User Interfaces for Generic Proof Assistants
Part II: Displaying Proofs*

Bernard Sufrin †   Richard Bornat ‡

March 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 the first part of this paper we introduced JAPE, illustrated some aspects of proof development by direct manipulation, and described some of the infrastructure provided for interpreting the gestures which users make at proofs. In this part we discuss the problem of displaying proofs at appropriate levels of abstraction, and describe the infrastructure JAPE provides for doing this. We also show how specialised modes of display for proofs which use *cut* and *identity* rules can be exploited to give the illusion of forward (natural deduction style) proof.

1 Introduction

In the first part of this paper we gave an account of the interaction tactics which are used by JAPE to interpret users' gestures in context in order to invoke proof tactics and rules. We also explained how such tactics provide support for interactive proof activity at level of abstraction closer to provers' intuitions than that of the basic inference rules of a logic might be.

Just as it is important to be able to define tactic programs which implement intuitive proof moves, so it is important to be able to *show* proofs at an appropriate level of abstraction: for otherwise the human engaged in a proof has to abstract the essence of a proof situation from a display which may well present too much detail.

We start this part of the paper by describing the mechanisms which JAPE currently provides for displaying proofs at a level of abstraction chosen by the interaction designer. Then we describe JAPE's linear presentation style for proofs in which *identity* and *cut* rules have been used – and show how the use of a simple display transformation coupled with appropriate interaction tactics can be used to give the illusion of forward proof.

2 Displaying Equational Proofs

Logical presentation masks equational essence

Consider the following proof state, reached during a proof that list-reversal is self-inverting:

---

*This is the second part of a paper whose first part[1] was 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

$$\dfrac{\dfrac{\dfrac{}{id\ x = x}\,id \qquad \dfrac{\dfrac{}{(rev \bullet rev)x = rev(rev\ x)}\,\bullet \quad \dfrac{}{rev(rev\ x) = x}}{(rev \bullet rev)x = x}\,rewrite}{(rev \bullet rev)x = id\ x}\,rewrite}{rev \bullet rev = id}\,ext$$

At an intuitive level we might describe the history of this proof as follows: "The extensionality rule is followed by unfolding the term *id x* into *x*, and this is followed by unfolding the term $(rev \bullet rev)\ x$ into $rev(rev\ x)$."

But the display shows so much detail that it can be hard for someone who didn't do the proof (or who doesn't care that an unfolding proof step happens to be implemented by an application of the *rewrite* rule) to follow what went on. What's needed is a way of suppressing the irrelevant logical detail from the proof display whilst preserving its equational essence. .

One way of doing so would be to augment the basic rules (and definitions) of the the logic in question with specialised derived rules. For example, from the rule `id X = X` which defines *id*, and the *rewrite* rule

```
RULE  rewrite (X, Y, ABSTRACTION P)
FROM  X=Y
AND   P(Y)
INFER P(X)
```

we can derive the rule[1]

```
RULE  "Fold id"(X, Y, ABSTRACTION P)
FROM  P(X)
INFER P(id X)
```

Using this rule, and the corresponding rule for compositions, the proof would become.

$$\dfrac{\dfrac{\dfrac{rev(rev\ x) = x}{(rev \circ rev)x = x}\,Fold \circ}{(rev \circ rev)x = id\ x}\,Fold\ id}{rev \circ rev = id}\,ext$$

But this isn't really a practical proposition: in any interesting logic there will simply be too many derived rules for us to install before we can start work on the conjectures which really interest us. What is needed is a way of transforming the *presentation* of the *logical* proof state into a presentation in the equational style.

Subtree Selection – a partial solution

In order to explain the partial solution which JAPE currently offers we shall first have to explain the machinery for displaying proofs. In effect there's a two-stage pipeline.

$$\begin{array}{c}\text{Logical}\\\text{Proof}\end{array} \longrightarrow \boxed{\text{Proof Abstraction}} \longrightarrow \begin{array}{c}\text{Abstracted}\\\text{Proof}\end{array} \longrightarrow \boxed{\text{Proof Rendering}} \longrightarrow \begin{array}{c}\text{Displayed}\\\text{Proof}\end{array}$$

The rendering stage depends on the proof display style currently in use, and constructs a proof layout from the abstracted proof. The details of this stage of the transformation will become important when we come to discuss automatic elision of redundant formulae, but they're not important here.

The abstraction stage is guided by abstraction descriptions which are installed in the logical proof tree as it is being constructed. This is done by tactics of the form

[1] Why is the rule called `Fold id`? Because when we read the proof *forward*, the transformation is from *x* to *id x*. Such a right-to-left use of a defining equation is generally referred to as a folding transformation.

$$(\texttt{LAYOUT}\ \ \mathit{reason}\ \ \mathit{branch{-}list}\ \ \mathit{tactic}_1\ \ \mathit{tactic}_2\ \ \ldots)$$

A tactic of this form is executed by running its component *tactic*s in sequence. If this succeeds, then the proof subtree which it generates is marked to be displayed as an abstraction of the logical proof tree, explained with (a text derived from) the given *reason*. The only form of abstraction currently provided is *subtree selection*, and the *branch−list* is a (possibly empty) list of the numbers of the branches of the generated subtree which are to be displayed.

For example, if the tactic form

```
(LAYOUT "Fold id" (1) (rewrite (id x)) id)
```

is executed in the proof state

$$\frac{(\mathit{rev} \circ \mathit{rev})x = \mathit{id}\ x}{\mathit{rev} \circ \mathit{rev} = \mathit{id}}\ \mathit{ext}$$

it transforms the logical proof into

$$\frac{\dfrac{\dfrac{}{\mathit{id}\ x = x}\ \mathit{id} \quad \dfrac{}{(\mathit{rev} \circ \mathit{rev})x = x}}{(\mathit{rev} \circ \mathit{rev})x = \mathit{id}\ x}\ \mathit{rewrite}}{\mathit{rev} \circ \mathit{rev} = \mathit{id}}\ \mathit{ext}$$

but the proof abstraction stage of the display process ensures that only the second branch (numbered 1) of the new proof subtree is selected for rendering, as shown below.

$$\frac{\dfrac{(\mathit{rev} \circ \mathit{rev})x = x}{(\mathit{rev} \circ \mathit{rev})x = \mathit{id}\ x}\ \mathit{Fold\ id}}{\mathit{rev} \circ \mathit{rev} = \mathit{id}}\ \mathit{ext}$$

It makes sense to generalise this tactic over the rule (or tactic) used to close the left branch of the *rewrite* tree, so we define:

```
TACTIC Unfold(term, rule) (LAYOUT "Fold %s" (1) (rewrite term) rule)
```

When an application of this tactic is succesful, the reason annotation of the displayed proof node is constructed by expanding `"Fold %s"` in the layout tactic – replacing the `%s` by the name of the rule used at the base of the branch which is suppressed. Although this simplistic way of constructing the annotation is nearly always good enough, we shall see later that there are circumstances where it can be uninformative or misleading.

In part 1 we developed a tactic, `UnfoldSelectionOrSearch`, to support the construction of “fold” moves in equational proofs. Below we present the production version of that tactic – this reports proof steps made with it at the appropriate level of abstraction. The last stage of development is very simple: the three three underlined sequential tactic forms of the original (underlined below) are just embedded in `LAYOUT` forms which select their second branch.

```
TACTIC UnfoldSelectionOrSearch(rulebase) IS
(WHEN
  (LETHYP (_L=_R)
          (LETSUBSTSEL _TERM
                 (LAYOUT "Fold with hyp" (1) (WITHSUBSTSEL rewrite) (hyp (_L=_R)))))
  (LETSUBSTSEL _TERM
        (LAYOUT "Fold %s" (1) (WITHSUBSTSEL rewrite) rulebase))
  (LAYOUT "Fold %s" (1) (UNFOLD rewrite rulebase)))
```

Shallow and weak though it is, the form of tree abstraction provided by LAYOUT has been sufficiently powerful to support coherent proof presentations in several theories based on equational reasoning. These range from a simple untyped theory of functional programming, to a reasoning system for typed category theory. In the latter theory, the typing antecedents of rules are decided automatically by tactic, but the proofs of well-typing are suppressed (unless they fail) in the presentation.

Universally Applied Tree Transformations

In one experimental JAPE implementation, the annotations which guide proof abstraction were described declaratively in a tree-transformation notation. For example, the following transformation could be used to present an unfolding – an application of rewrite whose left subgoal is closed directly by a rule.

```
TRANSFORM Unfolding (rulename, proof)
  Γ ⊢ Q BY  rewrite
  FROM
    Γ ⊢ X=Y BY rulename.
  AND
    Γ ⊢ P FROM proof.
  END
INTO
  Γ ⊢ Q BY (Fold rulename)
  FROM
    Γ ⊢ P FROM proof
  END
```

The following transformation could be used to present a more deeply-nested proof – a fold, in fact.

```
TRANSFORM Folding (rulename, proof)
  Γ ⊢ Q BY rewrite
  FROM
    Γ ⊢ X=Y BY "= Symmetric"
    FROM
      Γ ⊢ Y=X BY rulename.
    AND
      Γ ⊢ P FROM proof.
  END
INTO
  Γ ⊢ Q BY (Unfold rulename)
  FROM
    Γ ⊢ P FROM proof
  END
```

The declarative style seemed, for a while, to be inappropriate: partly because the notation is rather long-winded, but mainly because transformations were applied without discrimination to all proof trees of the right form – even if the tree in question was constructed by applying primitive rules by hand. The resulting display could misleadingly suggest that high-level proof steps like "unfold" have been employed when they hadn't.

Selectively Applied Tree Transformations

It took us a while to realize that the *selective* application of declarative transformations could help us solve a class of awkward problems which arise from the simplemindedness of the way in which LAYOUT forms construct their explanations.

Consider programming a *Fold* tactic – designed to replace an occurence of the right hand side of a definition by the corresponding left hand side. This tactic is much the same as the *Unold* described earlier, but we will use the rule `"= symmetric"` (defined as `FROM X=Y INFER Y=X`) before applying the definitional rule.

```
TACTIC Fold(term, rule)
       (LAYOUT "Unfold %s" (1) (rewrite term) "= symmetric" rule)
```

The tactic does the right job, but unfortunately the `"%s"` in the reason component of this tactic picks up the name of the `"= symmetric"` rule – and explains the move as `Unfold "= symmetric"` when what we really want is the name of the rule which was actually used.

In situations like this the `LAYOUT` form is just inadequate. The problem is remedied by introducing the `TRANSFORM` tactic form, which is analogous to `LAYOUT` save that the tree abstraction it performs is described by a declaratively described transformation of the kind discussed above. The `Fold` tactic inow written as follows[2]

```
TACTIC Fold(term, rule)
       (TRANSFORM Folding (rewrite term) "= symmetric" rule)
```

Having decided to rehabilitate the more general form of tree transformation described above, several issues are worthy of investigation. For example, apart from the obviously-useful alternation combinator, what ways of composing transformations will be useful? What other ways of applying transformations will be useful?

3 Natural Deduction Displays of Sequent Proofs

As we demonstrated in the previous section, there are certain proofs whose presentations can be improved if the interaction designer takes the trouble to program interface tactics in a certain way. In this section we show how JAPE can *automatically* exploit the presence of certain kinds of structural rule in a logic to eliminate uninformative or redundant material from proof displays. One consequence of the display transformations we describe in this section is that it is easy to present a convincing simulacrum of forward proof in the natural deduction-style – despite the fact that JAPE is a backward proof system based on sequents!

The simplest automatically applicable display transformation is the removal of uninformative identity rules. We can best explain it by giving a concrete example.

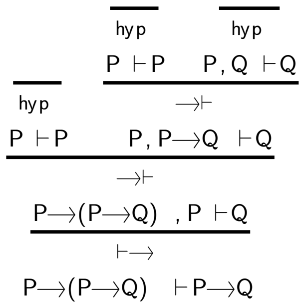


Figure 1: A proof in the tree style

The proof tree in figure 1 built using the following rules is shown rendered in the (unameliorated) linear style in figure 2.

```
RULE    "⊢→"    FROM Γ, A ⊢ B INFER Γ ⊢A→B
RULE    "→⊢"    FROM Γ, A→B ⊢ A AND Γ, B ⊢ C INFER Γ, A→B ⊢ C
RULE    hyp(A)  INFER Γ, A ⊢ A
```

In this style the proof of each sequent $\gamma_1, ...\gamma_n \vdash C$ is shown in a box, with the hypotheses at the top and the conclusion at the bottom, and the proofs of any antecedent sub-sequents recursively shown between them (ellipsis

[2]"`Folding`" is the proof transform we described on the previous page.

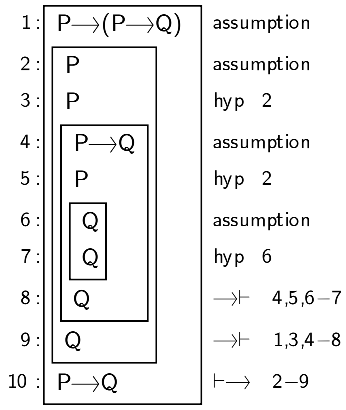


Figure 2: The same proof in the linear style

represents an unclosed subtree). If any of the hypotheses have already appeared at the top pf an enclosing box, then they are omitted, and if this results in a box with no hypotheses at the top, then the conclusion is presented unboxed.[3]

Notice that line 8 is justified in part by the P established on line 5 by *hyp* from the *identical* assumption on line 2. Likewise line 8 is justified in part by the P established on line 3 by *hyp* from the assumption on line 2. Finally notice that line 7 simply duplicates the assumption Q made on line 6. We say that *hyp* is an *identity* rule because its conclusion is identical to one of its assumptions. It seems fair to say that the use of *hyp* at lines 3, 5, and 7 do not really add to our understanding of the proof, and that these lines could therefore be eliminated. Figure 3 shows what happens when this is done.

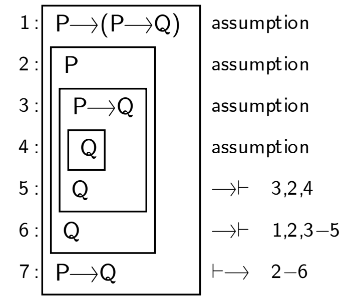


Figure 3: The same proof with *hyp* elided

Removing Cuts

A more interesting automatic display transformation is the elimination of *cut* rules – which take the form

```
FROM Γ ⊢ μ AND Γ, μ ⊢ C INFER Γ ⊢ C
```

It is used to "cut" the task of finding a proof of $\Gamma \vdash C$ into two parts: that of establishing an intermediate formula μ from the hypotheses Γ, and that of establishing the conclusion C from the hypotheses augmented with the intermediate formula. In figure 4 we show a *Cut* proof, and its standard linear display. Notice that the intermediate formula μ

[3]We note, in passing, that this presentation style is ambiguous for some logics: just because a hypothesis appears at the head of a box, that doesn't mean that it's in scope at the conclusion of all the boxes nested within. For example, the hypothesis $P{\to}(P{\to}Q)$ is not in scope at line 8 of the proof displayed above, because that line comes from the proof of the sequent $P, P{\to}Q \vdash Q$. In JAPE we resolve the ambiguity dynamically: when a particular conclusion is selected we "grey out" the hypotheses which aren't in scope at that conclusuion. Dually, when a particular hypothesis is selected we grey out those parts of the proof at which it isn't in scope.

appears once at the foot of its own proof, and again – as the new hypothesis opening the subproof which establishes C.

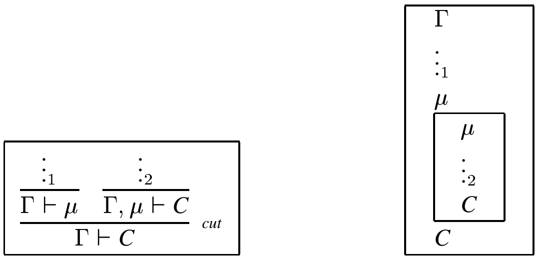


Figure 4: A *cut* proof and its standard linear display

The cut display transformation replaces the vertically adjacent (but structurally separate) occurences of μ with a single occurence, as shown in figure 5.

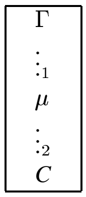
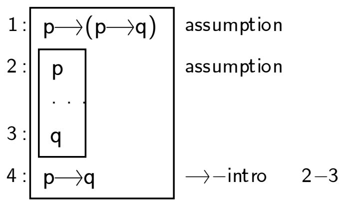


Figure 5: Linear display of a *cut* proof after transformation

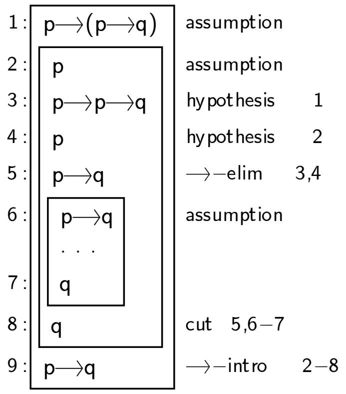
The following concrete example demonstrates the *cut* transformation in action during a proof of $p{\to}(p{\to}q) \vdash p{\to}q$ in a natural-deduction style logic, JnJ[3], which includes the following rules:

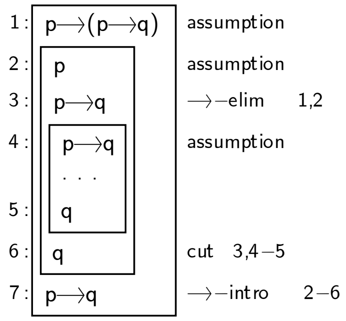
```
RULE  "→-intro"  FROM Γ, p ⊢ q INFER Γ ⊢ p→q
RULE  "→-elim"   FROM Γ ⊢ p→q AND Γ ⊢ p INFER Γ ⊢ q
RULE  hypothesis(p) INFER Γ, p ⊢ p
```

The intuitive approach to this proof for someone who has learned natural deduction is to use `"→-intro"` to establish the proof state



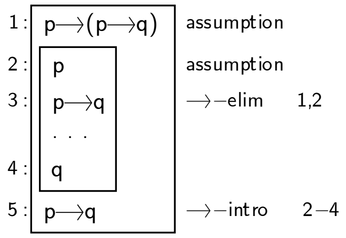
then to use implication elimination to establish $p{\to}q$. In a backward proof engine such as ours this has to be done by *cut*ting the proof first, then applying implication elimation. The resulting state is shown in complete detail in the diagram below on the left, and with applications of the *hypothesis* rule suppressed on the right.





Whilst these details may be illuminating to a specialist, someone who's simply trying to do a natural deduction proof would almost certainly be overwhelmed by the technicalities of the left-hand display, and would find the right-hand display uninformative.

Fortunately, application of the identity and cut-suppression transformations reveals just how simple the (natural deduction) proof situation is[4]



Now novices shouldn't have to know that forward application of elimination rules requires the use of *cut*! How do we arrange that simply invoking the implication elimination rule ends up doing the right thing?

We do so by defining an *interaction tactic* [1] which assumes that when both a hypothesis and a conclusion are selected, rules are to be applied in the forward direction, but when only a conclusion is selected, rules are to be applied backwards. One minor difficulty is the problem of deciding what is meant when the middle formula (μ) is selected in a transformed display such as the one in figure 5. Did the user intend to select the μ conclusion (in the left antecedent of *cut*) or the μ conclusion (in the right antecedent). Our approach to this is simple: if there's a selection below it in the display, then it is the hypothesis; if there is no other selection, or if the other selection is above it in the display, then it is the conclusion.

The forward application of a proof rule *R*, from aa hypothesis γ is implemented by the sequential composition of *cut*, *R*, and *hypothesis*(γ). This is easily arranged: an approximation to the interaction tactic used for this purpose in the presentation[4] of the logic is shown below

```
TACTIC ElimRule(rule)
(WHEN
  (LETHYP _p
    (ALT (SEQ  cut (WITHARGSEL rule) (WITHHYPSEL hypothesis))
         (FAIL ("%s is not applicable to %s", rule, _p))))
  (FAIL ("Select an assumption before applying %s", rule)))
```

The sequence of proof states which this evokes during the forward application of `"→-elim"` by `(ElimRule "→-elim")` described above is shown in Figures 6 through 8. Before the user sees the proof again, the automatic invocation of the *hypothesis* rule specified by the presentation designer will have disposed of the outstanding proof obligation in the left-hand subproof of figure 8.

[4] JAPE automatically applies the display transformations described above only when declarations are made which inform it of the existence (and names) of the structural rules in question.

$$\frac{\dfrac{p\to(p\to q),\ p \vdash \_p1 \qquad p\to(p\to q),\ p,\_p1 \vdash q}{p\to(p\to q),\ p \vdash q}\ \text{cut}}{p\to(p\to q) \vdash p\to q}\ \to\text{-intro}$$

Figure 6: Proof state after `cut`

$$\frac{\dfrac{\dfrac{p\to(p\to q),\ p \vdash \_p2\to\_p1 \qquad p\to(p\to q),\ p \vdash \_p2}{p\to(p\to q),\ p \vdash \_p1}\ \to\text{-elim} \qquad p\to(p\to q),\ p,\_p1 \vdash q}{p\to(p\to q),\ p \vdash q}\ \text{cut}}{p\to(p\to q) \vdash p\to q}\ \to\text{-intro}$$

Figure 7: Proof state after `(WITHARGSEL →-elim)`

$$\frac{\dfrac{\dfrac{\dfrac{}{p\to(p\to q),\ p \vdash p\to p\to q}\ \text{hypothesis} \qquad p\to(p\to q),\ p \vdash p}{p\to(p\to q),\ p \vdash p\to q}\ \to\text{-elim} \qquad p\to(p\to q),\ p,\ p\to q \vdash q}{p\to(p\to q),\ p \vdash q}\ \text{cut}}{p\to(p\to q) \vdash p\to q}\ \to\text{-intro}$$

Figure 8: Proof state after `(WITHHYPSEL (p→(p→q)))`

4 Conclusion and Prospects

Although JAPE is a generic tool, we believe that the techniques we have described here could easily be adapted for use in provers designed for specific logics, and would be of considerable benefit to users. Without the complexities induced by JAPE's generic nature they might also be much easier to implement!

I apologize to reviewers for having been forced to curtail this section of the submitted paper because several deadlines have hit me simultaneously.

I expect to be writing about two main topics here, namely the problem of displaying proofs formed from transitive relations, and the problem of designing "little languages" for describing display layouts and transforms.

References

[1] Richard Bornat & Bernard Sufrin. *User Interfaces for Generic Proof Assistants: Part I*. Presented at UITP-96, York, England
`http://www.comlab.ox.ac.uk/oucl/users/bernard.sufrin/JAPE/PAPERS/UITP96-paper.ps.gz.`

[2] Richard Bornat & Bernard Sufrin. *Displaying sequent-calculus proofs in natural-deduction style: experience with the Jape proof calculator*.
Presented at PTP-97, Schloss Dagstuhl, Germany
`http://www.comlab.ox.ac.uk/oucl/users/bernard.sufrin/JAPE/PAPERS/PTP97-paper.ps.gz.`

[3] Jim Woodcock and Jim Davies. *Using Z*. Prentice-Hall International.

[4] Bernard Sufrin & Richard Bornat. *JnJ in Jape*.
`http://www.comlab.ox.ac.uk/oucl/users/bernard.sufrin/JAPE/PAPERS/jnj.ps.gz.`