LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

The Interactive Proof Editor

An Experiment in Interactive Theorem

by

Brian Ritchie and Paul Taylor

LFCS Report Series

ECS-LFCS-88-61
(also published as CSR-273-88

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

July 1988

The Interactive Proof Editor
An Experiment in Interactive Theorem *

Brian Ritchie and Paul Taylor

1 Introduction

The past thirty years have seen a diversity of strategies applied to the problem of using computers as an effective vehicle for the performance of mathematical proofs. As the development of the discipline has reflected many trends, it might be useful to note some of the more important:

- The activity of working through examples has led to an increasing appreciation of the profundity of the basic problem. Often proofs presented by mathematicians appear simpler than is actually the case, possibly because absolutely formal proofs are seldom attempted.
- Today there is a deeper theoretical understanding of the activity of proof itself; computer based theorem proving should perhaps be seen as applied proof theory. An outstanding example of this is the Curry-Howard propositions as types isomorphism, which underlies systems such as AutoMath [3], Nuprl [13], Constructions [6] and Edinburgh Logical Framework [9].
- The capabilities of the equipment used by today's researchers exceeds by several orders that which was available in the sixties: compare the teletype (or batch mode) of the sixties with today's workstations with bitmap screens and mouse.

The wide range of approaches have resulted in many different styles of system, each of which have their adherents and each have supplied important paradigms with influences far beyond their immediate setting. One of todays tasks is to build proof systems integrating many of the important ideas so as to make the activity of proof accessible to a wider community.

Computer based proof assistants have a long history at Edinburgh (for instance the LCF [8] system), and recent years have seen the active development of interactive proof editors . This paper describes one of the initial results of this work, the Edinburgh IPE, a system built to explore the application of attribute grammars to proof (following the experience of Reps and Alpern with the Synthesiser Generator [15, 16]) and the possibility of performing "proof by clicking". It was implemented by Brian Ritchie, Tatsuya Hagino, John Cartmell and Claire Jones with the supervision of Rod Burstall and funding from SERC. Experiments with the IPE have highlighted the advantages of workstation technology for exploring proofs interactively, but also indicate a variety

*To appear in: Proceedings of Calgary Workshop on Hardware Verification 1988

of problems which have to be tackled in order to provide a powerful yet natural proof environment.

2 Some proofs with the IPE

The IPE runs on a SUN3 workstation under X-windows. We will give a few simple examples to indicate how "proof by clicking" has been realized within the IPE. As "proof by clicking" obviously makes extensive use of the SUN's mouse, throughout this paper we shall use **LM**, **MM** and **RM** to abbreviate references to the various mouse buttons (Left, Middle and Right). Upon start up the user is presented with an empty proof window (buffer) as in Figure 1.

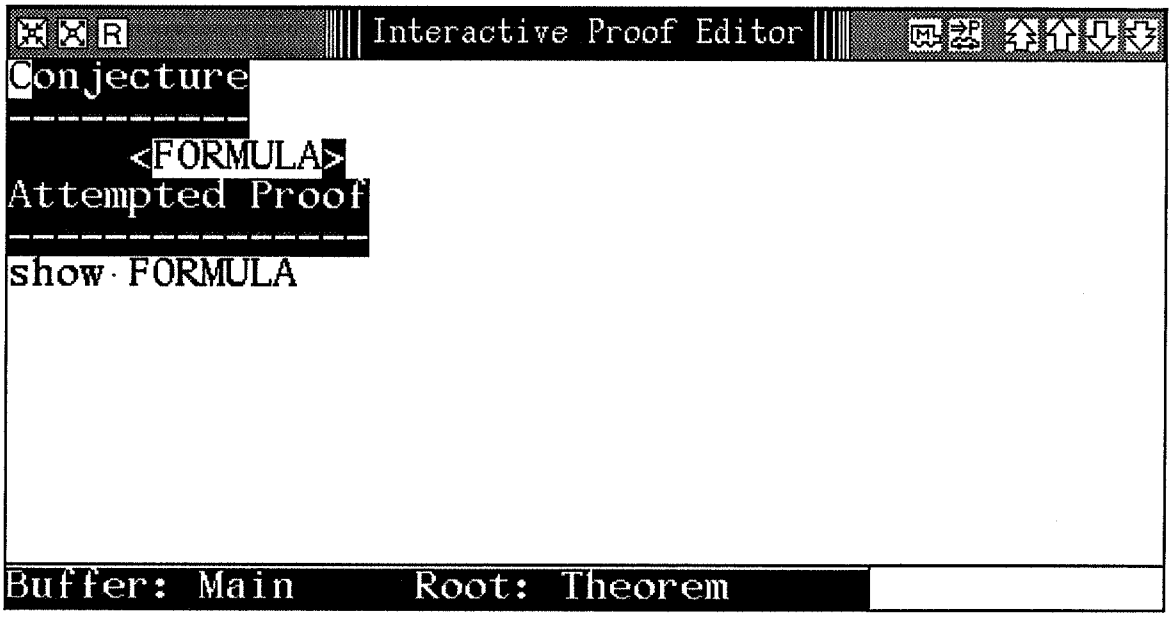

Figure 1

Notice that the display is broken into two areas, one for the conjectured formula and the other for the proof. Initially, as we have yet to conjecture anything, the conjecture is represented by a dummy displayed in angle brackets (`<FORMULA>`). Clicking MM at this location starts a simple editor which can be used to enter the formula you wish to prove.

For instance, type in the formula `A&B&C->(B|C)&A`: the character "|" is used to input the Or-operator, it is displayed with the conventional symbol. Upon exiting the editor the formula becomes the goal of the proof. Hence the first line of the proof is

```
show A&B&C->(B∨C)&A
```

The proof is developed by clicking MM on this first goal formula. The IPE will generate a subgoal from the goal by applying an appropriate rule (here Implies Introduction), which it selects by the top connective (an implication) of the formula. The second line and new goal of the proof will now be

```
show A&B&C entails (BvC)&A
```

note that the subformula `A&B&C` has become a premise which hopefully will entail `(BvC)&A` as a conclusion. Clicking MM at the premise `A&B&C` will cause IPE to generate a new goal, and further clicks will produce the complete proof shown in Figure 2.

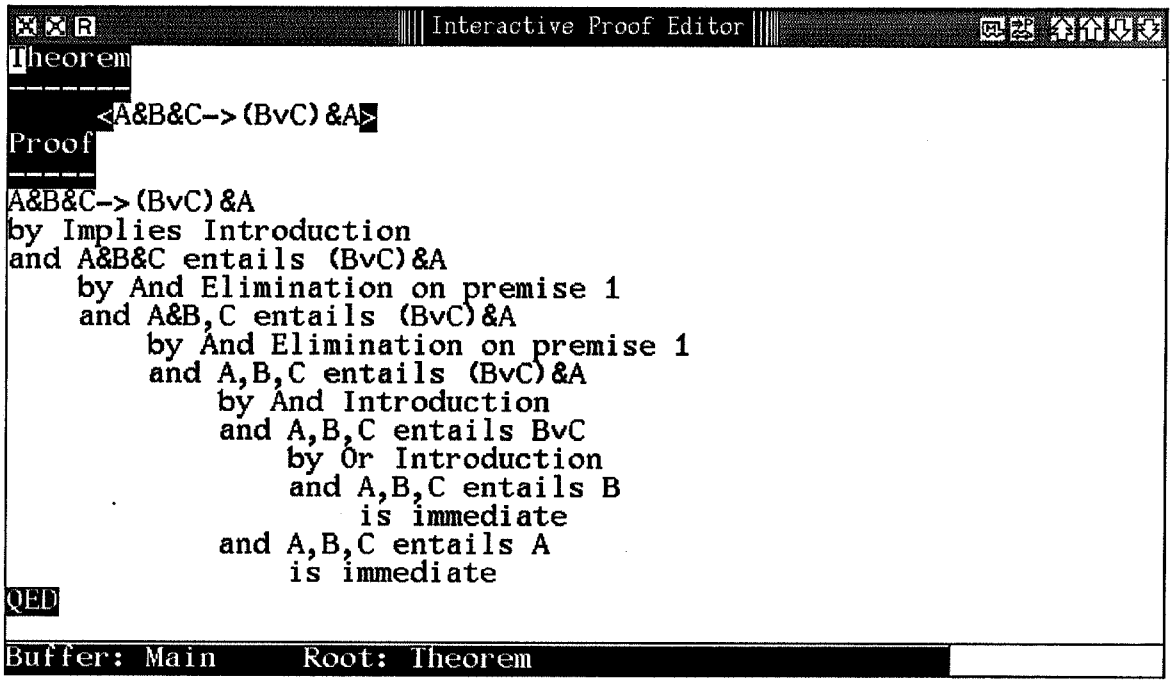

Figure 2

You should now see the style of proof which is implemented in the IPE. At any stage in the proof we have a number of outstanding goals to satisfy. A goal is a sequent, a collection of formulae which have been assumed (the premises) and a formula which hopefully they entail (the conclusion). Clicking MM at the conclusion or a premise will refine the goal with the appropriate introduction or elimination rule (Figure 6 gives a complete table of the built in rules).

It is possible to go back to the conjecture and edit it (notice it is displayed within angle brackets indicating editable data), for example change it to `A&B&C -> B&(C|A)`; the attribute evaluation in IPE will cause the proof to be reapplied with this new goal, much of it will succeed so only a few more clicks will be needed to complete the proof of the new conjecture. Similarly it is possible to explore alternative paths through the proof, by clicking upon the formulae in a different sequence: for instance we could first expand the operators in the conclusion.

2.1 Proofs, Quantifiers and Entering Terms

To achieve proofs of formulae involving quantifiers it is necessary for the user to supply some data to the IPE. Consider for instance

$$\forall x A(x) \& \exists a(A(a) \rightarrow B(a)) \rightarrow \exists z B(z)$$

which is input to IPE via the sequence

```
!x A(x) & ?a(A(a)->B(a)) -> ?z B(z)
```

By clicking in sequence on ->, & and ∃ the proof tree shown in Figure 3 will be obtained.

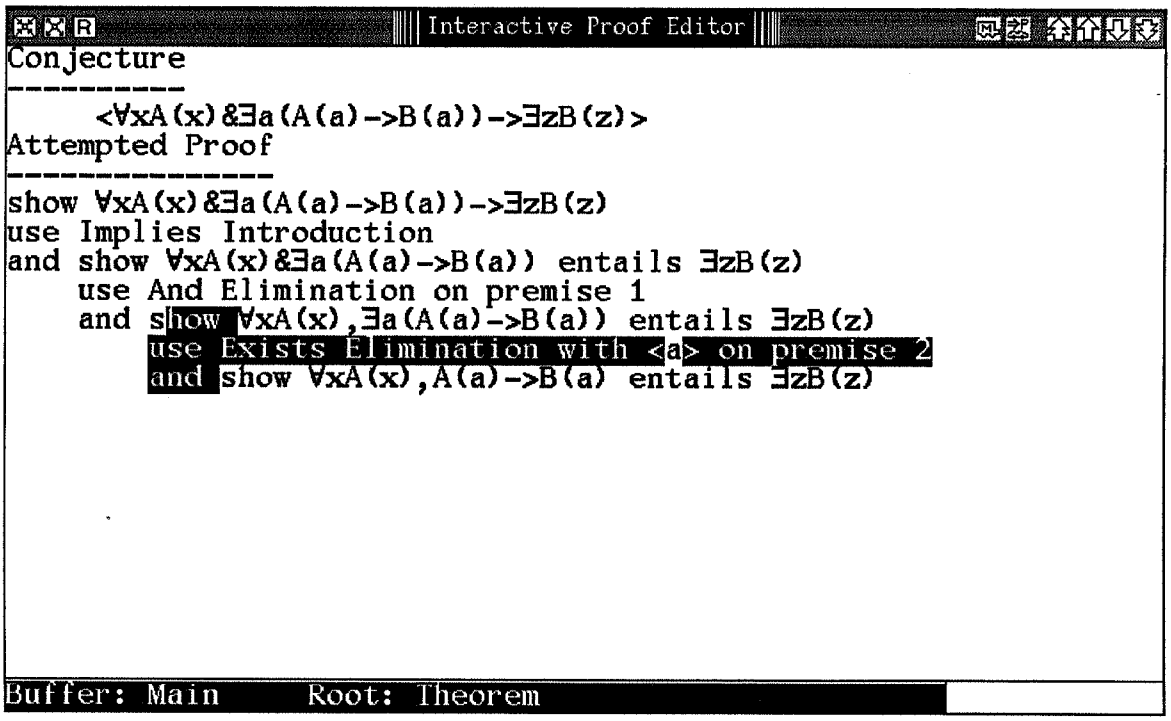

Figure 3

You should notice that the last rule, an Exist Elimination, has introduced a new free variable a; this should be thought of as a name for the object whose existence was asserted by the existential premise. The next step must exploit the fact that with any term x we may assume A(x), in particular x may be the free variable a. Clicking on the premise ∀x A(x) invokes All Elimination extending the proof tree as follows:

```
use All Elimination with <TERM_1> on premise 1
and show A(TERM_1),A(a)->B(a) entails ∃zB(z)
```

TERM_1 is enclosed by angle brackets, indicating data which may be edited; so by clicking MM at this point to start up the editor, and by typing in a, we make the appropriate instantiation.

Two further clicks and entering one more term will give the complete proof shown in Figure 4.

2.2 Using Facts, Rewriting and Proof by Induction

The proofs described in the previous sections have all been of tautologies in intuitionistic predicate logic - the logic built into the IPE. However it is possible to extend IPE with

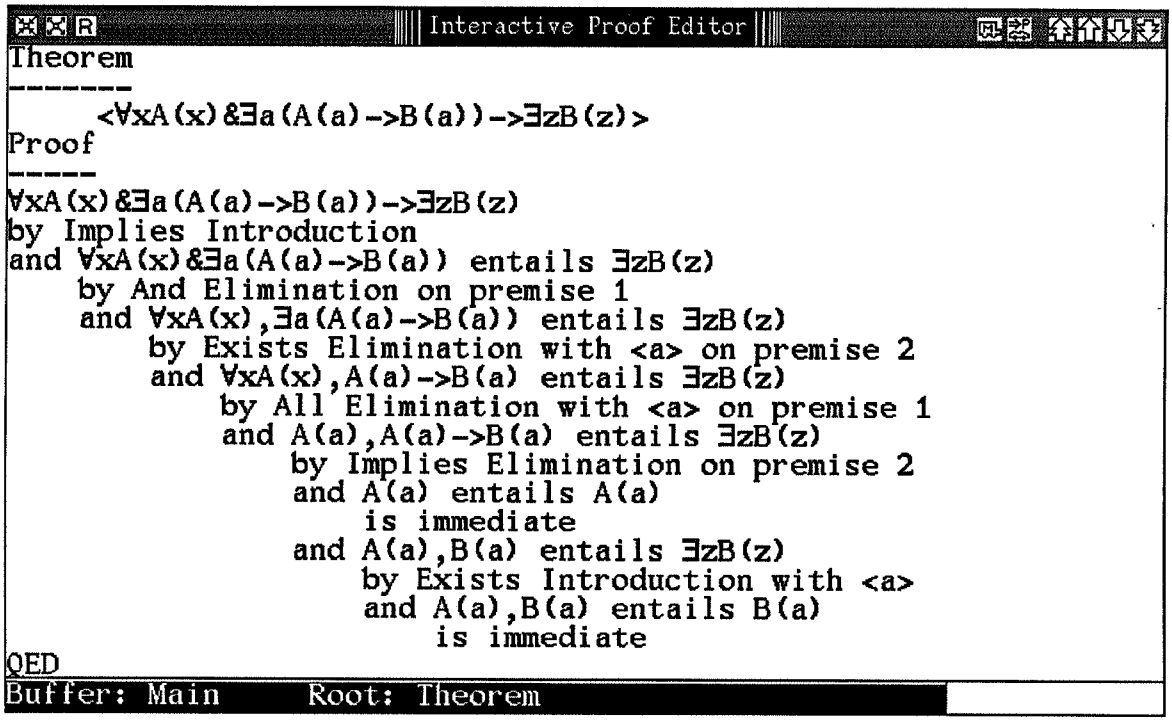

Figure 4

theories by supplying a collection of UNIX files. Usually the IPE will be set up with a collection of predefined theories amongst which should be List, a theory of lists.

We now describe how you would prove a new fact about lists with the IPE. Type in as a conjecture for IPE the following formula.

```
length(x@y) = length(x) + length(y)
```

The formula expresses the fact that when you append a list to the end of another list the length of the combined list is the sum of the lengths of the component lists (here `@` represents the appending operation). If you now attempt to prove this conjecture by clicking on the goal you will see IPE does nothing, apart from printing a rather obscure error message ("Immediate - Not so"); and indeed we can make no progress with this proof until the facts about lists have been loaded.

Loading a theory is one of the actions accessed via the menu. Pressing **RM** reveals the top level menu of a walk-through menu, this has a submenu of **Theory** operations, one of which can be used to load the theory of lists. Choosing this extends the IPE with the operations, axioms and rewrite rules significant for the theory of lists. Clicking **MM** upon the goal no longer results in the error message: in fact, as we are trying to prove an equality, IPE breaks our equality into two, a standard tactic for tackling proofs of general equalities. Here however this tactic is irrelevant; what is needed is a proof by induction over the structure of lists. Hence it is necessary to apply one of the facts about lists to our goal.

Again the facts of a theory are also accessed via the menu. To exploit a fact it is first necessary to indicate where we wish to apply the fact (in this case to the original goal). A formula can be selected for some action by clicking upon it with **LM**, selections

becoming highlighted in the display. Choosing the action **Choose from All Facts** from the **Facts** submenu, allows the goal to be seen as an instance of the schema for list induction. To achieve a proof by induction it is necessary to select a variable in the goal as the induction variable — thus you will see an editable term **<InductionVariable>**. Using the editor this can be replaced with x.
This leaves two goal formulae to prove:
the base case,

```
length(nil@y)=(length(nil)+length(y))
```

and the step case

```
∀L((length(L@y)=length(L)+length(y)->
    ∀a length(a:L@y)=length(a:L)+length(y))
```

The first of these is an equality, which follows directly by simplifying its left hand term using the definitions of the operations length and @. This action can be achieved by selecting the operation "Rewrite" from from the main menu for this goal. Having disposed of the first goal; the second can be tackled in a similar fashion once the logical constants have been stripped away. This can be done by repeated clicks with MM or more immediately by switching on autoprove (by clicking **LM** on the icon depicting a racing car). The outstanding goal now becomes

```
show length(L@y)=length(L)+length(y) entails
    length(a:L@y)=length(a:L)+length(y)
```

which becomes

```
show length(L@y)=length(L)+length(y) entails
   S(length(L@y))=S(length(L)+length(y))
```

upon rewriting the conclusion. We are almost finished; notice that the equality of the conclusion is a consequence of the premise if we use the premise to simplify it. This we do by clicking upon this premise with MM. Figure 5 shows the complete proof.

Figure 5

3 The Features of the IPE

These three proofs above illustrate the main characteristics of IPE, we will now discuss the features of the IPE more systematically. This account is mainly intended to provide a user's viewpoint of the system, so the details of the internals of the implementation will only be touched upon briefly. An extended account of these may be found in [17].

3.1 The Logic, Proofs and the Proof Display

Proofs in the IPE are generated in a top down fashion with goals being refined into subgoals. The logic is intuitionistic first order logic defined as a natural deduction system, each logical connective being defined by a pair of refinement tactics implementing an elimination rule and an introduction rule. Negation is defined by $\neg A$ is `A -> false`, and the negation elimination rule. Goals in the proof are represented by sequents, which can be thought of as conforming to the SML type definition

```
type sequent = {premises:formula list,conclusion:formula}
```

The introduction rules refine the formula in the conclusion whilst the elimination rules refine a formula from the premises. Figure 6 tabulates the rules implemented in IPE.

Structural Rules

Immediate	$\frac{}{\Gamma, A \textit{ entails } A}$
Weakening	$\frac{\Gamma, A \textit{ entails } C}{\Gamma \textit{ entails } C}$
Duplication	$\frac{\Gamma, A \textit{ entails } C}{\Gamma, A, A \textit{ entails } C}$

Rules For Logical Connectives

Connective	*Elimination*	*Introduction*
And &	$\frac{\Gamma, A\&B \textit{ entails } C}{\Gamma, A, B \textit{ entails } C}$	$\frac{\Gamma \textit{ entails } A\&B}{\Gamma \textit{ entails } A \quad \Gamma \textit{ entails } B}$
Or $\vee$ (|)	$\frac{\Gamma, A \vee B \textit{ entails } C}{\Gamma, A \textit{ entails } C \quad \Gamma, B \textit{ entails } C}$	$\frac{\Gamma \textit{ entails } A \vee B}{\Gamma \textit{ entails } A \textit{ or } \Gamma \textit{ entails } B}$
Implies ->	$\frac{\Gamma, A \rightarrow B \textit{ entails } C}{\Gamma \textit{ entails } A \quad \Gamma, B \textit{ entails } C}$	$\frac{\Gamma \textit{ entails } A \rightarrow B}{\Gamma, A \textit{ entails } B}$
Not $\neg$ (~)	$\frac{\Gamma, \neg A \textit{ entails } C}{\Gamma \textit{ entails } A}$	$\frac{\Gamma \textit{ entails } \neg A}{\Gamma, A \textit{ entails contradiction}}$
Forall $\forall$ (!)	$\frac{\Gamma, \forall x A(x) \textit{ entails } C}{\Gamma, A(t) \textit{ entails } C}$	$\frac{\Gamma \textit{ entails } \forall x A(x)}{\Gamma \textit{ entails } A(x')}$
Exists $\exists$ (?)	$\frac{\Gamma, \exists x A(x) \textit{ entails } C}{\Gamma, A(x') \textit{ entails } C}$	$\frac{\Gamma \textit{ entails } \exists x A(x)}{\Gamma \textit{ entails } A(t)}$

For the quantifier rules:-

x' indicates a new free variable name introduced by IPE

t indicates an arbitrary term which the user must input

Figure 6

Notice the rules are presented with the goal above the bar and the subgoals below the bar, which is consistent with the way proofs evolve in a refinement editor, but opposite to the usual convention for presenting deduction rules. The rules are a fairly direct translation of those found in Gentzen [7], and are complete for intuitionistic logic. The Duplicate rule is necessary to ensure completeness of the logic, because in IPE the application of an elimination rule to a premise removes that premise from the subgoal sequents. This makes the search that the immediate tactic has to perform more efficient and, more importantly, prevents the AutoProving from looping; but removing the duplication rule would render certain tautologies unprovable ($\neg\neg(A \vee \neg A)$ for instance).

As noted in the previous section proofs in IPE are extended by clicking the mouse upon a premise or conclusion. Internally this results in the construction of an attributed tree for an attribute grammar. An attribute grammar is a context free grammar where each syntactic category has an associated set of attributes, and with each production is associated an equation which defines how the attributes can be calculated. Attributes are classified as being either inherited (defined prior to the elaboration of a production of the grammar), or synthesised (computed from the attributes of the subtrees). Attribute grammars were originated by Knuth [10] to define programming language semantics.

In proof editors there is a correspondence between the proof rules for the logic and the productions of an attribute grammar, moreover the structure of the proof rule ultimately defines the equation which computes the attribute. The advantage of this technique in the implementation of proof editors is that the effect of modifying parameters can be propagated through a proof, enabling the user to see interactively the effect of changing the conjecture, modifying the instantiation of a variable or using an alternative axiom or lemma. Attribute grammars form a reasonable vehicle for implementing interactive proofs as algorithms exist to incrementally recalculate attributes (see [14]). Thus if a new branch is grafted into the proof tree only the minimum computation necessary to restore the consistency of the attributes will occur.

Internally the IPE grammar makes use of many attributes, and it is unnecessary for the user to have knowledge of any of the attributes. However it is possibly easier to understand operation of IPE if you are aware of certain of the attributes of the proof grammar:

- **Goal** is an inherited attribute of a proof - the goal sequent to which it is being applied.
- **Proven** is a synthesised attribute for a proof - a boolean which is true when each branch of the proof is resolved.
- **Appropriate** is a synthesised attribute for a proof - a boolean which is true when the expansion operation is consistent with the goal.
- **Self** is an inherited attribute for Formulae, Variables or Terms - essentially the string which defines the value of these items and which can be edited by the user.

The IPE display gives a complete presentation of the status of the current proof tree. The sequent associated with each node is shown, plus the rule used to expand it to its subgoals. The rules embodied in IPE will result in a binary and/or tree. Whilst an Or-node is unresolved both of its branches will be displayed, so that the alternative branch can be expanded if desired, but upon the successful conclusion of an Or-branch the other branch will disappear. Editable inputs to the proof (instantiations of terms and formulae, and the names of lemmas) are displayed within angle brackets and can be changed by invoking the text editor, by clicking MM at that point. For certain operations such as saving and restoring proofs or copying into and applying proof buffers, it is necessary that a node in the proof tree be selected. This can be done by clicking LM at the word "show" or the proof expansion rule associated with that node.

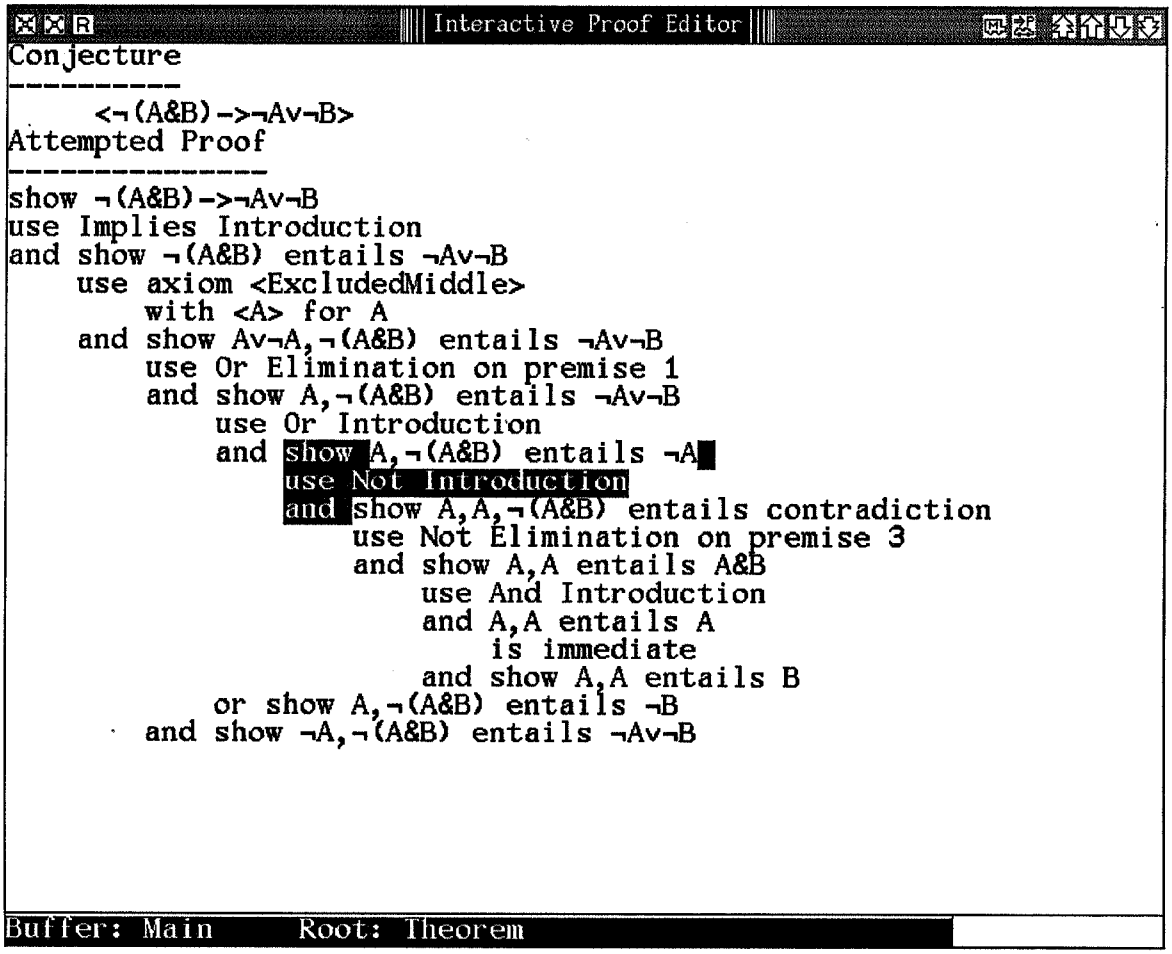

Figure 7

Figure 7 presents the display for an attempted proof of $\neg(A\&B) \rightarrow \neg A \vee \neg B$ in the context of the theory Classical. Note that an OR-branch has been completely explored whilst its companion is still unexpanded. From the highlighting we can see the attempted proof of the OR-branch is the currently selected point of the proof tree.

Of course this form of display will rapidly fill the screen. It is necessary to scroll the screen over such a display, and focus in and out to selected nodes of the proof tree. This capability is provided by the icons to right and left of the titlebar.

3.2 AutoMove and AutoProve

In its basic mode of operation the path taken by IPE through a proof is the sole responsibility of the user, however as one becomes more experienced with the system this mode of operation quickly becomes tedious. The **Automove** and **Autoprove** modes (which can be switched on and off by the use of icons on the titlebar) provide a facility for the system to assist in the elaboration of the proof. The Automove is purely a housekeeping device; it highlights the next unresolved branch in a depth first traversal of the proof tree. Whilst simple, this is very useful, in the context of the IPE's proof display, for directing the user's attention to nodes requiring work.

The Autoprove mode is more sophisticated. In keeping with the IPE's philosophy of leaving control of the proof solely in the hands of the user it is not an automatic theorem proving strategy. Rather the intention is that it should carry out sequences of proof expansions which are inevitable in some sense (i.e. no real choice is involved). As a corollary to this property Autoprove should not take the proof down a blind alley (i.e. a path which fails but which has a successful alternate path), for the choice of which point to backtrack to might present difficulties. A simple tactic, which has these properties, is to repeatedly expand a sequent whilst it is not immediately valid and there is just one valid expansion. However consideration of a proof of `A&B->B&A` shows that this *BoringTactic* is too simple. After one step we have

```
show A&B entails B&A
```

and further progress is blocked because of the tactic's inability to choose between using *AndElimination* and *AndIntroduction*. Autoprove should continue to elaborate the proof by those expansions where the order of application doesn't matter, because the rules commute in some sense. To implement this for IPE required the introduction of tactics and tacticals as in the LCF system, see [8, 17]. (This tactical subsystem is invisible and unavailable to the user.)

The autoprove tactic is then defined as

AlternateTac(*BoringTactic*,AlternateTac(*AutoIntros*,*AutoElim*))

where

fun AlternateTac (t1,t2) = REPEAT(ORELSE(THEN(t1,t2),t2))

REPEAT, THEN and ORELSE are essentially the standard tacticals defined in LCF [8], so AlternateTac will alternately swop between trying tactic t1 and tactic t2 until both fail. *AutoIntros* tries to expand the goal by an introduction rule (but not *OrIntro*), if it is not an immediate consequence. Similarly *AutoElim* will try to expand by an elimination rule (but only for And and Exist).

The resulting tactic is remarkably successful, often what appear to be quite subtle tautologies can be proved with just one or two clicks with Autoprove mode engaged. Also, routine sequences which strip away logical constants are well adapted to the use of Autoprove. However the tactic is not completely foolproof, as it may for example expand the wrong branch of an OR-node. As Autoprove rapidly changes the look of the display it can be quite confusing for beginners to use.

3.3 Proof Buffers and Reapplying Proofs

As noted previously the incremental re-evaluation of attributes makes it feasible to graft arbitrary chunks of tree into the proof tree. Thus there is no need to simply develop a proof with single basic steps; fragments of proofs should be reusable to prove an analogous subgoal of a proof. To implement this facility IPE has been provided with the concept of a buffer. Buffers can contain arbitrary fragments of an attributed proof tree - i.e. a "Theorem" - a formula plus an associated proof, or a "Proof" an actual fragment of a proof tree, or a "Formula" a simple formula. Operations with buffers

require consistency between the type of the currently selected node of the display and the type of the buffer. Of course a buffer in IPE can be completely independent of any other buffer, in the sense that the user can spawn a new Theorem buffer. Thus in the course of a proof you can switch your attention to another buffer for the proof of some relevant lemma. The important operations with buffers are

Change To Buffer Displays an alternative buffer, either pre-existing or new.

Copy Copy the selected subtree of the display into a buffer.

Apply Graft the contents of a buffer at the currently selected node of the display.

The IPE also has what is in effect an implicit buffer, invisible to the user, which retains the subtree excised by the last operation that overwrote the proof tree. The operation **Yank** applies this buffer at the currently selected node of the proof. This gives a one step undo facility, which can be quite useful as "proof by clicking" also makes unintended changes easy. Yanking itself will not affect the yank buffer.

As attributed proof trees tend to become rather large internal data structures, the IPE also has the notion of a proof skeleton. A skeleton is a sequence which records sufficient information regarding the proof steps to recreate the proof in the same context. Like buffers, skeletons reflect the type of node of the attributed tree selected at their creation. Skeletons can be written to and read from files, and so provide an ability to retain a proof state between sessions, or to maintain a kind of tactic library. However the skeletons probably should preserve more information about the context to provide a worthwhile reprove facility.

Additionally there is a facility to print out a conventional bottom up presentation of the proof, to provide documentation for completed proofs.

3.4 The Theory Database, Using Facts and Rewriting

Of course most proofs occur relative to the hypotheses of some theory, and IPE has the capacity for working with a theory database. Theories are not completely integrated into the system, partly because of certain shortcomings of the interface between UNIX and the version of ML used to implement IPE. An IPE theory is a UNIX directory containing a file, the ".environment" file, recording names of constants and operations specific to the theory and the names of subtheories upon which it depends, and further files each of which define an axiom or lemma of the theory. Axioms are created externally of the system, whilst lemmas are created by the system but both have the same textual form. A typical Axiom file is shown below:

```
axiom ListInduction is
    phi(nil) & (!L (phi(L) -> (!a phi(a:L)))) -> phi(InductionVariable)
generic terms InductionVariable
generic formulae phi(L)
```

Notice that certain formulae and terms are classified as generic, this implies that their value can be set by a substitution, which is either input interactively by the user or discovered via a matching algorithm. This enables axiom schemas to be defined and universally quantified expressions to be presented as schemas. Once a theory (and

recursively all its subtheories) has been loaded into the system its facts are available for use in proofs. Facts may be used to resolve some goal sequent: the conclusion or a premise has to be selected from the display of the sequent by clicking **LM**. Selecting the conclusion lets the fact be used in a backwards direction, whilst selecting a premise lets the fact be used in a forwards direction. The IPE expands a fact by use of Implies and And eliminations to transform it into a form which is usually most convenient for the progress of the proof. A variety of mechanisms have been supplied to aid the user in selecting the most appropriate fact.

- **Choose Fact by Name** Here the user must select a fact from a list of names of all the available facts. All the generic terms and formulae have to be filled in interactively.

- **Choose Matching Fact** Here the system will present a list of all the matches between the facts and the goal. The user can then select which of these should be used to extend the proof.

- **Choose from all Facts** Here each fact is interactively presented in turn to the the user, showing the form it takes after the best partial match between it and the formula selected from the sequent. Again the user can choose which fact should be used in the proof.

- **Match Chosen Fact** This is similar to choosing a fact by name, but now a list of facts is passed on to the matcher which can supply the substitution for the generic variables.

The order in which facts are searched reflects the order in which the theories have been loaded into the system. Each of these operations make use of a chooser window, such as shown in Figure 8

Quite frequently proofs involve equalities. To prove these at the lowest levels of the logic is quite tedious so the IPE has a number of adaptations for this case.

- If an equality in a goal doesn't directly match one of the facts of the current theory, the usual tactic is to use transitivity to decompose the goal into two equalities. This is the default action initiated by clicking upon a conclusion.

- A premise which is an equality is often used to simplify the conclusion. Hence clicking upon an equality premise will substitute any occurrence in the conclusion of the premise's left hand term by its right hand term.

- Often equalities in theories of data have a computational intent, they describe how terms should be simplified to a normal form. To take advantage of this observation the IPE has been extended with a very simple rewriting facility. Theorems which are equalities can be added to a list of rewrite rules. The menu action **Rewrite** will apply a simple leftmost outermost rewriting function, with all the loaded rules, to simplify the terms of an equality. It is up to the user to take care that the set of rewrite rules has nice termination properties.

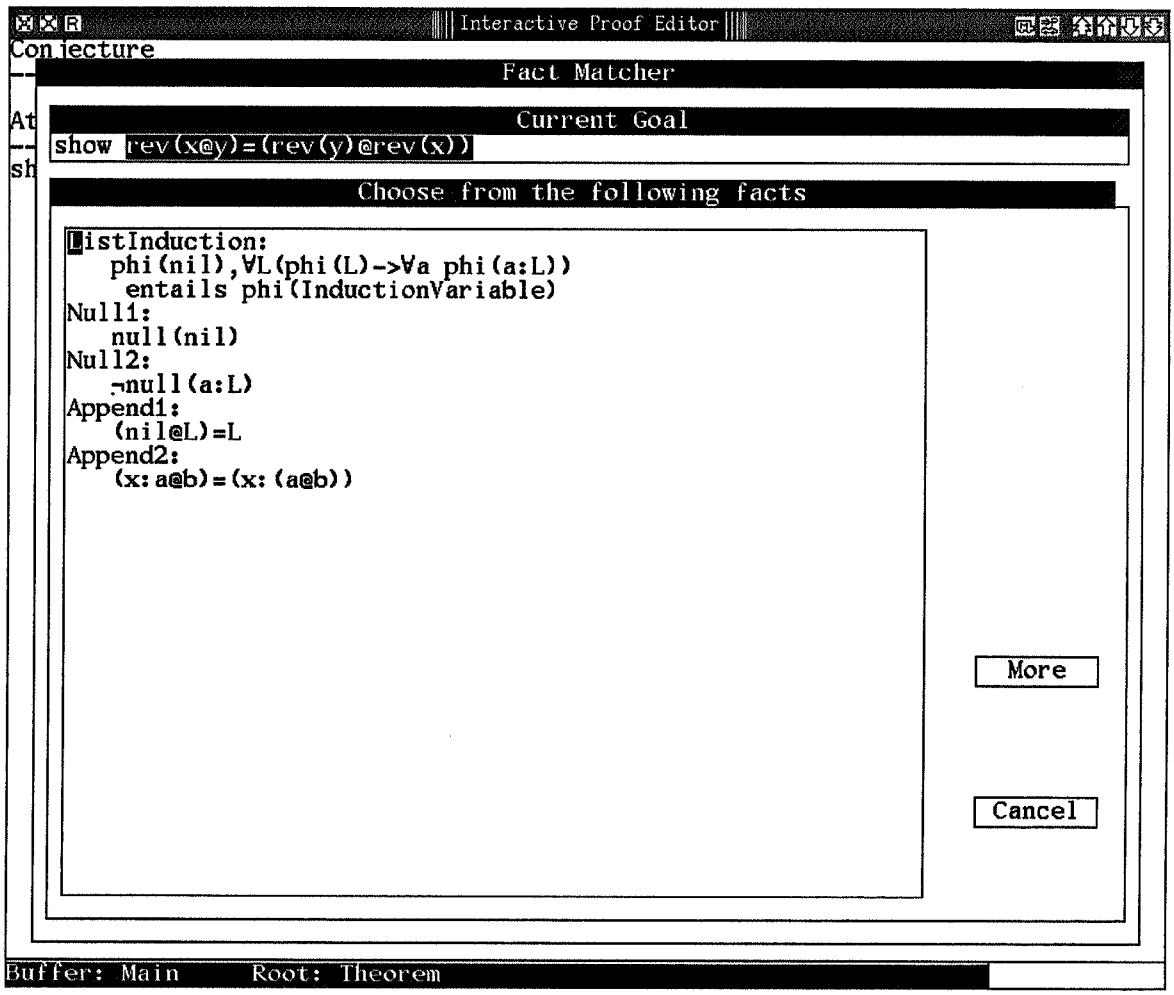

Figure 8: A Chooser

4 Some Reflections on Interactive Proof Development

4.1 Using IPE

The IPE was a first attempt in applying workstation and attribute grammar technology in the construction of interfaces to proof assistants. It has shown that "proof by clicking" provides a very immediate entry into the activity of proof. An understanding of the possibilities and mechanics of proof, can be developed incrementally through a graded sequence of examples. The ease with which proofs are constructed, and the lack of penalty involved in jumping about the search space, allows students to develop an understanding of the logical operations via experiment. Also presenting the more sophisticated operations of IPE via structured menus allows students to unfold as much or as little of the system as their experience requires. Compare this with the tactical style of proof embodied in system such as LCF, where the activity of proof is guarded by an intimidating array of operations with obscure technical titles, presupposing some understanding of logic, and the use of a new programming language ML.

Although IPE clearly has advantages as a teaching tool, more work is required to prove its worth in a more general setting. The most complex case studies attempted have been of a simple parser after Cohn [1] and a simple compiler for expressions [5]. This is in the main a reflection of the difficulties in formulating problems in the IPE's logic which lacks notions of set or type. However it is possible to see how IPE style proof editors would affect the proof process.

IPE lends itself to an exploratory style of proof. Thus one will often attempt a proof without a clear understanding of which lemmas will be necessary for its successful completion. However the introduction of a lemma or a new axiom, or a change to the conjecture will frequently be indicated in the course of the proof. The mutability of proof objects and the facility to stack proofs embodied in the IPE, becomes particularly useful in such situations. This is in marked contrast to the style of proof adopted and recommended by practitioners of LCF and Boyer-Moore theorem provers (see [12, 4]) where a plan of the key lemmas and concepts is required in advance of the actual proof. Thus a proof will frequently reveal an error or inconsistency or incompleteness in the formulation of a definition. This is perhaps a true reflection of the significance of the role of proof in formal semantics: it is the chief experimental tool available to reassure formalists of the validity of their formalisations. See for instance Avra Cohn's discussion of the verification of the Viper specification [2], or in a wider setting Lakatos's analysis of role of proof in mathematics [11]. It is tempting to conjecture that formal methods will only play a prominent role in information technology when accessible systems which support realistic proofs become generally available.

4.2 The Structure of Proofs

The IPE development identified various classes of features of general application in a proof development environment. It is interesting to speculate how these can be made more powerful and yet more consistent with one another. Indeed it would be a worthwhile exercise to make a more formal analysis of the proof process in the light of the IPE experience. The collection of attributes associated with a proof tree in IPE is essentially very simple: with the goal and input terms as inherited attributes and the status

(proved or unresolved) of the goal a synthesised attribute. There are other attributes concerned with how the proof is displayed on the screen but of no direct relevance to the proof. However certain enhancements suggest that a more sophisticated collection of attributes is required.

At present the instantiation of terms must be supplied by the user of IPE ; except in the case of application of a fact where matching can supply the instantiation of the generic terms of a ForAll Elimination. This means that the IPE can provide no help for the user in satisfying existential goals - or in specialising universals in the most general setting. Hence Autoprove will never completely resolve a conjecture involving quantifiers, and so for example user input is necessary to prove something as simple as

```
∀y A(y) -> ∀x A(x)
```

as the system does not test equivalence of formulae up to α-conversion. An obvious solution to this would be to extend the Immediacy test with the facility to unify a premise with a conclusion. This would imply the necessity for a further synthesised attribute of a proof recording the substitutions discovered by unification. However such an enhancement requires care, because the substitution discovered in a subtree might be relevant to the proof recorded in a parallel subtree: so the attribute must propagate both up and down the proof tree. There is a danger of defining a circular attribute, especially if the user is still to be allowed to interactively input instantiations for terms. There is further discussion of this problem in [17].

Elimination rules are applied at a particular premise of a goal sequent, so in IPE each elimination rule is defined not solely by its connective, but also in terms of the absolute position of the premise upon which it acts. This causes no problems whilst developing a proof, but if we wish to reapply a proof in a slightly different context it causes obstructions. Hence we see in the case of elimination rules the need for another inherited attribute identifying the position of the relevant premise, so the user can edit the proof to conform to the new situation. Alternatively the position of the premise might be a synthesised attribute whose value is discovered by investigating the structure of each of the premises. Of course the real significance in a hypothesis is in its information content - conveyed by its structure. In informal practise we often recognise this by attaching names to hypotheses and axioms in the course of a proof, would this be useful in this situation for the IPE? More precisely it is how the subformulae of a premise are exploited subsequently in a proof which is the real significance of a premise; it might be possible to synthesis a minimal pattern encoding this data, which the IPE could use when reapplying a proof.

The simple model of proof editing implemented in IPE assumes that proofs grow in a tree like manner. Actual proofs may be much more complex, for instance an extra step may have to be inserted into a proof, perhaps because the conjecture is edited to include an extra hypothesis. The effects of even such a simple change might propagate throughout the proof (the positions of all the hypotheses may be shifted one place). Frequently one will elaborate a sequence of steps in slightly the wrong order, and then discover a proof step is blocked because it should have occurred earlier. Although buffers and yanking provide a capability for dealing with this, sometimes a need is felt for higher level proof operations.

Further questions arise when one considers how theory structures should be inte-

grated into the system. Should there be attributes expressing the dependency of a theorem upon a theory? Should the maintenance of the consistency of the theory database be via the attribute system?

4.3 The Interface

For IPE the display is a vital component of the system, since the path through the proof is selected by clicking upon formulae. In order to backtrack, some alternative formula must be selected from a previous state of the proof. This feature has the unfortunate consequence that all of the information relevant to the proof has to be present within the display. As a result the screen rapidly becomes packed with redundant data. Additionally a proof is often dependent upon facts contained within a particular theory, and there is no convenient method for displaying this larger context concurrently with the local state of the proof. Thus devising an effective display for an interactive proof editor is an interesting challenge. At any point in the proof both local and global data are of interest: to choose the next step it is necessary to see all the local data in the context; whilst to backtrack in the search space it is necessary to know how the proof developed, and at what stage the premises were introduced into the context (i.e. what was the subgoal at that time). The display might also convey other information of interest, such as whether a premise has been exploited in a proof.

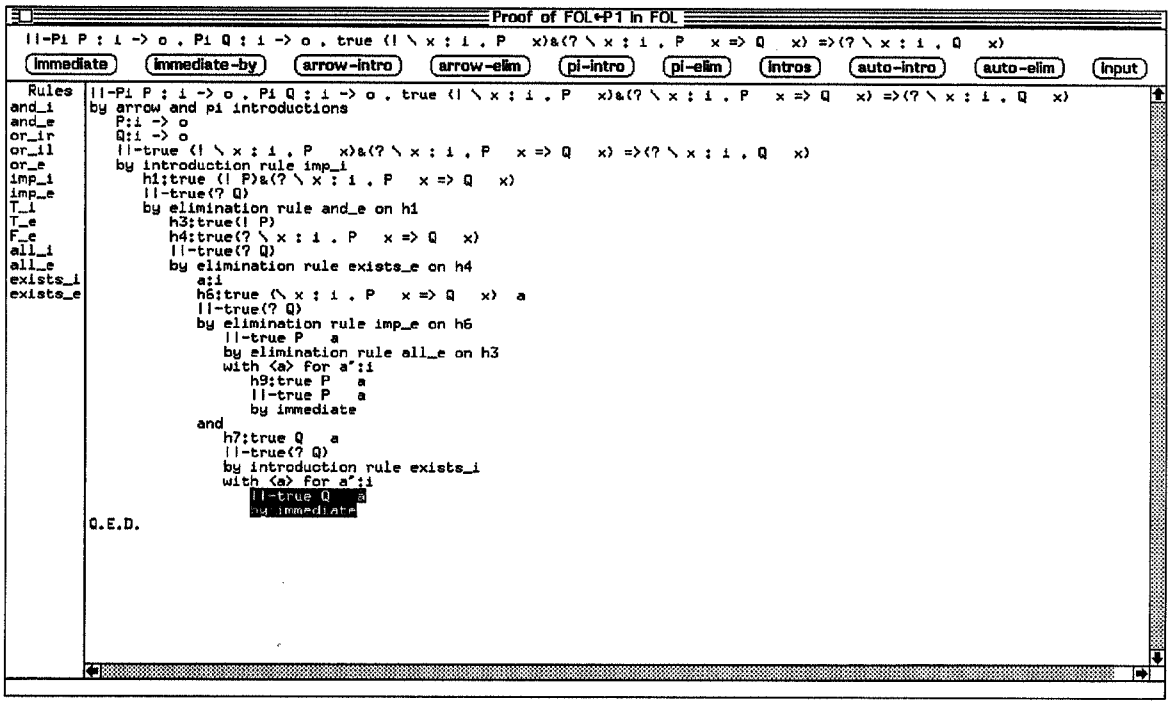

Figure 9

Figure 9 shows an alternative style of display (devised by Lin Huimin and Tatsuya Hagino for use with a prototype), which is somewhat more economic than the IPE's in presenting the proof data. However yet more efficient layouts which make better

use of the capabilities of the technology can be imagined e.g. different shadings or colour, multiple displays. We haven't considered the issue of pretty printing formulae in IPE, however how a formula is displayed can often be significant in deciding the next step of a proof, especially as formulae become large and complex. Similarly it is possible to imagine alternative schemes for associating the operations with the mouse buttons: the IPE could be driven using a mouse with just a single button and a variety of menus. These interface aspects may appear to have little theoretical relevance, but can significantly affect the way users understand and exploit the system.

4.4 The Role of Tactics

It has already been noted that IPE was built to explore an alternative to tactic driven proof systems, but we also pointed out that there is an underlying tactic level to the IPE. The questions naturally arise

> "Should this tactic level be made available to users of the IPE? "
> "If it is, in what way should it be made accessible? "

Initially the immediacy of proof by clicking, especially with the Autoprove feature, leads one to question the need for certain classes of syntactic tactics and hence possibly all tactics. Still the Autoprove and the Rewrite tactics, despite their simplicity, have proved their worth in the system. But perhaps they are useful because they have been used only in simple settings: sometimes one might need to switch off part of the Autoprove, and there are situations where one would like to supply interactively the rules Rewrite uses. Again it might be useful to be able to supply a default tactic for a non-logical operator (as is already the case with equality) or, in general, define a default tactic to be used in the context of a particular theory. All these considerations suggest that the answer to the first question should be "Yes", but the creation of tactics should be guarded. Users shouldn't have the ability to usurp the lowest level of operation of the system and hence compromise the consistency of the logic. Again it should be possible to backtrack over a tactic, or interrupt a tactic and still maintain the consistency of the proof and all its attributes.

Of course even if the problems of integrating tactics into the system were solved, a mechanism also has to provided to create new tactics. The simplest solution might be to make ML and the tactic level directly accessible to the users; however that seems somehow contrary to the spirit of an IPE style of system, and would almost certainly interfere with the attribute consistency of the system. An alternative might be to provide the ability to create buffers where fragments of proof trees (or proof tree skeletons) may be constructed by elaborating a null goal, by selecting from the basic operations embodied in the IPE. In addition it might be possible to provide iterators, conditionals and facilities to preset some of the proof inputs to make it look more like a language of tacticals. The standard attribute evaluation mechanism would still have to be invoked to apply these buffers.

The facility the IPE already has for copying fragments of proofs into buffers and saving proof skeletons provides an ability to create tactics. These tactics are in some sense "proofs by example". Considering how this works in the case of an inductive proof, might point out the attractiveness of such a scheme for creating tactics and also some

of its shortcomings. A proof by induction tactic will have the form :-

Break into cases by inductive definition of the data
Simplify the base case
With Each Step Case
  Clear the logical operators
  Simplify the conclusion of the goal
  Simplify the conclusion by the induction premises

In the course of an inductive proof this will be created interactively, possibly using Autoprove, but certainly using Rewrite; saved as proof skeleton and reapplied in a different setting by reading in the proof skeleton. However what information should the proof skeleton record? Presently the skeleton records the trace generated by Autoprove, not a reference to the use of it. This is fine for Autoprove, but for Rewrite the skeleton must record the use of the tactic not its trace. This behaviour is vital, as the details of simplifications of terms will differ for every proof. Thus if a proof by example is to be reusable, it must have within itself some flexibility, generated by the appeal to general tactics. List induction is similar in form to Natural Number induction, so one would hope that by just editing the name of the axiom we could reuse this proof by induction tactic, however this will not work because the skeleton records the trace of Autoprove, not its use. Thus even this simplistic method of creating tactics requires a richer set of built in tactics and more flexible parameterisation of proofs to be worthwhile.

5 Conclusion

IPE revealed the challenge involved in the construction of truly interactive yet powerful proof development environments. The problems that arise combine both theoretical and practical issues, whilst the systems themselves are truly multilayered: a logical kernel, controlled by a proof management system, with which one interacts via a window based interface. Appropriate solutions to these can only be found by experiments with prototypes, supported by the development of the underlying theory. Whilst further development of IPE itself has ceased, work on its successors is well under way at LFCS. Future editors will support refinement proofs, based upon use of typed lambda calculus to represent logics as in Constructions and Edinburgh LF.

References

[1] Cohn, A.J. and Milner, R., On Using Edinburgh LCF to Prove the Correctness of a Parsing Algorithm, University of Edinburgh Computer Science Department Internal Report CSR-113-82.

[2] Cohn, A.J., A Proof of Correctness of the Viper Microprocessor, Proceedings of the Calgary Hardware Verification Workshop, Calgary 1987.

[3] de Bruijn, N.G., A Survey of the Project Automath, To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism, Academic Press 1980.

[4] The User's Manual for A Computational Logic, Boyer, R.S. and Moore, J.S., University of Texas at Austin 1987.

[5] Burstall, R.M., Taylor, P. and Jones, C., Interactive Proof Editing with the Edinburgh IPE, course organised by the Laboratory for Foundations of Computer Science, University of Edinburgh 1987.

[6] Coquand, T. and Huet, G., Constructions:a higher-order proof system for mechanising mathematics, EUROCAL'85: European Conference on Computer Algebra, 151-184, Springer-Verlag 1985.

[7] Gentzen, G.,Investigations into Logical Deduction, in: The Collected Works of Gerhart Gentzen, edited by M. E. Szarbo, North Holland 1969.

[8] Gordon, M., Milner, R. and Wadsworth, C. Edinburgh LCF: A Mechanised Logic of Computation, Springer-Verlag 1978.

[9] Harper, R., Honsell, F. and Plotkin, G., A Framework for Defining Logics. IEEE Symposium on Logic in Computer Science, Ithaca 1987.

[10] Knuth, D.E., The Semantics of Context-Free Languages, Math. Syst. Theory 2, 1968.

[11] Lakatos I., Proofs and Refutations, Cambridge University Press 1976.

[12] Paulson L.C., Interactive Theorem Proving with Cambridge LCF, Technical Report 80, University of Cambridge Computer Laboratory 1985.

[13] The PRL Group (Constable et. al.) Implementing Mathematics with the Nuprl Proof Development System, Prentice Hall, New Jersey 1986.

[14] Reps, T., Generating Language Based Environments, Ph.D. Thesis, Cornell University 1982, and MIT Press 1984.

[15] Reps, T. and Alpern, B., Interactive Proof Checking, ACM Symposium on Principles of Programming Languages, Salt Lake City 1984.

[16] Reps, T. and Teitelbaum T., The Synthesiser Generator. Proceeding ACM SIGPLAN Symposium on Software Development Environments, Pittsburgh, 1984

[17] Ritchie B., The Design and Implementation of an Interactive Proof Editor, Ph.D. Thesis , Edinburgh University 1987.