uparrow}$$

the result of our potential match. Looking at the left hand side of this wave-rule we are able to determine that the difference came from the second argument of exp, so we replace the $\mathcal{Y}$ with the meta term $M_1(r, y, \mathcal{Y})$ giving the new candidate invariant:

$$r = exp(\mathcal{X}, M_1(r, y, \mathcal{Y}))$$

Second Approximation (Result of replacement of constants critic)

This new candidate invariant is used to set up a verification condition:

$$(r = exp(\mathcal{X}, M_1(r, y, \mathcal{Y})) \wedge y \neq 0) \rightarrow r * x = exp(\mathcal{X}, M_1(r * x, y - 1, \mathcal{Y}))$$

which is annotated to give:

$$(r = exp(\mathcal{X}, M_1(r, y, \mathcal{Y})) \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, M_1(\boxed{\underline{r} * x}^{\uparrow}, \boxed{\underline{y} - 1}^{\uparrow}, \mathcal{Y}))$$

Wave-rule (4) can be applied. This has the effect of finding an instantiation for M_1 to be $\lambda X.\lambda Y.\lambda Z.\ M_2(X, Y, Z) - Y$. This gives the new verification condition:

$$(r = exp(\mathcal{X}, M_2(r, y, \mathcal{Y}) - y) \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = exp(\mathcal{X}, \boxed{\underline{(M_2(\boxed{\underline{r} * x}^{\uparrow}, \boxed{\underline{y} - 1}^{\uparrow}, \mathcal{Y}) - y)} + 1}^{\uparrow})$$

Wave-rule (2) can now be applied giving the new verification condition:

$$(r = exp(\mathcal{X}, M_2(r, y, \mathcal{Y}) - y) \wedge y \neq 0) \rightarrow \boxed{\underline{r} * x}^{\uparrow} = \boxed{\underline{exp(\mathcal{X}, M_2(\boxed{\underline{r} * x}^{\uparrow}, \boxed{\underline{y} - 1}^{\uparrow}, \mathcal{Y}) - y)} * \mathcal{X}}^{\uparrow}$$

Now we are able to apply the wave-rule (3) as we hoped giving:

$$(r = exp(\mathcal{X}, M_2(r, y, \mathcal{Y}) - y) \wedge y \neq 0) \rightarrow \boxed{\underline{r = exp(\mathcal{X}, M_2(\boxed{\underline{r} * x}^{\uparrow}, \boxed{\underline{y} - 1}^{\uparrow}, \mathcal{Y}) - y)} \wedge x = \mathcal{X}}^{\uparrow}$$

The first conjunct in the conclusion appears in the hypothesis so we can apply the fertilize method, which has the effect of projecting the meta-variable onto it's third argument i.e. M_2 is instantiated to $\lambda X.\lambda Y.\lambda Z.\ Z$. We pass the remaining conjunct back up since we are unable to fertilize it:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge y \neq 0) \rightarrow x = \mathcal{X}$$

Now we are stuck again, this proof obligation is not trivial and we are unable to annotate it. The strengthening critic now fires. So as a last resort we try to show that it is invariant, by adding it to our previous invariant giving the next approximation:

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

Program	Post Condition	Invariant Discovered
`exp`	$r = exp(\mathcal{X}, \mathcal{Y})$	$r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X}$
`sum`	$r = sum(\mathcal{X})$	$r = sum(x)$
`times`	$r = \mathcal{X} * \mathcal{Y}$	$r = (\mathcal{X} - x) * \mathcal{Y} \wedge y = \mathcal{Y}$
`factorial`	$r = fac(\mathcal{X})$	$r = fac(x)$
`min-array`	$m = min(i, 0 \leq i < \mathcal{N} : a[i])$	$m = min(i, 0 \leq i < n : a[i])$
`sum-array`	$s = \sum_{i=0}^{i<\mathcal{N}} a[i]$	$s = \sum_{i=0}^{i<n} a[i]$
`min-sum-seg`	$s = min(i, j, 0 \leq i < j \leq \mathcal{N} : \sum_i^j a[i])$	$s = min(i, j, 0 \leq i < j \leq n : \sum_i^j a[i])$ $\wedge c = min(i, 0 \leq i < n : \sum_i^n a[i])$

Program `exp` is given in figure (1). Programs `sum` and `factorial` computes the sum and factorial of the first $\mathcal{X}$ natural numbers respectively. Program `times` multiplies two natural numbers by successive addition. Programs `min-array`, `sum-array` and `min-sum-seg` find the least element of an array, the sum of an array and the minimum sum segment of an array respectively.

Table 1: Results

Third Approximation (Result of strengthening critic)

Again we set up a verification condition:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow (r * x = exp(\mathcal{X}, \mathcal{Y} - (y-1)) \wedge x = \mathcal{X})$$

The simplify method removes the $x = \mathcal{X}$ from the conclusion since it appears in the hypothesis giving:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow r * x = exp(\mathcal{X}, \mathcal{Y} - (y-1))$$

The invariant method now is used. This lead to the subgoal:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow (r * x = exp(\mathcal{X}, \mathcal{Y} - \boxed{(\underline{y} - 1)}^{\uparrow}) \wedge x = \mathcal{X})$$

Now the proof proceeds exactly as before applying the wave-rules (4), (2) then (3) resulting in:

$$(r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X} \wedge y \neq 0) \rightarrow \boxed{\underline{r = exp(\mathcal{X}, \mathcal{Y} - y)} \wedge x = \mathcal{X}}^{\uparrow}$$

Now the fertilize method removes both conjuncts in the conclusion completing the proof.

So we have shown that $r = exp(\mathcal{X}, \mathcal{Y} - y) \wedge x = \mathcal{X}$ is an invariant to the loop body. To finish the proof we check that our invariant satisfies the first and third premises of the while rule.

6 Results

This work on invariant discovery has been implemented in the CLAM proof planner. It describes a systematic framework for refining initial approximations to a loop invariant automatically in a principled way. Initial results using examples from the literature are promising, see table (1). The heuristics described have been used for more complex problems involving arrays, notably Gries' minimum sum segment problem [8]. Our approach is syntactically driven and requires defining equations and properties needed to complete the proof. There is no restriction that these defining equations are recursive or the specification is immediately executable. In the example above for the program (1) we use a constructive definition of *exp* to prove a destructive version of the program.

7 Related Work

Early research into Automatic Programming investigated heuristic based methods for discovering loop invariants. Wegbreit [21] 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 [17]. 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.

Our technique is very tightly coupled to the syntactic structure of the initial invariant. A more semantic approach to verification is described by Mili [19], where program semantics are found independently of their specifications and heuristics are given for discovering invariant properties. We are currently investigating the relationship between these alternatives and whether a combined approach has merit.

8 Conclusion

Discovering a loop invariant is a bottle neck for automatic imperative program verification. Building upon the proof planning framework we have shown that by productive use of failed proof attempts we are able to automate the synthesis suitable invariants.

References

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

[2] David Basin and Toby Walsh. Annotated rewriting in inductive theorem proving. *Journal of Automated Reasoning*, 16(1–2):147–180, 1996.

[3] A. Bundy. The use of explicit plans to guide inductive proofs. Research Paper 349, Dept. of Artificial Intelligence, University of Edinburgh, 1988. Short version published in the proceedings of CADE-9.

[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. Research Paper 507, Dept. of Artificial Intelligence, University of Edinburgh, 1990. Appeared in the proceedings of CADE-10.

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

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

[8] D. Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2:207–214, 1982.

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

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

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

[12] 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.

[13] 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. Also available from Edinburgh as DAI Research Paper 786.

[14] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, 1996. Also available as DAI Research Paper No 716, Dept. of Artificial Intelligence, Edinburgh.

[15] A. Ireland and J. Stark. On the Automatic Discovery of Loop Invariants. In *Fourth NASA Langley Formal Methods Workshop*, number 3356 in NASA Conference Publications, 1997. Also available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/97/1.

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

[17] S.M. Katz and Z. Manna. A heuristic approach to program verification. In *Proceedings of IJCAI-73*. International Joint Conference on Artificial Intelligence, 1973.

[18] I. Kraan, D. Basin, and A. Bundy. Logic program synthesis via proof planning. In K.K. Lau and T. Clement, editors, *Logic Program Synthesis and Transformation*, pages 1–14. Springer-Verlag, 1993. Also available as Max-Planck-Institut für Informatik Report MPI-I-92-244 and Edinburgh DAI Research Report 603.

[19] A. Mili, J. Desharhais, and F. Mili. *Computer Program Construction*. Oxford University Press, 1994.

[20] 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.

[21] Wegbreit. Heuristic methods for mechanically deriving inductive assertions. In *Proceedings of IJCAI-73*. International Joint Conference on Artificial Intelligence, 1973.