Formal Aspects of Computing (1999) 3: 1–000

Interactive Proof Critics

Andrew Ireland[1] and Michael Jackson[2] and Gordon Reid[3]

[1] Department of Computing & Electrical Engineering, Heriot-Watt University, Edinburgh, A.Ireland@hw.ac.uk;
[2] School of Computing, Napier University,Edinburgh, mikej@dcs.napier.ac.uk;
[3] Division of Informatics, University of Edinburgh, Edinburgh, G.D.Reid@ed.ac.uk

Keywords: Theorem proving; User interfaces; Proof planning; Proof patching; Inductive proof

Abstract. The key to a successful proof often lies within the analysis of failed proof attempts. Motivated by this observation we have developed and evaluated an interface to an inductive theorem prover which supports a collaborative style of failure analysis. Our work builds upon an automatic proof patching mechanism and extends the capabilities of an existing theorem proving interface. Our approach is multi-disciplinary, we draw upon work from both the automated theorem proving and human computer interaction communities.

1. Introduction

The benefits of theorem proving are recognized by formal methods practitioners [AF97] and have borne fruit within niche markets [CW96]:

"Theorem provers are increasingly being used today in the mechanical verification of safety-critical properties of hardware and software designs."

General purpose theorem provers remain, however, primarily the tool of the academic researcher. This is particularly true of provers which support inductive proof [BM88, KM97, BR93, OSR92, ORA96, BvHHS90, GSG91, KZ95, HS96] where a high level of skill is required on the part of the user. Addressing this skills issue is of crucial importance if practitioners are to make more widespread use of the tools and techniques developed within the theorem proving community.

Correspondence and offprint requests to: Andrew Ireland, Department of Computing & Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS

One approach to addressing the skills issue is to build theorem provers which support effective high level interaction as well as automation. The crucial question is how to achieve an effective balance between automation and high level interaction. Here we describe an interface which, we believe, makes progress towards the goal of achieving an effective balance. The semi-automatic style of theorem proving which we advocate draws upon research from two distinct disciplines:

- The Human Computer Interaction (HCI) community directly addresses the need for "effective" interaction. Researchers have noted that the automated theorem proving community has been slow to capitalize upon the results of the HCI community in building effective interfaces [AGMT95, AGMT98, Sym95]. The development of our interface has been significantly influenced by the work of the HCI community.
- The Automated Theorem Proving community has produced a wide spectrum of techniques for controlling the search for proofs, ranging from the machine-oriented through to more human-oriented approaches. To achieve our goal of "high level" interaction as well as automation we build upon *proof planning* [Bun88], a technique which promotes a human-oriented approach to search control.

Here we investigate the use of HCI ideas in extending the capabilities of the CLAM [BvHHS90] proof planner, an implementation of the proof planning technique. Our contribution is to introduce what we call *interactive proof critics*. This mechanism extends the notion of automatic proof critics which has been used very successfully in automating the activity of patching failed proof attempts [Ire92, IB96b, IB96a, IS97, SI98]. While this paper deals with the issues of interaction within the context of inductive theorem proving we believe that our ideas are relevant to the development of deductive tools in general.

2. What is Proof Patching?

We begin by considering in more detail what we mean by proof patching. We draw upon a theorem, motivated by Matt Kaufmann [Kau90], which is deceptively simple to state yet surprisingly hard to prove using current automatic techniques. The theorem concerns the function *rotate*, i.e. given a natural number N and a list L, then $rotate(N, L)$ computes a list which is constructed by concatenating the first N elements of L onto the end of L. We define *rotate* equationally in the appendix. The property we are interested in proving is that the rotation of any list by its length yields an identical list:

$$\forall T : list(\tau).\ rotate(length(T), T) = T \tag{1}$$

Given our recursive definition of *rotate*, a proof of (1) not surprisingly requires mathematical induction. The definition of *rotate* and *length* suggest a structural induction. While the resulting base case is trivial the step case fails. In order to overcome this failure we must first generalize the conjecture. There are many forms a generalization can take. Indeed generalization represents an infinite branching point within the search space which makes it a very challenging research topic for the automatic theorem proving community. Here the required generalization involves the introduction of an additional universally quantified variable:

$$\forall T : list(\tau).\forall L : list(\tau).\ rotate(length(T), app(T, L)) = app(L, T) \qquad (2)$$

Note that *app* denotes list concatenation. Making the connection between the failure to prove the step case of (1) and the need for the generalized conjecture (2) is typically seen as a *eureka step*. Proof patching is the process of discovering such eureka steps. The process can be viewed in terms of the identification of a failure and the subsequent synthesis of a patch. We are interested in how an automatic approach to proof patching can be adapted so as to enable a user to collaborate in the synthesis task.

3. Limitations of Automatic Inductive Provers

Current automated inductive theorem provers [BM88, KM97, KZ95, ORA96, BR93, BvHHS90] fail to find the generalization step presented in §2. However, experienced users of such provers typically know by observing the prover's output when to interrupt a proof attempt and where to look in order to gain insight into how the failure can be patched. It is this kind of experience which led to the development of the automatic proof critics mechanism. As will be explained in §4.3, the CLAM proof planner augmented with the automatic proof critics [IB96a, IB96b] can discover the required generalization. However, the discovery is dependent upon the appropriate lemmas being available within the CLAM library. This is not the case with our interactive proof critics where the user is given the opportunity of assisting with the proof patching process. More precisely, the interface focuses attention on key failure points and suggests partial proof patches while the user is invited to participate in the selection and instantiation of the suggestions. Consequently, interactive proof critics extend the scope of the proof patching capabilities of CLAM by facilitating a collaborative style of interaction.

4. Automatic Proof Critics

As mentioned above the notion of a proof critic was developed in order to support proof patching through the analysis of proof planning failures. Before describing our interactive extension to this mechanism in §5, we firstly outline proof planning and the automatic proof critics mechanism.

4.1. Proof Plans: Methods and Critics

Proof planning builds directly upon the *tactic*-based [GMW79] approach to proof construction. A proof plan is defined in terms of proof *methods* which are associated with a set of parameterized tactics. Typically a proof plan is defined in terms of a hierarchy of methods. When planning a proof, methods are used to constrain the selection and instantiation of tactics drawn from the tactic set. The output from planning a proof is a composite tactic tailored to the given conjecture. Note that while tactic execution is necessary in order to ensure soundness, all the search is taken care of at the planning level. The constraints of a method are represented as preconditions to the applicability of a tactic. It is the failure, or partial success, of method preconditions which is exploited by the critics mechanism in guiding the process of proof patching.

In order to make the ideas outlined above more concrete we return to the rotate-length example outlined in §2. Firstly, in §4.2 we show how methods can be used to guide the construction of the step case of the generalized conjecture (2). Secondly, in §4.3 we address the harder issue of how the analysis of the failure to prove (1) can be used to generate (2), the required generalization.

4.2. A Step Case Method

We begin by considering the step case proof obligation resulting from an inductive proof attempt of (2). Assuming structural induction on the list T, then the step case takes the form:

$$\begin{array}{rcl} \forall L : list(\tau).\ rotate(length(t), app(t, L)) = app(L, t) & \vdash & \\ rotate(length(h :: t), app(h :: t, l)) & = & app(l, h :: t) \end{array}$$

A successful proof requires the manipulation of the conclusion so that a match with an instance of the hypothesis can be achieved. The manipulation requires the application of rewrite rules. In the context of automatic proof we require a strategy for selecting which rewrite rules to apply. One such strategy involves identifying the term structure within the conclusion which prevents a match with the hypothesis and then restricting the rewriting of the conclusion to rewrite rules which only reduce the mismatching term structure. This strategy is known as *rippling* [BSvH$^+$93, BW94], and in terms of proof planning is called the *ripple* method. A successful application of the ripple method will enable the application of the *fertilize* method, the method which controls the use of induction hypotheses. Ripple and fertilize provide a composite method for dealing with step case proof obligations. The preconditions to rippling are expressed in terms of meta-level annotations, called *wave fronts*, which indicate the mismatching term structures. Here we use shading to represent wave fronts, as shown below:

$$\begin{array}{rcl} \forall L : list(\tau).\ rotate(length(t), app(t, L)) = app(L, t) & \vdash & \\ rotate(length(\boxed{h :: t}^{\uparrow}), app(\boxed{h :: t}^{\uparrow}, l)) & = & app(l, \boxed{h :: t}^{\uparrow}) \end{array}$$

The unannotated term structure within the conclusion is called the *skeleton*. The key property of rippling is *skeleton preservation*. The motivation for skeleton preservation is that it maximizes the chances of fertilization, i.e. matching with an induction hypothesis. The arrow attached to a wave front is used to indicate the direction in which a wave front can be moved with respect to the skeleton. Directed wave fronts enable the termination of rippling to be guaranteed even when bi-directional rewriting is required [BW94].

There are three basic patterns of rewriting associated with rippling which are summarized schematically in figure 1. In general a proof may require all three forms of rippling, as will be illustrated below. Note that the transverse and longitudinal inward patterns enable universally quantified induction hypotheses to be used in eliminating wave fronts. That is, if a wave front is moved to a position which corresponds to a universally quantified variable within the hypothesis, then it can be eliminated by the appropriate instantiation of the hypothesis. We use annotations called *sinks*, i.e. $\lfloor \ldots \rfloor$, within the conclusion to identify potential

longitudinal (outward):

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

before after

transverse:

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

before after

longitudinal (inward):

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

before after

Fig. 1. The three basic rippling patterns

longitudinal (outward):

$$length(\boxed{U :: \underline{V}}^{\uparrow}) \Rightarrow \boxed{s(\underline{length(V)})}^{\uparrow} \tag{3}$$

$$app(\boxed{U :: \underline{V}}^{\uparrow}, W) \Rightarrow \boxed{U :: \underline{app(V,W)}}^{\uparrow} \tag{4}$$

transverse:

$$app(U, \boxed{V :: \underline{W}}^{\uparrow}) \Rightarrow app(\boxed{app(\underline{U}, V :: nil)}^{\downarrow}, W) \tag{5}$$

$$rotate(\boxed{s(\underline{U})}^{\uparrow}, \boxed{V :: \underline{W}}^{\uparrow}) \Rightarrow rotate(U, \boxed{app(\underline{W}, V :: nil)}^{\downarrow}) \tag{6}$$

longitudinal (inward):

$$\boxed{U :: \underline{app(V,W)}}^{\downarrow} \Rightarrow app(\boxed{U :: \underline{V}}^{\downarrow}, W) \tag{7}$$

$$\boxed{app(\underline{app(U,V)}, W)}^{\downarrow} \Rightarrow app(U, \boxed{app(\underline{V}, W)}^{\downarrow}) \tag{8}$$

The wave rules shown illustrate the three basic patterns of rippling presented schematically in figure 1. Note that in general a wave rule may involve any combination of the three basic patterns.

Fig. 2. Example wave rules

positions where wave fronts can be "absorbed" in this way. As will be illustrated below, sinks provide additional meta-level constraints on the rewriting of the conclusion. The skeleton preserving property of rippling is achieved by restricting rewriting to a syntactic class of rewrite rules called *wave rules*. Wave rules are annotated rewrite rules which are guaranteed to be *skeleton preserving* while *making progress towards reducing differences*. Example wave rules derived from the *rotate-length* example are presented in figure 2. Note that the process of annotating rewrite rules is completely automatic. The ripple method controls the application of a sequence of wave rules. Each wave rule application is controlled by a submethod of ripple called *wave*. The preconditions associated with the

wave method are as follows:
Preconditions: wave method (wave rule application)

1. *there exists a wave occurrence within the conclusion;*
2. *and matching wave rule;*
3. *and any condition attached to the wave rule is provable;*
4. *and any inward directed wave occurrences are sinkable.*

Note that precondition *2* holds only if both object-level and meta-level term structures match. Precondition *4* relates to transverse and longitudinal (inwards) rippling, that is, the movement of a wave front sideways or inwards is only allowed if progress towards a sink is guaranteed.

We now return to the *rotate-length* example. By including the sink annotations, the step case proof obligation generated for (2) now takes the form:

$$\begin{aligned} &\forall L : list(\tau).\ rotate(length(t), app(t, L)) = app(L, t) \quad \vdash \\ &\qquad rotate(length(\boxed{h :: \underline{t}}^{\uparrow}), app(\boxed{h :: \underline{t}}^{\uparrow}, \lfloor l \rfloor)) \quad = \quad app(\lfloor l \rfloor, \boxed{h :: \underline{t}}^{\uparrow}) \end{aligned}$$

By wave rules (3) and (4), the left-hand side of the conclusion ripples outwards to give:

$$rotate(\boxed{s(\underline{length(t)})}^{\uparrow}, \boxed{h :: \underline{app(t, \lfloor l \rfloor)}}^{\uparrow}) = app(\lfloor l \rfloor, \boxed{h :: \underline{t}}^{\uparrow})$$

Wave rules (6) and (5) give rise to transverse ripples:

$$rotate(length(t), \boxed{app(\underline{app(t, \lfloor l \rfloor)}, h :: nil)}^{\downarrow}) = app(\left\lfloor \boxed{app(\underline{l}, h :: nil)}^{\downarrow} \right\rfloor, t)$$

Note that the occurrence of the sink term $\lfloor l \rfloor$ is a necessary condition for the application of each of these wave rules. While the wave front on the right-hand side has been absorbed by the sink, the inward directed wave front on the left-hand side requires further rippling to enable fertilization. This is achieved by wave rule (8), the conclusion now becomes:

$$\ldots rotate(length(t), app(t, \left\lfloor \boxed{app(\underline{l}, h :: nil)}^{\downarrow} \right\rfloor) \quad = \quad app(\left\lfloor \boxed{app(\underline{l}, h :: nil)}^{\downarrow} \right\rfloor, t)$$

Rippling has terminated successfully, i.e. all wave fronts have been absorbed, so fertilization is now applicable. Note that by instantiating L to be $app(l, h :: nil)$ a match between conclusion and hypothesis is achieved. This completes the step case proof.

4.3. A Step Case Critic

We now consider how a proof of (2), the generalized conjecture, may be discovered by analyzing the failure to prove (1) directly. Again we focus on the step case proof obligation which now takes the form:

$$rotate(length(t), t) = t \vdash rotate(length(\boxed{h :: \underline{t}}^{\uparrow}), \boxed{h :: \underline{t}}^{\uparrow}) \quad = \quad \boxed{h :: \underline{t}}^{\uparrow}$$

No wave rules are applicable on the right-hand side. On the left-hand side rippling-out becomes blocked after the application of wave rule (3):

$$rotate(\boxed{s(\underline{length(t)})}^{\uparrow}, \underbrace{\boxed{h :: \underline{t}}^{\uparrow}}_{\text{no sink}}) = \boxed{h :: \underline{t}}^{\uparrow} \tag{9}$$

Note that while there is a wave rule which matches, *i.e* (6), it does not apply because there is no sink within the second argument position of *rotate*. In terms of the preconditions for wave rule application, this translates into preconditions *1* to *3* succeeding and *4* failing. The failure of precondition *4* is strongly suggestive of patching the proof by modifying the conjecture, i.e. generalizing the conjecture through the introduction of a new universally quantified variable (sink). Once the kind of patch required has been identified a *patch schema* is constructed. This is achieved through the use of higher-order meta-variables, e.g. the schematic patch for conjecture (1) takes the form:

$$\forall T : list(\tau).\forall L : list(\tau).\ rotate(length(T), \mathcal{M}_1(T, L)) = \mathcal{M}_2(T, L) \tag{10}$$

Note that here we only require second-order meta-variables, i.e. $\mathcal{M}_1$ and $\mathcal{M}_2$. Identifying the need for a particular kind of patch and the construction of the associated patch schema is performed by a proof critic. The instantiation of the patch schema is guided by a subsequent proof planning attempt. It is here that the tight constraints of rippling pay-off. In the case of (10) the ripple and fertilize methods constrain $\mathcal{M}_1$ and $\mathcal{M}_2$ to be $\lambda x.\lambda y.app(x, y)$ and $\lambda x.\lambda y.app(y, x)$ respectively. Note that the use of meta-variables delays the generalization step till later in the construction of a proof. This is an application of a technique known as *middle-out reasoning* [BSH90]. The technique of middle-out reasoning has been implemented for a number of different kinds of proof patches, including the generalization patch outlined above [IB96b, IB96a]. We include a short description below of three other proof critics associated with rippling which will be referred to later in this paper:

Lemma speculation: when a ripple proof is blocked because of the lack of an appropriate wave rule (lemma) one can attempt to patch the proof by constructing a schematic wave rule. The construction of a schematic wave rule is based upon the failure pattern and the skeleton preserving property of rippling. The use of schematic wave rules allows missing lemmas to be discovered incrementally during the course of a proof.

Lemma calculation: this patch involves the use of an induction hypothesis in discovering missing lemmas and is based upon the Boyer-Moore notion of cross-fertilization [BM79].

Induction revision: the selection of an induction rule based upon a local analysis of a conjecture may result in a failure later in a proof attempt. Analyzing such failures within the context of rippling can lead to an appropriate revision of the initial induction selection.

5. Interactive Proof Critics

Given the effectiveness of automatic proof critics [IB96b, IB96a] why have we investigated an interactive approach? Well in general, for a given proof failure, a number of proof critics may be applicable. Moreover, each proof critic may give rise to a number of distinct patch schemas. In such situations a user may usefully prioritize the "more promising" candidates and thus reduce the search. In addition, a user may be able to short-circuit the proof planner's incremental instantiation of a patch schema, either partially or fully. More crucially, however, patches may be mutually related. For example, lemmata (12) and (13) (see appendix) play a pivotal role in the instantiation of (10), but where do these non-definitional properties come from? As mentioned above, a lemma speculation critic has been implemented [IB96b] and can discover the required lemmata as a side-effect of planning a proof of (2), the generalized conjecture. So with the generalized conjecture the auxiliary lemmata can be discovered automatically but finding the generalization relies on the auxiliary lemmata — Catch-22! With interactive proof critics we at least allow for the possibility of a user overcoming such Catch-22 situations.

As mentioned in §1, the automated theorem proving community has been slow to exploit the work of the HCI community in building theorem prover interfaces. We have attempted to rectify this situation by using HCI knowledge to inform the design and evaluation of our interface. There are many possible ways in which one could interact with a proof critic. We have focused upon allowing users to select and instantiate patch schemas which are generated automatically by the existing proof critics. The main motivation for the design of our interface was the important distinction which exists between *necessary* information and *available* information. Necessary information is the information that the user needs to complete their task, in our case, the proposed patches. Available information is information that is not necessary, but may potentially assist them in their task, e.g. the reason why a particular critic is being invoked. In the following subsections we describe our approach to interacting with proof critics. We again use the rotate-length example.

5.1. A Proof Failure

When a method fails and an associated proof critic is triggered the user is presented with the interactive proof critic interface. This interface is illustrated in figure 3. The reader should note that the proof critic interface is only a subcomponent of the overall interface to the proof planner. This interface provides information about why a particular critic was invoked together with the patches being proposed. The interface enables the user to query a proof critic as to why the associated method failed. In response a proof critic will provide high level explanations as to why the associated method failed and why the critic was invoked. These explanations are presented in terms of the method and critic preconditions. Relating this back to the rotate-length example, consider again blocked goal (9), and in particular the following subterm:

$$rotate(\boxed{s(\underline{length(t)})}^{\uparrow}, \boxed{h :: \underline{t}}^{\uparrow})$$

Note that, while wave rule (6) matches the subterm, it fails to apply because of precondition *4*. In other words, the application of the wave rule gives rise to an inward directed wave front which is not sinkable:

$$\ldots rotate(length(t), \underbrace{\boxed{app(\underline{t}, h :: nil)}^{\downarrow}}_{\text{not sinkable}}) \ldots$$

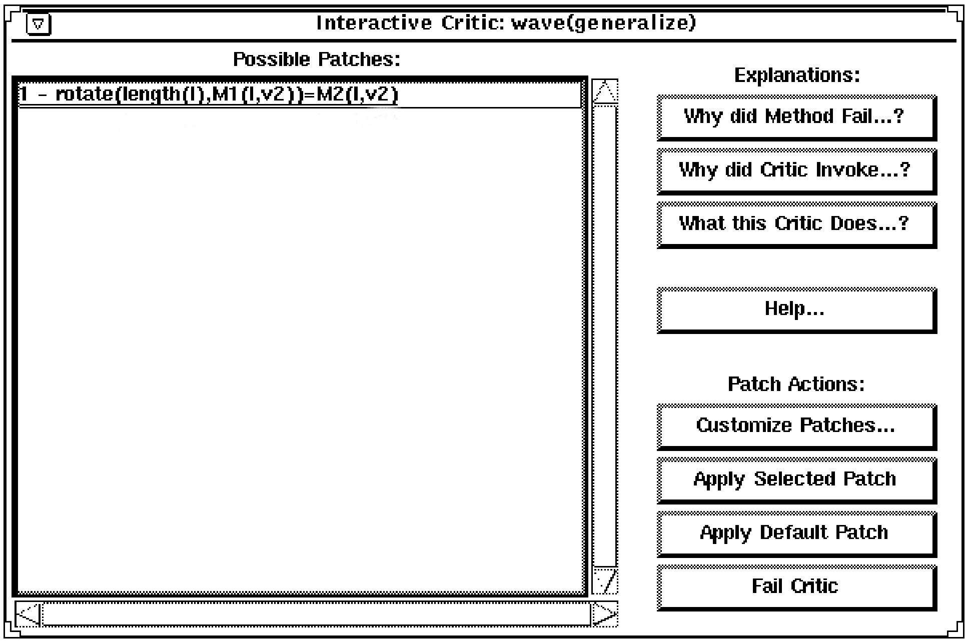


Fig. 3. The Interactive Proof Critic Interface

Figure 4 demonstrates how the critic interface communicates this rationale to the user.

5.2. Patch Schema Presentation

The patch schemas proposed by a proof critic are presented to the user as illustrated in figure 3. In the case of the rotate-length example only one patch is proposed, which takes the form:

$$rotate(length(T), \mathcal{M}_1(T, L)) = \mathcal{M}_2(T, L) \tag{11}$$

Note that typically a user will be presented with a number of alternative patch schemas. Further user assistance is provided by a third form of patch-specific explanation. This explanation informs the user as to how the proof critic will use the patch to overcome the proof failure. An example of this form of explanation, for the patch in our rotate-length example, is shown in figure 5.

5.3. Patch Schema Selection and Instantiation

As mentioned above, the user is invited to select which of the suggested patches should be applied. Prior to applying their selected patch the user has the option of providing an instantiation for the meta-variables in the patch. The interface to the proof critic allows the user to request this. Interaction is provided in such a way as to ensure that no ill-formed instantiations are accepted. The part of the interface which deals with the instantiation of patch schemas is illustrated

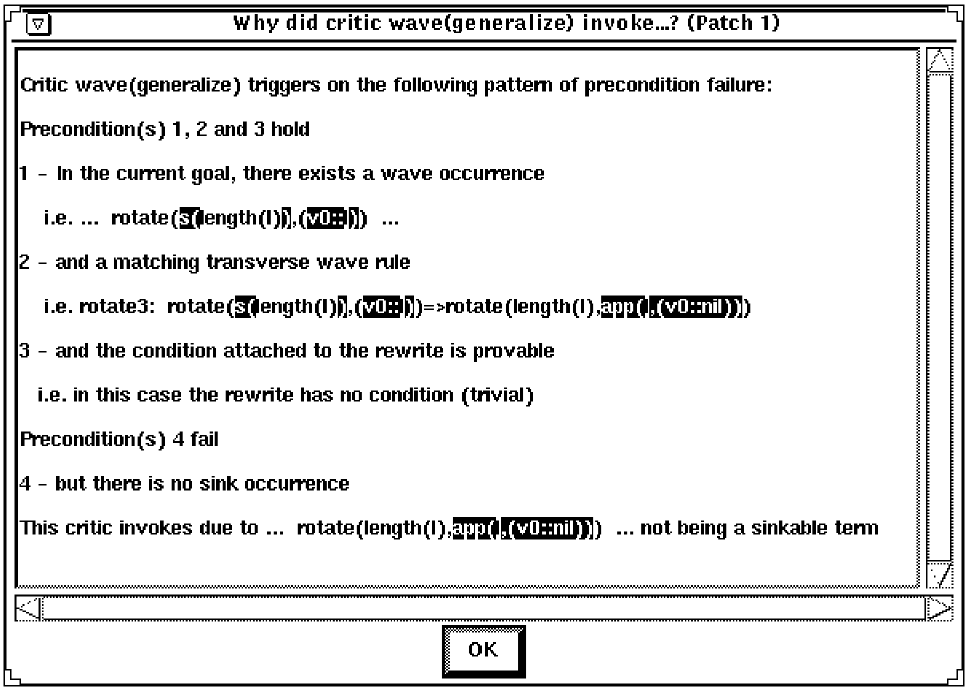


Fig. 4. Explanations - Why is the Patch Applicable?

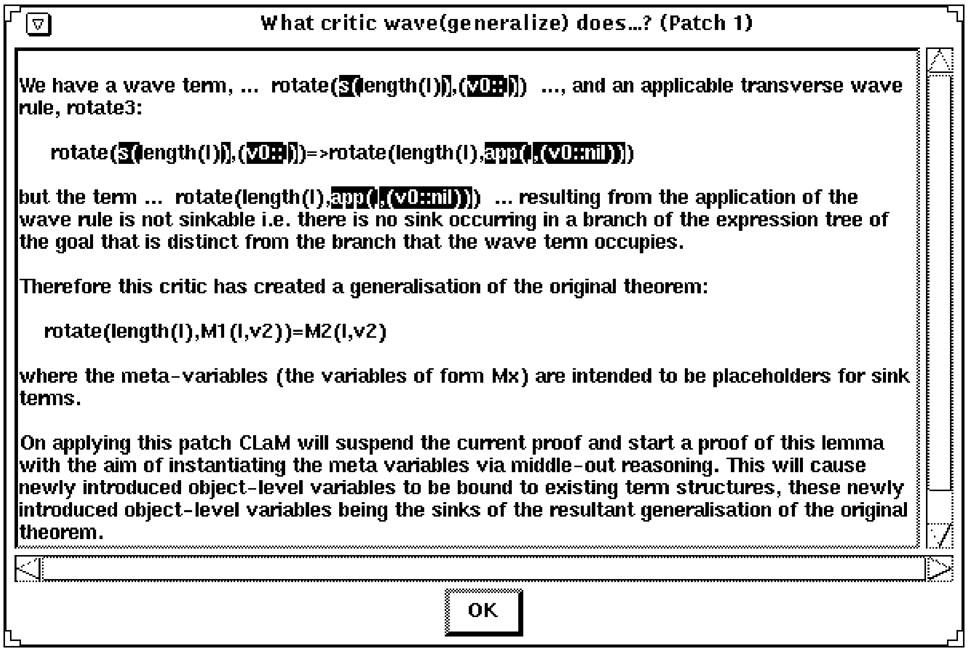
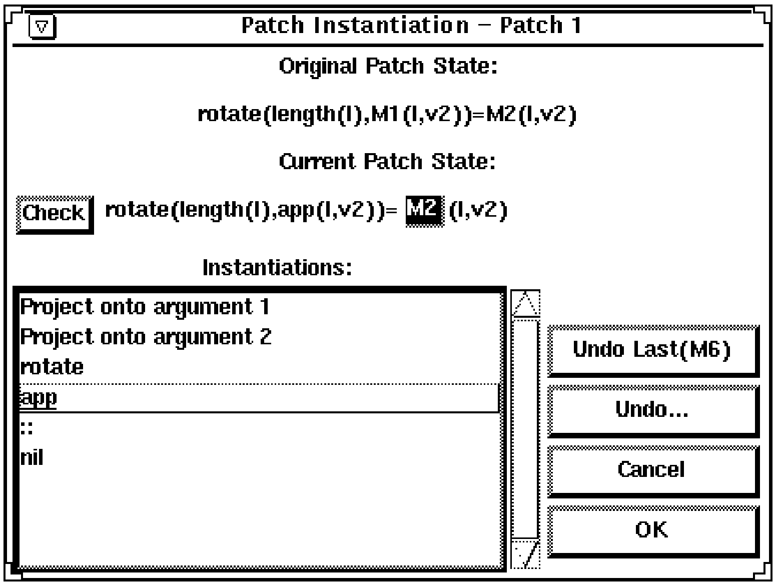


Fig. 5. Explanations - What will the Critic do?



The patch to be customized is shown at the top of the interface. The patch is also shown beneath, with meta-variables rendered as buttons. On clicking a meta-variable button, type inference is used to determine the allowable instantiations for that meta-variable based upon the function definitions currently loaded and the types of the variables occurring in the patch. Selecting one of these performs the instantiation. In this example the user is customizing the rotate-length patch, i.e. the user has completed the instantiation for the first meta-variable and is about to instantiate the second.

Fig. 6. Instantiating Meta-variables

in figure 6 where the partial instantiation of schema (11) is shown:

$$rotate(length(T), app(T, L)) = \mathcal{M}_2(T, L)$$

Note that the completed instantiation corresponds to (2).

5.4. Applying Proof Patches

In terms of applying proof patches a user has three options. Firstly, they can select the default patch suggested by the critic. Secondly, they can select a patch of their choice, either uninstantiated or instantiated as described above. Thirdly, they can reject the suggested patches in favour of another critic associated with the failure of the current method.

5.5. User-Critic Collaboration

In the case of the rotate-length example, if the user provides the appropriate instantiations for the meta-variables within (11) then the lemma speculation critic will be invoked. This is illustrated in figure 7 where multiple patch schemas, in

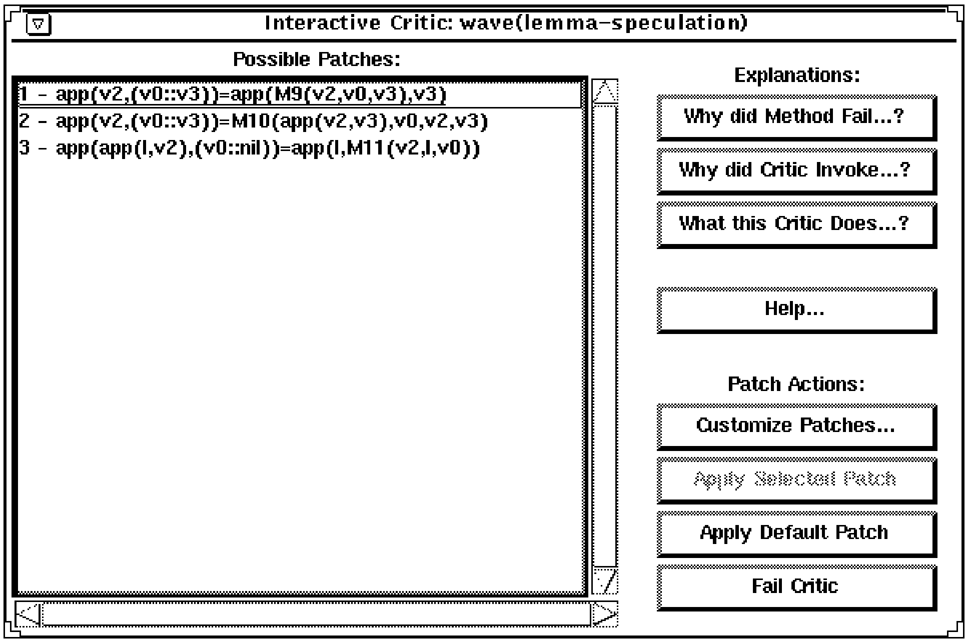


Fig. 7. Example Lemma Speculation Patch Schemas

this case schematic lemmata, are presented to the user. Here if the user provides the appropriate instantiation for the default patch schema then the subsequent middle-out proof planning automatically discovers the required lemmata, i.e. (13) and (12) (see appendix). This example, we believe, clearly shows the potential interactive proof critics have for realizing the goal of a truly collaborative theorem proving environment.

6. A User-Centred Evaluation

We described in the previous sections how interacting with proof critics offers the potential for a user to augment or even extend (using the terminology of [Wic84, MJR87]) the power of a proof critic. However, the utility of this user-critic collaboration only holds under the assumption that the user would know what information to provide when interacting with a proof critic, this information being in the form of selecting patches or supplying instantiations for meta-variables. In reality though, we do not have such "ideal" users. We must therefore assess user-critic interaction to see if it is reasonable to expect actual users to supply the required information. This consultation with users is also vital to assess whether the users themselves perceive interacting with proof critics as a natural and useful task to perform, making a positive contribution to the task of proof patching. If users do not view this interaction as being constructive then, irrespective of the advantages shown in §5, they will not be inclined to use it. In this section we describe how we addressed these issues by studying a small group of users interacting with proof critics. This evaluation is comprehensively described in [Jac98b].

6.1. Quantitative and Qualitative Evaluation

Evaluating human interaction with a computer system may yield two contrasting types of data, quantitative data and qualitative data, depending upon the approach taken and the issues that the evaluation is intended to address.

Quantitative data is typically objective data including task performance times, feature usage rates and error rates. Subjective data such as user opinions of interface features may also yield quantitative data such as usability scores or rankings. Quantitative data is often collected when comparisons are required between systems, for example to state that one system allows tasks to be performed faster than another, that one interface results in less errors for the user, or that users prefer one interface to another. Such measures also lend themselves to statistical justification of claims made regarding the usability of a system.

One limitation of using quantitative data in the analysis of human interaction with a computer system is that such data provides no information as to why one interface leads to faster or slower, or more accurate or inaccurate performance than another [SD87]. Such data requires subsequent elaboration and explanation in order to provide this information [YA87]. This is important information, as is stated in [WWD91]:

"... it is important that developers have some understanding of why*, so that they can direct their redesign efforts appropriately."*

Also of importance is the fact that quantitative information does not provide information relating to user experiences, for example the nature of a user's work, their intentions behind their work and their view of their work [Nix95, WBH87]. This is despite the fact that such an understanding is vital if the design of both usable and useful systems is to be realized.

This understanding may be obtained via the collection of qualitative data. In contrast to quantitative data, qualitative data is typically subjective in nature comprising of users' comments, views, complaints and descriptions of users' experiences, behaviour and activity.

While the gathering and analysis of qualitative data may be time-consuming and open to evaluator bias [YA87] [DFAB93] (page 395) there are real benefits to be gained. This is emphasized in [PRS+94] (page 694) where it is stated that qualitative analysis can:

"... provide new insights into ways in which technology is used enabling designers to produce more appropriate products".

The literature contains many examples of the utility of qualitative data in analyzing human computer interaction. For example, [MPB95] identified from qualitative data the key activities and patterns of activities involved in the performance of a collaborative writing task. This knowledge supports understanding of the user's task and lends itself towards the more effective design of support for such a task. In analyzing qualitative data from a study of human interaction with a public access system Buckley and Long [BL90] were able to identify important variables that seem to affect the usability of the system. Through quantitative analysis this allows the effects such variables have on usability to be determined.

Since participant groups of a size and composition which would yield statistically significant data were not available this cast doubts on the utility of using a quantitative approach to evaluating interactive proof critics. It was decided therefore to focus on a qualitative approach. Not only would this support the gathering of information relating to the user's perception of interacting with proof critics it would also yield information relating to the use of interactive proof critics, for example, the information sources which users require when using proof critics, or any patterns of behaviour which arise.

Another important motivation for this decision was that user opinions typically provide information on the acceptability of a system [SMS93]. Acceptability is a much more important issue than usability since a system or approach to interaction may be usable without necessarily being acceptable. For example, an interface to a system may be poorly designed. However users are willing to make do because the system itself provides an effective means of achieving some task. Conversely, a system may be neglected even if its interface is well designed since the system itself does not provide an effective means of achieving some task. Our evaluation therefore focuses on the question of the acceptability, rather than usability, of interactive proof critics as an approach to using CLAM.

6.2. Co-operative Evaluation

The style of the evaluation was modelled after the co-operative evaluation technique proposed by [MWHD93]. Co-operative evaluation lacks the formality of laboratory based experiments in that experimenter and user engage in a dialogue while evaluating the system. Furthermore, this evaluation takes place in a natural setting for the user. Our reasons for selecting co-operative evaluation are as follows:

- Evaluation sessions could be performed in locations, and with equipment, familiar to the participants. Indeed, participation of one participant was dependent upon this;
- The evaluator already had some experience of performing this kind of evaluation;
- This evaluation technique is well suited for the elicitation of qualitative data regarding user interaction with a computer system, i.e. user comments, complaints, suggestions and their behaviour.

6.3. Aims of the Evaluation

The main aim of the evaluation was to gather evidence as to whether users can be expected to provide the information needed to ensure that the benefits of interacting with proof critics are realized. This was in conjunction with gathering comments, complaints, suggestions and compliments with respect to this interaction. Analysis and classification of these utterances would then provide information about many issues relating to interacting with proof critics. The main issue of course would be whether interacting with proof critics is viewed by users as having positive benefits in assisting them in the process of proof patching.

6.4. Profile of the Participants

The Mathematical Reasoning Group of the Division of Informatics at the University of Edinburgh was approached for volunteers to participate in the evaluation.

The group conducts work in the areas of proof planning and proof critics and so are familiar with the concepts and terminology which may arise when interacting with a proof critic.

Six participants volunteered to take part in the evaluation. Three of these were classed as experts, one being an expert in the field of proof planning and the other two being major developers and users of automatic inductive theorem provers. The other three participants were classed as non-experts. Note that none of the participants were directly involved with the development or implementation of interactive proof critics.

6.5. Tasks

The participants were given as tasks ten failed proof attempts to patch. These were proofs where proof planning had become blocked and a proof critic had been invoked. The selection criteria for the conjectures on which the failed proof attempts were based is as follows:

Automatic success: Three conjectures were chosen which could be proved using automatic proof critics. In such cases interaction would only increase the time taken to find a proof. These conjectures were chosen to determine whether interacting with proof critics would lead to negative effects, e.g. the users' actions resulting in a failure to find a proof;

Automatic success with backtracking: Four additional conjectures were chosen which could be proved using automatic proof critics. However, unlike the first three, the proof patching associated with these conjectures results in backtracking if the user decides not to collaborate. So in these cases user interaction had the potential to reduce task time. Again, negative effects of interaction could also be looked for. One of the tasks in this category was the rotate-length conjecture, but with lemmas 12 and 13 loaded;

Automatic failure: Four conjectures were chosen such that they could not be proved using automatic critics. However, in all cases information provided to a critic by the user could potentially lead to a successful proof, i.e. meta-variable instantiation and patch selection.

The users were studied interacting with proof critics for each failed proof. A "think-aloud" protocol was adopted in which the users verbalized their thoughts, explained their actions and made any other relevant comments. These comments were recorded using a Dictaphone and interactions with the proof critics were also logged.

6.6. Results

The participants' comments made during their sessions were classified according to the level of interaction which they address. We adopted the interaction levels of [AGMT95, AGMT98]:

Logical level: This is concerned with interaction in terms of task level concepts, for example applying a proof patch;

Abstract interaction level: This is concerned with more interface-specific interaction, but still at a high level, for example selecting items from lists;

Concrete level: This is concerned with interaction at the level of keystrokes, mouse presses and perceptual elements of the display, or interface concerns at a relatively low level.

Comments were then analysed within these levels and the important issues arising at each level identified. Some of these issues had already been identified as having the potential to arise during the evaluation and others were revealed during the analysis of the results. This is a common form of analysis for such qualitative data (see for example [MPB95, WS95, How97]).

We shall present a summary only of the results relating to issues of logical level interaction since this relates to interacting with proof critics and its contribution to the task of proving theorems. Lower level issues are more directly related to the implementation of the interface and can be addressed via a suitable re-implementation phase. It should be stated, however, that such lower level issues may have an impact on the users' interaction at the logical level. The reader is referred to [Jac98b] for detailed analyses of the comments occurring at all three levels.

6.6.1. Allowing Users to Select Patches

Generally, the participants accepted the default patches proposed by the generalization, induction-revision and lemma-calculation critics. This was the expected behaviour for such critics.

For patches proposed by the lemma-speculation critic most participants were quick to notice when a poor default patch was proposed and to identify the required patch. For example, in a task where selection of the second patch was vital for the proof to succeed five of the six participants applied the second patch and ignored the first. Similarly, during another task an expert participant applied the desired patch identifying correctly that the first was inappropriate. On the same task another expert expressed dislike of the first patch but approval of the second though went on to apply the default patch. This could have been an error, due to interface design, as other participants in other tasks also applied the default patch when intending to apply a selected one.

The evidence from the evaluation suggests that lemma-speculation seems to be the most profitable critic for interaction, in terms of selecting the patch to apply. The users can actually make a difference by overriding the poorer default patches that are occasionally suggested by critics.

6.6.2. The Tendency to Customize Patches

There was much evidence to support the view that expert users tend towards providing instantiations for meta-variables (or some form of patch customization) when using interactive proof critics. One expert always provided complete instantiations for any meta-variables occurring in lemma-speculation and generalization patches. Another expert, due to no prior familiarization with interacting with proof critics, only realized the significance of the customization option at the end of their fourth task. However, from then on the participant customized patches whenever possible. The third expert based the need to supply instantiations on prior experience with interacting with proof critics. The participant stated:

"Generally my experience has been that I need to give the instantiations..."

Evidence of the tendency of experts to customize is also strengthened by comments from the other experts during lemma-calculation tasks, for example:

"...there's nothing I can do to alter that one is there..."

these comments hinting that these participants would have done any customizing or alteration if at all possible.

Why experts should tend towards customization is likely answered in part by experts' comments. Expert knowledge about the limitations of proof critics to find instantiations automatically facilitates behaviour on the part of the experts to support the critics at their weak points. Another reason could be the desire to play a more active part in a proof patching task and, as an expert stated, *"to give the instantiation when I know what [it] is"* with the motivation that if problems are encountered these will perhaps be encountered more quickly than if leaving a critic to do all the work.

As for non-experts, one participant, after being informed of how the early tasks presented (including rotate-length) would run into problems, stated:

"I'm getting an idea that it is better if you can see [an instantiation] just to input it yourself ... rather than to let it go ahead with an uninstantiated patch ..."

This may of course be a consequence of the order in which the tasks are presented to the user. That is, if confronted early on by time-consuming search on the part of the planner a user may become more pro-active in supplying schema instantiations in subsequent tasks. This is borne out by the fact that the same participant always provided instantiations during subsequent tasks. This was even in cases where the participant was not sure what to provide:

"We could try instantiating [one of the meta-terms], that would cut down the time."

Though the participant is not sure what instantiation to provide a partial instantiation is thought to at least assist the proof critic in cutting down the work it must do. Although another non-expert did not customize they did state:

"I'm thinking I ought to instantiate it to something but I can't see what it needs."

These results lend weight to the view that interacting with proof critics can have benefits in increasing the number of provable theorems and cutting down task times since the tendency for expert users, at least, is towards customization and supplying the information that proof critics need in problematic examples such as rotate-length.

6.6.3. Non-productive User-Critic Interaction

Comments relating to patches produced by the lemma-calculation and induction-revision critics indicate a general acceptance of the critics' suggestions by the participants. The participants made comments that the patches are *"reasonable"* or *"okay"*, though one participant expressed dislike at some of the, non-default, patches suggested by the induction-revision critic. The general perception as to the effectiveness of the patches suggested by these critics is underlined by the fact that all the participants accepted the critics' suggestions in such cases.

This gives rise to the question as to whether or not it is worth considering interaction with these critics given that the automatic versions are highly effective. One non-expert participant stated:

"Well, I'm beginning to wonder why it just doesn't go ahead and try the default patch and only come back to me when it's got a problem."

This suggests a more powerful critics architecture which allows a critic to try a few patches before requesting assistance from a user. This participant proceeded to suggest research into the effectiveness of proof critics in proposing patches, allowing a more precise determination as to which critics are worth interacting with and which are just as effective if their patches are applied without user intervention.

6.6.4. Straightforward Patches and User Behaviour

We have noted that there is a tendency, particularly among experts, to customize patches when possible. A problem that occurs however is that occasionally this behaviour does not arise. In such cases this is related to the perception of the simplicity of the patch by the user. This problem arose with lemma-speculation patches. For example, during a task in which a meta-variable instantiation was necessary for the proof to complete successfully the straightforward appearance of the uninstantiated patch caused an expert participant to remark:

"If I tell it to apply the default patch ... it should be able to find the instantiation."

This highlights the problem of how the lemma-speculation critic may propose credible patches whose application is hampered by limitations in middle-out reasoning. However, the deceptive simplicity of such patches may persuade the user to apply these without providing an instantiation in the belief that middle-out reasoning will succeed.

6.6.5. User Trust of Proof Critics

The implications of the comments made by the participants, alongside their behaviour, is that for generalization, lemma-calculation and induction-revision critics the first patch suggested is the most effective one for the problem and that user interaction is unnecessary with regards to patch selection. However interaction could still be useful in the case of the generalization critic by allowing users to provide instantiations for meta-variables.

This indicates that the participants do trust proof critics to propose patches that are applicable to the problem at hand and that therefore they view the proof critics as making a productive contribution to the proof patching task. One expert participant stated:

"Critics were better at suggesting patches than I had feared."

The participant also commented that there was a good match between the patches suggested and the problem at hand. In some cases where participants were perhaps unsure of what to apply they seemed quite happy to put their trust in the proof critics to apply a patch successfully. For example, non-experts stated:

"Well, I don't feel I can make much of an assessment between the possible patches so I'll trust the critics mechanism and apply the default."

"From an interactive point of view I don't mind just sending it away to the theorem prover and seeing if it works."

It should be remembered, however, that in some cases, as we have shown, such trust in proof critics can be misplaced.

6.6.6. *Increases in Domain Knowledge*

The only way to effectively assess whether interacting with proof critics led to an increase in domain knowledge for the users would be through the use of pre- and post-evaluation testing. However, some anecdotal evidence was elicited. Firstly, one non-expert participant stated that the interface had increased their understanding of how lemmas are used during a proof as well as where the lemma speculation and generalization proof patches are most appropriate. Secondly, one expert participant stated that using interactive proof critics had clarified their understanding of some of the heuristic choices involved within inductive proof.

Of course, such statements must be viewed with a certain scepticism. Nevertheless this may give a hint as to the utility of interacting with proof critics, particularly the explanation component, as part of a way to teach certain proof planning concepts, such as the differences between various critics, or their utility as a reminder for certain proof planning concepts.

6.6.7. *Participants View on Interacting with Proof Critics*

User comments reflected a generally favourable view amongst participants towards interactive proof critics, one non-expert making the comment that:

"... the critic is a really good thing in that even if it doesn't tell you how to fix the proof it gives you ideas [of how] to fix the proof."

An expert stated:

"...[the] critic made some interesting suggestions I would not normally consider..."

One expert participant did remark, however, that just because a proof critic is applicable does not mean that accepting its suggestions is the best thing to do:

"... just because the critic pops up it [does not mean that it is] a good idea to do that. I mean I might have something that's a better idea to do."

This implies that work should be done to elicit the other activities that users might want to perform at a patch point and to provide support for these.

6.7. Summary

The findings of the user-centred evaluation of interactive proof critics may be summarized as follows:

- real users can be expected to provide information required by proof critics to enable the potential benefits identified in §5 to occur;
- users, especially experts, are actively inclined to provide this information and view interacting with a proof critic as making a positive contribution towards the task of proof patching;
- users view the proof critics as making a valuable contribution to the proof patching task;
- interactive proof critics provide information that acts as a reminder of proof planning concepts to both experts and novices, hinting at the educational benefits of such interaction;
- it should be borne in mind that users may have ideas, that differ from those suggested by proof critics, as to what needs to be done to patch a proof and

support should be provided to allow the exploration of the implications of such ideas;
- some critics, for example lemma-speculation or generalization, support more productive interaction than others.

7. Implementation Issues

The proof planning ideas [Bun88] were first implemented within the CLAM proof planner [BvHHS90], a Prolog based system. Initially CLAM was used to guide the construction of tactics for the Oyster proof editor [BvHHS90], a minimal re-implementation of the NUPRL proof development environment [CAB+86]. Other object-level theorem provers, however, have also been targeted, most recently HOL [BSBG98]. In addition, various experimental versions of CLAM have been developed.

Our work builds upon three particular strands of this experimental development. Firstly, the automatic proof critics mechanism [Ire92, IB96b] provided the theorem proving basis for our approach and in turn exploited higher-order unification indirectly through λ-Prolog [MN88]. Secondly, in terms of graphical user interfaces (GUI) we began by developing xCLAM [Rei96] which was developed as a prototype teaching tool. xCLAM was implemented in the scripting language Tcl/Tk [Ous94] as it provided an effective basis for rapid-prototyping and also seemed to be emerging as a standard. An additional perceived advantage of Tcl/Tk was the ease with which it enabled students to experiment with the interface. Thirdly, while xCLAM formed the basis for the first version of the interactive proof critics interface [Jac96] the second version, which is illustrated within this paper, was re-implemented within the XBarnacle system [LCSV96, LD97, Low97], another GUI to CLAM. The switch to XBarnacle enabled us to portray wave front annotations using shading more effectively.

8. Related Work

Interactive proof critics represent an approach to building semi-automatic theorem provers. An alternative approach is demonstrated by the original XBarnacle [LD97] interface to CLAM. XBarnacle allows users to monitor the progress of CLAM and to step in and direct CLAM if the user considers that an unproductive path in the search for a proof is being explored. The user may control current proof steps being performed by CLAM and provide guidance as to how earlier steps may be revised. Evaluations as to the effectiveness of such an approach to interacting with XBarnacle have been performed [Low97, Jac97] and the initial results are favourable. The interactive proof critics version of XBarnacle differs from the original XBarnacle in that the critics, and not the user, decide when a particular path within the search space has become unproductive. In addition, the interactive proof critics version also supports the proof patching capability.

The importance we place upon the analysis of failed proof attempts is also reflected in the work of Matt Kaufmann [Kau92] where he provides a mechanism to assist the navigation of proof scripts generated by NQTHM [BM88]. This work is based upon the notion of a "checkpoint", essentially a tag which can be used to identify points within a particular proof attempt where the prover may have made an unproductive heuristic choice, e.g. a generalization leading to a non-theorem. These checkpoints are used in conjunction with *emacs* to enable a user to navigate more effectively through a failed proof attempt. While the mechanism provides a useful navigation facility, it relies upon the user to identify the root cause of the proof failure and provide the proof patch.

9. Future Work

Using the terminology of expert-support critics from [FLMM91b, FLMM91a] the approach to interacting with proof critics described in this paper is termed *active*, e.g. the proof critics decide when to be invoked. In contrast, a *passive* approach is also possible where the user decides when a critic should be invoked. Some of the participants in the evaluation expressed interest in this alternative. A subsequent study of the passive critics approach has been undertaken [Jac98a].

Having demonstrated that interacting with proof critics is viewed positively by users one important direction for further work is to perform a complementary evaluation, using a much larger group of participants, focusing on quantitative aspects of user performance This would allow more information to be gathered with regard to user behaviour when interacting with proof critics, e.g. task performance times and number of theorems proved. This in turn would enable us to develop statistical justification for our claims regarding the utility of interacting with proof critics.

As described in §7, the proof planning ideas have led to a number of experimental implementations. In order to integrate the best aspects of these implementations a new version of CLAM, called λCLAM [RSG98], is under development. λCLAM is implemented directly in λ-Prolog [MN88] which provides better support for middle-out reasoning. Work is currently under way to implement proof critics within λCLAM and also to provide an XBarnacle front-end. This gives rise to the possibility of interactive proof critics being supported by λCLAM in the future.

10. Conclusion

Inductive theorem provers rely upon the skill of the user in being able to exploit the insights which are typically embedded within failed proof attempts. Building upon this observation we have developed interactive proof critics which promote a collaborative approach to the activity of proof patching.

The design and evaluation of our interactive proof critics draws upon the experience of the HCI community. Our initial evaluation has been very positive and as far as we know the kind of interactive capability which we have developed is supported by no other semi-automatic theorem prover.

We have focused upon inductive theorem proving and the rippling proof plan in particular. However, interactive proof critics are relevant to any theorem proving application where proof planning is appropriate.

We see the potential beneficiaries of our work as primarily researchers working in the area of theorem proving as well as computer science students learning proof by mathematical induction. However, in the longer term we strongly believe that the basic ideas underlying interactive proof critics will be of benefit to mainstream formal methods practitioners.

Acknowledgements

The work reported in this paper is supported financially by a number of sources: EPSRC grants GR/J80702 and GR/L11724; the Department of Computing & Electrical Engineering, Heriot-Watt University; the Student Awards Agency for Scotland Postgraduate Student Awards Scheme; and the Napier University Research Scholarship Scheme. In developing our ideas we are indebted to David Benyon, Richard Boulton, Alan Bundy, Andrew Cook, Alison Crerar, Jeremy Gow, Ian Green, Helen Lowe, Rod McCall, Julian Richardson, Alan Smaill, Toby Walsh, Simon Wilkinson.

Appendix: Equational Definitions & Lemmata

$$
\begin{aligned}
length(nil) &= 0 \\
\forall X : \tau.\forall Y : list(\tau).\ length(X :: Y) &= s(length(Y)) \\
\forall Z : list(\tau).\ app(nil, Z) &= Z \\
\forall X : \tau.\forall Y, Z : list(\tau).\ app(X :: Y, Z) &= X :: app(Y, Z) \\
rotate(0, nil) &= nil \\
\forall X : nat.\ rotate(s(X), nil) &= nil \\
\forall Y : \tau.\forall Z : list(\tau).\ rotate(0, Y :: Z) &= Y :: Z \\
\forall X : nat.\forall Y : \tau.\forall Z : list(\tau).\ rotate(s(X), Y :: Z) &= rotate(X, app(Z, (Y :: nil))) \\
\forall X, Y, Z : list(\tau).\ app(app(X, Y), Z) &= app(X, app(Y, Z)) \quad (12) \\
\forall Y : \tau.\forall X, Z : list(\tau).\ app(X, Y :: Z) &= app(app(X, Y :: nil), Z) \quad (13)
\end{aligned}
$$

References

[AF97] S. Agerholm and J. Frost. An Isabelle-based Theorem Prover for VDM-SL. In E. L. Gunter and A. Felty, editors, *Proceedings of the 10th International Conference on Theorem Pro ving in Higher Order Logics (TPHOLs'97)*, volume 1275 of *Lecture Notes in Computer Science*, pages 1–16, Murray Hill, NJ, USA, August 1997. Springer.

[AGMT95] J.S. Aitken, P. Gray, T. Melham, and M. Thomas. Interactive proof discovery - an empirical study of hol users. In P. Gray, editor, *Proceedings of the First International Workshop on User Interface Design for Theorem Provers*. Department of Computer Science, University of Glasgow, July 1995.

[AGMT98] J.S. Aitken, P. Gray, T. Melham, and M. Thomas. Interactive theorem proving: An empirical study of user activity. *Journal of Symbolic Computation*, 25(2):263–284, February 1998.

[BL90] P. Buckley and J. Long. Using videotex for shopping - a qualitative analysis. *Behaviour and Information Technology*, 9(1):47–62, 1990.

[BM79] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979. ACM monograph series.

[BM88] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*. Academic Press, 1988. Perspectives in Computing, Vol 23.

[BR93] A. Bouhoula and M. Rusinowitch. Automatic case analysis in proof by induction. In *Proceedings of the 13th IJCAI*. International Joint Conference on Artificial Intelligence, 1993.

[BSBG98] R. Boulton, K. Slind, A. Bundy, and M. Gordon. An interface between CLAM and HOL. In J. Grundy and M. Newey, editors, *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98)*, volume

1479 of *Lecture Notes in Computer Science*, pages 87–104, Canberra, Australia, September/October 1998. Springer.

[BSH90] Alan Bundy, A. Smaill, and J. Hesketh. Turning eureka steps into calculations in automatic program synthesis. In S. L.H. Clarke, editor, *Proceedings of UK IT 90*, pages 221–6, 1990. Also available from Edinburgh as DAI Research Paper 448.

[BSvH+93] Alan Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[Bun88] Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.

[BvHHS90] Alan Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[BW94] David Basin and Toby Walsh. A calculus for and termination of rippling. Technical report, MPI, 1994. To appear in special issue of the Journal of Automated Reasoning.

[CAB+86] R. L. Constable, S. F. Allen, H. M. Bromley, et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

[CW96] Edmund M. Clarke and Jeanette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, Dec 1996.

[DFAB93] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice-Hall, 1993.

[FLMM91a] G. Fischer, A.C. Lemke, T. Mastaglio, and A.I. Morch. Critics: An emerging approach to knowledge-based human-computer interaction. *International Journal of Man-Machine Studies*, 35(5):695–721, 1991.

[FLMM91b] G. Fischer, A.C. Lemke, T. Mastaglio, and A.I. Morch. The role of critiquing in co-operative problem solving. *ACM Transactions on Information Systems*, 9(2):123–151, 1991.

[GMW79] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.

[GSG91] S. Garland, J. Stephen, and John V. Guttag. *A Guide to LP, The Larch Prover*, November 1991.

[How97] S. Howard. Trade-off decision making in user interface design. *Behaviour and Information Technology*, 16(2):98–109, 1997.

[HS96] D. Hutter and C. Sengler. INKA: The Next Generation. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of CADE-13*. Springer Verlag, 1996. (LNAI vol. 1104).

[IB96a] 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.

[IB96b] 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.

[Ire92] 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.

[IS97] A. Ireland and J. Stark. On the Automatic Discovery of Loop Invariants. In *Proceedings of the Fourth NASA Langley Formal Methods Workshop – NASA Conference Publication 3356*, 1997. Also available from Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/97/1.

[Jac96] M. Jackson. HCI Techniques for Theorem Proving. Master's thesis, Department of Computing & Electrical Engineering, Heriot-Watt University, Edinburgh, 1996.

[Jac97] M. Jackson. The evaluation of a semi-automatic theorem prover (part ii). In Y. Bertot, editor, *Proceedings of the Third International Workshop on User In-*

terfaces for Theorem Provers, pages 59–66, September 1997.

[Jac98a] M. Jackson. An evaluation of a passive approach to critiquing proofs in xbarnacle 2.6, December 1998. Unpublished report, Department of Computing, Napier University, Edinburgh. Available on-line at `http://www.dcs.napier.ac.uk/~mikej/`.

[Jac98b] M. Jackson. An evaluation of interactive proof critics, May 1998. Unpublished report, Department of Computing, Napier University, Edinburgh. Available online at `http://www.dcs.napier.ac.uk/~mikej/eval/eval.html`.

[Kau90] M. Kaufmann. An instructive example for beginning users of the Boyer-Moore theorem-prover. Internal Note 185, Computational Logic Inc., Austin, 1990.

[Kau92] M. Kaufmann. An assistant for reading nqthm proof output, November 1992. Technical Report 85.

[KM97] M. Kaufmann and J. Moore. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, 1997.

[KZ95] D. Kapur and H Zhang. An Overview of Rewrite Rule Laboratory (RRL). *Journal of Computer and Mathematics with Applications*, 29(2):91–114, 1995.

[LCSV96] H. Lowe, A. Cumming, M. Smyth, and A. Varey. Lessons from experience: Making theorem provers more co-operative. In N. Merriam, editor, *Proceedings of the Second International Workshop on User Interfaces for Theorem Provers*, pages 67–74, July 1996.

[LD97] H. Lowe and D. Duncan. XBarnacle: Making theorem provers more accessible. In W. McCune, editor, *14th Conference on Automated Deduction*, pages 404–408. Springer-Verlag, July 1997.

[Low97] H. Lowe. The evaluation of a semi-automatic theorem prover. In Y. Bertot, editor, *Proceedings of the Third International Workshop on User Interfaces for Theorem Provers*, pages 50–58, September 1997.

[MJR87] T. Moody, M. Joost, and R. Rodman. Vigilance and its role in artificial intelligence technology: How smart is smart ? In G. Salvendy, S.L. Sauter, and J.J. Hurrell Jr, editors, *Social, Ergonomic and Stress Aspects of Work with Computers - Proceedings of the 2nd International Conference on Human Computer Interaction*, pages 263–270. Elsevier Science Publishing, September 1987.

[MN88] 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.

[MPB95] A. Mitchell, I. Posner, and R. Baecker. Learning to write together using groupware. In *Proceedings of the 1995 Conference on Human Factors and Computer Systems CHI'95*, pages 288–295. ACM Press, May 1995.

[MWHD93] A. Monk, P. Wright, J. Haber, and L. Davenport. *Improving Your Human-Computer Interface: A Practical Technique*. Prentice-Hall International, New York, 1993.

[Nix95] D. Nixon. Qualitative research methods in design and development. *Interactions*, 11(4):19–26, 1995.

[ORA96] ORA. Introduction to EVES: Eercises and Notes. In *Odyssey Research Associates, Ottawa Ontario, Canada*, 1996.

[OSR92] S. Owre, N. Shankar, and J. Rushby. Pvs: A prototype verification system. In Deepak Kapur, editor, *Proceedings of CADE-11*. Springer Verlag, 1992. (LNAI vol. 607).

[Ous94] J.K. Ousterhout. *Tcl and the TK Toolkit*. Addison-Wesley, 1994.

[PRS+94] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey. *Human Computer Interaction*. Addison-Wesley Publishing Company, 1994.

[Rei96] G. Reid. The representation of the rippling heuristic in proof planning using coloured annotations. In *Proceedings of the 2nd International Workshop on User Interfaces for Theorem Provers — University of York*, pages 91–98, 1996.

[RSG98] J. Richardson, A. Smaill, and I. Green. System description: proof planning in higher-order logic with λClam. In *15th Conference on Automated Deduction*. Springer-Verlag, 1998.

[SD87] M. Sweeney and A. Dillon. Methodologies employed in the psychological evaluation of hci. In H-J. Bollinger and B. Shackel, editors, *Proceedings of IFIP INTERACT'87*, pages 367–373. North-Holland, September 1987.

[SI98] J. Stark and A. Ireland. Invariant discovery via failed proof attempts. In P. Flener, editor, *Logic-Based Program Synthesis and Transformation*, number 1559 in LNCS, pages 271–288. Springer-Verlag, 1998. An earlier version is available from the Dept. of Computing and Electrical Engineering, Heriot-Watt University, Research Memo RM/98/2.

[SMS93] M. Sweeney, M. Maguire, and B. Shackel. Evaluating user-computer interaction: a framework. *International Journal of Man-Machine Studies*, 38(4):689–711, 1993.

[Sym95] D. Syme. A new interface for hol - ideas, issues and implementation. In E.T. Schubert, P.J. Windley, and J. Alves-Foss, editors, *Proceedings of the 8th International HOL Workshop on Higher Order Logic and its Applications*, volume 971 of *Lecture Notes in Computer Science*, pages 324–339. Springer-Verlag, September 1995.

[WBH87] J. Whiteside, J. Bennett, and K. Holtzblatt. Usability engineering: our experience and evolution. In M. Helander, editor, *Handbook of Human-Computer Interaction*, pages 791–817. North-Holland, 1987.

[Wic84] C.D. Wickens. *Engineering Psychology and Human Performance*. Merrill Publishers, Columbus, Ohio, 1984.

[WS95] S. Whittaker and H. Schwarz. Back to the future: Pen and paper technology supports complex group coordination. In *Proceedings of the 1995 Conference on Human Factors and Computer Systems CHI'95*, pages 495–502. ACM Press, May 1995.

[WWD91] A. Whitefield, F. Wilson, and J. Dowell. A framework for human factors evaluations. *Behaviour and Information Technology*, 10(1):65–80, 1991.

[YA87] N. Yamagishi and M. Azuma. Experiments in hci evaluation. In G. Salvendy, editor, *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, pages 167–176. Elsevier, August 1987.