Proof Plans for the Correction of False Conjectures

– Submitted to LPAR'94 – ⋆

Raul Monroy, Alan Bundy, & Andrew Ireland

Department of Artificial Intelligence
The University of Edinburgh
80 South Bridge, EH1 1HN
Scotland, U.K
raulm, bundy, & air@aisb.ed.ac.uk

Abstract. Theorem proving is the systematic derivation of a mathematical proof from a set of axioms by the use of rules of inference. We are interested in a related but far less explored problem: the analysis and correction of false conjectures, especially where that correction involves finding a collection of antecedents that, together with a set of axioms, transform non-theorems into theorems. Most failed search trees are huge, and special care is to be taken in order to tackle the combinatorial explosion phenomenon. Fortunately, the planning search space generated by proof plans, see [1], are moderately small. We have explored the possibility of using this technique in the implementation of an abduction mechanism to correct non-theorems.

1 Introduction

The problem of building an artificial mathematician to find a mathematical proof has been a topic of much interest in Artificial Intelligence. We are interested in a related but far less explored problem: the analysis and correction of false conjectures, especially where that correction involves finding a collection of antecedents that, together with a set of axioms, transform non-theorems into theorems. More formally:

> Given a set of axioms $\mathcal{A}$ and a false conjecture G, i.e. $\mathcal{A} \rightarrow G$ does not hold, our aim is to identify C such that[2]:
> 1. $\mathcal{A} \wedge C \rightarrow G$ is a theorem, i.e. the addition of C turns the non-theorem into a theorem.
> 2. $\mathcal{A} \wedge C$ is satisfiable, i.e. C is **consistent** with the set of axioms.

⋆ We are grateful to Jane Hesketh for her comments on an earlier draft of this paper. The research reported here was supported by SERC grant GR/H/23610 and ITESM & CONACyT studentship 64745 to the first author.

[2] The notions of consistency, triviality, minimality, and basicness have been defined in [5].

3. $C \rightarrow G$ does not hold, i.e. C is **nontrivial**.
4. C is **minimal** in that it does not contain any redundant literals.
5. there does not exist nontrivial explanations of C: **basicness**.

As a way of motivation, consider the following non-theorem

$$\forall N : \text{Nat. double}(\text{half}(N)) = N \tag{1}$$

together with the Peano axioms for the natural numbers. Clearly, a condition like $N < 0$ does not meet our requirements because it is inconsistent with the set of axioms. Also, the formula

$$\forall N : \text{Nat. } (\text{double}(\text{half}(N)) = N) \rightarrow (\text{double}(\text{half}(N)) = N)$$

it is not a valid solution since the condition is trivial. The abduction mechanism we present in this paper is capable of finding the condition even(N), which is clearly *fundamental*, i.e. consistent, nontrivial, minimal, and basic. Note that even$(N) \wedge N = 0$ would not be minimal. Note also that $\mathcal{A} \rightarrow$ even(N) does not hold, thus ensuring that this condition is basic.

2 Proof Plans

Reasoning and searching are necessary for the solution to the problem of correcting a false conjecture. *Abduction* seems to be a candidate mechanism for the former. Abduction, as proposed by C.S. Peirce[9], is a fundamental form of logical inference that allows us to find hypotheses that account for some observed facts. Its simplest form is:

From A $\rightarrow$ B, and B
Infer A as a possible justification of B

However, most failed proof search spaces are huge and logic based abduction mechanisms are severely affected by the combinatorial explosion phenomenon, see [10]. Fortunately, the planning search spaces generated by *proof plans* are moderately small, see [1]. Furthermore, the meta-level reasoning used to guide the proof plan formation provides us with a basis for analysing the failure and the partial success derived from a proof attempt. The proof plans technique guides the search for a proof in the context of tactical style reasoning, see [6]. It has been implemented in a proof plan formation system called CLAM [3], and successfully applied to the domain of inductive proofs [2].

2.1 Rippling

The key idea behind inductive proofs is the use of the induction hypothesis in its proof. *Rippling*, see [4], is a heuristic which does this work. It works by applying a special syntactic class of rewrite rules called *wave-rules*. The most simple form of such wave-rules is:

$$F(\boxed{S(\underline{U})}^{\uparrow}) \Rightarrow \boxed{T(\underline{F(U)})}^{\uparrow} \tag{2}$$

where F, S, and T are terms with one distinguished argument. T may be empty but S and F must not be. F and $\boxed{S(\underline{U})}^{\uparrow}$ are called *wave-function* and *wave term*, respectively. Wave-terms are composed of a *wave-front* and one or more *wave-holes*. Wave-holes are the underlined sub-terms of wave terms. Sub-expressions of the induction conclusion that also appear in the hypothesis are either underlined or not enclosed by boxes. For our current wave-rule example, F and U would match such sub-expressions. Note how the application of (2) has the effect of put them closer together. Also, note how the arrow indicates the direction in which wave-fronts are moved within the term structure. i.e. they are rippled-out all the way up to the very top of the formula.

By marking these wave-terms and tracking their movements, we can ensure that our rewriting makes progress towards the desired effect: the removal of the obstructive wave fronts so that *fertilization* can be applied. Fertilization, according to Boyer and Moore, is the use of the induction hypothesis in its proof.

2.2 Proof Critics

Experience has shown that a failed proof plan attempt may hold the key for discovering a complete proof. In [7], the author propose the use of *planning critics* as a mechanism to provide the means of exploiting failure and partial success in the search for a proof. Proof critics can be used either to modify the plan structure, the given conjecture, or the theory. Indeed, the abduction mechanism to correct faulty conjectures that we introduce in this paper constitutes another application of this mechanism.

3 Correcting Faulty Conjectures

Our abduction mechanism to correct faulty theorems is built upon CLAM. We have exploited the information derived from a failed proof attempt by looking for unprovable goals. We have also made use of the meta-level control information of rippling to focus the process of locating, and correcting a fault. False conjectures that exhibited faults in boundary values were successfully corrected using the information provided by the base case of inductive proofs. We worked by refinement when a suggested condition from a previous patching attempt turned out to be necessary but not sufficient. We also corrected false conjectures in which the fault exhibited arguments in wrong positions within the conjecture structure; this sort of fault can be found in attempts at proving commutativity in operators that are not Abelian. Our abduction mechanism consists of a collection of proof critics that define heuristics to detect, locate and correct these sort of faults.

3.1 Exploiting Contradictory Blocked Goals

Consider the non-theorem:

$$\forall A, B : \text{list}(DataType).\ \text{length}(A <> B) > \text{length}(A) \qquad (3)$$

The recursive definitions of $<>$, $>$, and length give rise to the rewrite rules:[3]:

$$\boxed{X :: \underline{U}}^{\uparrow} <> V \Rightarrow \boxed{X :: \underline{U <> V}}^{\uparrow} \tag{4}$$

$$\mathrm{nil} <> U \Rightarrow U$$

$$\boxed{\mathrm{s}(\underline{X})}^{\uparrow} > \boxed{\mathrm{s}(\underline{Y})}^{\uparrow} \Rightarrow X > Y \tag{5}$$

$$X > 0 \Rightarrow X \neq 0$$

$$0 > X \Rightarrow false$$

$$\mathrm{length}(\boxed{X :: \underline{U}}^{\uparrow}) \Rightarrow \boxed{\mathrm{s}(\underline{\mathrm{length}(U)})}^{\uparrow} \tag{6}$$

$$\mathrm{length}(\mathrm{nil}) \Rightarrow 0$$

From which we establish the following induction hypothesis:

$$\mathrm{length}(a <> \lfloor b \rfloor) > \mathrm{length}(a) \tag{7}$$

Following the Prolog convention, we denote variables with symbols that start with an upper-case letter. Note how universally quantified variables are denoted by this piece of notation $\lfloor \ldots \rfloor$.

We attempt to prove (7) using a $hd :: tl$ induction rule of inference, selecting a as the induction variable[4]. The proof fails in the base case, ($a = \mathrm{nil}$).

$$\begin{aligned} &\mathrm{length}(\mathrm{nil} <> b) > \mathrm{length}(\mathrm{nil}) \\ &\mathrm{length}(b) > \mathrm{length}(\mathrm{nil}) \\ &\mathrm{length}(b) > 0 \\ &\mathrm{length}(b) \neq 0 \end{aligned} \tag{8}$$

With (8), a nested induction is suggested, $v_n :: b$. This time the base case, $b = \mathrm{nil}$, gives rise to a contradictory blocked goal:

$$\begin{aligned} \mathrm{length}(\mathrm{nil}) &\neq 0 \\ 0 &\neq 0 \end{aligned}$$

Definition 1 Contradictory Blocked Goals. A goal G is said to be *contradictory blocked* if it is in canonical form, i.e. it cannot be further rewritten, all its variables are instantiated, and it is false in the domain of the theory in which we are working.

This contradiction suggests our first patch, namely, to introduce $b \neq \mathrm{nil}$, i.e. the negation of the base case for the most recent induction, as a condition to the original conjecture. The process of correcting a fault is guided by failure in that

[3] The operators ::, $<>$, and s() represent the infix list constructor function, the lists concatenation function, and the successor constructor function, respectively.

[4] This will be abbreviated as $IndScheme[IndVar]$; where $IndVar$ is the induction variable, and $IndScheme$ is the suggested induction rule of inference.

it would not lead to the contradictory blocked goal experienced in the first proof attempt. This gives us a new goal of the form:

$$\forall a, b : \text{list}(DataType).\ b \neq \text{nil} \rightarrow \text{length}(a <> b) > \text{length}(a) \tag{9}$$

With the revised conjecture (9), a $v_n :: a$ induction schema is again suggested. This time the base case proof obligation goes through and so does the step case. The initial induction conclusion takes the form:

$$\lfloor b \rfloor \neq \text{nil} \rightarrow \text{length}(\boxed{v_0 :: \underline{a}}^{\uparrow} <> \lfloor b \rfloor) > \text{length}(\boxed{v_0 :: \underline{a}}^{\uparrow}) \tag{10}$$

Rippling-out (10) with (4) results in:

$$\lfloor b \rfloor \neq \text{nil} \rightarrow \text{length}(\boxed{v_0 :: \underline{a <> \lfloor b \rfloor}}^{\uparrow}) > \text{length}(\boxed{v_0 :: \underline{a}}^{\uparrow})$$

By wave-rule (6) this rewrites both, the right-hand side (RHS) and the left-hand side of the above formula, to give us:

$$\lfloor b \rfloor \neq \text{nil} \rightarrow \boxed{s(\underline{\text{length}(a <> \lfloor b \rfloor)})}^{\uparrow} > \boxed{s(\underline{\text{length}(a)})}^{\uparrow}$$

and finally, wave-rule (5) gives us:

$$\lfloor b \rfloor \neq \text{nil} \rightarrow \text{length}(a <> \lfloor b \rfloor) > \text{length}(a)$$

Note how this expression matches the induction hypothesis. We can appeal therefore directly to the hypothesis to complete the proof. This process is called *strong fertilization.*

Definition 2 Exploiting Contradictory Blocked Goals. The preconditions and associated patch for exploiting contradictory blocked goals through the use of the information derived from failed cases are as follows:

CRITIC induction (correction of faulty conjecture)
Precondition:
1. *The goal G is blocked contradictory,* e.g.
   $\ldots \vdash 0 \neq 0$

Patch:
Negate the condition for the most recent induction case and add it as a condition to the original conjecture, e.g.
$\forall a, b : \text{list}(DataType).\ b \neq \text{nil} \rightarrow \text{length}(a <> b) > \text{length}(a)$

3.2 On Fixing Non-Theorems by Refinement

As the reader may now suspect, it is possible to have a false conjecture in which the patch suggested by the above heuristic is not sufficient to transform the non-theorem into a theorem. This situation is likely to occur whenever the condition consists of either a predicate other than equality or a combination of predicates.

As a solution to this problem, we have defined a strategy which refines previous patching attempts. As will become clear later, our strategy exploits both syntactic (rippling) and semantic information. Consider again (1), the example conjecture shown in Sect. 1. The recursive definitions of double and half give rise to the following rewrites:

$$\begin{aligned} \mathrm{double}(0) &\Rightarrow 0 \\ \mathrm{double}(\boxed{\mathrm{s}(\underline{X})}^{\uparrow}) &\Rightarrow \boxed{\mathrm{s}(\mathrm{s}(\underline{\mathrm{double}(X)}))}^{\uparrow} \qquad (11) \\ \mathrm{half}(0) &\Rightarrow 0 \\ \mathrm{half}(\mathrm{s}(0)) &\Rightarrow 0 \\ \mathrm{half}(\boxed{\mathrm{s}(\mathrm{s}(\underline{X}))}^{\uparrow}) &\Rightarrow \boxed{\mathrm{s}(\underline{\mathrm{half}(X)})}^{\uparrow} \qquad (12) \end{aligned}$$

In addition, we assume that our theory of natural numbers includes the predicates even and odd[5]:

$$\begin{aligned} \mathrm{even}(0) &\Rightarrow \mathrm{true} \\ \mathrm{even}(\mathrm{s}(0)) &\Rightarrow \mathrm{false} \qquad (13) \\ \mathrm{even}(\boxed{\mathrm{s}(\mathrm{s}(\underline{X}))}^{\uparrow}) &\Rightarrow \mathrm{even}(X) \qquad (14) \\ \mathrm{odd}(0) &\Rightarrow \mathrm{false} \\ \mathrm{odd}(\mathrm{s}(0)) &\Rightarrow \mathrm{true} \qquad (15) \\ \mathrm{odd}(\boxed{\mathrm{s}(\mathrm{s}(\underline{X}))}^{\uparrow}) &\Rightarrow \mathrm{odd}(X) \qquad (16) \end{aligned}$$

Furthermore, we assume the wave-rule for the cancellation of the successor function:

$$\boxed{\mathrm{s}(\underline{X})}^{\uparrow} = \boxed{\mathrm{s}(\underline{Y})}^{\uparrow} \Rightarrow X = Y \qquad (17)$$

We attempt to prove (1) using $\mathrm{s}(\mathrm{s}(x))$ induction. The first base case ($n = 0$) is trivial. It is the second base case ($n = \mathrm{s}(0)$) which is interesting since it gives rise to a contradiction, as shown below.

$$\begin{aligned} \mathrm{double}(\mathrm{half}(\mathrm{s}(0))) &= \mathrm{s}(0) \\ \mathrm{double}(0) &= \mathrm{s}(0) \\ 0 &= \mathrm{s}(0) \end{aligned}$$

[5] The predicate odd is not needed, but is included to show that the technique does not fail in the presence of irrelevant information.

This suggests our first patch attempt of introducing the condition $n \neq \mathrm{s}(0)$ using the strategy defined in the previous section. This gives a new conjecture of the form:

$$\forall n : nat.\ n \neq \mathrm{s}(0) \rightarrow \mathrm{double}(\mathrm{half}(n)) = n \tag{18}$$

With the revised conjecture, (18), a $\mathrm{s}(\mathrm{s}(n))$ induction rule of inference is again suggested. This time both base cases go through. In the step case, however, rippling gets blocked. The initial induction conclusion takes the form:

$$\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0) \rightarrow \mathrm{double}(\mathrm{half}(\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow})) = \boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow}$$

By wave-rule (12) we get:

$$\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0) \rightarrow \mathrm{double}(\boxed{\mathrm{s}(\underline{\mathrm{half}(n)})}^{\uparrow}) = \boxed{s(s(\underline{n}))}^{\uparrow}$$

Rippling-out this formula with (11) results in:

$$\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0) \rightarrow \boxed{\mathrm{s}(\mathrm{s}(\underline{\mathrm{double}(\mathrm{half}(n))}))}^{\uparrow} = \boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow}$$

Finally, two applications of wave-rule (17) give us:

$$\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0) \rightarrow \mathrm{double}(\mathrm{half}(n)) = n$$

Note how this formula matches the induction hypothesis *modulo* the antecedents. Although strong fertilization is not possible we are *potentially* in a position to perform what is defined as *conditional fertilization.* Conditional fertilization extends strong fertilization with conditional equations.

Definition 3 Conditional Fertilization. The preconditions to apply conditional fertilization are the following:

METHOD fertilize (conditional)
Preconditions:

1. *The conclusion and hypothesis match modulo the antecedents,* e.g.
   $(n \neq \mathrm{s}(0)) \rightarrow \mathrm{double}(\mathrm{half}(n)) = n$
   $\vdash (\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0)) \rightarrow \mathrm{double}(\mathrm{half}(n)) = n$
2. *The antecedent of the hypothesis is logically implied by the antecedent of the conclusion.*

For our example the first precondition holds while the second is obviously false. The failure of the fertilize method suggests that our initial condition, $n \neq$ s(0), was *necessary* but not *sufficient* in order to make (1) into a theorem.

Our second attempt at patching (1) is syntactically driven and represents a refinement of our first patch. We analyse the second failure with the aim of finding a wave-function which will not lead to the blockage experienced in the second proof attempt, i.e.

$$\underbrace{\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0)}_{\text{blockage}} \rightarrow \ldots$$

We are looking for a wave-rule of the form:

$$F(\boxed{\mathrm{s}(\mathrm{s}(\underline{X}))}^{\uparrow}) \Rightarrow \ldots$$

Note how this wave-rule would allow further rippling. In addition, we know that F must be of type Nat$\rightarrow$ Bool. Taking these constraints into consideration there are two[6] candidate wave-rules within our theory: (14) and (16). So, for our current example, F may be

$$\lambda x.\mathrm{even}(x) \tag{19}$$

or

$$\lambda x.\mathrm{odd}(x)$$

Now we exploit our semantic knowledge. From the first patch attempt we know that[7] $F(\mathrm{s}(0))$ must evaluate to false. Looking at rewrites (13) and (15) we see that (19) is the correct instantiation for F. Note, however, that our strategy should be extended to cope with cases in which the actual patch consists of a combination of predicates. The corrected conjecture becomes:

$$\forall n : nat.\ \mathrm{even}(n) \rightarrow \mathrm{double}(\mathrm{half}(n)) = n \tag{20}$$

which is actually provable.

Definition 4 Fixing Theorems by Refinement. The preconditions and associated patch for refining previous patching attempts are the following:

---

[6] Note that wave-rule (12) is ruled-out for type reasons.
[7] This ensures that the second attempt at patching (1) subsumes the first one.

CRITIC fertilize (correction of faulty conjecture)
Preconditions:

1. *The conclusion and hypothesis match modulo the antecedents,* e.g.
$$\ldots(n \neq \mathrm{s}(0)) \rightarrow \mathrm{double}(\mathrm{half}(n))\ldots$$
$$\vdash (\boxed{\mathrm{s}(\mathrm{s}(\underline{n}))}^{\uparrow} \neq \mathrm{s}(0)) \rightarrow \mathrm{double}(\mathrm{half}(n))$$
2. *The antecedent of the hypothesis is* **not** *logically implied by the antecedent of the conclusion,* e.g.
$\mathrm{s}(\mathrm{s}(n)) \neq \mathrm{s}(0) \rightarrow n \neq \mathrm{s}(0)$ *is not elementary.*
3. *And there exists a wave rule and a rewrite of the form:*
$$F(\boxed{C(\underline{X})}^{\uparrow}) \Rightarrow \ldots$$
$$F(\mathrm{val}) \Rightarrow \ldots$$
respectively, where C is a constructor function determined by the selected induction schema, and val is the value associated with the problematic case condition (antecedent), e.g.
$$\mathrm{even}(\boxed{\mathrm{s}(\mathrm{s}(\underline{X}))}^{\uparrow}) \Rightarrow \mathrm{even}(X)$$
$$\mathrm{even}(\mathrm{s}(0)) \Rightarrow \mathrm{false}$$

Patch:

Replace the condition attached to the conjecture by $F(x)$, e.g
$$\forall n : nat.\ \mathrm{even}(n) \rightarrow \mathrm{double}(\mathrm{half}(n)) = n$$

3.3 Lochs and Dikes

Consider the following faulty conjecture:

$$\forall A, B : \mathrm{list}(T).\ \mathrm{rev}(\mathrm{rev}(A <> B)) \ = \ \mathrm{rev}(\mathrm{rev}(B)) <> \mathrm{rev}(\mathrm{rev}(A)) \tag{21}$$

From which we establish the following induction hypothesis:

$$\mathrm{rev}(\mathrm{rev}(a <> \lfloor b \rfloor)) \ = \ \mathrm{rev}(\mathrm{rev}(\lfloor b \rfloor)) <> \mathrm{rev}(\mathrm{rev}(a)) \tag{22}$$

This formula is false in that the RHS has two arguments in wrong positions. We attempt to prove (22) using a $v_n :: a$ induction. We assume the rewrite rules derived from the definition of $<>$ and the following rewrite rules:

$$\mathrm{rev}(\boxed{X :: \underline{U}}^{\uparrow}) \Rightarrow \boxed{\underline{\mathrm{rev}(U)} <> X :: \mathrm{nil}}^{\uparrow} \tag{23}$$
$$\mathrm{rev}(\mathrm{nil}) \Rightarrow \mathrm{nil}$$
$$\mathrm{rev}(\boxed{\underline{U} <> X :: \mathrm{nil}}^{\uparrow}) \Rightarrow \boxed{X :: \underline{\mathrm{rev}(U)}}^{\uparrow} \tag{24}$$
$$\boxed{X :: \underline{U}}^{\uparrow} = \boxed{X :: \underline{V}}^{\uparrow} \Rightarrow U = V \tag{25}$$

The base case is trivial. The step case proceeds as follows. First of all, the induction conclusion takes the form:

$$\mathrm{rev}(\mathrm{rev}(\boxed{v_1 :: \underline{a}}^{\uparrow} <> \lfloor b \rfloor)) = \mathrm{rev}(\mathrm{rev}(\lfloor b \rfloor)) <> \mathrm{rev}(\mathrm{rev}(\boxed{v_1 :: \underline{a}}^{\uparrow}))$$

By the application of wave rule (4) we get

$$\mathrm{rev}(\mathrm{rev}(\boxed{v_1 :: \underline{a <> \lfloor b \rfloor}}^{\uparrow})) = \mathrm{rev}(\mathrm{rev}(\lfloor b \rfloor)) <> \mathrm{rev}(\mathrm{rev}(\boxed{v_1 :: \underline{a}}^{\uparrow}))$$

Wave-rules (23) rewrites this twice to give us:

$$\mathrm{rev}(\boxed{\underline{\mathrm{rev}(a <> \lfloor b \rfloor)} <> v_1 :: nil}^{\uparrow}) =$$
$$\mathrm{rev}(\mathrm{rev}(\lfloor b \rfloor)) <> \mathrm{rev}(\boxed{\underline{\mathrm{rev}(a)} <> v_1 :: nil}^{\uparrow})$$

Two applications of (24) transform the above formula into:

$$\boxed{v_1 :: \underline{\mathrm{rev}(\mathrm{rev}(a <> \lfloor b \rfloor))}}^{\uparrow} = \mathrm{rev}(\mathrm{rev}(\lfloor b \rfloor)) <> \boxed{v_1 :: \underline{\mathrm{rev}(\mathrm{rev}(a))}}^{\uparrow} \tag{26}$$

At this point, no further rewriting is possible but *weak fertilization* is applicable. The use of the induction hypothesis as a rewrite rule is called weak fertilization. Having fertilized (26), the resulting formula is considered as a sub-goal to be proved using (a nested) induction. However, any proof attempt will be fruitless because the conjecture is false. The problem is that we cannot assume this in advance. As a partial solution we have implemented a simple counter-example finder that evaluates a few standard instantiations to check whether a given formula is trivially unprovable. The counter-example finder provides us with the means of detecting a faulty occurrence.

It is clear that the left-hand side (LHS) of (26) is fully rippled, whereas its right-hand side (RHS) is blocked. According to the rippling paradigm, we say that the wave fronts on the RHS cannot ripple-out all the way up to the very top of that side. We may think that there is a *dike* in the middle of the loch such that it is not possible for the waves to raise up in the conjecture structure.

Our failure location process is guided by the partial use of the induction hypothesis. This process is called *lemma calculation* and it is described in [8]. Lemma calculation is aimed at providing missing wave-rules which complete the proof for a given conjecture. It is invoked whenever rippling gets blocked and there exists the opportunity to exploit the induction hypothesis.

For our example, the lemma calculation technique would first apply the induction hypothesis to get:

$$v_1 :: (\mathrm{rev}(\mathrm{rev}(b)) <> \mathrm{rev}(\mathrm{rev}(a))) = \mathrm{rev}(\mathrm{rev}(b)) <> v_1 :: \mathrm{rev}(\mathrm{rev}(a))$$

which generalises to the following lemma:

$$\forall X : T,\ \forall U, V : \mathrm{list}(T).\ X :: (U <> V) = U <> X :: V \tag{27}$$

If an induction proof is able to establish this conjectured formula, the following wave-rule would be available:

$$U <> \boxed{X :: \underline{V}}^{\uparrow} \Rightarrow \boxed{X :: \underline{U <> V}}^{\uparrow}$$

Note how this wave-rule would allow further rewriting and completing the proof. As the reader may now notice, (27) is not a valid lemma. But even if it is not valid we can still exploit the information that it provides. If we look carefully at it, we will notice that the wave-front term, *i.e.* $X :: \ldots$, introduced by the step case proof obligation has to move outwards past both $<>$ and U. This observation enables to deduce that correcting (22) can be achieved by performing one of the following actions:

- Emptying one of the lochs, i.e. to force $a = nil$ or $b = nil$.
- Eliminating the dike, i.e. to force $a = b$.

From the above actions, we prefer the latter. This strategy has been implemented by switching the positions of these variables in one side of the expression, looking for a pattern of the form:

$$F1(A, F2(X, B)) = F2(X, F1(A, B))$$

or any plausible combination, e.g. $F2(X, F1(A, B)) = F1(A, F2(X, B))$.

Definition 5 Lochs and Dikes. The preconditions and associated patch for exploiting blocked goals derived from the permuting arguments error are the following:

CRITIC wave (lochs and dikes)
Preconditions:
1. *The current sub-goal can be disproved without much effort;*
2. *The principal connective in the goal is equality or implication;*
3. *Rippling is blocked in one side of the formula but the other side is fully rippled;*
4. *The lemma calculation technique suggests a lemma of the form:*
   $$F1(A, F2(X, B)) = F2(X, F1(A, B))$$
   or any plausible combination;
5. *A and B are of the same type;*
6. *Exchange the positions of A and B in one side of the formula; and*

allõw *The resulting formula cannot be disproved without much effort.*

Patch:
Initiate the plan formation for the modified conjecture.

4 Implementation Aspects

The strategies presented in the above section have been built upon CLAM v3.1, [11]. They have been captured as a collection of critics. CLAM v3.1 was especially designed to realise the proof critics technique described in [7].

However, correcting faulty conjectures by adding conditions gives rise to the problem of finding a proof for conditional equations. Generally speaking, we now have goals of the form:

$$C[N] \rightarrow P[N] \ \vdash \ C[\boxed{S(\underline{N})}] \rightarrow P[\boxed{S(\underline{N})}]$$

where C, P, and S are terms with a distinguished argument, C is the antecedent, and S any constructor function.

These sort of goals introduce technical problems in proofs by induction. This is because the antecedents get in the way in an actual proof. We have extended the capabilities of the proof planner to cope with these situations. We use two different strategies. In the first one, we allow fertilization once we have proved that the condition of the induction hypothesis holds, we called this *conditional weak fertilization*. In the second one, we split a proof into cases using the condition of the induction hypothesis and its negation. These strategies have also been implemented as proof critics, thus preserving the core the system.

5 Results and Conclusions

We tested our mechanism by making it correct a set of 45 faulty conjectures that included the sort of faults that were mentioned in Sect. 3. It proved to be capable of correcting 80% of them. It corrected 72.3% of false conjectures with wrong definitions in boundary values; 72.3% of faulty conjectures with wrong definitions beyond boundary values; and 91.67% of non-theorems in which the fault consisted of wrong definitions in the properties of operators.

OYSTER has been especially designed to be applied in the problem of computer program synthesis. We would like to apply the strategies outlined in this paper in the correction of faulty computer program specifications. This process may involve the creation of guards to constrain the input domain of the synthesised code. Note the similarity between these guards and the conditions that transform non-theorems into theorems. Table 1 shows some interesting non-theorems that were corrected using the strategies outlined in Sect. 3.

References

1. Bundy, A.: The Use of Explicit Plans to Guide Inductive Proofs. In 9th Conference on Automated Deduction. Lusk, R. and Overbeek, R.(Eds.). (1988) 111–120. Longer version available from Edinburgh as DAI Research Paper No. 349
2. Bundy, A. and van Harmelen, F. and Hesketh, J. and Smaill, A.: Experiments with Proof Plans for Induction. Journal of Automated Reasoning **7** (1991) 303–324.

Table 1. Example non-theorems successfully corrected

Non-Theorems	Theorems
$\mathrm{length}(a <> b) > \mathrm{length}(a)$	$b \neq \mathrm{nil} \rightarrow \mathrm{length}(a <> b) > \mathrm{length}(a)$
$\mathrm{length}(a) < \mathrm{length}(a <> b)$	$b \neq \mathrm{nil} \rightarrow \mathrm{length}(a) < \mathrm{length}(a <> b)$
$\mathrm{half}(x) < \mathrm{double}(x)$	$x \neq 0 \rightarrow \mathrm{half}(x) < \mathrm{double}(x)$
$\mathrm{half}(x) < x$	$x \neq 0 \rightarrow \mathrm{half}(x) < x$
$x < \mathrm{double}(x)$	$x \neq 0 \rightarrow x < \mathrm{double}(x)$
$x + y > x$	$y \neq 0 \rightarrow x + y > x$
$x + y > \mathrm{s}(x)$	$y > \mathrm{s}(0) \rightarrow x + y > \mathrm{s}(x)$
$\neg\mathrm{even}(x)$	$\mathrm{odd}(x) \rightarrow \neg\mathrm{even}(x)$
$\neg\mathrm{odd}(x)$	$\mathrm{even}(x) \rightarrow \neg\mathrm{odd}(x)$
$\mathrm{double}(\mathrm{half}(x)) = x$	$\mathrm{even}(x) \rightarrow \mathrm{double}(\mathrm{half}(x)) = x$
$\mathrm{double}(\mathrm{half}(x)) \neq x$	$\mathrm{odd}(x) \rightarrow \mathrm{double}(\mathrm{half}(x)) \neq x$
$\mathrm{even}(x) \rightarrow \mathrm{even}(x + y)$	$\mathrm{even}(y) \rightarrow (\mathrm{even}(x) \rightarrow \mathrm{even}(x + y))$
$\neg\mathrm{even}(\mathrm{length}(a))$	$\mathrm{oddl}(a) \rightarrow \neg\mathrm{even}(\mathrm{length}(a))$
$\mathrm{odd}(\mathrm{length}(a))$	$\mathrm{oddl}(a) \rightarrow \mathrm{odd}(\mathrm{length}(a))$
$a <> (b <> c) = (a <> c) <> b$	$a <> (c <> b) = (a <> c) <> b$
$\mathrm{rev}(\mathrm{rev}(a <> b)) = b <> a$	$\mathrm{rev}(\mathrm{rev}(b <> a)) = b <> a$
$\mathrm{rev}(a <> b) = \mathrm{rev}(a) <> \mathrm{rev}(b)$	$\mathrm{rev}(a <> b) = \mathrm{rev}(b) <> \mathrm{rev}(a)$
$a <> \mathrm{rev}(b) = \mathrm{qrev}(b, a)$	$rev(b) <> a = \mathrm{qrev}(b, a)$
$a <> b = b <> a$	$b <> a = b <> a$
$\mathrm{rev}(a <> x :: \mathrm{nil}) = \mathrm{rev}(a) <> x :: \mathrm{nil}$	$\mathrm{rev}(a <> x :: \mathrm{nil}) = x :: \mathrm{nil} <> \mathrm{rev}(a)$

The predicate oddl returns true whenever its input, a list of objects, is of length odd. qrev is the tail reverse function. All variables in this table are universally quantified and range over either the Peano natural numbers or lists as it should be clear from the formulae.

3. Bundy, A. and van Harmelen, F. and Horn, C. and Smaill, A.: The Oyster-Clam system. In Proceedings of the 10th International Conference on Automated Deduction. Springer-Verlag. Stickel, M.E. (Ed.). (1990) 647–648.
4. Bundy, A. and Stevens, A. and van Harmelen, F. and Ireland, A. and Smaill, A.: Rippling: A Heuristic for Guiding Inductive Proofs. Artificial Intelligence **62** (1993) 182–253.
5. Cox, P.T. and Pietrzykowski, T.: Causes for Events: Their Computation and Applications. Lecture Notes in Computer Science: Proceedings of the 8th International Conference on Automated Deduction. Siekmann, J. (Ed.) Springer-Verlag. (1986) 608–621.
6. Gordon, M.J. and Milner, A.J. and Wadsworth, C.P.: Edinburgh LCF - A mechanised logic of computation. Lecture Notes in Computer Science **78** (1979).
7. Ireland, A.: The Use of Planning Critics in Mechanizing Inductive Proofs. International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg. Lecture Notes in Artificial Intelligence **624**. Voronkov A. (Ed.). Springer-Verlag. (1992) 178–189
8. Ireland, A. and Bundy, A.: Using Failure to Guide Inductive Proof. Technical Report, Department of Artificial Intelligence (1992). Available from Edinburgh as DAI

Research Paper 613.
9. Peirce, C.S.: Collected papers of Charles Sanders Peirce. Vol. 2, 193. Harston, C. and Weiss, P. (Eds.) Harvard University Press. (1959).
10. Selman, B. and Levesque, H.L.: Abductive and Default Reasoning: A Computational Core. In Proccedings of the 8th National Conference on Artificial Intelligence. (1989) 343–348.
11. van Harmelen, F.: The CLAM Proof Planner, User Manual and Programmer Manual. Technical Paper 4. Department of Artificial Intelligence, Edinburgh University. 1989