A Comparison of Two Proof Critics: Power vs. Robustness

Louise A. Dennis[1] and Alan Bundy[2]

[1] School of Computer Science and Information Technology, University of Nottingham, Nottingham, NG8 1BB lad@cs.nott.ac.uk
[2] Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh, EH1 1HN bundy@dai.ed.ac.uk

Abstract. Proof critics are a technology from the proof planning paradigm. They examine failed proof attempts in order to extract information which can be used to generate a patch which will allow the proof to go through.
We consider the proof of the "whisky problem", a challenge problem from the domain of temporal logic. The proof requires a generalisation of the original conjecture and we examine two proof critics which can be used to create this generalisation. Using these critics we believe we have produced the first automatic proofs of this challenge problem.
We use this example to motivate a comparison of the two critics and propose that there is a place for specialist critics as well as powerful general critics. In particular we advocate the development of critics that do not use meta-variables.

1 Introduction

Proof planning is an AI-search based technique for automated proof. It has been applied with particular success to proofs by mathematical induction. A significant component of proof planning is the use of critics to patch failed proof attempts based upon the way in which the proof is proceeding. Proof critics exist which provide generalisations, speculate missing lemmas and revise induction schemes.

The "whisky problem" is a challenge problem from the domain of first order temporal logic. Its proof requires induction and a generalisation of the original conjecture. Proof critics appeared to be an obvious technology to use in an attempt to provide an automated proof of this theorem. We use the whisky problem here as a basis for the examination of two generalisation critics. The first of these uses meta-variables which are gradually instantiated as a proof progresses. The second is a more specialised equivalent which did not use such meta-variables. The second critic proved more robust to slight changes to the context in which the proof attempt was performed and illustrated that it is sometimes beneficial to favour a less general solution (if one is available).

2 The Whisky Problem

Temporal logics are extensions of classical logic. They add a concept of time to classical logic and include syntax for the way things may persist or change as time progresses. They are closely related to modal logics. In general such logics introduce additional quantifiers: in this case box, $\Box$, and circle, $\bigcirc$. In the domain of linear discrete temporal logic $\Box$ is generally taken to mean "always" and $\bigcirc$ as "next". Time is modeled as a sequence of discrete states so $\bigcirc P$ means that P will be true at the next point in time. There is not the space here to give a full description of temporal logics. The interested reader is refered to [10].

Several effective automatic approaches have been developed for propositional modal and temporal logic (e.g. [9]). There has been relatively little work in the field of first-order temporal logics, many of which are undecidable. Contributions in this area include [6, 7].

The whisky problem[1] is a challenge problem in the domain of linear, discrete first-order temporal logic. Its statement is:

$$\begin{gathered} p(a)\wedge \\ \forall x.\ (p(x) \Rightarrow p(h(x)))\wedge \\ \Box\forall x.\ (p(h(x)) \Rightarrow \bigcirc p(x)) \Rightarrow \\ \Box p(a). \end{gathered}$$

The challenge here is that, in order for the proof to go through, the conjecture $p(a)$ has to be generalised to $p(h^n(a))$ where h^n indicates n applications of the function h. This solution is actually outside the first order domain but a similar solution can be obtained by enriching the First-Order signature to contain a new function[2].

We have investigated two *proof critics* which interrupt the course of the proof to suggest generalisations of the original conjecture based on information gained by the failure of the naive proof attempts. Both the generalisation mechanisms eventually proposed the generalisation used in the hand proof.

3 Proof Planning and The Lambda-Clam Proof Planner

Proof planning was first suggested by Bundy [2]. A proof plan is a proof of a theorem at some level of abstraction presented as a tree. A proof plan is generated using AI-style planning techniques. The planning operators used by a proof planner are called *proof methods*.

The first and most famous proof plan is that for induction with the associated rippling heuristic (a form of rewriting constrained to be terminating by meta-logical annotations). This was implemented in the *Clam* [5] proof planner.

1 This problem is attributed to Regimantas Pliuskevicius. But the problem as stated here is considerably more concrete than the general problem posed by Dr. Pliuskevicius (Private Communication). Reputedly Dr. Pliuskevicius has promised a bottle of whisky for the first automatic proof of the problem – hence its name.

2 Anatoli Degtyarev, Private Communication

λ*Clam* [14], which we used for this work, is a higher-order descendant of *Clam* which incorporates both a hierarchical method structure and proof critics (which have not previously been available together in a proof planning system).

λ*Clam* is written in Teyjus λProlog. It works by using depth-first planning with proof methods. Each node in the search tree is a subgoal under consideration at that point. The planner checks the preconditions for the available proof methods at each node and applies those whose preconditions succeed to create the child nodes. The plan produced is then a record of the sequence of method applications that lead to a trivial subgoal. λ*Clam*'s proof methods are believed to be sound although they are not currently reducible to sequences of inference rule applications. We consider this to be comparable to Jones' notion of rigorous argument [12][3].

4 Proof Methods

Proof method application is governed by the preconditions (which may be either legal or heuristic in nature) and by a *proof strategy* which restricts the methods available depending on the progress through the proof. For instance we may wish to simplify a goal as much as possible by applying a rewriting method exhaustively before considering other procedures such as checking for tautologies.

A proof method can be *atomic* or *compound*. If it is compound then it is a *methodical expression* [13] (or sub-strategy) which refers to other methods (e.g. `(repeat_meth sym_eval)` – which can be interpreted as repeatedly apply symbolic evaluation). In this example `sym_eval` is itself another compound method while `repeat_meth` is a *methodical*[4]. Other methodicals exist for sequencing methods, creating OR choices etc. and so complex proof strategies for controlling the search for a proof can be created.

The Proof Strategy for Induction The proof strategy for induction can be seen in figure 1. This is implemented as a selection of atomic and compound methods. The diagram shows a top level repeat which attempts a disjunction of methods. These include basic tautology checking, generalisation of common subterms and also symbolic evaluation and the induction strategy (`ind_strat`). Within the induction strategy, the induction method performs ripple analysis [3] to choose an induction scheme (from a selection specified in λ*Clam*'s theories) and produces subgoals for base and step cases. In brief, ripple analysis applies the induction scheme and then looks to see how many of the instances of the new structure associated with the induction variable can be rewritten – from this it constructs a heuristic score which it uses to choose an appropriate scheme and induction variable.

The top level strategy is then applied once more to the base cases. The step cases are annotated with *skeletons* and *embeddings* and then the wave method

[3] Jones describes a rigorous argument as one that while not fully formal appears in sufficient detail that the writer knows how to formalise it.

[4] Analogous to a tactical in an LCF style theorem prover.

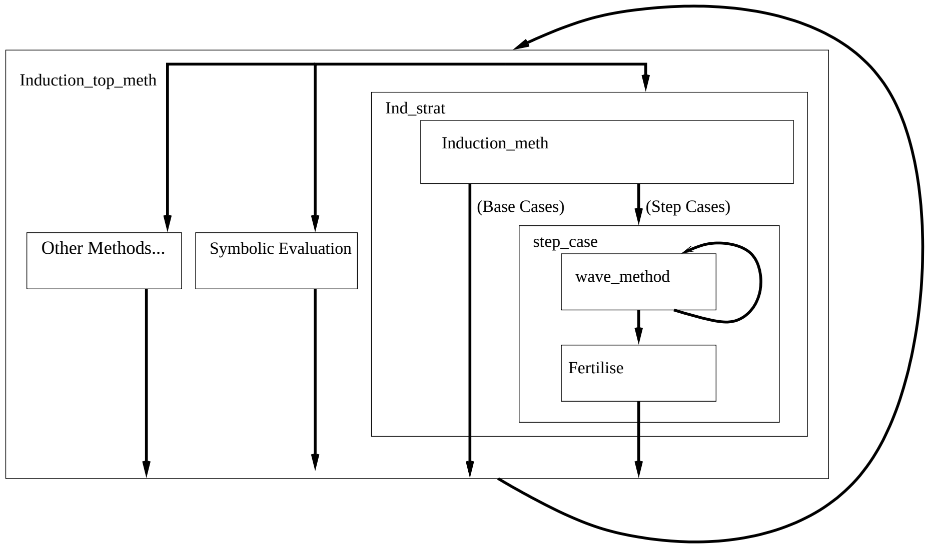

Fig. 1. The Proof Strategy for Induction

is repeatedly applied to them followed by fertilisation (exploitation of the induction hypothesis). Annotations are then removed. The methods (`set_up_ripple` and `post_ripple`) which place and remove annotations are omitted from the diagram. The results are then passed out to the top level strategy again. The process terminates when all subgoals have been reduced to *true* (or *false* in the case of failure).

We will discuss the `wave_method` in more detail since it underpins the generalisation critics we used. The wave method embodies the rippling heuristic. Rippling was first introduced in [4]. We use the theory as presented by Smaill & Green [15] who proposed a version that naturally coped with higher-order features. Rippling steps apply rewrite rules to a target term (called the *erasure*) which is associated with a skeleton and an embedding that relates the skeleton to the erasure (e.g. rippling rewrites an induction conclusion which has an induction hypothesis embedded in it so the induction hypothesis is the skeleton and the conclusion is the erasure). After rewriting, a new embedding of the skeleton into the rewritten term is calculated. There is a measure on embeddings and any rewriting step must reduce this *embedding measure* (written as $<_\mu$).

Rippling is terminating [1]. Rippling either moves differences outwards in the term structure so that they can be canceled away or inwards so that the differences surround a universally quantified variable (or *sink*). If it is possible to move differences inwards in this way the embedding is said to be *sinkable*. The measure on embeddings allows differences that are being moved outwards to be moved inwards but not vice versa.

The `wave_method` method has five preconditions. It finds a rewrite rule that rewrites the goal. It then checks that there is still an embedding of the skeleton into the rewritten goal and that this new embedding is less, according to the embedding measure, than the original embedding. It checks that the embedding is sinkable and that any conditions for the application of the rewrite rule are trivial. This is shown in figure 2[5].

Input

$$ripple\_goal(H \vdash G, S, E).$$

where *ripple_goal* is a triple of a sequent, a skeleton, S and an embedding, E, of that skeleton into the goal, G.

Preconditions

1. The conditional rewrite rule *Rule*, $Cond \Rightarrow X \Leftarrow:\Rightarrow Y$ instantiated with some substitution σ, applies to G and rewrites it to G'.
2. There is an embedding E' that embeds S in G'.
3. $E' <_\mu E$.
4. $\sigma(Cond) = True$ or $\sigma(Cond) \in H$.
5. E' is sinkable.

Output

$$ripple\_goal(H \vdash G', S, E')$$

Fig. 2. The Wave Method

The method will backtrack in order to try to satisfy all requirements, and if it is successful returns a new goal. The main advantages of rippling are that it allows an equation to be treated as a rewrite in both directions without loss of termination and provides useful information for automatically patching failed proof attempts.

We will annotate the goals in our presentation of λ*Clam*'s proof in order to indicate to the reader how the current embedding relates the skeleton to the erasure. If one expression can be embedded into another then this is indicated in the notation by placing boxes (or *wave fronts*) round the additional parts of the larger expression. The embedded expression can be identified as those

[5] We use the notation $\Leftarrow:\Rightarrow$ and $:\Rightarrow$ to indicate rewrite rules. $\Leftarrow:\Rightarrow$ indicates that the rule is derived from an equality and can be used in either direction while $:\Rightarrow$ implies it is derived from an implication.

parts of the expression that are outside the boxes or those parts within boxes which are underlined (called *wave holes*). As an example the result of difference matching $s(x)$[6] and $s(x+y)$ would be $s(\boxed{\underline{x}+y})$. Here the first expression, $s(x)$, appears outside the wave fronts or in the wave holes of the annotated expression. If the annotations of $s(\boxed{\underline{x}+y})$ are stripped away then the second expression, $s(x+y)$, is left. $s(x)$ is the skeleton and $s(x+y)$ is the erasure. During rippling the direction in which a difference is moving is indicated using an arrow on the wave fronts. If a difference is moving outward in the term structure the arrow is upwards and if the difference is moving inward then the arrow is downwards. We indicate variables which are universally quantified in the skeleton (i.e. sinks) with the symbol $\lfloor\rfloor$. So if for instance the induction hypothesis were $\forall y.n+y=y+n$ then the conclusion might be annotated as $\boxed{s(\underline{n})}^{\uparrow} + \lfloor y \rfloor = \lfloor y \rfloor + \boxed{s(\underline{n})}^{\uparrow}$ [7].

4.1 Proof Critics

A proof strategy provides a guide to which proof methods should be chosen at any given stage of the proof. Knowing which method is expected to apply gives additional information should the system generating the plan fail to apply it. *Proof critics* [11] can be employed to analyse the reasons for failure and propose alternative choices. Critics are expressed in terms of preconditions and patches. The preconditions examine the reasons why the method has failed to apply; ideally they do this with reference to the preconditions of the proof method although they may also include extra information. The proposed patch suggests some change to the proof plan. It may choose to propagate this change back through the plan and then continue from the current point, or it may choose to jump back to a previous point in the proof plan.

As an example of a proof critic, consider the generalisation critic suggested by Ireland [11]. The development of this critic is motivated by consideration of the proof of the correctness of *qrev*.

The rewrite rules for *qrev* are as follows:

$$qrev(nil, L) \Leftarrow:\Rightarrow L, \tag{1}$$
$$qrev(H :: T, L) \Leftarrow:\Rightarrow qrev(T, H :: L), \tag{2}$$

The goal is

$$\forall l.\ rev(l) = qrev(l, nil).$$

In order for a proof to go through this must be generalised to

$$\forall l, m.\ rev(l) <> m = qrev(l, m).$$

[6] The semantics of the symbols here is unimportant to the description but s can be viewed as the successor function on natural numbers and $+$ as addition on natural numbers. This is consistent with the use of these symbols elsewhere in this paper.

[7] Note that in the conclusion the universal quantifier has been removed by skolemisation.

During the inductive proof of the original conjecture the rippling becomes "blocked" at the goal

$$h : \alpha,\ t : list(\alpha),\ rev(t) = qrev(t, nil) \vdash$$
$$\boxed{\underline{rev(t)} <> h :: nil}^{\uparrow} = qrev(\boxed{h :: \underline{t}}^{\uparrow}, nil),$$

while the rewrite rule (2) applies here, it cannot be used because the resulting annotated term $qrev(t, \boxed{h :: \underline{nil}}^{\downarrow})$ is unsinkable: nil is a constant not a variable. The generalisation critic is motivated by the failure of the sinkability precondition to speculate a new goal:

$$\forall x, y.\ F(rev(x), y) = qrev(x, G(y)).$$

Here capital letters indicate meta-variables which will be instantiated during the course of the proof. This generalisation is created by placing a new variable (called the *generalised variable*) in the place of the wave hole of the unsinkable wave front, this is a subterm (called the *blocking subterm*), and creating a meta-variable indicating that there may be some function (as yet unknown) applied to the generalised variable. We call this new term the *generalised subterm.* Since the skeleton appears in the wave holes of the erasure the blocking subterm will also appear in the induction hypothesis and (in the same place) in the goal to which induction was originally applied. It is possible from this to work out a generalisation of this original goal where the generalised subterm appears in place of the blocking subterm – so in the example $G(y)$ (the generalised subterm) replaces nil (the blocking subterm). In the case of a proof involving equality, the generalised variable is also available on the other side of the equality and again a meta-variable is introduced to indicate there may be some function involving that variable. The critic is shown in figure 3.

5 Representing the Whisky Problem in λ*Clam*

λ*Clam* contains no theory of temporal logic. This means it has no induction schemes, rewrite rules, inference rules etc. specialised to the syntax and semantics of temporal logic. However it is fairly trivial to translate problems like the whisky problem into problems in arithmetic (for which λ*Clam* has considerable support). This is done primarily by including a new argument (usually t) in all predicates indicating the point in time at which they are true. The whisky problem is represented as three axioms and a goal to be proved. The axioms are:

$$p(a, 0),$$
$$\forall x : nat.\ p(x, 0) \Rightarrow p(h(x), 0),$$
$$\forall x : nat, t : nat.\ p(h(x), t) \Rightarrow p(x, s(t)).$$

These are treated as the following rewrite rules in λ*Clam*:

$$p(a, 0) :\Rightarrow T, \tag{7}$$
$$p(h(X), 0) :\Rightarrow p(X, 0), \tag{8}$$
$$p(X, s(T)) :\Rightarrow p(h(X), T). \tag{9}$$

Fires if rippling has failed.

Preconditions

- The first four preconditions for rippling have successfully applied.
- Precondition 5 has failed (i.e. the embedding is unsinkable).

Patch

Let t be the blocking subterm (i.e. the subterm of the conclusion which lies in the wave hole of the unsinkable wave front). The goal to which induction was originally applied is $P(t)$. t is of type α. Generalise this goal to $\forall x : \alpha.\ P(G(x))$. If the goal is of the form $X = Y(t)$ then generalise it to $\forall x : \alpha.\ F(X, x) = Y(G(x))$ and vice versa if t appears in X. Restart the proof from that goal with a generalisation step to this new goal.

Fig. 3. The Ireland Generalisation Critic

λ*Clam* also expects all variables and constants to be typed. In what follows we have chosen the a to be of type `nat` (natural numbers) and so h is of type `nat -> nat` etc. Note that the implications in the rewrites are in the opposite direction to those in the axioms because λ*Clam* uses backwards proof.

The goal to be proved is

$$\forall t : nat.\ p(a, t).$$

The `induction_meth` method is applied. This chooses to perform induction on t using the standard induction scheme for natural numbers. This produces two goals: a base case and a step case:

$$p(a, 0), \tag{11}$$

$$n : nat,\ p(a, n) \vdash p(a, s(n)). \tag{12}$$

Base Case (11)

The base case rewrites easily to true using symbolic evaluation and the rule (7).

Step Case (12)

The step case is annotated as:

$$n : nat,\ p(a, n) \vdash p(a, \boxed{s(\underline{n})}^{\uparrow}).$$

The rewrite rule (9) matches but rippling fails because the resulting annotation, $p(\boxed{h(\underline{a})}^{\downarrow}, n)$ is unsinkable since a is a constant.

It is at this point that a critic can be applied.

6 The Ireland Generalisation Critic

The first proof critic we investigated was the Ireland critic (already described). This speculated the following generalisation:

$$\forall x : nat, t : nat.\ p(F(x), t),$$

which is introduced as a new subgoal where the `induction_meth` method was previously used and the proof is restarted from that point.

The `induction_meth` method succeeds as before with the induction rule on natural numbers to give the following base and step cases:

$$\forall x : nat.\ p(F(x), 0), \tag{13}$$

$$n : nat,\ \forall x : nat.\ p(F(x), n) \vdash \forall x : nat.\ p(F(x), s(n)). \tag{14}$$

6.1 Base Case (13)

Once again all the proof methods fail until the system attempts induction. This time ripple analysis suggests the use of a rather unusual induction scheme:

$$\frac{P(B) \qquad n : nat,\ P(G^n(B)) \vdash P(G(G^n(B)))}{\forall n : nat.P((\lambda x.G^x(B))(n))}, \tag{15}$$

F^n represents n applications of F. Its use within λ*Clam* is governed by the rewrite rules:

$$F^0(X) \Leftarrow:\Rightarrow X \tag{16}$$

$$F^{s(N)}(X) \Leftarrow:\Rightarrow F(F^N(X)) \tag{17}$$

The application of this new induction scheme (15) instantiates F to $\lambda x.G^x(B)$ where G and B are new meta-variables yet to be instantiated. This creates the following base and step case:

$$p(B, 0), \tag{18}$$

$$n : nat,\ p(G^n(B), 0) \vdash p(G(G^n(B)), 0). \tag{19}$$

Base Case (18) This is rewritten to true using (7) and instantiating B to a.

Step Case (19) The step case is now

$$n : nat,\ p(G^n(a), 0) \vdash p(G(G^n(a)), 0),$$

since B has been instantiated as a. This is annotated as:

$$n : nat,\ p(G^n(a), 0) \vdash p(\boxed{G(\underline{G^n(a)})}^{\uparrow}, 0),$$

and then rippled using (8)

$$n : nat,\ p(h^n(a), 0) \vdash p(h^n(a), 0).$$

This instantiates G to h. The hypothesis and conclusion are now identical and the goal is proved using fertilisation.

Step Case (14)

This is the step case of the previous induction. Recall F has now been instantiated (via instantiations of B and G) to $\lambda x.h^x(a)$. So the goal is:

$$n : nat\ \forall x : nat.\ p((\lambda n.h^n(a))(x), n) \vdash \forall x : nat.\ p((\lambda n.h^n(a))(x), s(n)).$$

We move the universally quantified variable in the conclusion to the meta-level. At this point we β-reduce both the hypothesis and conclusion before we continue. The `set_up_ripple` method which performs annotations was modified to perform β-reduction at the same time (if appropriate)[8] giving the annotated goal:

$$x' : nat,\ n : nat,\ \forall x : nat.\ p(h^x(a), n) \vdash p(h^{\lfloor x' \rfloor}(a), \boxed{s(\underline{n})}^{\uparrow}).$$

Rippling proceeds with (9) and (17):

$$x' : nat,\ n : nat,\ \forall x : nat.\ p(h^x(a), n) \vdash p(\boxed{h(\underline{h^{\lfloor x' \rfloor}(a)})}^{\downarrow}, n),$$

$$x' : nat,\ n : nat,\ \forall x : nat.\ p(h^x(a), n) \vdash p(h^{\boxed{s(\underline{\lfloor x' \rfloor})}^{\downarrow}}(a), n).$$

This then succeeds by fertilisation matching $s(x')$ in the conclusion with the universally quantified x in the hypothesis.

6.2 Discussion

Using the Ireland critic, λ*Clam* can successfully prove the whisky problem in an automated fashion. However it has to be "set up" carefully in order for this to work. For instance the induction scheme (15) had to be added to the program's knowledge base. In principle this is not a problem since we can assume that an automated theorem prover should operate with a large knowledge base.

The conclusion of this new rule, however, is suspect. The conclusion is $\forall n : nat.\ P((\lambda x.G^x(B))(n))$ and is expressed in this way so it can match $\forall t : nat.\ p(F(n), t)$ with P instantiated to $\lambda x.p(x, y)$ and F instantiated to $\lambda x.G^x(B)$. This only matches the goal because n in the goal is of type `nat`. This is the case because we happened to have arbitrarily chosen a to be of type `nat` and so a variable of this type was used in the generalisation. Had we chosen a to have been of some other type then the rule could not have matched and no proof would have been found.

There were also several less serious complications. The default context in which λ*Clam* works had to be altered to remove (8) from the list of rules considered by symbolic evaluation. If (8) had been available to the symbolic evaluation process then it would have rewritten $p(F(x), 0)$ to $p(x, 0)$ instantiating F to h

[8] This did not adversely effect any of our existing benchmark theorems so can be assumed to be harmless.

before induction was applied. This is an incorrect generalisation of the initial conjecture[9]. The proof attempt then fails to show that the generalisation implies the initial conjecture at the end of the planning process (a step that we've omitted from the presentation of the proofs).

We had to alter the order in which the base and step cases were considered. As a general heuristic we have found it better to consider step cases first in an induction attempt since they tend to be more complex to prove and it is usually more important that meta-variables are instantiated correctly during their proof. In this example it was in the base case that the instantiation of the meta-variable was critical. We also had to add the ability for the `set_up_ripple` method to perform some β-reduction.

Of these problems we consider the first, the fact that the proof only succeeded because we chose a to be a natural number, to be the most significant although the rest indicate that the amount of customisation of the planner required to get a proof to work means that a proof like this could only really be found when the planner was used by an expert. This throws some doubt on the extent to which the process can be considered automatic.

7 The Walsh Generalisation Critic

Given the technical problems involving the use of the Ireland critic it seemed worth investigating alternative approaches to the problem.

A family of critics has been suggested for trapping divergence in proofs. The first critic in this family was suggested by Walsh [16] and was used in proof by implicit induction. A similar critic has also been used with proof by coinduction [8]. This divergence occurs when the pursuit of the proof contains a sequence of goals each involving some accumulating structure e.g.

$$\begin{gathered} p(a,0), \\ p(h(a),0), \\ p(h(h(a)),0). \end{gathered}$$

The divergence critic for proof by coinduction speculated a generalisation of the form $h^n(a)$ which is the desired generalisation in this case. It therefore seemed interesting to explore whether such a critic could be used with the inductive proof strategy.

Divergence is spotted by embedding subsequent goals in each other (e.g. $p(a,0)$ and $p(h(a),0)$). The resulting embedding tells us that anything contained in a wave front of the second term and not in a wave hole is some sort of accumulating structure. The suggested patch is then to treat the contents of the wave front as a function and apply it n times as the generalisation.

We have the added advantage when applying this critic during a ripple proof that we have already calculated the embedding of the induction hypothesis into

[9] I.e. It is not a generalisation of the initial conjecture and so the initial conjecture can not be proved from it.

the conclusion which performs the difference matching process for us. The critic is shown in figure 4.

Fires if rippling has failed.

Preconditions

- The first four preconditions for rippling have successfully applied.
- Precondition 5 has failed (i.e. the embedding is unsinkable).

Patch

Let t be the blocking subterm. From the unsinkable embedding determine the function represented by the unsinkable wave front $\lambda x.F(x)$. The goal to which induction was originally applied is $P(t)$. Generalise this goal to $P(F^n(t))$. Restart the proof from this goal with a generalisation step to this new goal.

Fig. 4. The Walsh Generalisation Critic

Notice that the preconditions for this critic are identical to those for the Ireland critic. However it will not fire in all the same cases. For instance in the original example of the correctness of $qrev$, F is instantiated to $\lambda x.\ h :: x$ – this captures a variable h which was introduced by the `induction_meth` itself. λProlog's internal unification algorithm will not allow this variable to get out of scope and so will prevent the instantiation taking place. This is a good thing since it is unlikely that $(\lambda x.\ h :: x)^n$ will be an appropriate generalisation since the h will change each time an element is removed from the list.

The proof proceeds as with the first proof until the introduction of the generalisation. Here the generalisation is:

$$\forall x : nat, t : nat.\ p(h^x(a), t).$$

The `induction_meth` method then performs induction on t

$$n : nat,\ \forall x : nat.\ p(h^x(a), n) \vdash \forall x : nat.\ p(h^x(a), s(n)), \tag{20}$$
$$\forall x : nat.\ p(h^x(a), 0). \tag{21}$$

Step Case (20)

The step case is annotated as

$$x' : nat,\ n : nat,\ \forall x : nat.\ p(h^x(a), n) \vdash p(h^{\lfloor x' \rfloor}(a), \boxed{s(\underline{n})}^{\uparrow}),$$

and its proof proceeds exactly as the last part of the proof using the Ireland generalisation critic.

Base Case (21)

Again induction succeeds this time on x

$$n : nat,\ p(h^n(a), 0) \vdash p(h^{s(n)}(a), 0), \tag{22}$$
$$p(h^0(a), 0). \tag{23}$$

Step Case (22) The step case is annotated as

$$n : nat,\ p(h^n(a), 0) \vdash p(h^{\boxed{s(\underline{n})}^{\uparrow}}(a), 0),$$

and then rippled as follows using (17) followed by (8):

$$n : nat,\ p(h^n(a), 0) \vdash p(\boxed{h(\underline{h^n(a)})}^{\uparrow}, 0),$$
$$n : nat,\ p(h^n(a), 0) \vdash p(h^n(a), 0),$$

fertilisation then completes the case.

Base Case (23) Rewrites to true using symbolic evaluation and (16) followed by (7).

7.1 Discussion

This critic is considerably less general than the Ireland critic and will apply in fewer situations because of the restrictions involving variable capture. However it has the advantage of avoiding many of the problems caused by the use of meta-variables and is therefore more robust. It achieves an automated proof of the whisky problem in the standard set up and context of $\lambda Clam$.

The only alteration we had to make was the addition of the new critic which means we have to ask whether it will apply in a range of situations or whether it is tailored to this particular problem. Observing that the whisky problem depends upon the use of freely generated terms it is possible to come up with similar problems[10]. We developed a small corpus of about half-a-dozen such problems. Our methods and critic are able to handle some of these. The proofs of the others tended to depend on techniques not yet implemented in $\lambda Clam$ such as lemma speculation or mutual recursion and so were not attempted. The critic was also unable to handle one problem where an alternating structure was required. This problem was created by adding the rule

$$p(X, s(T)) :\Rightarrow p(k(X), T),$$

[10] This observation was made by Alexei Lisitsa who also assisted with generating similar problems.

to the set of rewrites used in the whisky problem and altering rewrite rule (8) to

$$p(h(k(X)), 0) :\Rightarrow p(X, 0).$$

In this case the necessary generalisation is $p((h \circ k)^n(0), t)$ while the critic speculates $p(h^n(0), t)$.

8 Using the Critics Together

Since both critics are triggered by the same failed precondition it would be ideal if we could determine situations when one should be preferred over the other and build that into the critic preconditions. We adapted our existing default set up in λ*Clam* so that it would attempt both critics (first the Walsh critic and then the Ireland critic). In this set up the existing (small) set of benchmarks for testing the Ireland critic continued to plan, suggesting that the new critic did not negatively affect the overall ability of the inductive proof strategy.

Brotherston[11] has been investigating alternatives to the depth-first approach currently adopted by λ*Clam*. In particular he has developed a best-first style planner which could potentially be used to decide between critics and between the use of symbolic evaluation or induction (and possibly between whether to attempt the base or the step case first in an induction). This would require the development of heuristic functions by which the goodness of a given method, goal or critic could be assessed. We need to find a wider range of examples before this could take place.

9 Related Work

Castellini and Smaill [6] have written a tactic-based theorem prover, `FTL`, for first-order modal and temporal logics in Teyjus Prolog. This performs proof in a family of sequent based calculi for quantified temporal logics.

They have created several high-level tactics for automating large parts of a proof. `FTL` has proved a simplified version of the whisky problem:

$$p(a) \wedge \Box\forall x : nat.\ (p(a) \rightarrow p(h(x))) \wedge \Box\forall x : nat.\ (p(h(x)) \rightarrow \bigcirc p(x)) \rightarrow \Box p(a).$$

This presentation of the problem allows a proof where the accumulating h can be simplified away at any point in time instead of working back to the original starting time point. This proof is within the scope of the `auto_tac` developed for `FTL` which exhaustively applies the induction tactics. Castellini and Smaill are investigating integrating their system with λ*Clam* and so hopefully the methods and critics here could be adapted to provide a formal automatic proof of the full whisky problem. Clearly their logic would also have to be extended to cope with the higher-order features of the proof.

[11] Private Communication and MRG BBNotes 1405 and 1409.

10 Conclusion

We have investigated the use of two generalisation critics to provide an automated proof of the whisky problem, a challenge problem for induction from the domain of temporal logic.

The first of these critics is more general but its use of meta-variables made it very sensitive to the context in which the proof was taking place and caused the user to do a certain amount of "fiddling" in advance of the proof attempt in order for it to go through. The second critic was more specialised and had not been used in the context of explicit induction before. With this critic the λ*Clam*'s "default" context was able to produce a proof of the theorem automatically. The new critic comes from a family of similar critics used in other domains and we are convinced that it is sufficiently general that it could be used on a class of problems and is not tailored specifically to this conjecture.

This research suggests that we should look to develop critics which do not use meta-variables where this is possible. It seems plausible to assume that such alternatives may well be less powerful than versions using meta-variables but where they can be used they may prove to be more robust.

As far as we are aware these two proofs provide the first automated solutions to the whisky problem and demonstrate the power and flexibility of the existing proof planning methods and critics.

Acknowledgments

This research was funded by EPSRC grants GR/M45030 and GR/M46624. We thank the members of the Logics Group at Liverpool University and Michael Fisher and Anatoli Degtyarev in particular for drawing our attention to this challenge problem.

References

1. D. Basin and T. Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(1–2):147–180, 1996.
2. A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.
3. A. Bundy. The automation of proof by mathematical induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning, Volume 1*. Elsevier, 2001.
4. A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.
5. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

6. C. Castellini and A. Smaill. Tactic-based theorem proving in first-order modal and temporal logics. In *IJCAR 2001, workshop 10*, 2001.
7. A. Degtyarev and M. Fisher. Towards first-order temporal resolution. In F. Baader, G. Brewka, and T. Eiter, editors, *KI 2001: Advances in Artificial Intelligence, Joint German/Austrian Conference on AI, Vienna, Austria, September 19-21, 2001, Proceedings*, volume 2174 of *Lecture Notes in Computer Science*. Springer, 2001.
8. L. A. Dennis, A. Bundy, and I. Green. Making a productive use of failure to generate witness for coinduction from divergent proof attempts. *Annals of Mathematics and Artificial Intelligence*, 29:99–138, 2000. Also available from Edinburgh as Informatics Report RR0004.
9. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic*, 1(4), 2001.
10. D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, 1994.
11. 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.
12. C. B. Jones. *Software Development: A Rigorous Approach*. Prentice-Hall international series in Computer Science. Prentice-Hall, 1980.
13. J. Richardson and A. Smaill. Continuations of proof strategies. In M. P. Bonacina and B. Gramlich, editors, *Proc. 4th International Workshop on Strategies in Automated Deduction (STRATEGIES 2001)*, Siena, Italy, june 2001. Available from `http://www.logic.at/strategies/strategies01/`.
14. J.D.C. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with lambda-clam. In C. Kirchner and H. Kirchner, editors, *Conference on Automated Deduction (CADE'98)*, volume 1421 of *Lecture Notes in Computer Science*, pages 129–133. Springer-Verlag, 1998.
15. A. Smaill and I. Green. Higher-order annotated terms for proof search. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1275 of *Lecture Notes in Computer Science*, pages 399–414, Turku, Finland, 1996. Springer-Verlag. Also available from Edinburgh as DAI Research Paper 799.
16. T. Walsh. A divergence critic for inductive proof. *Journal of Artificial Intelligence Research*, 4:209–235, 1996.