Displaying sequent-calculus proofs in natural-deduction style: experience with the Jape proof calculator.

Richard Bornat (QMW, University of London)

Bernard Sufrin (PRG, Oxford University)

Presented at PTP-97, Schloss Dagstuhl, Germany

1. Uncommitted, interactive, transparent, lightweight calculator

Jape[1,2,3] is a proof calculator: a system which can make correct steps in a proof under interactive user control. The steps may be incomplete, in the sense that they can introduce unknowns into the proof. The internal data structure of Jape is a Gentzen tree, and the program is designed to operate at the surface of logical proof so that, for example, it uses explicit visible provisos rather than encodings of restrictions on future proof steps. Our intention was to construct a system which would support the construction of proofs on screen in just the sort of way that proofs are constructed on paper or blackboard.

Jape is uncommitted to any particular logic: at present it is restricted to logics which can be presented using two-sided sequents. The syntax of formulæ in an encoding is under the encoder's control (and by use of specialised fonts, the screen display can be made to look very like printed proofs). Figure 0 shows the rules section of a sample encoding.

For the most part interaction with Jape is by clicking on a formula in a displayed partial proof, to select it to be worked on, and/or by selecting rules and other commands from menus. The interaction is always at the level of the proof and is always about proof steps: Jape doesn't give its users access to a proof plan, and it doesn't require them to decide in advance how they are going to tackle a proof. Like a numerical calculator it doesn't offer help or advice, and it forces its users to make the proof step-by-step. This is a conscious design choice: it keeps the user interface simple, and it is claimed to have advantages in an educational setting.

Jape is lightweight compared to much modern interactive application software – about 1.5Mbytes when compiled by the not-very-frugal SML-NJ. It isn't very fast as a proof-searching theorem prover, because it interprets the encoder's syntax description, rules and tactic definitions But speed is rarely a problem when working step-by-step using a modern personal computer, and by making it an interpreter we make it simple for our encoders to experiment.

Jape did not spring full-fledged from our brows: our most seminal influences were [5,6,7,8].

RULE hyp(A) INFER Γ,A⊢A
RULE "⊢∧" FROM Γ⊢A AND Γ⊢B INFER Γ⊢A∧B
RULE "∧⊢" FROM Γ,A,B⊢C INFER Γ,A∧B⊢C
RULE "⊢∨(L)" FROM Γ⊢A INFER Γ⊢A∨B
RULE "⊢∨(R)" FROM Γ⊢B INFER Γ⊢A∨B
RULE "∨⊢" FROM Γ,A⊢C AND Γ,B⊢C INFER Γ,A∨B⊢C
RULE "⊢¬" FROM Γ⊢A→ ⊥ INFER Γ⊢¬A
RULE "¬⊢" FROM Γ,A→ ⊥⊢B INFER Γ,¬A⊢B
RULE "⊢→" FROM Γ,A⊢B INFER Γ⊢A→B
RULE "→⊢" FROM Γ⊢A AND Γ,B⊢C INFER Γ,A→B⊢C
RULE "⊢≡" FROM Γ⊢A→B AND Γ⊢B→A INFER Γ⊢A≡B
RULE "≡⊢" FROM Γ,A→B,B→A⊢C INFER Γ,A≡B⊢C
RULE "⊥⊢" INFER Γ,⊥⊢A
RULE "⊢∀"(OBJECT m) WHERE FRESH m FROM Γ⊢P(m) INFER Γ⊢∀x.P(x)
RULE "∀⊢"(B) FROM Γ,P(B)⊢C INFER Γ,∀x.P(x)⊢C
RULE "⊢∃"(B) FROM Γ⊢P(B) INFER Γ⊢∃x.P(x)
RULE "∃⊢"(OBJECT m) WHERE FRESH m FROM Γ,P(m)⊢C INFER Γ,∃x.P(x)⊢C

figure 0: rules of the single-conclusion sequent calculus, encoded for Jape

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{hyp}}}{\neg P\to Q(\_B1),\ P\vdash P}\qquad \neg P\to Q(\_B1),\ P,\ Q(\_B)\vdash Q(m)}{\to\vdash}}{P\to Q(\_B),\ \neg P\to Q(\_B1),\ P\vdash Q(m)}\qquad P\to Q(\_B),\ \neg P\to Q(\_B1),\ \neg P\vdash Q(m)}{\vee\vdash}}{P\vee\neg P,\ P\to Q(\_B),\ \neg P\to Q(\_B1)\vdash Q(m)}}{\forall\vdash}}{P\vee\neg P,\ \forall x.\neg P\to Q(x),\ P\to Q(\_B)\vdash Q(m)}}{\forall\vdash}}{P\vee\neg P,\ \forall x.P\to Q(x),\ \forall x.\neg P\to Q(x)\vdash Q(m)}}{\vdash\forall}$$

$$P\vee\neg P,\ \forall x.P\to Q(x),\ \forall x.\neg P\to Q(x)\vdash \forall x.Q(x)$$

x NOTIN P

figure 1: a small partial proof in the single-conclusion sequent calculus (tree form)

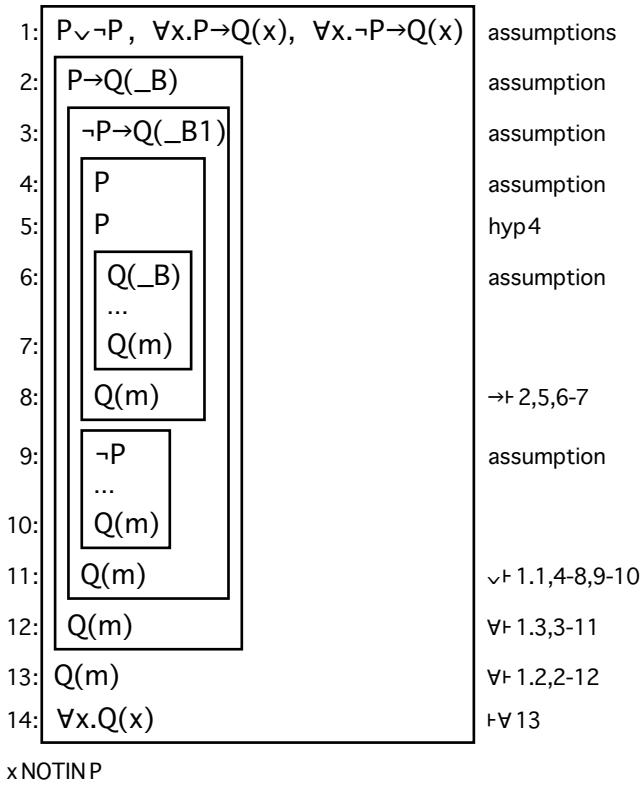


figure 2: a small partial proof in the single-conclusion sequent calculus (box form)

2. Tree to box

A Gentzen tree (for example, figure 1) is notoriously wasteful of space, because the hypotheses are written at every node of the tree. Jape saves a lot of space by using mathematical symbols rather than ASCII approximations, and with tricks such as writing the reasons in a smaller font it saves a little more. In Jape's tree display the reasons are in the middle of the line, rather than at the end, because the tree is normally much wider than it is high. In other respects the depiction is perfectly standard. Note the use of a visible proviso – proofs on paper use visible provisos and so, therefore, must Jape – and of explicit unknowns – we don't force our users to make decisions about the detail of a proof step until it is necessary to do so.

A simple postfix traversal – subtree, subtree, ... , subtree, root – converts figure 1 into a structure of lines. Writing hypotheses on separate lines, drawing a box whenever a subtree has a left-hand side formula which it doesn't share with its parent, and including left-hand-side principal terms in the reason attached to each line, leads to figure 2. Even though this is a proof in the single-conclusion sequent calculus, it looks remarkably like the box form of a natural-deduction proof, as described by Fitch[4]. The similarity is because most of the rules used have immediate natural-deduction analogues. Lines 4-8 are more peculiar, not only because they use the left-hand-side implication, which is not very like implication elimination in natural deduction, but also because line 5 is proved by the 'hyp' rule. Line 8 refers to line 5, which refers to line 4; if all lines proved by 'hyp' are eliminated we have figure 3, which is Jape's default box-display mechanism. The completed proof (figure 4) is relatively compact, and hides four occurrences of 'hyp'.

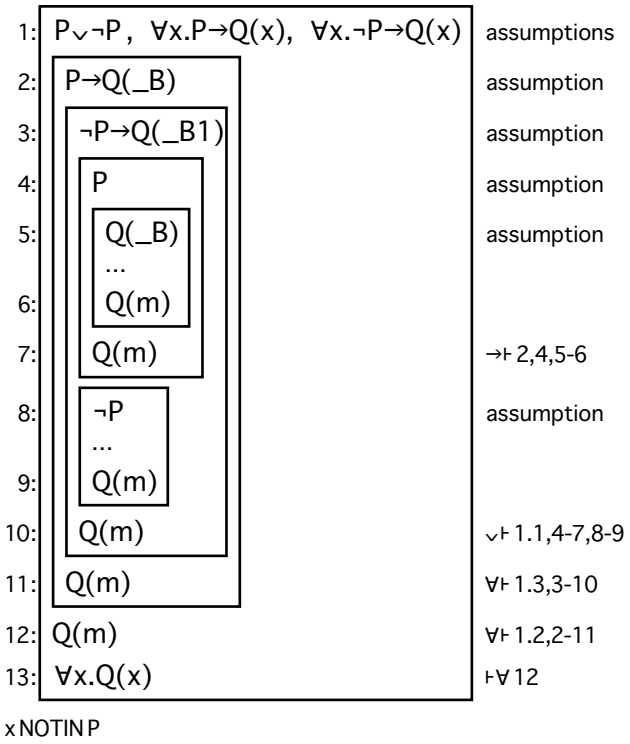


Figure 3: a small partial proof without 'hyp' lines

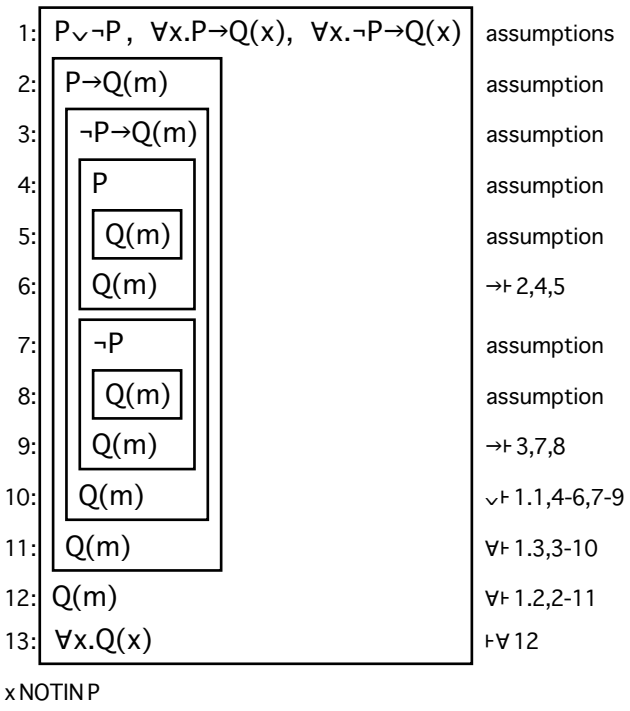


Figure 4: a completed small proof

Jape is able to take a standard presentation of natural deduction in a simplified sequent-calculus style (figure 5) in which only the 'hyp' rule is a concession to sequent calculus[1]; note in passing that by the encoder's choice this logic includes a classical treatment of negation, and that by the same choice it does not include an LF-style treatment of variables (so that, for example, ∀x.P(x) ⊢ ∃x.P(x) is a theorem in this logic). Proofs using this encoding appear just like proofs in natural deduction – for example, figure 6. This style of presentation seems to us to have significant advantages over natural deduction trees: the treatment of hypotheses is more easily explicable in the box presentation, and for educators that makes a conclusive argument.

---

1 The name 'hyp' has been used in our encodings both of single-conclusion sequent calculus and of natural deduction; this is an historical accident, not a necessary feature.

```
RULE "→-E"(A) IS FROM A AND A→B INFER B
RULE "∧-E(L)"(B) IS FROM A ∧ B INFER A
RULE "∧-E(R)"(A) IS FROM A ∧ B INFER B
RULE "∨-E"(A,B) IS FROM A ∨ B AND A ⊢ C AND B ⊢ C INFER C
RULE "¬-E" IS FROM ¬¬A INFER A
RULE "∀-E"(B) IS FROM ∀x. A(x) INFER A(B)
RULE "∃-E"(OBJECT c) WHERE FRESH c AND c NOTIN ∃x.A IS FROM ∃x.A(x) AND A(c) ⊢ C INFER C
RULE "→-I" IS FROM A ⊢ B INFER A→B
RULE "∧-I" IS FROM A AND B INFER A ∧ B
RULE "∨-I(L)"(B) IS FROM A INFER A ∨ B
RULE "∨-I(R)"(A) IS FROM B INFER A ∨ B
RULE "¬-I"(B) IS FROM A ⊢ B ∧ ¬B INFER ¬A
RULE "∀-I"(OBJECT c) WHERE FRESH c IS FROM A(c) INFER ∀x .A(x)
RULE "∃-I"(B) IS FROM A(B) INFER ∃x.A(x)
RULE hyp(A) IS INFER A ⊢ A
```

Figure 5: rules of natural deduction encoded for Jape

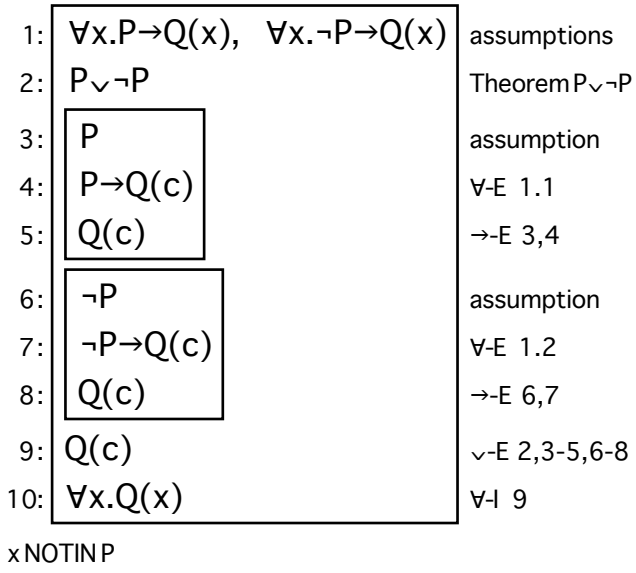


Figure 6: a small proof in the above encoding of natural deduction.

Note that Jape supports backward proof in a number of simple ways. First, it is able to apply rules automatically in certain circumstances. This is useful particularly for 'closing rules' such as hyp (axiom, identity), and in most cases encoders restrict the automatic application of such rules to situations where no explicit unification is necessary. We also allow a limited notion of proof-by-pointing: in our case we do not want our users to lose sight of the step-by-step construction of the proof, so we restrict it to a single level; we allow a double-click on a formula, in either box or tree mode, to fire a rule chosen by pattern-matching on the chosen formula. As with all our proof automation, this is entirely and completely under the encoder's control.

3. Equational reasoning by hiding 'facts'

The boxes shown above each incorporate information in the reasons attached to each proved conclusion: the reason quotes the lines which prove the subgoals, and any left-hand-side principal term in the rule that makes the step. In encoding equational reasoning, we have included more information in the reason attached to each proof step.

Our example is an encoding of equational reasoning in functional programs; we treat function definitions as equations, and we base most proof steps on the rewrite rule[1]

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

[1] The ABSTRACTION annotation directs Jape to interpret the juxtaposition formulæ AA(Y) and AA(X) as shorthand for substitution forms. The parameter X allows us to provide an argument formula which helps when unifying the conclusion formula, which will be a substitution form, with the conclusion of a problem sequent.

When we use this rule, we direct Jape:

- to hide the proof of X=Y, which is normally a simple function definition;
- to incorporate the name of the rule used to prove X=Y into the reason attached to the step;
- to show only the inference from AA(Y) to AA(X).

Figure 7 shows the beginning of a proof which uses the rewrite rule; figure 8 shows the hidden detail in that proof. The encoder controls which subgoals of a proof step should be hidden and what should be displayed as the reason for the proof step.

```
  ...
1: map F(map G x)=map(F•G)x
2: (map F•map G)x=map(F•G)x    Fold • 1
3: map F•map G=map(F•G)        ext 2
```

Figure 7: partial proof with hidden 'facts'

```
1: (map F•map G)x=map F(map G x)    •
  ...
2: map F(map G x)=map(F•G)x
3: (map F•map G)x=map(F•G)x         [rewrite] 1,2
4: map F•map G=map(F•G)             ext 3
```

Figure 8: proof with 'facts' revealed

The completed proof (figure 9) includes a good deal of repeated information. The equational parts of the proof can be read most easily 'up the left-hand sides, down the right-hand sides'. For example, lines 11 to 16 would be easier to read if they were written

$$
\begin{aligned}
& map\ F\ \big(map\ G\ (xs +\!\!+ ys)\big) \\
&= map\ F\ \big(map\ G\ xs +\!\!+ map\ G\ ys\big) && [map.2] \\
&= map\ F\ \big(map\ G\ xs\big) +\!\!+ map\ F\ \big(map\ G\ ys\big) && [map.2] \\
&= map\ (F \bullet G)\ xs +\!\!+ map\ F\ \big(map\ G\ ys\big) && [\text{inductive hypothesis}] \\
&= map\ (F \bullet G)\ xs +\!\!+ map\ (F \bullet G)\ ys && [\text{inductive hypothesis}] \\
&= map\ (F \bullet G)\ (xs +\!\!+ ys) && [map.2]
\end{aligned}
$$

– and it doesn't take any more lines to do so. Section 5 below indicates how Jape might be extended to represent equational proofs in this way.

```
1: []=[]                                                          = reflexive
2: map F[]=[]                                                     Fold map'0 1
3: map F(map G[])=[]                                              Fold map'0 2
4: map F(map G[])=map(F•G)[]                                      Fold map'0 3
5: [F(G x1)]=[F(G x1)]                                            = reflexive
6: [F(G x1)]=[(F•G)x1]                                            Fold • 5
7: map F[G x1]=[(F•G)x1]                                          Fold map'1 6
8: map F(map G[x1])=[(F•G)x1]                                     Fold map'1 7
9: map F(map G[x1])=map(F•G)[x1]                                  Fold map'1 8
10:  map F(map G xs)=map(F•G)xs ,  map F(map G ys)=map(F•G)ys     assumptions
11:  map(F•G)xs++map(F•G)ys=map(F•G)xs++map(F•G)ys                = reflexive
12:  map(F•G)xs++map F(map G ys)=map(F•G)xs++map(F•G)ys           Fold with hypothesis 10.2,11
13:  map F(map G xs)++map F(map G ys)=map(F•G)xs++map(F•G)ys      Fold with hypothesis 10.1,12
14:  map F(map G xs++map G ys)=map(F•G)xs++map(F•G)ys             Fold map'2 13
15:  map F(map G(xs++ys))=map(F•G)xs++map(F•G)ys                  Fold map'2 14
16:  map F(map G(xs++ys))=map(F•G)(xs++ys)                        Fold map'2 15
17: map F(map G x)=map(F•G)x                                      listinduction 4,9,10-16
18: (map F•map G)x=map(F•G)x                                      Fold • 17
19: map F•map G=map(F•G)                                          ext 18
```

Figure 9: a complete equational proof

4. Forward proof by hiding cuts

The proof in figure 6 would hardly be *natural* natural deduction if it had to be made forward. Fortunately, Jape is able to simulate forward proof in many circumstances. We begin with a trivial example, requiring two forward steps:

```
1: P→Q,  Q→R,  P   assumptions
   ...
2: R
```

The first step is to select one of the hypothesis formulæ to be worked on. In this case it is reasonable to select P→Q:

```
1: P→Q,  Q→R,  P   assumptions
   ...
2: R
```

Then we apply the →-E rule. The user sees a forward step, using the formula that has been selected:

```
1: P→Q,  Q→R,  P   assumptions
2: Q               →-E 1.3,1.1
   ...
3: R
```

Jape has automatically picked up the hypothesis *P* to complete the step. We note in passing that the slogan 'select the formula you want to work on and apply a rule' appears to be readily grasped by novices.

If we now select the other implication formula in the assumption line we can apply the same rule again, completing the proof:

```
1: | P→Q,  Q→R,  P |  assumptions
2: | Q             |  →-E 1.3,1.1
3: | R             |  →-E 2,1.2
```

It is possible to construct the same proof display, but in the other order. If we select Q→R first and apply →-E the effect is apparently to make a backward step:

```
1: | P→Q,  Q→R,  P |  assumptions
   | ...           |
2: | Q             |
3: | R             |  →-E 2,1.2
```

Selecting P→Q in this state, and then applying →-E, generates what is apparently the same completed proof as before.

It has to be stressed that the mechanism which supports this activity – apparent choice between forward and backward steps, apparent 'intelligent' choice of proof direction – is utterly mechanical and utterly devoid of intelligence. It is simply a matter of hiding lines generated by cut and hyp, and automatically generating hyp steps whenever it is obviously possible to do so.

Consider the first proof step above: forward from P and P→Q to Q. It is shown in box display as a single line, but behind the scenes there is much hidden activity. In tree form this is the true picture:

$$\dfrac{\dfrac{\dfrac{}{P\to Q,\ Q\to R,\ P \vdash P}\ \text{hyp} \qquad \dfrac{}{P\to Q,\ Q\to R,\ P \vdash P\to Q}\ \text{hyp}}{P\to Q,\ Q\to R,\ P \vdash Q}\ \to\text{-E} \qquad P\to Q,\ Q\to R,\ P,\ Q \vdash R}{P\to Q,\ Q\to R,\ P \vdash R}\ \text{cut}$$

and in box form, with hyp lines hidden but cuts displayed, Jape shows the same proof as:

```
1: | P→Q,  Q→R,  P |  assumptions
2: | Q             |  →-E 1.3,1.1
3: |  | Q          |  assumption
   |  | ...        |
4: |  | R          |
5: | R             |  cut 2,3-4
```

In proof terms the rightmost of the two hyp steps in the tree above moves P→Q from left to right; the →-E step generates the formula Q; and finally the cut step moves conclusion Q from right to left, making it available for use in the rest of the proof. Working backwards, all Jape has to do is to apply the sequence cut, then →-E, then hyp on the right-hand subgoal; afterwards the automatic application of hyp will close the other subgoal of →-E.

It should be clear how the 'normal' – that is, deceitful, cut-hiding – display is generated from the fuller version. In the full display above there are two lines carrying the formula Q; one of them is a completed conclusion line and the other is a hypothesis line. At the same time there are two lines carrying the formula R, one an unproved conclusion and the other proved by cut. The conclusion Q is now proved, so we don't have to do any more work on it, and it can't be called into service to make any more proof steps because it is a conclusion line; that line is useless to the further development of the proof. The proved conclusion R is useless in exactly the same way. By conflating the two Qs, and labelling the conflation with the reason of the conclusion line, we introduce a hypothesis which can be used to prove the conclusion R. At the same time we conflate the two Rs, and treat the conflation as an unproved conclusion.

Exactly the same sequence of steps, but this time using the other hypothesis formula, produces a different tree and an apparent backward step:

$$\dfrac{\dfrac{\dfrac{}{P\to Q,\ Q\to R,\ P\vdash Q}\ \text{hyp} \qquad \dfrac{}{P\to Q,\ Q\to R,\ P\vdash Q\to R}\ \text{hyp}}{P\to Q,\ Q\to R,\ P\vdash R}\ \to\text{-E} \qquad \dfrac{}{P\to Q,\ Q\to R,\ P,\ R\vdash R}\ \text{hyp}}{P\to Q,\ Q\to R,\ P\vdash R}\ \text{cut}$$

This time the box display with cuts displayed is even more confusing:

```
1: | P→Q,  Q→R,  P |  assumptions
   | ...            |
2: | Q              |
3: | R              |  →-E 2,1.2
4: |  | R |         |  assumption
5: | R              |  cut 3,4
```

The only difference in this example is that the automatic application of hyp has closed the rightmost subgoal of cut, leaving the leftmost subgoal of →-E as the problem to be solved, and this time we have to conflate three occurrences of R. The box-display mechanism carries out this task by a straightforward recursive traversal of the tree.

Unfortunately, we can't always hide cuts in this way: if we trace through the steps that Jape takes when carrying out the tactic that is bound to the menu entry labelled →-E, we can see immediately why this is so. The first step, working backwards, is to apply cut:

```
1: | P→Q,  Q→R,  P |  assumptions
   | ...            |
2: | _B             |
3: |  | _B |        |  assumption
   |  | ... |       |
4: |  | R |         |
5: | R              |  cut 2,3-4
```

Jape can't conflate the two unknown _B formulæ: to do so would make a line which had to be both conclusion (for the step or steps which lead to line 2) and hypothesis (for the step or steps which lead from line 3 to line 4). Jape's user interface gesture mechanisms wouldn't allow the prover to work in that situation. We discuss below some ways in which we might be able to solve this problem.

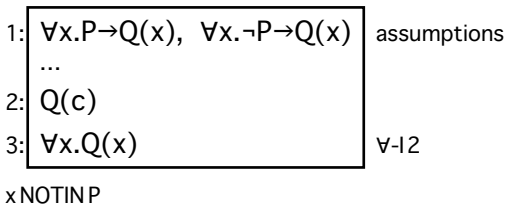
The next step is to apply →-E. Now the cut can be hidden, because one of the _B lines is a proved conclusion, and thus that part of the conflated whole is redundant:

```
1: | P→Q,  Q→R,  P |  assumptions
   | ...            |
2: | _A             |
   | ...            |
3: | _A→_B          |
4: | _B             |  →-E 2,3
   | ...            |
5: | R              |
```

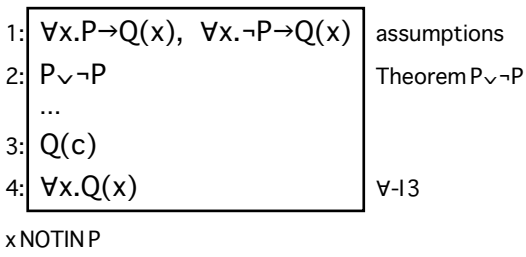
Now if we unify line 3 with P→Q, most of the proof disappears: line 2 becomes P, is proved by hyp and disappears; line 3 is proved by hyp and disappears; line 4 becomes Q and remains. If we unify line 3 with Q→R: line 2 becomes Q and remains, line 3 is proved by hyp and disappears; line 4 becomes R, so line 5 is proved by hyp and disappears.

The way in which Jape's tactic language supports this kind of helpful deception is discussed in [3].

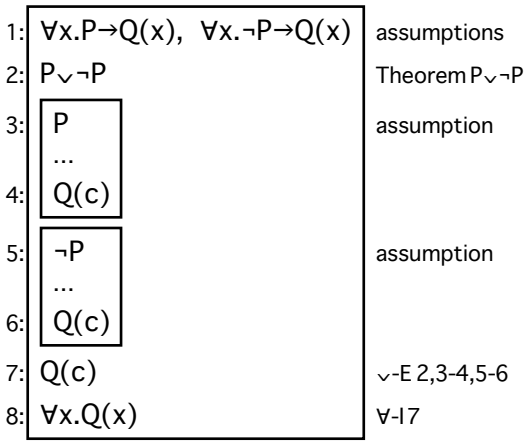
The way in which Jape supports a mixture of forward and backward proof is illustrated by a version of the example of figures 1-4 . The first step is backwards:



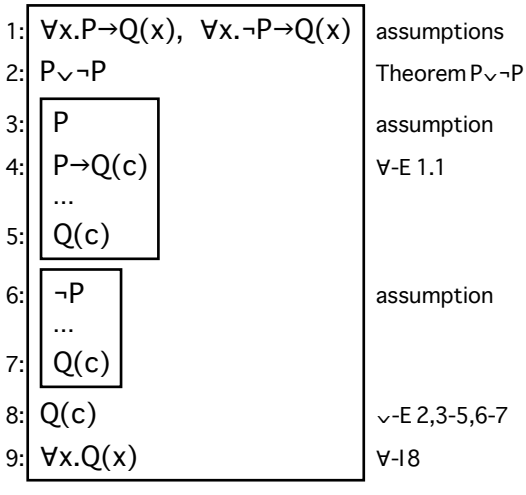
The next step is forward – in effect, a cut with a previously-proved theorem[1]:



Forward again:



and again:



and so on until the proof is complete as before. This style of proving is the closest we have got, so far, to truly natural deduction.

---

[1] Jape's theorems are proved schematically, so can be used as derived rules. In this particular case the theorem and its instance are identical..

5. True forward proof: a user-interface problem

As illustrated above, Jape cannot hide all cuts in its box style. The reason is that it is necessary to translate the user's gestures towards the box display into equivalent gestures towards the tree which is Jape's internal representation of the developing proof. Each line in the proof is either a conclusion line or a hypothesis line, and we use that fact to guide the tactics which decide whether to make a pseudo-forward or a truly-backward step.

The result is that in Jape a line may *not* always be used in subsequent proof steps, even though the proof structure would allow it: we can't combine the two A formulæ, for example, in

```
  ...
1:A
  ...
2:A
3:A∧A   ∧-I1,2
```

In such circumstances, Jape will require the user to make two proofs[1]. The fact that the underlying structure is a tree appears to be poking out, frustrating our attempts to hide it behind the box display.

But the truth is otherwise: it's a user-interface issue, not a proof engine problem. We can imagine how we might construct our display mechanism, using and hiding cuts, so that every line in the proof was in fact a hypothesis line for those that followed. Instead of applying a rule like ∧-I backwards, we would cut with its first subgoal, then its second, then apply the rule, then close its subgoals with hyp:

```
  ...
1:A
2:| A          assumption
  | ...
3:| B
4:| | B        assumption
5:| | A∧B      ∧-I2,4
6:| A∧B        cut 3,4-5
7:A∧B          cut 1,2-6
```

Clearly line 2 can be used as a hypothesis in a proof of line 3, and line 4 could be a hypothesis for anything that followed line 5. The cut on line 6 can be hidden because lines 3 and 4 can be conflated: there is no need to preserve the rôle of line 4 as an hypothesis, since it has no live conclusions. But we can't conflate lines 1 and 2, so the best that Jape can do at present is

```
  ...
1:A
2:| A          assumption
  | ...
3:| B
4:| A∧B       ∧-I2,3
5:A∧B          cut 1,2-4
```

Actually there's more to the problem than this. When we make a forward step, we introduce a formula into the proof by a cut. The position at which we introduce the formula matters: at present we require that the user indicates a position by selecting an open conclusion formula. If we begin to allow arbitrary cuts it seems reasonable that they might be moved around in the proof: out of a box, for example, so that an intermediate result becomes more widely available. In the same way, the order in which the antecedents of a backward step are written might matter, and the user might want to alter it.

---

1 You need only one proof if you work forwards, establishing *A* first and then calling upon it twice to prove *A*∧*A*.

If we allow arbitrary cuts to be made in a proof, we have opened a vista to proof editing and transformation which Jape's present mechanisms cannot support. We can make some progress, however, by considering the user interface solutions which might allow Jape to hide all cuts.

One solution might be to refine the treatment of selection. At present Jape has two kinds of selection: formula-selection, illustrated in this paper, is used to point out principal terms and to indicate a position in the tree at which action should take place; character-selection is used to select subformulæwhich can then be arguments to rules or can indicate a way of making a selection form out of a formula[1] . If formula-selection were modified, so that in ambiguous cases the user could choose between 'conclusion selection' and 'hypothesis selection', then it would be possible to conflate lines 1 and 2 in the proof above. That would be an ugly mechanism, however, and it seems to us to go against our aim to give Jape a 'quiet' interface in which every action has a simple, easily explained semantics and in which there is not very much to learn. Novices at present have difficulty with two kinds of selection given the possibility of selecting something in a hypothesis or a conclusion; three kinds of selection might very well be too much.

It might be possible, on the other hand, to use Jape's tactic language to discriminate more carefully, to infer the purpose of a selection. Suppose, for example, that we decided that elimination rules *must* be applied forward and introduction rules backward. Then if we select an ambiguous line and apply an elimination rule, we must be working forwards, using its assumption component; if an introduction line, we must be working on its conclusion component. It is not yet clear how well that mechanism would work: we might in any case have to use formula-selection to indicate where in the proof the step should take place. In some logics there are elimination rules which are sometimes used backward – ¬-E,. for example, in the natural deduction encoding above[2].

Neither of these mechanisms, at first sight, seems to offer an attractive mechanism to support the use of arbitrary cuts. We still have to investigate them in detail, though: it may be that they contain the germ of a solution to our problem.

It would not do to discriminate on the form of hypotheses and conclusion formulæ: Jape can already do this, but we don't use the results to direct the course of the proof. The reason isn't far to seek. Suppose, for example, that we did conflate lines ambiguously, and suppose that we decided that, when applying an elimination rule, we would work forward if the selected formula was an implication formula, backward if not. There are two circumstances in which that discrimination would be misleading: if the formula is an unknown, we can't decide whether it is an implication or not; if we want to use implication-elimination on some other formula to prove a conclusion formula which is itself an implication, the form of the conclusion is misleading.

6. Transitive reasoning by hiding transitivity steps

Jape hides cuts in its box display by conflating lines. It should be possible to extend that technique to cover equational and transitive reasoning. There is an obvious analogy between cut and transitivity on the one hand, and between hyp/axiom/identity and reflexivity on the other:

$$\frac{\Gamma \;\; C \quad \Gamma, C \;\; A}{\Gamma \;\; A}\; cut \qquad\qquad \frac{\Gamma \;\; X = Y \quad \Gamma \;\; Y = Z}{\Gamma \;\; X = Z}\; = transitive$$

$$\frac{}{\Gamma, A \;\; A}\; hyp \qquad\qquad \frac{}{\Gamma \;\; X = X}\; = reflexive$$

Cut moves a formula from left to right of the turnstile; transitivity moves a formula from left to right of equality. Hyp/axiom/identity and reflexivity close a proof by matching a formula from left and right-hand side. The mechanisms which hide cut and identity ought to be easily adaptable to hide transitivity and reflexivity.

---

1 Jape really ought to have proper subformula-selection in place of character selection, and it will happen Real Soon Now.

2 It won't do to criticise this encoding. For sure, that particular rule is not very convenient, and another presentation would not have the difficulty: but it is taken from a respectable logic textbook, it is the encoding that a particular encoder wants to use, and Jape has to be able to cope. We would not be satisfying our design aims if we only treated sensible logic encodings.

By using a different rewrite rule:

$$\frac{\Gamma \quad X = Y}{\Gamma \quad A(X) = A(Y)}$$

we could ensure that all steps of equational reasoning use the transitivity rule, just as all forward steps at present use cut; then a transitivity-hiding display mechanism would do most of the work.

Despite the simplicity of the insight, it isn't yet completely obvious how to modify Jape's pointing and display mechanisms to cope with this proposal. It isn't clear, for example, whether equality problems would be displayed as one line or two, whether the first or last line of an equality chain should carry one formula or two. It is obvious that when the antecedent of the rewrite rule is a simple 'fact', it ought to be hidden; it isn't obvious what ought to be displayed in other cases.

7. There is much more

Jape is a user interface project: our aim is to construct an interface which makes proof steps transparent and straightforward. We have already demonstrated that some proof activity can be carried out by point-and-click actions using the vocabulary of existing high-quality user interfaces. Jape is already simple enough to be given to complete novices.

Our aim is to make Jape simpler still, by expanding its internal mechanisms to cope with more of the sort of proof actions that our users might think are obviously possible. The aim is not to "expand the functionality" of our interface by adding more windows and more buttons, nor is the aim to automate more of Jape's actions to make it more attractive to experts. Our aim is, rather, to display information about the progress of a proof in such a way that a very simple vocabulary of gestures is obviously enough to make the proof steps that its users, novice or expert, wish to make. Our treatment of naive set theory (not illustrated in this paper) is a step in that direction, combining forward proof and substitution of equivalent formulæ. The steps toward a more appealing presentation of equational reasoning outlined above should be another.

We can see, at present, two major ways in which this research can be continued. One, alluded to above, is to refine our treatment of forward proof, and in doing so to investigate some ways of proof editing and transformation. Another is to improve our treatment of 'families of rules': rules which are schematic in their structure (numbers of antecedents, number of components in consequent, etc.). Jape can at present deal with such families but it requires some tricky tactic-language programming and it is limited in its effectiveness.

References

1. Bornat R and Sufrin B (1996) *Jape's quiet interface* . Presented at UITP 96, University of York, August 1996; available from ftp://ftp.dcs.qmw.ac.uk/programming/jape/papers/
2. Bornat R and Sufrin B (1997) *Jape: a calculator for animating proof-on-paper*. System description to be presented at CADE-14, 1997.
3. Sufrin B and Bornat R (1996) *User Interfaces for generic proof assistants, part 1, interpreting user gestures*. Presented at UITP 96, University of York, August 1996; available from http://www.comlab.ox.ac.uk/oucl/users/bernard.sufrin
4. Fitch F.B. (1952) *Symbolic Logic*, Ronald Press, New York
5. Dawson W.M.G. (1990) A Generic Logic Environment, PhD thesis, Imperial College, University of London
6. Dyckhoff, R. (1987) *Implementing a simple proof assistant*, in Workshop on Programming for Logic Teaching, Leeds, July 1987, Centre for Theoretical Computer Science, University of Leeds (program available FROM Machine Assisted Logic Teaching Project, Computational Science Division, University of St Andrews)
7. Jones C.B. et al (1985) *mural: A Formal Development Support System*. Springer-Verlag
8. EPC Ltd (1992) *The B Tool User Manual*, EPC Ltd, 17 Alva St. EDINBURGH EH2 4PH