User Interfaces for Generic Proof Assistants
Part I: Interpreting Gestures*

Bernard Sufrin † Richard Bornat ‡

April 3, 1998

Abstract

JAPE is a generic proof editor which (amongst other things) offers the teacher, user, or logic designer the opportunity of constructing a direct manipulation interface for proofs.

In part I of this paper we introduce JAPE, illustrate some aspects of proof development by direct manipulation, and describe some of the infrastructure provided for interpreting the gestures which users make at proofs. In part II we discuss the problem of displaying proofs at appropriate levels of abstraction, and describe the infrastructure JAPE provides for eliding proof steps from the display.'

1 Introduction

JAPE is a generic interactive proof editor designed to assist teachers, students, and practitioners of formal reasoning. It accepts a description of a logic as a system of inference rules, and provides support for the discovery of proofs in that logic. It is not an automatic theorem prover: on the contrary, our goal has been to support human-directed proof-discovery – the machine does as much of the donkey-work as is appropriate in a given proof setting, and the human provides insight at crucial[1] moments – if necessary by making a few simple remarks, but preferably by gesturing.

We like to think of Jape as a bit like a pedantic (but uninspired) student beavering away at a proof, with an experienced teacher providing support. From time to time the teacher points at a formula and says something like "concentrate on reducing this using the appropriate list theorems", or "distribute that map through the catenations", or "reduce the scope of that quantifier" – enough of the teacher's insights can be communicated by these means to make progress possible when the next step to take isn't completely obvious to the student.

The description of a logic may be accompanied by a description of a direct manipulation interface – a mapping, if you will, between the user's gestures and the assistant's proof actions. In this paper we discuss some of the gesture-interpreting infrastructure JAPE makes available to the interface designer. Although our discussion is centred on JAPE, we believe that many of the techniques we've used can be used in other settings.

There are few constraints on the kinds of logic with which Jape can cope: our goal was to make it possible for some-one who isn't expert in Jape to code a logic more or less directly from its presentation in a textbook or paper. Several specimen logics accompany the standard distribution of Jape [Bornat and Sufrin, 1996a], and we have selected two of these logics to provide the setting for most of the discussion below.

An important limitation we imposed on ourselves when implementing JAPE was that although it must embody a metatheory of formulae, bindings, and deduction, it should not have any special knowledge of any particular kind

* This paper is a revised and enlarged version of a paper of the same name presented at the workshop "User Interfaces for Theorem Provers", York, U.K., July 1996.

† Computing Laboratory and Worcester College, Oxford

‡ Queen Mary and Westfield College, London

[1] "Crucial" is in the eye of the beholder, of course. For a student taking first steps in learning to discover proofs in a simple predicate logic, almost every step is crucial, and it's important that the student participates in each decision. For an experienced functional programmer, the crux of an inductive proof may come after a good deal of automatically invoked rewriting.

$$\dfrac{\dfrac{\dfrac{}{P\,,\;Q \vdash P}\,hyp \qquad \dfrac{}{P\,,\;Q \vdash Q}}{P\,,\;Q \vdash (P \land Q)}\vdash\land}{P \vdash Q {\to} (P \land Q)}\vdash\to$$

1	P	*assumption*
2	Q	*assumption*
3	P	*hyp* 1
	...	
4	Q	
5	$(P \land Q)$	$\vdash \land\ 3, 4$
6	$Q{\to}(P \land Q)$	$\vdash \to 2 - 5$

Figure 1: A partial proof in tree and box forms

of object logic or theory – not even a theory of equality. This constraint gave rise to a number of challenges in the design of the infrastructure to support direct manipulation interfaces, and we tried at every stage to meet these challenges in a principled, and logically justifiable way.

By this we intend to convey the idea that JAPE should not make any move in a proof, unless that move is justifiable by rules and theorems of the its currently-loaded object logic. We wanted to avoid "magic" – proof transformations implemented by JAPE's internal algorithms which might not be justifiable in some object logics.

In sections 4 and 5 we discuss some of the more important problems, and the extent to which our solutions can indeed be described as logically justifiable.

2 Proof Display

It is very hard to experiment with proof tactics and strategies without being able to see the location and structural consequences of decisions taken during the proof, and this suggests that an interactive proof tool should always display the proof(s) under construction. On the other hand this seems to be impractical for a tool in which high level proofs are conducted by tactics which invoke several or many low-level inference rules – information overload is as hard to cope with as an information vacuum!

We conclude that the display of the complete proof – at an appropriate level of abstraction – is essential to support fluent proof discovery, and we provide a good deal of support for showing appropriately abstracted proofs. This support is discussed in part II of this paper [Bornat and Sufrin, 1996c].

During a proof, the user is shown a view of the proof tree; this can be literally tree-shaped, or – more usefully – presented in the boxed style pioneered by Fitch [Fitch, 1952]. For example, Figure 1 shows a partial proof in both tree and box forms.[2] We discuss the relationship between these two views in [Bornat and Sufrin, 1996b].

3 Interpreting Gestures

The user's side of the interaction is conducted by making gestures at JAPE: these are made by typing, by mouse-clicks, or by mouse-drags. The most important form of gesture results in the identification (and eventual invocation) of a tactic – which might be one of the inference rules, or might be something more complicated. Tactics may be chosen from menus or panels or by double-clicking on a displayed hypothesis or conclusion. Before identifying the tactic, the user may have selected a hypothesis and/or a conclusion formula, and may also have text-selected parts of other formulae visible in the proof.[3] The situation at the point the gesture is made is called the *gesture context*, and in what follows we shall be describing some of the tactic language constructs which give access to it.

[2] This example, and the other examples in the first part of the paper are drawn from the intuitionistic predicate calculus – part of whose JAPE description is in Appendix C.

[3] By dragging the mouse over them in the usual way.

3.1 Menu-based gestures

The simplest form of user interface places the names of the inference rules on menus or panels whence they may be invoked in the usual way. It's very easy to set up such an interface; all we need to do is to enclose the declarations of the inference rules in a `MENU` declaration:

```
MENU Rules
        RULE hyp(A)   A ⊢ A
        RULE "⊥⊢"     ⊥ ⊢ A
        RULE "⊢∧"     FROM ⊢A AND ⊢B INFER ⊢A∧B
        ...
END
```

3.2 Direct manipulation

Of more interest here is the style of interface where the user signals which rule to apply by double-clicking on an appropriate formula. In JAPE such interfaces can be constructed by making a few additional declarations. For example

```
HYPHIT  A∧B ⊢ C   IS "∧⊢"
CONCHIT     ⊢ B∨C IS "⊢∨"
```

The first of these declarations associates the proof rule `"∧⊢"` with a double-click on a hypothesis which is a conjunction. The second is more interesting – it associates the *tactic* `"⊢∨"` with a double-click on a disjunctive conclusion. This tactic tries to complete the proof of the disjunction by using the left or-introduction rule followed by the hypothesis rule, or (if that doesn't work) by using the right or-introduction rule followed by the hypothesis rule.[4]

```
TACTIC  "⊢∨"
(ALT (PROVE "⊢∨(L)" hyp)
     (PROVE "⊢∨(R)" hyp)
     (FAIL "⊢∨" does not lead to an immediate conclusion))
```

If neither of these methods succeeds, then the user has to exercise a little more insight, and decide which of the branches of the disjunct to aim at: the corresponding or-introduction rule has to be invoked by some other means (which is why we rarely dispense with the menu interface completely)

3.3 Parameters

An interesting situation arises when we want to invoke a rule whose antecedents cannot be completely determined by unifying its consequent with a goal sequent. For example

```
RULE "⊢∃"(B) FROM P[x\B] INFER ∃x.P
```

The purpose of the formal parameter `(B)` of the exists-introduction rule, is to enable the extra information to be supplied as an explicitly-chosen argument when the rule is applied. For example, if the current goal is

$$\exists y.\textit{undefined}\ y$$

[4]The `ALT` form takes a list of tactics (which may be rules), and tries them in sequence until one of them succeeds – in which case it succeeds, or it runs out of rules – in which case it fails.

and we run the tactic[5] `"⊢ ∃"`$(foo \div 0)$ then a new subgoal

$$undefined(foo \div 0)$$

is generated from the antecedent.

If, on the other hand, we run the tactic `"⊢ ∃"` without supplying any additional information, then JAPE will invent a new proof unknown `_B` whose name is derived from `B`,[6] and generate a new goal with conclusion `P[x \ _B]` – in the case of our example this is *undefined _B*. The unknown is generated to permit the user to delay choosing `B` until later in the discovery of the proof – when the choice may be easier to make.

But we often know in advance what the `B` is going to be – and there is frequently a candidate `B` in a visible formula already. We want to make it as convenient as possible for the user to invoke the tactic *with the appropriate parameter*. The guarded tactic form

(`LETARGSEL` *pattern tactic*)

is designed for use in just these circumstances. It succeeds if a formula has been text-selected, and the selection unifies with *pattern* (which may contain proof unknowns), and the *tactic* (which may contain occurences of proof unknowns bound by the unification) also succeeds. Thus, for example, the direct-manipulation declaration

```
CONCHIT ∃x.B IS (LETARGSEL _TERM "⊢∃"(_TERM))
```

causes the tactic `"⊢ ∃"(_TERM)` to be invoked, whenever an existential conclusion is double-clicked in the presence of a text-selected formula. In the situation outlined at the start of this section we could therefore run the tactic `"⊢ ∃"`$(foo \div 0)$ by text-selecting the formula $(foo \div 0)$ then double-clicking on $\exists y.undefined\ y$.

The tactic needs a little refinement, however, because the conclusion-hit will fail (because the `LETARGSEL` fails) if there's no text selection.[7]

```
CONCHIT ∃x.B IS (WHEN (LETARGSEL _TERM "⊢∃"(_TERM))
                      ("⊢∃"))
```

Now the exists-introduction rule is invoked when there's a double-click on an existential conclusion: if a formula has been *text selected* it is provided as an argument to the rule; if not, then the rule is invoked without an argument.

3.4 Resolving ambiguity in the application of rules

When working on the hypothesis side of a proof in a single-conclusion calculus we frequently find ourselves in situations where the rule we invoke does not uniquely determine the hypothesis to which it is to be applied.[8] For example, in the proof situation shown below the rule `"→⊢"` can be used to eliminate either of the implication hypotheses.

$$\cfrac{\cfrac{\overline{P\to(Q\to R)\ ,\ \ (P\to Q)\ ,\ \ P\ \vdash\ R}}{P\to(Q\to R)\ ,\ \ (P\to Q)\ \vdash\ (P\to R)}\ \vdash\to}{P\to(Q\to R)\ \vdash\ (P\to Q)\to(P\to R)}\ \vdash\to$$

[5] Rules are the simplest form of tactic.

[6] The derivation method is safe: it avoids re-using unknowns which are still present in the proof – albeit at the cost of deriving abominable names like _B1, _B2, ... The names of Jape proof unknowns are always prefixed with an underscore, and they are the only things which appear in a proof which are so decorated. We should like to have used some other typographic convention for displaying them, but the austerity of our implementation infrastructure precludes that at present.

[7] The `WHEN` tactic form constructs a composite tactic from a list of simple guarded tactics followed by a default tactic. The effect of a `WHEN` is the effect of the first component whose guard is satisifed. If no guard is satisfied then the effect is the effect of the default tactic.

[8] The dual situation exists when we're working on the conclusion side of a multi-conclusion calculus, and JAPE provides mechanisms dual to those being discussed here for coping with it.

Use of implication-elimination would be ambiguous here

In such situations it's useful to be able to select the hypothesis which is to be acted on before applying the rule.[9]

The guarded tactic form

(`LETHYP` *pattern tactic*)

is used to facilitate such selections. It succeeds if the selected hypothesis[10] matches the pattern, and the given tactic succeeds. The tactic form

(`WITHHYP` *tactic*)

runs *tactic* in such a way that if there is any ambiguity over which hypothesis to use when *tactic* applies a rule, then the selected hypothesis is the one that is chosen.

An appropriate direct-manipulation declaration for implication hypotheses is therefore

```
HYPHIT P→Q ⊢ C IS
(WHEN (LETHYP _FORM (WITHHYP "→⊢"))
      ("→⊢"))
```

In fact, the `WITHHYP` form is fairly forgiving – if there's no hypothesis selected, then the tactic under its control gets run in the normal way. So the declaration above could be simplified, with no loss of functionality, to

```
HYPHIT P→Q ⊢ C IS (WITHHYP "→⊢")
```

3.5 Guarding the application of tactics

It is often useful to restrict the situations in which a particular rule or tactic is invoked. We can do so by guarding the tactic with a non-trivial pattern in one of the (many) `LET` forms.

For example, in a gesture context where an equivalence hypothesis has been selected, the tactic `HypRewrite` whose body is

```
(LETHYP (_A ≡ _B) (substitutivity _A)(hyp (_A ≡ _B)))
```

invokes the following sequential[11] tactic.

```
(SEQ (substitutivity _A)(hyp (_A ≡ _B)))
```

The `substitutivity` rule is:[12]

```
RULE  substitutivity(A, OBJECT x)
FROM  A≡B
AND   F[x\B]
INFER F[x\A]
```

The intention is that `HypRewrite` should use the selected hypothesis, if it's an equivalence, as a left-to-right rewrite rule – replacing all occurences of its left subformula with its right subformula.

It is worth seeing what happens when we use the `HypRewrite` tactic in the following (contrived) situation with the left hypothesis selected

[9] If that isn't done then JAPE presents the user with a choice box which shows the candidate hypotheses and the corresponding outcomes – the sudden switch in the style of interaction which this enforces can be disconcerting.

[10] At present there can be at most one selected hypothesis and at most one selected conclusion in the gesture context.

[11] The `SEQ` is implicit.

[12] In the general case the unification of a formal substitution such as that in the consequent of the substitutivity rule may have to be deferred. Provision of a formula `A` to abstract over (*i.e.* replace with *x*) during the unification makes an immediate unification possible.

1	$P \equiv Q\ ,\ \ Q \equiv R$	*assumptions*
	$\ldots$	
2	$(P \rightarrow R) \wedge (R \rightarrow P)$	

About to use `substitutivity P`

After the first tactic in the sequence – `substitutivity P` – succeeds, the proof looks like this:

1	$P \equiv Q\ ,\ \ Q \equiv R$	*assumptions*
	$\ldots$	
2	$P \equiv \_B$	
	$\ldots$	
3	$(\_B \rightarrow R) \wedge (R \rightarrow \_B)$	
4	$(P \rightarrow R) \wedge (R \rightarrow P)$	*substitutivity* $2, 3$

After using `substitutivity P`

The `_B` is a proof unknown, ready to be unified with any term, and line 2 (the leftmost unclosed subgoal of the previously applied rule) has (automatically) become the current goal. The application of `(hyp (_A ≡ _B))` to this goal closes it, leaving us with

1	$P \equiv Q\ ,\ \ Q \equiv R$	*assumptions*
	$\ldots$	
2	$(Q \rightarrow R) \wedge (R \rightarrow Q)$	
3	$(P \rightarrow R) \wedge (R \rightarrow P)$	*substitutivity* $1.1, 2$

After `hyp ...`

which is the box-style presentation of the following proof tree.[13]

$$\dfrac{\dfrac{}{P \equiv Q\ ,\ \ Q \equiv R\ \vdash\ P \equiv Q}\,\mathit{hyp} \qquad \dfrac{}{P \equiv Q\ ,\ \ Q \equiv R\ \vdash\ (Q \rightarrow R) \wedge (R \rightarrow Q)}}{P \equiv Q\ ,\ \ Q \equiv R\ \vdash\ (P \rightarrow R) \wedge (R \rightarrow P)}\,\mathit{substitutivity}$$

As a tree after `hyp ...`

4 Support for Equational Reasoning

This section of the paper is (loosely) based on the the system for working with functions and symmetric lists which is distributed with JAPE. This presented the interaction designer with several interesting problems – including those of controlling search with a light touch – which we shall address here. The problems aren't confined to the realm of equational proof, and the solutions we propose here were designed to be used in other realms as well.

4.1 The evolution of an Unfold gesture

Most equational theories lean heavily on the use of a rule which embodies Leibniz' law of substitutivity of equals. It can be formulated as follows in JAPE.

```
RULE  rewrite (X, OBJECT v, Y)
FROM  X=Y
AND   P[v\Y]
INFER P[v\X]
```

[13]The alert reader will wonder where the `hyp` step vanished to in the box style presentation. The answer is that it was elided – a full explanation of elision is given in part II of this paper [Bornat and Sufrin, 1996c].

In this section we illustrate how the JAPE tactic language facilitates the deployment of this rule – despite the fact that JAPE has no special knowledge of equality. We'll do so by giving an account of the evolution of a particularly useful gesture-interpreting tactic.

To provide a concrete setting for our account we shall work on a proof of the conjecture $rev \bullet rev = id$ – first by considering the rules which we would apply if we were going to do everything by hand, and then by developing an "unfolding" tactic which makes the interaction rather simple.

We start by applying the extensionality rule – giving a proof tree of the form

$$\dfrac{\overline{(rev \bullet rev)x = id\;x}}{rev \bullet rev = id}\;ext$$

Next we decide that we want to rewrite the right hand side, using the *id* rule, which is $id\;x = x$. This is done in two steps. In the first we apply the tactic `rewrite (id x)`, leaving the proof tree in the following state[14]

$$\dfrac{\dfrac{\overline{id\;x = \_Y} \qquad \overline{(rev \bullet rev)x = \_Y}}{(rev \bullet rev)x = id\;x}\;rewrite}{rev \bullet rev = id}\;ext$$

Selecting the leftmost unclosed subgoal ($id\;x = \_Y$) we now apply the *id* rule ($id\;x = x$). This closes the goal – unifying $\_Y$ with x in the process, – and we're left in the following proof state.

$$\dfrac{\dfrac{\overline{id\;x = x}\;id \qquad \overline{(rev \bullet rev)x = x}}{(rev \bullet rev)x = id\;x}\;rewrite}{rev \bullet rev = id}\;ext$$

Next we must rewrite the left hand side of the remaining equation, using the definition of composition. The first step is to apply `rewrite((rev • rev) x)`, and this leaves the proof in the following state.[15]

$$\dfrac{\dfrac{\overline{id\;x = x}\;id \qquad \dfrac{\overline{(rev \bullet rev)x = \_Y} \qquad \overline{\_Y = x}}{(rev \bullet rev)x = x}\;rewrite}{(rev \bullet rev)x = id\;x}\;rewrite}{rev \bullet rev = id}\;ext$$

The next step is to select the leftmost unclosed goal, and apply the composition rule `"•"`, which is $(F \bullet G)x = F(G\,x)$. This closes the goal in question – this time unifying $\_Y$ with $rev(rev\;x)$, and leaves the proof in state

$$\dfrac{\dfrac{\overline{id\;x = x}\;id \qquad \dfrac{\overline{(rev \bullet rev)x = rev(rev\;x)}\;\bullet \qquad \overline{rev(rev\;x) = x}}{(rev \bullet rev)x = x}\;rewrite}{(rev \bullet rev)x = id\;x}\;rewrite}{rev \bullet rev = id}\;ext$$

At this point we'll call a halt to the developing proof, and observe two things. The first observation is that *given that we're proceeding by equational reasoning* the tree has a lot of redundant information present in it – we explain the remedies JAPE offers for this in part II of this paper [Bornat and Sufrin, 1996c].

The second observation – which is more important in this context – is that there's a certain ritualised form to the steps we're taking: each user level move involves a decision about a subterm to rewrite, followed by a decision about a rule to use in the rewriting. But once we have chosen the subterm to rewrite, the rule to use is *obvious*.

[14] A detailed explanation of the way in which this works is given in appendix A.

[15] Notice that the proof unknown $\_Y$ is re-used – its earlier occurences in the proof tree simply vanished when we applied the *id* rule.

This gives rise to an interesting way of using the LETARGSEL form discussed earlier. The tactic defined below tries to rewrite the text-selected (sub)-term using one of the rules defined in the Functions theory.

```
TACTIC RewriteSelection IS
(WHEN (LETARGSEL _TERM (SEQ (rewrite _TERM) Functions))
      (FAILWITH "Please text-select a term"))
```

Its behaviour depends on two important features of JAPE, namely:

1. During the application of a SEQuential tactic form, JAPE always works on the leftmost remaining unclosed subgoal unless told to do otherwise.
2. A JAPE theory implicitly defines an ALT tactic (whose name is the theory name) containing the names of all the rules defined in it.

So if there is a text selection, then the rewrite rule is applied first – leaving an unclosed leftmost subgoal of the form *textselection* $= \_Y$ as the next to be worked-on. Next the Functions tactic simply applies each of the rules in the *Function* theory in turn, until one of them matches the goal (or it runs out of rules).[16]

We can make the rule more useful if we make it possible to use the currently selected hypothesis – if it's an equation – as a rewrite rule. Another useful refinemenet is to abstract over the tactic used to search for a suitable rewriting rule. Making both these refinements yields the tactic

```
TACTIC UnfoldSelection(rulebase) IS
(WHEN  (LETHYP (_L = _R) (LETARGSEL _TERM (SEQ (rewrite _TERM) (hyp (_L = _R)))))
  (LETARGSEL _TERM (SEQ (rewrite _TERM) rulebase))
  (FAILWITH "Please text-select a term"))
```

Another refinement has the rule do something sensible even if there's no text-selection.

```
TACTIC UnfoldSelectionOrSearch(rulebase) IS
(WHEN (LETHYP (_L = _R) (LETARGSEL _TERM (SEQ (rewrite _TERM) (hyp (_L = _R)))))
      (LETARGSEL _TERM (SEQ (rewrite _TERM) rulebase))
      (UNFOLD rewrite rulebase))
```

If there's no selected hypothesis or text selection, then the UNFOLD form, which is the last branch of the WHEN, searches the current goal, in leftmost outermost order, for a subterm which matches the left hand side of one of the rules in *rulebase*. If the search is successful – say because the subterm *t* matches the left hand side of the rule *r*, then the effect is as if the tactic (SEQ (rewrite *t*) *r*) had been run.

There is nothing unique about the ability to do a search in this way – what's interesting from the point of view of generic interface construction is the way in which the language of gesture interpretation supports its controlled deployment.

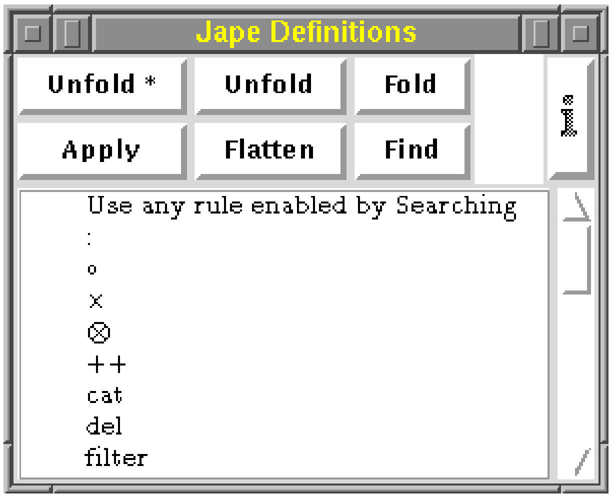
The "folding" counterpart of the tactic is just as straightforwardly programmed, using the FOLD form to replace UNFOLD, and using the rewritebackwards rule to replace replace.

4.2 Controlling Search

One remaining detail concerns the way in which the user can decide on which rule base to use for the rewriting. Here JAPE offers an abundance of mechanism. In this section we explain one style of using the mechanism which has proven useful.

[16]Theory designers have to be careful to order the presentation of rules so that the more general a rule is the later it appears in the theory – JAPE should take the responsibility for this itself.

First we need to explain the concept of *panel* – which is a kind of free-floating list of rules and conjectures decorated with one or more buttons. For example, here's the *Definitions* panel from JAPE's functional programming presentation.



Entries on a panel can be selected (by clicking on them), and buttons on the panel may be pressed in the usual way. The `Unfold` button on the panel shown above can be associated with the tactic we developed in the previous section by making the following declaration in the scope of the declaration of the *Definitions* panel.

```
PREFIXBUTTON Unfold IS apply UnfoldSelectionOrSearch
```

When a `PREFIXBUTTON` is pressed, JAPE checks that one of the rules in the panel has been selected, and then supplies that rule as an argument to the tactic associated with the button. For example, clicking on `Unfold` having selected `"•"` is equivalent to applying the tactic `(UnfoldSelectionOrSearch "•")`. In a context where there's no text selection, this results in a search for a place where the definition of `"•"` can be applied left-to-right, and if there is a text selection it results in an attempt to rewrite that selection[17] using the rule.

It's frequently the case (particularly in equational proofs) that there's really only one term worth unfolding[18], or that the user can indicate an appropriate site for an unfolding by making a text selection. It is tedious to have to indicate precisely which rule should be used in the unfolding – often a simple search through a collection[19] of rules will resolve the question. But the collection of rules to use may change during the course of a proof – sometimes it's appropriate to include theorems (derived rules) in the search, sometimes it's not; sometimes it's appropriate to include functions defined for lists, sometimes it's not.

One method of providing choice would be to bind each collection of rules to a panel button for each action, but this leads to "button bloat" – a multiplicity of highly specialised buttons which overwhelms the user's powers of comprehension or the interface designer's capacity for inventing meaningful labels.

The alternative is to provide a convenient means of editing the content of a single collection – the `SEARCH` collection. We then place this collection as an entry in the appropriate panel(s):[20]

```
ENTRY "Any rule enabled by Searching'' IS SEARCH
```

Pressing `Unfold` with this entry selected invokes `UnfoldSelectionOrSearch SEARCH`, and pressing `Fold` with it selected invokes `FoldSelectionOrSearch SEARCH`.

[17] To be more precise, *all instances* of the selection – we shall take this question up in section 4.3.
[18] Or folding – all the remarks we make about unfolding apply equally well to folding.
[19] Identified by an `ALT` form.
[20] The details go a little beyond the scope of this discussion; they are presented in Appendix B.

4.3 Directing attention to specific terms

The unfolding tactic which we've been developing rewrites *all* instances of a text-selected term, and although this is often good enough, there are sometimes situations where we want to direct the attention of the tool to a *specific instance* of a term.

For example, suppose we are starting a proof of a conjecture such as

$$\begin{array}{ll} & \dots \\ 1 & \mathit{ref} \bullet \mathit{ref} = \mathit{id} \end{array}$$

and (for some reason – the example is contrived) wish to unfold the right `ref` but not the left one.

The present `UnfoldSelectionOrSearch ref` is too crude. If we text select the right `ref` then it rewrites *both* occurences – yielding

$$\begin{array}{lll} 1 & \mathit{ref} = (\mathit{rev} \times \mathit{rev}) \bullet \mathit{swap} & \mathit{ref} \\ & \dots & \\ 2 & (\mathit{rev} \times \mathit{rev}) \bullet \mathit{swap} \bullet ((\mathit{rev} \times \mathit{rev}) \bullet \mathit{swap}) = \mathit{id} & \\ 3 & \mathit{ref} \bullet \mathit{ref} = \mathit{id} & [\mathit{rewrite}]\ 1, 2 \end{array}$$

If we select nothing, and rely on `UNFOLD`, then the *left* `ref` will be expanded.

What we want to be able to do is to direct attention to the exact occurence which we text-selected. Rather than invent a special notation for describing occurences, we find it very appealing to exploit the substitution notation, and here's how we do it.

If one or more identical subformulas have been text-selected in a host formula within the current goal, then the tactic form

$$(\mathtt{LETSUBSTSEL}\ \mathit{pattern\ tactic})$$

matches *pattern* against the host formula in which they have been selected and invokes *tactic* if the match succeeds. So far there's nothing too remarkable about this. The magic part is that while the host formula is matched, it is presented in the form of an explicit substitution

$$\mathit{abstraction}[\mathit{variable} \backslash \mathit{selection}]$$

which, *if simplified* would yield the original host.

In the above case this would yield:

$$(\mathit{ref} \bullet \mathit{newvariable} = \mathit{id})[\mathit{newvariable} \backslash \mathit{ref}]$$

A *pattern* of the form `_P[_v\_X]` will unify with this, binding $\_P$ to $\mathit{ref} \bullet \mathit{newvariable} = \mathit{id}$, $\_v$ to *newvariable*, and $\_X$ to *ref*.

It's convenient to think of the host formula being converted into an "stable" substitution, which resists simplification for just one unification step (the match with `LETSUBSTSEL`'s pattern). The tactic form

$$(\mathtt{WITHSUBSTSEL}\ \mathit{rule})$$

has an analogous effect on the formula in which a selection has been made – turning it into a stable substitution for just long enough to run the *rule*. For example, running `WITHSUBSTSEL rewrite` in the original proof state, with the right `ref` text-selected yields

$$\begin{array}{lll} & \dots & \\ 1 & \mathit{ref} = \_Y & \\ & \dots & \\ 2 & \mathit{ref} \bullet \_Y = \mathit{id} & \\ 3 & \mathit{ref} \bullet \mathit{ref} = \mathit{id} & \mathit{rewrite}\ 1, 2 \end{array}$$

What happens is that the pattern `_P[_v\_X]` generated by the consequent of `rewrite` matches the stable substitution generated by `WITHSUBSTSEL`, then the antecedent of `rewrite` – which immediately after the unification is in the form substitution $(ref \bullet newvariable = id)[newvariable \backslash \_Y]$ – gets simplified.

Invoking the rule which defines `ref` yields the following proof state, which is just what we want.

$$
\begin{array}{lll}
1 & ref = (rev \times rev) \bullet swap & ref \\
 & \ldots & \\
2 & ref \bullet ((rev \times rev) \bullet swap) = id & \\
3 & ref \bullet ref = id & [rewrite]\ 1, 2
\end{array}
$$

The more discriminating behaviour of the `SUBSTSEL` tactic forms makes them more appropriate than `LETARGSEL` to use in the tactics described in section 4.1. The unfolding tactic now becomes:

```
TACTIC UnfoldSelectionOrSearch(rulebase) IS
(WHEN
  (LETHYP (_L = _R) (LETSUBSTSEL _TERM (SEQ (WITHSUBSTSEL rewrite) (hyp (_L = _R)))))
  (LETSUBSTSEL _TERM (SEQ (WITHSUBSTSEL rewrite) rulebase))
  (UNFOLD rewrite rulebase))
```

5 Dealing with Associativity

JAPE has no special internal representation for associative terms.[21] Nor, as we noted at the beginning of this paper, does it have any built-in assumptions about any particular object theory – even equality. All JAPE proofs are, at bottom, purely logical proofs conducted using the currently-installed axioms and rules.

One the face of things it would seem that exploiting algebraic properties such as commutativity and associativity might be rather difficult under these circumstances. But things aren't as bad as they seem, and in this section we explain how JAPE facilitates the exploitation of associative rules.

The tactic forms we describe are not *logically* necessary. The built-in tactic forms which support *flattening*[22], and *finding*,[23] are built-in for the sake of efficiency and generality, and to spare the user from the distractions of carrying out tedious activity at too low a level of abstraction.

5.1 The `FLATTEN` Tactic Form

Consider the proof state

$$
\begin{array}{lll}
 & \ldots & \\
1 & filter\ P = map\ fst \bullet (cat \bullet map(if\ snd(one, none))) \bullet zip \bullet (id \otimes map\ P) & \\
2 & filter\ P = map\ fst \bullet (filter\ snd) \bullet zip \bullet (id \otimes map\ P) & \textit{Fold using filter}\ 1 \\
3 & filter\ P = map\ fst \bullet filter\ snd \bullet zip \bullet (id \otimes map\ P) & \textit{Associativity}\ 2
\end{array}
$$

We'd like to use the theorem $map\ F \bullet cat = cat \bullet map(map\ F)$ to rewrite the leftmost $map\ fst \bullet cat$ – but at present there are parentheses in the way, so it isn't a proper subterm of the current right hand side, and can neither be text-selected, nor matched by an unfolding tactic.

But composition ($\circ$) is associative, and this means we can eliminate the (semantically) redundant parentheses. One way of doing so would be to proceed by hand, selecting subterms and either folding or unfolding with composition's associativity rule until JAPE no longer needs to use parenthesis to indicate structure. This would be tedious and error-prone, so we provide a means of automating the job.

[21] Terms formed with associative connectives.

[22] *i.e.* reduction of terms to associative normal form.

[23] *i.e.* the transformation of a term to a form in which a particular term is a proper subterm.

The tactic form (`FLATTEN` *term*) transforms the *term* – which should be a subterm of the current goal – into *assocative normal form*: a form in which no parenthesis is necessary to indicate structure. It justifies the transformation by using the associative laws of the topmost operator of the term, if there are any. `FLATTEN` will only conduct logically sound associative transformations: if there is no associative law for the topmost operator, or if there is a law but it hasn't yet been proven then `FLATTEN` fails.

In this case `FLATTEN` – which records its use as "Associativity" – gets us to the proof state

	$\ldots$	
1	$filter\ P = map\ fst \bullet cat \bullet map(if\ snd(one, none)) \bullet zip \bullet id \otimes map\ P$	
2	$filter\ P = map\ fst \bullet (cat \bullet map(if\ snd(one, none))) \bullet zip \bullet (id \otimes map\ P)$	*Associativity* 1
3	$filter\ P = map\ fst \bullet (filter\ snd) \bullet zip \bullet (id \otimes map\ P)$	*Fold using filter* 2
4	$filter\ P = map\ fst \bullet filter\ snd \bullet zip \bullet (id \otimes map\ P)$	*Associativity* 3

In this state *map fst* • *cat* is a proper subterm, and can be text-selected, or discovered by one of the other branches of `UnfoldSelectionOrSearch` – giving a proof which starts as follows

	$\ldots$	
1	$filter\ P = cat \bullet (map(map\ fst)) \bullet map(if\ snd(one, none)) \bullet zip \bullet id \otimes map\ P$	
2	$filter\ P = map\ fst \bullet cat \bullet map(if\ snd(one, none)) \bullet zip \bullet id \otimes map\ P$	*Fold* $(map\ F \bullet cat = \ldots)$ 1
3	$filter\ P = map\ fst \bullet (cat \bullet map(if\ snd(one, none))) \bullet zip \bullet (id \otimes map\ P)$	*Associativity* 2
4	$filter\ P = map\ fst \bullet (filter\ snd) \bullet zip \bullet (id \otimes map\ P)$	*Fold using filter* 3
5	$filter\ P = map\ fst \bullet filter\ snd \bullet zip \bullet (id \otimes map\ P)$	*Associativity* 4

Insight tells us that for our next move we'd like to use the theorem $map\ F \bullet map\ G = map(F \bullet G)$ to rewrite the leftmost outermost composition of *map*s. If JAPE were equipped with associative unification then we could use our existing unfold tactic – having selected the theorem in the appropriate panel. But JAPE doesn't have that facility, and the composition we have our eye on isn't a proper subterm (of the current representation) of the goal.

It seems that we are going to have to use the associative law of composition to transform the proof into one in which $(map\ (map\ fst) \circ map(if\ \ldots))$ is a proper subterm of the goal, namely

	$\ldots$	
1	$filter\ P = cat \bullet ((map(map\ fst)) \bullet map(if\ snd(one, none))) \bullet zip \bullet id \otimes map\ P$	
2	$filter\ P = cat \bullet (map(map\ fst)) \bullet map(if\ snd(one, none)) \bullet zip \bullet id \otimes map\ P$	*Associativity* 1
3	$filter\ P = map\ fst \bullet cat \bullet map(if\ snd(one, none)) \bullet zip \bullet id \otimes map\ P$	*Fold* $(map\ F \bullet cat = \ldots)$ 2
4	$filter\ P = map\ fst \bullet (cat \bullet map(if\ snd(one, none))) \bullet zip \bullet (id \otimes map\ P)$	*Associativity* 3
5	$filter\ P = map\ fst \bullet (filter\ snd) \bullet zip \bullet (id \otimes map\ P)$	*Fold using filter* 4
6	$filter\ P = map\ fst \bullet filter\ snd \bullet zip \bullet (id \otimes map\ P)$	*Associativity* 5

We are reluctant to insist that such transformations be performed by hand, so we must provide a means of automating the job. One way of doing so is the subject of the next section.

5.2 The FindSelectionInConclusion Tactic

If there is a text selection in the current conclusion, and the current conclusion is an equation which can be transformed by the application of associative laws to one in which the text selection is a proper subterm, then the following tactic performs that transformation.

```
TACTIC FindSelectionInConclusion IS
(WHEN
(LETCONCFIND (_XOLD=_YOLD, _XNEW=_YNEW)
   (ALT
    (SEQ (rewriteEquation _XOLD _XNEW _YOLD _YNEW) EVALUATE EVALUATE)
    (FAIL "the term you selected isn't a subterm of the conclusion"))))
```

This time the tactic is a little more complicated – requiring the use of a built-in tactic form, a specialised form of judgement, and a built-in proof procedure for that judgement. In order to explain how it works in the situation outlined above we need to explain its parts in some detail.

5.2.1 The `LETCONCFIND` tactic form

The tactic form

$$(\texttt{LETCONCFIND}\ (pattern_{old}, pattern_{new})\ tactic)$$

succeeds if

- there is a text selection, and
- the current conclusion unifies with $pattern_{old}$, and
- the result of putting brackets around the selected text and reparsing the whole conclusion unifies with $pattern_{new}$, and
- the form's body (*tactic*) then succeeds.

5.2.2 The `rewriteEquation` rule

The proof rule `rewriteEquation` captures the idea that the formula $X' = Y'$ is equivalent to the formula $X = Y$ providing that both X, X' and Y, Y' are associatively equivalent.

```
RULE    rewriteEquation(X, X', Y, Y', OBJECT x) IS
FROM    ASSOCEQ (X, X')
AND     ASSOCEQ (Y, Y')
AND     X'=Y'
INFER   X=Y
```

5.2.3 The `ASSOCEQ` judgement

The judgement $ASSOCEQ(X, X')$ is intended to hold if the terms X and X' have the same associative normal form. A decision procedure for such a tactic is programmable in JAPE's tactic language, but experience has shown it to be rather slow. So we have extended the scope of the built-in tactic form `EVALUATE` – which is the hook on which decision procedures for specialised forms of judgement are hung – to include *ASSOCEQ* judgements.

The operation of `EVALUATE` is quite simple. If presented with a judgement of a form for which a decision procedure has been declared, then it applies the appropriate decision procedure. In this case the decision procedure for *ASSOCEQ* judgements is built-in – its method is to apply the flattening transform to both its arguments and check the identity of the resulting terms.

5.2.4 `FindSelectionInConclusion` in operation

The transformation with which we began the discussion of `FindSelectionInConclusion`, namely from the goal

$$filter\ P = cat \circ (map(map\ fst)) \circ map(if\ snd(one, none)) \circ zip \circ id \otimes map\ P$$

into the goal

$$filter\ P = cat \circ ((map(map\ fst)) \circ map(if\ snd(one, none))) \circ zip \circ id \otimes map\ P$$

is accomplished by text-selecting the term $(map(map\ fst)) \circ map(if\ \ snd(one, none))$ then invoking the selection-finding tactic.

The first thing that happens, during the unification phase of `LETCONCFIND` is that the following bindings take place:

`_XOLD`	to	$filter\ P$
`_YOLD`	to	$cat \circ (map(map\ fst)) \circ map(if\ \ snd(one, none)) \circ zip \circ id \otimes map\ P$
`_XNEW`	to	$filter\ P$
`_YNEW`	to	$cat \circ ((map(map\ fst)) \circ map(if\ \ snd(one, none))) \circ zip \circ id \otimes map\ P$

The rule `rewriteEquation` is invoked next, and this gives rise to the three subgoals:

$$\begin{array}{lll} ASSOCEQ & (& filter\ P, \\ & & filter\ P) \end{array}$$

$$\begin{array}{lll} ASSOCEQ & (& cat \circ (map(map\ fst)) \circ map(if\ \ snd(one, none)) \circ zip \circ id \otimes map\ P, \\ & & cat \circ ((map(map\ fst)) \circ map(if\ \ snd(one, none))) \circ zip \circ id \otimes map\ P \\ &) & \end{array}$$

$$filter\ P = cat \circ ((map(map\ fst)) \circ map(if\ \ snd(one, none))) \circ zip \circ id \otimes map\ P$$

The first two of these are solved immediately by the following two applications of `EVALUATE`, leaving the last subgoal for the user's attention.

The full detail of the proof tree generated by running the tactic as described above is usually uninteresting, so in the production variant of the tactic the two `ASSOCEQ` subtrees are suppressed from the display, using the display transformation machinery described in the second part of this paper.

6 Envoy

The design of the language forms which support direct manipulation is clearly not yet finished. The small number of tactic forms described above, buttressed by analogous forms which deal with hypotheses, seem to make it possible for an interaction designer to program sensible interpretations of users' gestures rather easily, but no doubt these forms will undergo some refinement (and pruning) as our experience with them grows.

In the realm of proof discovery we have shown that it's possible in practice to dispense with non-logical descriptions of term occurences in the design of gesture-interpreting tactics. The full scope of the technique of stable substitutions has yet to be explored, but it is already clear to us that some control over the extent of their stability is going to be needed.

The soundness of the `FLATTEN` tactic and tactics like `FindSelectionInConclusion` ultimately depends on the soundness of the transformation into associative normal form which JAPE performs. We have not formally verified the algorithm concerned – the only reassurance we can offer is the fact that it's a very simple algorithm, and is "obviously" right. But we can offer no more reassurance than that for any other component of JAPE, at present and that is a weakness we must acknowledge. On the other hand, the fact that the superficially "magic" behaviour of `FLATTEN` and `ASSOCEQ+EVALUATE` cannot be evoked when there is no assocative rule (or theorem) to justify the transformations performed should increase the user's confidence in the soundness of any results which depend on them.

In closing, it's worth remarking that at this stage it wouldn't be appropriate to try to answer the sensible question posed to us by readers of an earlier draft of this paper: "for what class of user is a direct manipulation interface a good thing?" We believe that it's just beginning to be possible to make empirical studies of such questions using JAPE. Further improvements in *Jape*'s expressive power and in the quality and range of its display machinery should make such studies easier, and we look forward to collaborating with those who want to try.

A Unification of Formal Substitutions

When rules (such as `rewrite`) with formal substitutions in the consequent are used, they are frequently given an explicit formal parameter (the `X`, in the case of `rewrite`) so that the unification of the consequent with a goal can proceed smoothly when an actual parameter is given.[24]

In one of the examples in the body of the paper we saw that the goal $(rev \bullet rev)\ x = id\ x$ was transformed by the rule `(rewrite (id x))` into the subgoals $id\ x = \_Y$ and $(rev \bullet rev)x = \_Y$. What happens during the application of the rule is as follows:

1. A new object v and proof unknown $\_Y$ are invented (because neither was supplied as an actual parameter).
2. The consequent $P[v\backslash(id\ x)]$ is matched to the goal by abstracting *every* occurence of *id x* from the goal and replacing it with v – in other words the consequent is $((rev \bullet rev)x = v)[v\backslash i\ x]$.
3. The antecedents $id\ x = \_Y$ and $((rev \bullet rev)x = v)[v\backslash \_Y]$ are now generated, and the latter is immediately simplified to $((rev \bullet rev)x = \_Y$.

B Editing the "Search" collection

The simplest way of arranging for a named collection of rules to be dynamically configurable is to define the collection as an alternation of named tactics which can themselves be configured to fail (which permits the search defined by the alternation to continue), or to apply a rule (which may itself be an alternation).[25]

```
TACTIC SEARCH IS (ALT fun list ...)
```

JAPE's radio-button interface elements provide a way of defining a named tactic dynamically. For example, the element defined below defines a menu button with three distinct labels. It allows the user to choose the definition of `fun` to be either the tactic `Functions` (an implicitly-defined `ALT` tactic which searches the rules defined in the `Functions` theory), or an explicitly defined `ALT` tactic which searches the rules and the theorems of that theory, or the tactic `JUSTFAIL` – which does what its name suggests.

```
RADIOBUTTON fun IS
        "Search Function rules"              IS Functions
AND     "Search Function rules and Theorems" IS (ALT Functions FunctionsThm)
AND     "Ignore Function rules"              IS JUSTFAIL
END
```

[24] In fact JAPE defers the unification if the actual parameter isn't supplied.

[25] For those who like hearing about such things in algebraic terms, `ALT` is associative, and has unit `JUSTFAIL`.

C An extract from the Intuitionistic Sequent Calculus

```
RULE     hyp(A)           A ⊢ A
RULE     "⊥⊢"              ⊥ ⊢ A

RULE     "⊢∧"              FROM ⊢A AND ⊢B INFER ⊢A∧B
RULE     "∧⊢"              FROM A, B ⊢ C INFER   A∧B ⊢ C

RULE     "⊢∨(L)"           FROM ⊢A INFER ⊢A∨B
RULE     "⊢∨(R)"           FROM ⊢B INFER ⊢A∨B
RULE     "∨⊢"              FROM A ⊢ C AND B ⊢ C INFER A∨B ⊢ C

RULE     "⊢≡"              FROM ⊢A→B AND ⊢B→A INFER ⊢A≡B
RULE     "≡⊢"              FROM A→B,  B→A ⊢ C INFER A≡B ⊢ C

RULE     "⊢∀"(OBJECT m)  WHERE FRESH m
FROM     P[x\m]
INFER    ∀x . P

RULE     "∀⊢"(B)           FROM ∀x.P, P[x\B] ⊢ C INFER ∀x.P ⊢ C

RULE     "⊢∃"(B)           FROM ⊢P[x\B] INFER ⊢∃x.P

RULE     "∃⊢"(OBJECT m)   WHERE FRESH m
FROM     P[x\m] ⊢ C
INFER    ∃x.P ⊢ C
```

References

[Bornat and Sufrin, 1996a] Richard Bornat and Bernard Sufrin 1996a. *The Jape Web Sites.*

`http://www.comlab.ox.ac.uk/oucl/users/bernard.sufrin/jape.html`
`ftp://ftp.dcs.qmw.ac.uk/programming/jape/index.html`

[Bornat and Sufrin, 1996b] Richard Bornat and Bernard Sufrin 1996b. *Jape's Quiet Interface.* Proceedings of this Workshop

[Bornat and Sufrin, 1998b] Richard Bornat and Bernard Sufrin 1998b. *User Interfaces for Generic Proof Assistants – Part II: Displaying Proofs.* (Revised Version: Available at the Jape Web sites)

[Fitch, 1952] Fitch, F.B. 1952.*Symbolic Logic.* Ronald Press, New York

[Stevens, 1996] Andrew Stevens 1996. *Towards Mechanised Revision of Proofs and Theories* Proceedings of this Workshop

Contents