Evaluation of a Semi-Automated Theorem Prover (Part I)

Helen Lowe
Department of Computing, Napier University, Edinburgh, EH14 1DJ
h.lowe@dcs.napier.ac.uk

Abstract

So little evaluation has been carried out on user interfaces to theorem provers that even the question of what should be evaluated appears open. We took a user/task related approach, where subjects in a pilot study were given three theorems to prove, which might or might not require intervention by the user, and logged some of their interactions. In this paper, we describe the facets of semi-automated theorem prover illustrated by these theorems, and how we might expect our system to aid users in making appropriate interventions in proof attempts.

1 Introduction

This paper discusses the use of semi-automated theorem provers, where the ideal situation faced by the user is one in which they do nothing other than enter the theorem, but the usual situation is that this is insufficient and they must intervene in some way, possibly frequently.

One of the main problems seen by users is in understanding the output of an automated system, and one of the main problems seen by system designers is how to allow intervention in an otherwise automated system. Our guiding aim has been to build a system which *allows but does not force* users to interact with an otherwise automated system. We are thus firmly at the automated end of the interactive-automated continuum of theorem proving.

The question of evaluating interfaces to theorem provers in terms of users and tasks is largely unexplored. Evaluation hitherto has been sketchy, and confined to existing user groups. We are particularly interested in evaluating our interface work for *potential* user groups, such as members of research groups interested in non-implementational aspects of theorem provers, and the formal methods community.

This paper describes a pilot study in evaluating a user interface to the CLAM theorem proving system (Bundy *et al.*, 1990), where five users with different backgrounds were given three theorems and set the task of guiding CLAM to a proof in each case. In this part, we focus on the choice of tasks for the evaluation.

2 Semi-automated theorem proving

The major problem in building a co-operative interface to an automated theorem prover is in identifying when and where to pass control to the user, and what information to communicate. On this depends the user's effectiveness in supplementing the abilities of the automaton to prove theorems which would not otherwise be provable by the system. We need

- an understanding of how the user thinks about the problem.
- a good way of displaying the progress of a proof.

We have built the XBarnacle system (Lowe & Duncan, 1997) to meet the challenge of incorporating interactive features in the automated theorem prover CLAM while preserving the advantages of automation. The main parts of the system are the library browser and the planner. Once a theorem has been loaded using the browser, it may be selected and the planner activated.

The interface is very simple to use; in any case it is implemented largely using Tcl/Tk so we hope it will prove quick and easy for us to adapt once proper user evaluation of the interface aspects of the system has been carried out.

The proof is displayed on a canvas, and in general is a gradually growing tree structure. Colour coding of nodes helps to reinforce which nodes are complete and which parts of the proof are still to be completed. There are three buttons associated with planning a proof –

- a *play* button, which commences planning, drawing a tree on the canvas as each node is expanded or solved;
- a *single-step* button, which plans one node (the current node) and then halts;
- a *stop* button, which is active only after the play button has been pressed, and will halt the proof if the user wishes to take stock or intervene.

In fact, the user may get various pieces of information without stopping the planner, although true multitasking is a bit limited at the moment. Actions that the user can take are to change the current node – by default the planner attacks nodes depth-first style, but the user can switch the focus if they prefer; redo nodes, first examining what methods, apart from the one that was chosen automatically by the planner, are applicable at a node; get explanations; and expand sub-plans.

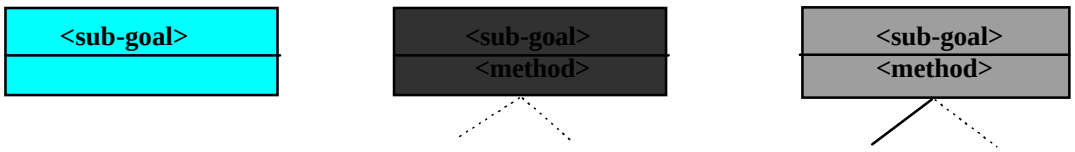
The guiding principle has been to *allow but not force* users to interact with the system, providing information in a form which enhances and assists them to intervene easily, appropriately, and effectively. Under this principle, the user is always in control, even in the role of passive observer, and can at any stage decide

1. whether to observe the proof – or not.
2. whether to intervene in the proof – or not.
3. what to see – and what not.

We explain these below.

2.1 Observing the proof

We wish to use automation to its full extent. When we have incorporated *critics* into our system as suggested in Jackson (1997), then this will also include the use of fully-automatic, sophisticated mechanisms for patching a proof, the equivalent of which in systems such as Nqthm (Boyer & Moore, 1988) would require user intervention. However, this assumption that the theorem prover will do most of the work requires the output to be in some eye-catching readily digestible form. We address this by the use of colour as *reinforcement* as follows. Observe the form of the three nodes shown in Figure 1.



open

closed
(may have descendants)

partial
(one or more descendants)

Figure 1: format of proof tree nodes

The first node in Figure 1 indicates an “open” node, i.e. one where a method has not been selected. A blue colour reinforces the idea that it is open. The second node is “complete”, i.e. a method has been selected which completes the branch of the proof. The dark grey colour reinforces this notion. The third node is “partial”, i.e. it has been expanded but one of more open sub-goals exist below it in the tree. We can see this without searching the rest of the tree because it is coloured light grey.

This extremely simple use of colour means that we can observe, even "out of the corner of the eye" the progress of a proof attempt such as that depicted in Figure 2. Here the nodes are shown minimally, i.e. with no text. If I am otherwise engaged on another task, I can glance at this from time to time and see how it is progressing. I can see at a glance the unexpanded node, and if I notice anything untoward in the pattern or complexity of the tree, then I can zoom in on it and examine it in more detail.

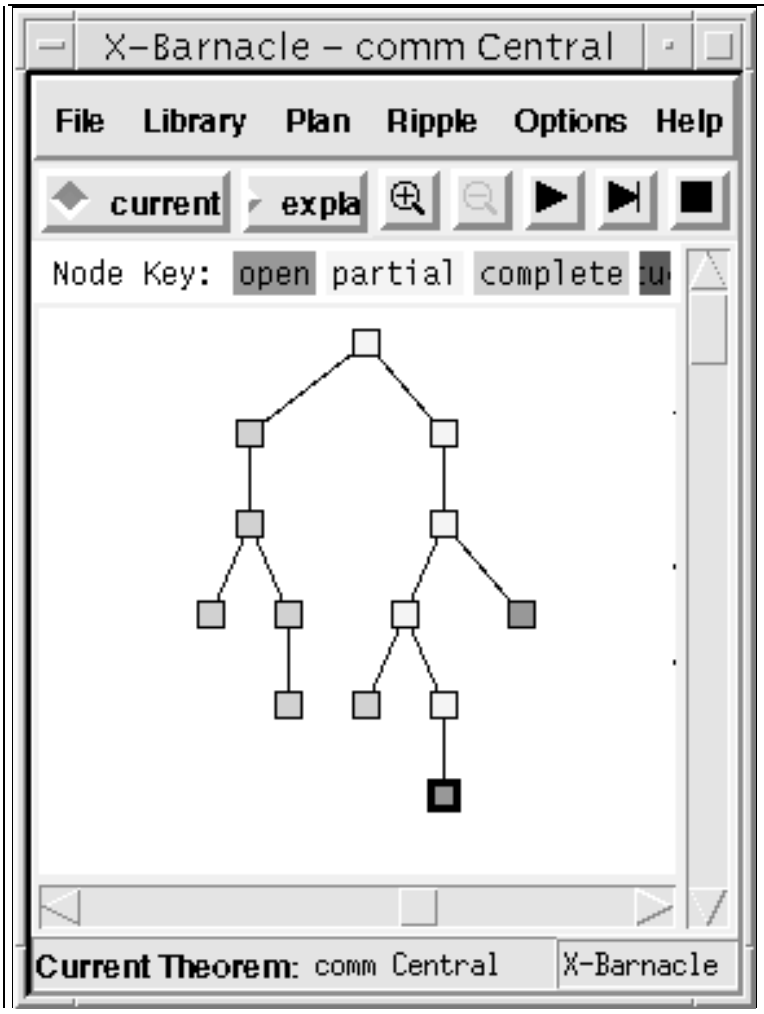


Figure 2: Skeleton proof tree

2.2 Intervention in proofs

The user is permitted to alter the course of the proof attempt by mouse clicks as appropriate. This implies that the user may mull over whether or how this may be necessary while the system is working. Sometimes a surprising (to the user) choice may be made. If the proof proceeds well, the user may be disinclined to interfere, but may wish to examine alternatives in the meantime. The *doubletimes1* example, below, gives an example of where CLAM finds a perhaps unexpected proof, but a perfectly valid one. Another observation is that a step may not be obviously wrong, but when the subsequent output is observed by the user, it may then, with the benefit of hindsight, be seen as such. The ability to be able to intervene at any point, and redo only those parts which need to be redone, is important in a human-computer system.

2.3 Levels of presentation

The third aspect is what to display by default to the user and how customizable this should be. In principle, the user should be able to invoke any level of detail at any point, but it is not enough to rely

on the user if a poor choice of defaults is made by the system designer. To get a useable system in practice we will need to evaluate the various options by examining users carrying out specific tasks and observing their behaviour patterns.

The system is customizable to a limited extent. Choices which can be made are:

- whether to view the base and step cases of the induction method.
- display of *rippling* (Bundy *et al.*, 1993) annotations.
- amount of detail at each node.

3 Test cases for evaluation

We plan a series of experiments, with the aim of determining the answers (or range of options) to a number of design decisions for both the interface and the underlying theorem proving system. Since there has been little work done in evaluating theorem provers in this way we decided to run a pilot experiment as a learning experience. For this pilot study we simply asked for volunteers, and did not press people in our target group to take part. Clearly this biased the sample (which cannot properly be called a sample), the bias being towards people who were less knowledgeable about CLAM (and who therefore had the most to gain from taking part and from encouraging our work).

It seems a reasonable hypothesis that different classes of user will require different functionality from a theorem proving system and that this will need to be factored out in future studies. As our work progresses, we hope to build up user profiles of subjects, including such factors as background, experience, and cognitive styles.

Jackson (1997) writes: “Barnacle … provides a ... natural task-domain oriented representation …”. This is the hypothesis to be tested, and we measure Barnacle’s functionality in terms of whether users can perform the tasks. This section describes how the tasks were chosen, whilst the paper by Jackson describes the subjects, the actual evaluation, and the results.

In our pilot study, the main question we addressed was whether users would intervene *appropriately*. Potential problems arise if:

1. Users intervene unnecessarily.
2. Users do not see the need to intervene when it is necessary.
3. Users see that they must intervene, but cannot find the right interaction to make.

3.1 Testing for unnecessary intervention

We told the subjects that they might – or might not – have to intervene in the proof, but as Jackson points out, the fact that the subjects are using a specifically interactive interface may predispose them to intervene, whereas in a more natural setting this may not be the case. We had already observed anecdotal evidence that people find machine proofs surprising in their length and complexity, even for such high level systems as CLAM, and wished to discover whether this would encourage people to make needless interventions. The proof tree for one of the three theorems we gave the subjects, *doubletimes1*, is shown in Figure 3. It contains three inductions. The proof begins by rewriting

$$\forall\, n \in \mathrm{N} \;\cdot\; double(n) = 2 \times n$$

to

$$\forall\, n \in \mathrm{N} \;\cdot\; double(n) = n + n$$

and then applying one step induction over the natural numbers. The step case:

$$double(n) = n + n \;\vdash\; double(s(n)) = s(n) + s(n)$$

becomes blocked at:

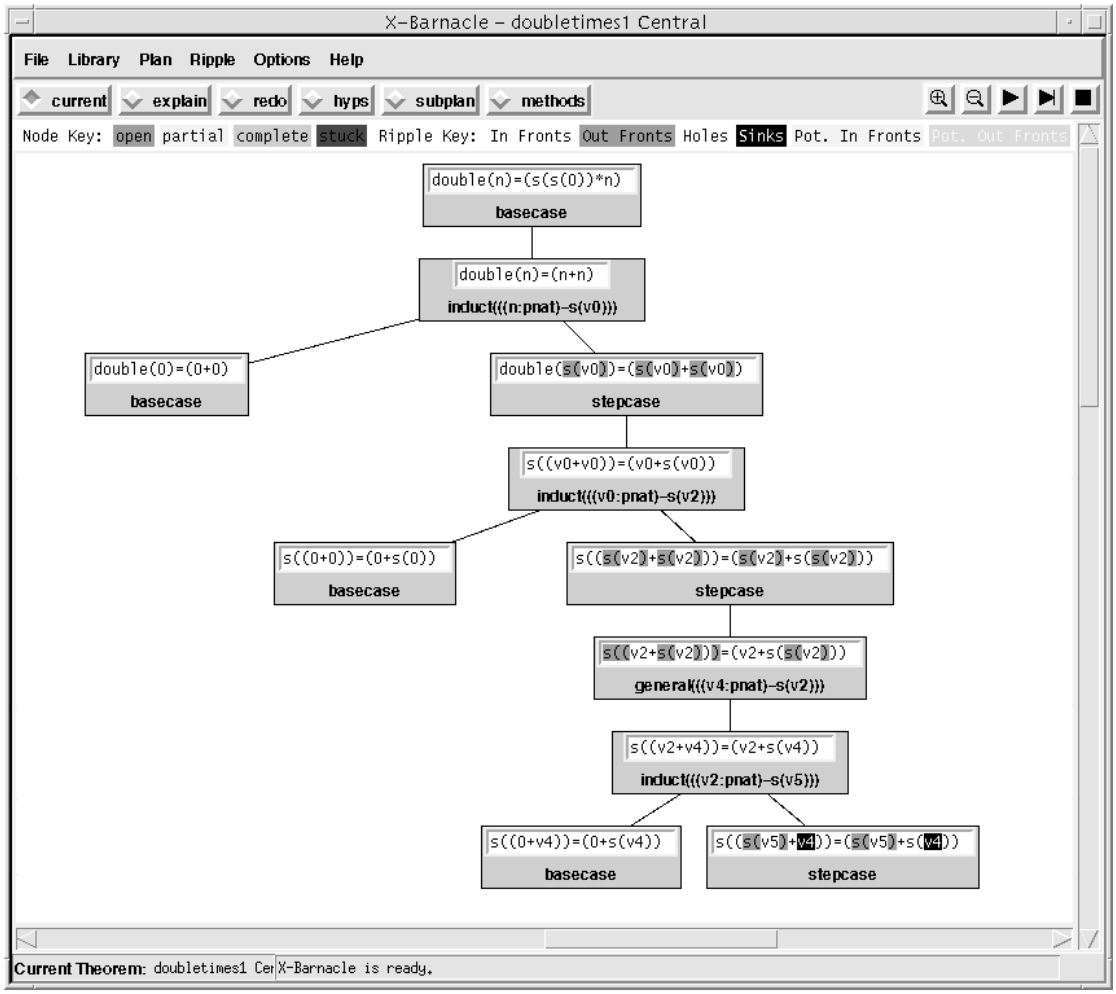


Figure 3: Long proof of *doubletimes1*

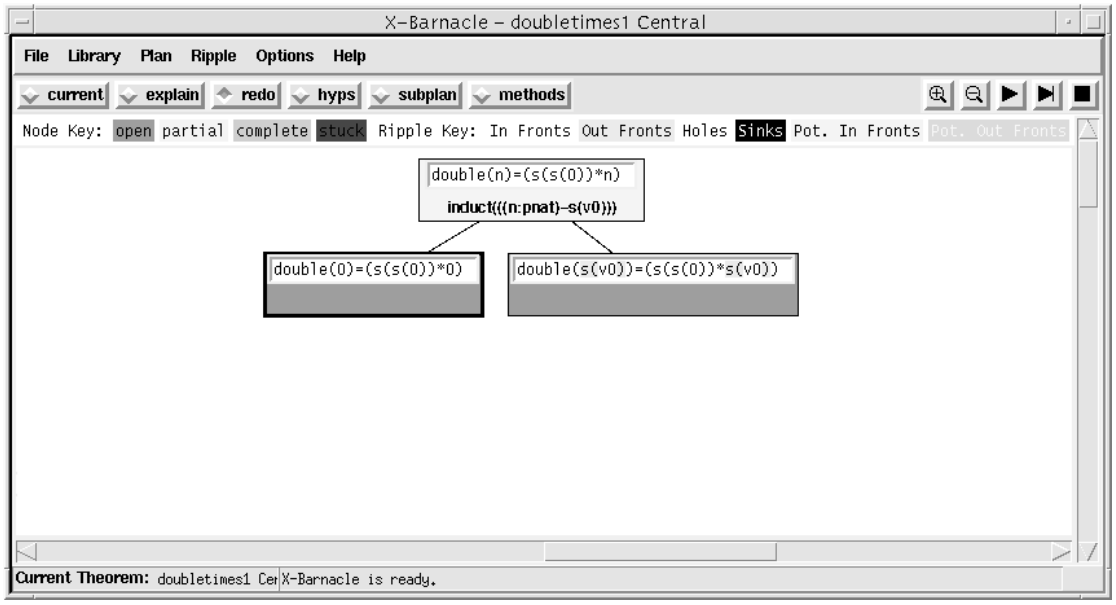


Figure 4: Short proof of *doubletimes1*

$$s(n + n) = n + s(n)$$

which, in the absence of any simple lemmas cannot be rewritten any further. The proof is found fairly quickly, however.

Note that if the first step in the proof is not the symbolic evaluation (unwisely named *basecase* in this version of CLAM) but induction, then a much shorter proof is obtained, as shown in Figure 4.

Whether this proof is spotted is probably dependent on the user's background; whether the user expects this proof or not, the first (longer) proof is often surprising; and whether users allow CLAM to continue could be expected to be a token of their recognition of, and faith in, the plan being followed by the automated system.

3.2 Correcting wrong steps

Here we concentrated on one weakness of CLAM, namely wrong choice of induction. There have been various heuristics used in determining which of possibly many rival induction schemas should be chosen, but none are perfect for all cases, as might be expected with a heuristic of this nature. The current situation is that a heuristic has been found which can distinguish between winning and losing induction schemas over a small number of such schemas, which are thus loaded by default: if none of these is appropriate, then a non-default schema must be loaded – either directly by the user, or by virtue of appearing in a dependency file, where users can store with each theorem the lemmas, schemas, etc. needed to prove it. It can this be seen that such so-called automated theorem proving is really semi-automatic, as it involves users, albeit off-line.

We have ideas, as yet untested on users, as to how induction schemas might be suggested and invoked in an explicitly semi-automated system. For now, given that we believe the heuristics need to be improved, we actually went backwards and stripped them out, using a version which simply orders the schemas lexiographically. Hence in a theorem of the form $\forall\, x \cdot \forall\, y \cdot \forall\, z \ldots$ induction on x, if applicable at all, will be preferred to y etc., and in each case one-step induction will be preferred to two-step induction over the naturals. By this ruse we knew, for the problem set we chose, that CLAM would choose the correct induction in the majority of cases! This would leave just a few cases for the user to spot and correct.

We wanted to see whether

1. The user could see when an obvious wrong induction was chosen.
2. The user could follow a complex proof, and intervene in the correct places.

To this end we chose two proofs, one where the wrong induction was chosen immediately, as the first step, and one involving a much more complex proof. The first proof was *evenplus*, where the theorem is

$$even(x) \wedge even(y) \rightarrow even(x + y)$$

where the correct induction is two-step induction on x, but where one-step induction is chosen, leading to the step case

$$even(s(x)) \wedge even(y) \rightarrow even(s(x) + y)$$

One of our hypotheses was that the unpromising look of the proof attempt (see Figure 5) as perhaps heralded by larger and larger amounts of annotation would alert the user.

Finally, we used the deceptively simple theorem

$$\forall\, x \cdot \forall\, y \cdot x \times y = y \times x$$

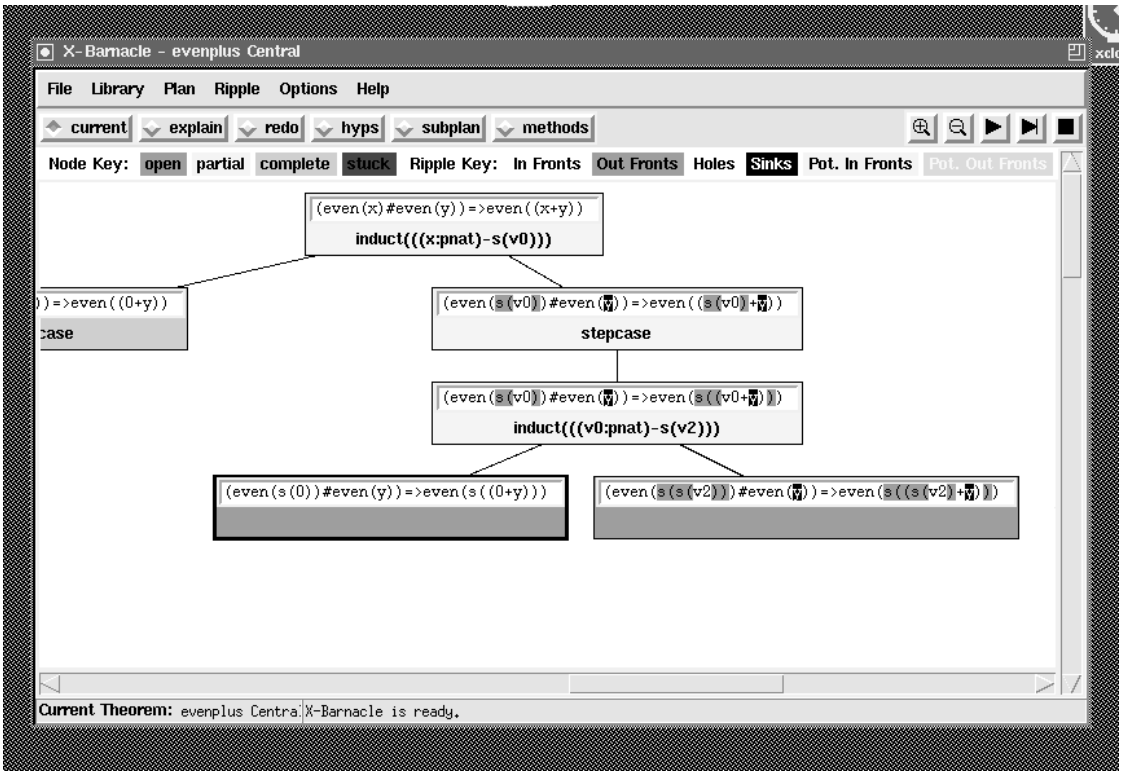


Figure 5: Aborted proof attempt of *evenplus*

This had both the property that, even when the proof attempt is going well, the proof appears strangely complex; and at least one requirement for user intervention. The user would need to be able to interpret the output, relate it to their picture of induction proof, and make the necessary intervention.

4 Summary and further work

Evaluating an interface for a theorem prover was a novel experience and much thought went into what should be evaluated. In the end, we decided that the function of the software was to prove theorems, and allow user intervention in a semi-automatic mode and that the effectiveness of performing this task should be what is eventually evaluated. The hypothesis to be tested was that the user would be able to intervene *as appropriate*, being able to distinguish between promising and unpromising steps by the automated system, and in the case of unpromising proof attempt, be able to find the wrong step and successfully redo the node. In a pilot study, we chose three theorems, one which did not require (but could usefully use) intervention, to see how the users might react to a strange, though correct, proof; one with an obvious wrong step to see when the users would spot this; and a third requiring concentration on the part of the user to find buried wrong moves.

The next step will be to set up experiments to evaluate specific task and user related functionality. The most fundamental of these are the use of lemmas in proofs, and the provision and choice of induction schema.

References

RS Boyer and J Moore. *A Computational Logic Handbook*. Academic Press, Perspectives in Computing, Vol. 23, 1988.

A Bundy, F van Harmelen, C Horn, and A Smaill (1990). The Oyster-Clam system. In ME Stickel (ed.), 10th Conference on Automated Deduction, pages 647-648. Springer-Verlag, Berlin. Lecture Notes in Artificial Intelligence No. 449.

A Bundy, A Stevens, F van Harmelen, A Ireland, and A, Smaill (1993). Rippling: a heuristic for guiding inductive proofs. *Artificial Intelligence*, 62, pages 185-253.

H Lowe and D Duncan. XBarnacle: Making theorem provers more accessible. To appear in the Proceedings of the 14th Conference on Automated Deduction, 1997.

M Jackson. A pilot study of an automated theorem prover interface. Submitted to UITP-97.