Rendering Tree Proofs in Box-and-Line Form

Richard Bornat[1]

School of Computing Science, Middlesex University. R.Bornat@mdx.ac.uk

Abstract. Some recent improvements in the design of Jape, a proof calculator, are described. Jape's internal data structure is a sequent tree, but it now supports an accurate box-and-line treatment of natural deduction. Changes to its internal workings, to its tactic language and to user interaction are described.

1 Background

In 1998 the proof calculator Jape[3, 8, 9, 1] was assessed by an educational researcher, as part of a project on visualisation[2]. It had been in existence for about eight years, and under constant development. Nevertheless, the effect of evaluation was salutary – as evaluations so often are – and led to several significant changes in the external and internal machinery of Jape.

First, it became clear that novices, unlike experts, don't benefit from the freedom to defer choices. The sort of 'exploration' offered to those learning formal proof is quite difficult enough when restricted to choices about which rule to apply and when. Jape's use of unknowns to defer the specialisation of a logical step (usually an unknown argument in a substitution) is a piece of meta-machinery which derails a novice's fragile understanding. Couple this with an awkward use of text substitution as a device to smuggle additional arguments past Jape's point-and-click primitivism, and few undergraduates could get beyond simple propositional examples. In one heartrending videotaped session a student wailed that she didn't know why 'it always does this' when she used a ∀ elimination rule (figure 1).[1]

So much meta-logical internal machinery is showing that it is no wonder she despaired. An entire section of the manual distributed to first years was described procedures for eliminating unknowns once they had appeared, and a subsection was dedicated to strategies for avoiding them in the first place. In this case the manual suggested that you should use multiple text selections and unification to replace *_c1* by *c*, or text-select *c* to provide it as a second argument in the rule step. In my course on GUI design I taught students that help text is an admission of defeat. Manuals are an even worse dereliction of duty, and I finally learnt to take my own medicine. To solve the problem I had to implement multiple antecedent selection in the GUI module, interpret it in Jape's tactic

[1] Figures 1 and 2 have been generated from a modern version of Jape. They suggest what was seen in 1998, but the reality was far worse.

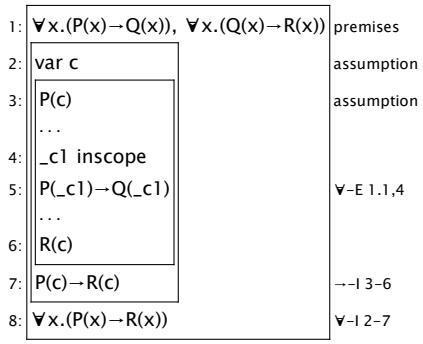

Fig. 1. An incomplete ∀ elimination step

language, and modify the logic so that certain steps became unnecessary (which last I wanted to do anyway, for other reasons). The benefit was that proof search strategies became more explicable in terms of the logic, and explanations had less to do with the difficulties of instructing Jape to do the right thing.

Second, those videotaped students showed that my advice to novices to read all Jape's error messages as "bang!!" wasn't working. They tried to understand what Jape said to them, and were understandably demoralised when they couldn't. Some error messages are generated by simple slops: many of us, for example, sometimes ask for an elimination step when we wanted an introduction step; the mistake is encouraged by the partical identification of introduction steps as 'backward' and elimination steps as 'forward'. In one session a pair of novices were confronted with an error dialogue box which looked something like figure 2. The dialog box used language which they had barely encountered to describe internal Jape objects that they couldn't see on the screen. It was designed to suit the needs of an expert logic encoder working in the sequent calculus, or able to envisage the internal workings of Jape behind a box-and-line proof display. To solve this problem I had to write lots of special tactics that tease apart possible causes of error and construct individual error messages appropriate to each different situation. In particular it was necessary to reorganise the menus, so that error messages which talked about backward and forward steps related to visible elements of the interface. And it meant more changes to the logic, particularly to the treatment of negation and contradiction, to remove some unnecessary causes of error.

These two changes – eliminating incomplete steps, improving error reporting – addressed the needs of novices learning a particular logic. They took a considerable amount of particularly focussed effort. Unfortunately, other logic encodings couldn't immediately benefit without similar expenditure of effort (Jape's tactic language doesn't allows much code reuse!).

The third significant problem affected every user, because it was about the correct display of box-and-line proofs. Simply, Jape didn't display

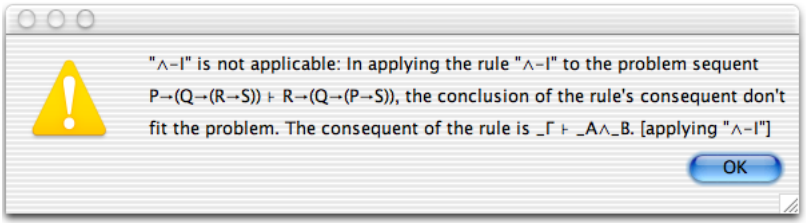

Fig. 2. An inappropriate error message

true box-and-line proofs, because you couldn't always call on earlier in-scope lines in later deductions. There were certain rules of thumb which an encoding expert could use to reduce the problem, like making elimination steps before introduction steps, but they didn't always work and in any case imposed a significant and unnecessary planning load. My Jape publicity claimed that you could choose what to do and where to do it, but it didn't say that innocent choices could have unjustifiable consequences. The problem needed fixing: it was already a problem for me as an expert user, and it would bite the novices once the other obstacles had been cleaned away. The complexities of the fix are described below. The final problem was gestural. Jape made no visible distinction between 'hypothesis' and 'conclusion' formula selections, although to command Jape it's necessary to observe and exploit the distinction. Jape now displays these diffferent selections differently, and this facilitates true forward steps (see below) which allow a line of the proof to be used either as hypothesis or as conclusion.

2 Rendering Tree Proofs in Box-and-Line Style

Jape's basic box-and-line proof rendering mechanisms are dealt with elsewhere [6, 8], and I give merely a summary. Sequent tree proofs make large displays, partly because the same left formulae are repeated over and over again in the tree and partly because of branching of the tree. For example, figure 3 is a tree proof in a natural-deduction style sequent calculus (introduction and elimination rules plus an identity rule $\Gamma, A \vdash A$, here called `hyp`). The same proof can be shown far more compactly as figure 5. Jape's rendering algorithm converts the tree into the compact box-and-line display in four stages.

2.1 Stage 1: Linearise

Jape renders each tree node as a sequence of boxes and lines. The linear rendering of figure 3 is shown in figure 4. If a node has more left formulae in its consequent than its parent has (the root and the node above it in figure 3, for example) then it is rendered as a box starting with the extra left formulae labelled as assumptions or premises. Otherwise it generates a sequence of the renderings of its subtrees, followed by a line containing

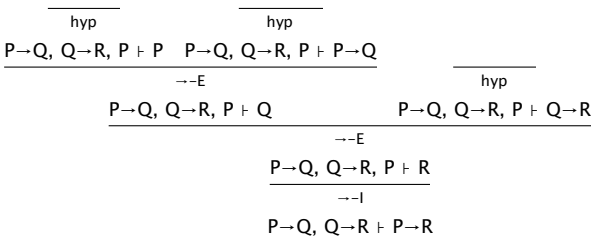

Fig. 3. A wide tree proof

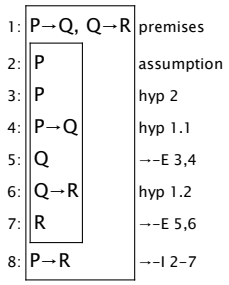

Fig. 4. A linearised proof

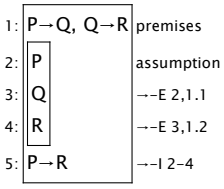

Fig. 5. A compact box proof

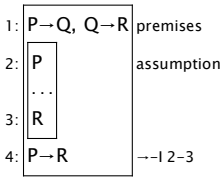

Fig. 6. First step of the proof

the consequent formula labelled with the name of the rule that generated the node. Tips generate a single unlabelled line.
Thus the root of figure 3 generates a box starting on line 1 of figure fig:boxproofwithhyp, followed by the rendering of its subtree (lines 2-7), followed on line 8 by the root formula labelled with →-I. The root's subtree is also rendered as a box: line 2 followed by its left subtree (lines 3-5), its right subtree (line 6) and line 7 labelled with →-E. Each consequent line refers to the subtrees either as *i*-*j*, the first and last line numbers of a subtree box, or *j*, the last line number of a sequence of lines. Rules which match a left formula (in this logic the only such rule is `hyp`) refer to an occurrence in the premises or assumptions. All these effects can be seen in the example, and it's easy to reconstruct the tree from its linearised form.

2.2 Stage 2: Hide Identity Lines

Lines 3, 4 and 6 of figure 3 are no more than indirections. Jape normally elides these lines, replacing references to them by references to the premise or assumption to which they refer. This converts figure 4 into figure 5. Most identity steps are carried out behind the scenes anyway, and they aren't really part of the natural deduction narrative, so the elision makes a lot of sense from the prover's point of view. After this stage it's only a little more difficult to reconstruct the tree from the linearised elided form.

2.3 Stage 3: Hide Cut Steps to Allow Forward Reasoning

Figure 6 shows how a user might make the first step in a proof. The tree produced is just the last three lines of figure 3. If the next step is backward, producing the lower five lines of figure 3, then the display is as shown in figure 7. But most novices would prefer to make a forward step, producing figure 8. The forward step is indicated by selecting a hypothesis formula – in this case $P \rightarrow Q$ – and invoking an appropriate rule.

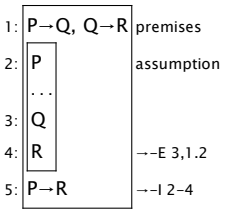

Fig. 7. A backward step from figure 6

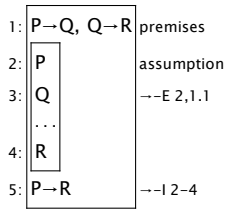

Fig. 8. A forward step from figure 6

Forward steps generate a new formula which can be called upon by deductions below it – that is, a new hypothesis formula. That's only possible if the new formula is a left formula. In a natural deduction style

logic, the only way to introduce a new left formula is to use the cut rule $(\Gamma \vdash A; \Gamma, A \vdash B) \Rightarrow \Gamma \vdash B$. Behind the scenes, figure 8 uses the tree of figure 9; the stage 2 rendering of that tree is shown in figure 10. To produce figure 8, Jape conflates the first antecedent of the cut with the assumption line, the conclusion of the second antecedent with the conclusion of the cut, elides the box, and adjusts the labelling of lines.

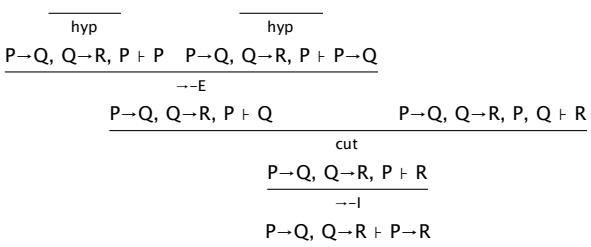

Fig. 9. A tree with a forward step

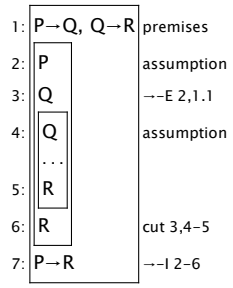

Fig. 10. A linearised cut step

2.4 Stage 4: Deal with Transitive Operations

In dealing with chains of reasoning such as $A = B$, $B = C$, ... $Y = Z$, it's possible to treat reflexivity $A = A$ and symmetry $A = B \Rightarrow B = A$l like logical identity and treat transitivity $(A = B; B = C) \Rightarrow A = C$ as a kind of cut, generating a sequence of lines $A =$, $= B$, $= C$, ..., $= Z$. Jape's rendering mechanism can do this, but it isn't directly relevant to this discussion.

2.5 What's Wrong with This Mechanism?

The rendering mechanism goes wrong in stage 1. In a box-and-line proof it's possible to use any previous line (boxes permitting) when making a

step: that is, the underlying structure is a DAG. In a tree, on the other hand, you can't refer to sibling subtrees. When a tree node is rendered, therefore, although the lines representing the deductions of subtree n come before the lines of subtree $n+1$, they must all be inaccessible to the lines of the sibling subtree below. In multiplicative (context-splitting) logics the inaccuracy is even worse, because the left formulae – shown as premises and assumptions – accessible from one subtree will not be the same as those accessible from another subtree, and the box-and-line rendering doesn't show the difference.

The problem is illustrated, and there is a hint of its possible solution, in the trees of figure 11 and 12. These different trees each generate the same box-and-line display, shown in figure 13. To generate figure 11 the first step is $\wedge$ introduction, and then a new left formula Q is generated in the left subtree by a forward $\rightarrow$ elimination step, closing that subtree. If the proof is to be completed, the same forward step will have to be duplicated in the second subtree. To generate figure 12 the first step is a forward $\rightarrow$ elimination, which generates a left formula Q usable by the whole of the proof. The next step is $\wedge$ introduction; the left formula of the cut closes the left subtree, and it's also available to the right subtree.

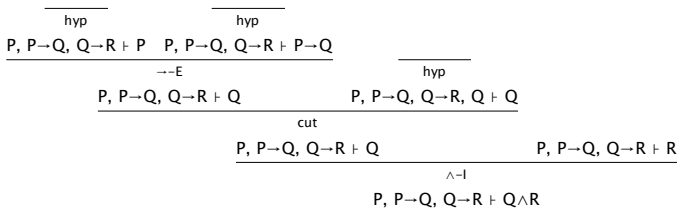

Fig. 11. Intro then elim

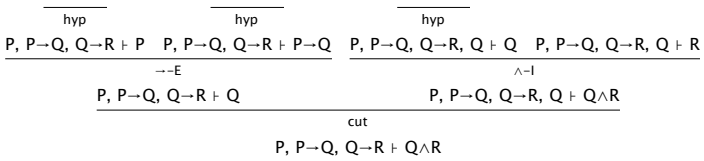

Fig. 12. Elim then intro

Figure 13 isn't quite as ambiguous as it seems. From the earliest days of Jape, Bernard Sufrin and I used a greying-out mechanism to show which left formulae were visible in a box-and-line proof but not accessible. If the tree behind figure 13 is figure 11, for example, then selecting line 3 will grey out line 2; if the tree is 12, line 2 will be undimmed. The manual advised users to select a position in the tree by clicking on an

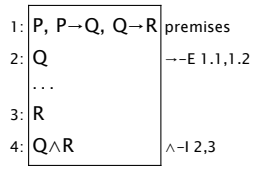

Fig. 13. An ambiguous display

active consequent – a formula below a line of three dots – and observe which formulae above the consequent remained visibly active and which were greyed out. If the results broke the rules of box-and-line proofs then novices perhaps thought it no more surprising than anything else Jape did, experts perhaps could explain it by thinking of the underlying working of the tree, and probably everybody was too grateful for the display of proof context to complain about its inaccuracies. At any rate I don't recall much criticism of our treatment.

This was the state of Jape for three or four years: inaccurately rendered proofs, and imperfectly executed forward steps. If the problem suited it, and if you were prepared, as most logical novice users are, to make all possible forward steps before any backward step,[2] then the misrepresentation of tree proofs seemed much less of a problem.

The problem isn't caused or made worse by the introduction of forward steps into a backwards-reasoning sequent-tree calculator. On the contrary, forward steps provide a way to increase efficiency, by choosing to do as much as possible by forward reasoning early on in the proof. Jape's deficiencies are perceived as inefficiency of the user's proof strategy caused by poor planning: that is, Jape is imposing a planning load on its users.

3 True Forward Steps

The first step towards a solution of the problem is to deal properly with forward steps. If a forward step always produces a tree like figure 12, never like figure 11, whichever order the forward and backward steps are undertaken, then at least one part of the problem is solved. In the latest versions of Jape, forward steps can insert a cut node at the lowest point possible in the tree. This means that, for the first time, users can make steps which are not directed towards a conclusion-tip. For example, the partial proof of figure 14 can evolve into figure 15, inserting a hypothesis formula directly below the line it derives from. This is a true forward step.

To make Jape take the step required some new internal mechanisms. Previously, the tactic which implemented forward steps was

[2] Novices believe with great fervour, no matter how they are instructed, in the unique efficacy of forward reasoning. They resist backward reasoning with the same conviction. I've concluded that they think backwards steps are cheating.

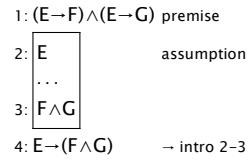

Fig. 14. Before forward step

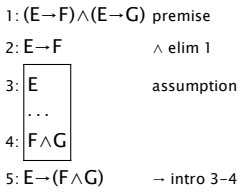

Fig. 15. After forward step

```
TACTIC ForwardCut (n,rule)
    SEQ cut
        (LETGOALPATH G (WITHARGSEL rule)
                       (GOALPATH (SUBGOAL G n))
                       (WITHHYPSEL hyp)
                       (GOALPATH G)
                       NEXTGOAL)
```

– apply cut; record the current position (left subgoal of the cut) by binding its tree path to G; make the user's proof step; move to the nth subgoal of the rule step; using the user's hypothesis selection, close with a selected left formula; move back to position G; look for the next unclosed tip using right-to-left recursive traversal of the tree.
The new tactic is superficially similar:

```
TACTIC ForwardCut (n,rule)
    CUTIN (LETGOALPATH G (WITHARGSEL Rule)
                         (GOALPATH (SUBGOAL G n))
                         (WITHHYPSEL hyp)
                         (GOALPATH G))
          )
```

Like all Jape tactics, `ForwardCut` executes at a position in a proof tree. To begin, the tree position is defined by the user's formula selections: a conclusion selection defines a tip; a hypothesis selection defines the lowest point in the tree at which it is present as a left formula; if there's more than one selected position then the greying-out mechanisms guarantee that they are all on the same path in the tree, and Jape uses the highest. Then the `CUTIN` operation, which can only be used if the logic has a suitable cut rule and has been specified in `ADDITIVELEFT` style, looks for the lowest point in the tree which has exactly the same left formulae as the selected position. It breaks the tree there, tops the break with a cut node using the old top section as its right subtree, and augments every left context in the old top section with the cut formula, generating new provisos where necessary.[3] Then it executes its argument tactic at the left subtree of the new cut node, and finally it returns to the point in the tree at which it was originally applied. The effect is to splice a forward step into the tree.

[3] In Jape's rule notation `FRESH c` requires that the name `c` can't appear free in left or right formulae of the consequent of a step which uses a rule. Nodes at which `FRESH` has been applied are marked, and `CUTIN` must add a proviso for the formula it is inserting, each time it encounters such a node.

The complication is navigation: Jape tactics navigate by storing tree paths (`LETGOALPATH` and `GOALPATH`, for example, in the tactics above), which are invalidated if a new node is inserted into the tree somewhere along the path. Rather than attempt to search for and correct all stored paths, I implemented a path-following algorithm which ignores inserted cut nodes unless they are explicitly mentioned in the path formula. (Easy to say, horrible to implement.)

4 Building a DAG

Forward steps are only half of the problem. Multi-antecedent backward steps are the other half. In Jape's rendering mechanism, a formula on a previous line is only accessible if it represents an immediate antecedent or it represents a left formula (a hypothesis, in the language we use to talk about Jape proofs). The lines and boxes describing subtree n aren't direct antecedents of anything in subtree $n+1$. There is nothing else for it: they have to be hypotheses; that is, they have to be introduced by cut steps. And then, by induction, it's clear that *every* backward step has to generate new hypotheses. This apparently absurd situation is the only solution to the problem that fits the bill.

The way to generate new hypotheses, low enough down in the tree that they can be accessed by lines which might need them, is of course to use `CUTIN`. The tactic which is behind the menu entry for $\wedge$ intro, for example, is

```
TACTIC "∧ intro backward" IS SEQ "∧ intro" fstep fstep
TACTIC fstep IS
    ALT (ANY (MATCH hyp))
        (trueforward SKIP)
MACRO trueforward(tac) IS
    LETGOAL _A (CUTIN (LETGOAL _B (UNIFY _A _B)
                                  tac))
               (ANY (MATCH hyp))
```

– apply the $\wedge$ intro rule, then apply the `fstep` tactic to each of its antecedents. The `fstep` tactic checks that the tip it is applied to can't be closed by an identity step (if it can, there's no need to introduce a new left formula) and then applies `trueforward`; that tactic records the consequent of the tip it is applied to (`LETGOAL`); then runs `CUTIN`, unifying the new cut formula with the original consequent and applying the argument tactic (in this case simply `SKIP` to the left subtree of the cut; finally, back at the original position in the tree, closes the tip with an identity step.

The effect, since this mechanism is used for *every* backward step, is to close every backward step by identity with a left formula generated lower down in the tree. All the identity steps will be invisible in the box-and-line display, because they are elided in stage 2 of the rendering algorithm; the lower position will be above the positions which inserted hypotheses visible to the backward step; stage 3 of the rendering algorithm, applied recursively, exposes hypotheses in lowest-first order; it follows that the lines of backward steps, if `fstep` is uniformly applied at every stage,

appear in the correct order. In effect, the antecedents of a backward step are inserted below and push the lines above them upwards, rather than growing the proof above the step itself.
The total effect, if we think of cuts as augmenting an environment and rule steps as generating nodes, is to generate a DAG. A cut which introduces a left formula A in effect produces a tree which is described by the formula "let $line$ = $treeA$ in $tree$" where $tree$ can refer as many times as necessary to $line$. This is a description of the spanning tree of a DAG. It seems inside out, and it is odd to use the proof tree in this way, but it does work!

5 Pointing to Formulae

In a sequent, left formulae are separated from right formulae by the turnstile, so that a selection-click on a left formula is spatially and obviously distinct from one on a right formula. Jape's tactic language was built from the first to allow discrimination between left (`LETHYP`) and right selections (`LETCONC`). When Bernard Sufrin and I first implemented box-and-line display of tree proofs, we retained the same distinction, since left formulae were labelled premise or assumption, and right formulae of tips were unlabelled and below a line of dots.
Before the developments described in this paper, stage 3 of the rendering mechanism hid two sorts of cut steps: one where the cut formula was a hypothesis alone, with no lines above it that could use it as a target conclusion; the other where the cut formula was a conclusion, with no lines below it that could call upon it as hypothesis. All formulae in the display, whether generated normally or by forward steps, could then be classified as hypothesis or conclusion, and clicks on them treated appropriately as left or right selections. This meant that the rendering mechanism couldn't deal with raw cut steps – ones which generated a formula that was a possible bridge between hypotheses and conclusion, but not yet connected to either – because it wasn't possible to classify the occurrence of the cut formula as purely hypothesis or purely conclusion. Under this rendering mechanism active conclusion formulae appeared below a line of dots, and hypotheses were those formulae above an active conclusion which didn't grey out when the conclusion was selected. All very well, provided that you clicked on a conclusion before every step, but novices didn't do that very often (remember, they prefer to reason forwards). Some more emphatic distinction seemed necessary, and even before implementing forward steps I began to make hypothesis selections – boxes open downwards – visibly distinct from conclusion selections – boxes open upwards. Also, following Bernard Sufrin's lead from his implementation of the Jape GUI module for Linux, I made the selection boxes red as shown in figure 16.
True forward steps made it essential to deal with raw cuts. In an $\wedge$ intro step, for example, the left-hand formula can be used as a conclusion to be proved from the lines above it, as in figure 17, or a hypothesis to be used in the lines below it, as in figure 18. As these examples show, Jape's rendering mechanism can now hide all cut steps, and interprets

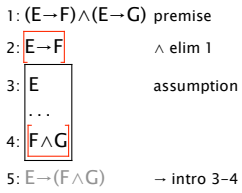

Fig. 16. Directed selection boxes

clicks on the top half of an ambiguously-usable cut formula as conclusion selection (upward-pointing), on the bottom half as hypothesis selection (downward-pointing). In either case the closed end of the selection is drawn with a dashed line, intended to suggest that it's only a temporary assignment of rôle.

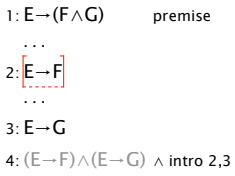

Fig. 17. A cut-conclusion

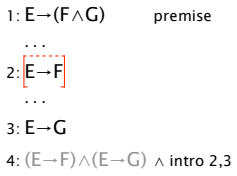

Fig. 18. A cut-hypothesis

The selection gesture now makes more demand on a prover's motor control than before, but the main issue is that the hypothesis/conclusion distinction sometimes has to be chosen by the prover. That means that novices now have to understand the distinction, where previously they might have been able to ignore it and blunder through. I think that perhaps this is a good thing.

6 Assessment

The four mechanisms discussed above made quite a difference to the usability of the logic encoding that I presented to my first-year undergraduate class, but they haven't been documented or widely publicised nor yet independently assessed, and so it's difficult to assess their eventual impact. The selection gesture improvement seems like a no-brainer, though a scientific evaluation would be needed to be certain. The modified logic encoding, with its improved error messages, its elimination of incomplete steps, its more rational set of logical rules and its correct display of box-and-line proofs, seems to be a success with novices and is unlikely to be completely misconceived, though my experience of educational evaluation warns me that there must be many points of detail which could be improved.

What is not to be celebrated is the means of implementing all this. Throughout the early development of Jape[4, 12, 5, 13, 7], Bernard Sufrin and I made great play of the notion that what is on the screen is an interpretation of what is in the machine, eliding unnecessary parts and showing tidied up versions of others. What was in the machine was supposed to be transparently an encoding of a logic; what appeared was supposed to fit the demands of proof development and display. It was and is the case that sequent trees are the easiest and most transparent way to represent formal logical proofs. It's easy to understand how to implement operations like applying a rule at a tip, or deleting the subtree(s) above a node. It's easy to implement a mechanism to enforce a proviso like "c must not occur in this sequent", and thus implement privacy conditions (eigenvariable conditions) on quantifier rules.

We also aimed to provide a lightweight way of encoding logics in Jape's rule and tactic notation. The developments described here seem to have subverted that aim: in my latest encoding of natural deduction the encoding of logical rules is a mere 45 lines and 1.4K characters; the tactics and definitions which support interaction via clicks and menu commands are 800 lines and 37K where, prior to improvement, they had been only 54 lines and 2K.

Much of the extra encoding volume is tactic programming that constructs error messages for particular situations. I'm confident that that is an inevitable cost of attempting to generate messages by interpreting user's intentions, though no doubt it would be worth searching for more transparent and economical representations of the search. The rest of the increase is caused by the complexity of the representation of forward steps, the need to use them everywhere, and the fact that they have to be carefully crafted if they aren't to break down with inexplicable error messages, to generate duplicate hypothesis lines, or loop for ever. In the particular case of natural deduction, which was the second example I tackled, it wasn't very difficult to get it right but then (a) I designed the mechanisms, so I understood them, and (b) in the other example, which was a treatment of Hoare logic and had to deal with large displays, I never quite got it right. It's hard to avoid the conclusion that for Jape tactic programming, this may already be a bridge too far.

Least satisfactory of all is the use of cuts to implement a DAG. The tree mechanism, which was until recently quite clearly behind the implementation of box-and-line proofs, is now little more than the scaffold on which quite a different structure is built. Worst of all, DAG generation is imperfectly implemented: in particular, the treatment of transitive reasoning is not integrated into it, and it does not support the next obvious step, which is moving lines around in a box-and-line display to produce a more pleasing arrangement.

7 Where Next?

Jape was designed as a sequent tree calculator, because Bernard Sufrin and I understood the syntax and semantics of trees. We knew we could get the tree part right and then work from a solid engineering basis. For a

long time this architecture was a success, and the difficulty of rendering a tree proof in box-and-line form was comparable to the difficulty of rendering a tree proof directly.[4] Now it's a very much more complicated business, and the complexity extends outside the module that actually renders the picture into the tactic interpreter (via `CUTIN`), the prooftree navigator (because of the need to carefully interpret paths once a cut step has been inserted), and worst of all the logic encoding (via `fstep` and the like).

One way forward is obvious. All that the complexity is accomplishing is the preservation of an illusion that what you see – a box-and-line proof, i.e. a DAG – is really there. If the illusion were reality, it would surely be much easier to implement the insertion of a line, which is all that `CUTIN` does. And if the DAG were the underlying representation, new mechanisms can be imagined which are impossibly daunting today: users could rearrange the lines of a displayed proof, for example, and it might be possible to satisfy undergraduate students' desire for a mechanism which would take in their pencil-and-paper proofs and comment on their correctness.

The drawback of such a development would be that Jape would no longer embrace forms of logic that don't fit DAGs. That would be a pity, but two lessons I have learnt from building Jape are to recognise the beauty of natural-deduction-style logics on the one hand, and to realise the range of logical invention and hence the difficulty of squeezing many useful logics even into Jape's sequent idiom. BI[11] and its cousin separation logic[10], for example, have trees in their left contexts, which is hard to mimic in Jape let alone implement effectively. Jape could usefully retreat from the hubris of universal aims and be made to support natural deduction reasoning even better than it does now.

All that is perhaps. It would be a significant effort to comprehend and implement the semantics of box-and-line proofs so that all the mechanisms Jape currently supports are accurately translated, and so that there is room for development.[5] It would be a still larger effort to re-design and re-implement all the parts of Jape that work on the tree. Maybe the recent port to Windows[1] is my last push, and Jape's swansong.

Acknowledgements

Bernard Sufrin and I designed and implemented Jape together for several years. His architectural insight and crucial early design decisions have been essential underpinning for everything I have managed to achieve since he moved on to other things. His continued advice helps me implement along the narrow path.

[4] So far Jape has a very simplistic recursive rendering of tree displays. I did manage to snuggle little subtrees under the branches of big ones to reduce width in some cases, but most tree displays are large, short and wide. It's possible to imagine a renderer which can draw a sequent as a block rather than a single line, which would make it possible to draw trees with a nicer aspect ratio, but they would still be large.

[5] Actually, since I began to write this paper, and in particular since I've noted the analogy between DAGs and let formulae, it doesn't seem so hard any more. Oh dear!

My colleagues at Queen Mary College, especially Adam Eppendahl, Mike Samuels, Graham White, Paul Taylor (who advised me on redesigning the logic rules) and Keith Clarke, all of whom discussed Jape's design and implementation with me in the ten years that it was developed and deployed there, helped me to see some of the ways that I pushed Jape forward.
James Aczel and his colleagues at the Open University evaluated what I thought was quite a good version of Jape. His exposure of its deficiencies, and his insights into the causes of students' difficulties, drove all the developments discussed above. The anonymous students who agreed to be videotaped did him and me and their successors a great service.
The mistakes in Jape, and the complexity of the current implementation, are nowadays all my fault (especially because I ignored so much wise advice over the years).

References

1. The jape web site. http://www.jape.org.uk. Latest versions of Jape for various platforms, including the encoding of natural deduction discussed in this paper.
2. J. C. Aczel, P. Fung, R. Bornat, M. Oliver, T. OShea, and B. Sufrin. Using computers to learn logic: undergraduates experiences. In G. Cumming, T. Okamoto, and L. Gomez, editors, *Advanced Research in Computers and Communications in Education: Proceedings of the 7th International Conference on Computers in Education*, Amsterdam, 1999. IOS Press.
3. R. Bornat and B. Sufrin. Jape: A calculator for animating proof-on-paper. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 412–415, Berlin, July 13–17 1997. Springer.
4. Richard Bornat. User interface principles for theorem provers. invited talk at UITP 95, Glasgow, 1995.
5. Richard Bornat and Bernard Sufrin. Jape's quiet interface. presented at UITP96, York, 1996.
6. Richard Bornat and Bernard Sufrin. Displaying sequent-calculus proofs in natural-deduction style: experience with the jape proof calculator. presented at International Workshop on Proof Transformation and Presentation, Dagstuhl, 1997.
7. Richard Bornat and Bernard Sufrin. Using gestures to disambiguate search and unification. presented at UITP 98, Eindhoven, 1998.
8. Richard Bornat and Bernard Sufrin. Animating formal proof at the surface: The Jape proof calculator. *The Computer Journal*, 42(3):177–192, 1999.
9. Richard Bornat and Bernard Sufrin. A minimal graphical user interface for the jape proof calculator. *Formal Aspects of Computing*, 11(3):244–271, 1999.
10. S. Ishtiaq and P.W. O'Hearn. BI as an assertion language for mutable data structures. In *28th POPL*, pages 14–26, London, January 2001.

11. D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.
12. Bernard Sufrin and Richard Bornat. User interfaces for generic proof assistants. part i: Interpreting gestures. presented at UITP96, York, 1996.
13. Bernard Sufrin and Richard Bornat. User interfaces for generic proof assistants. part 2: Displaying proofs. presented at UITP 98, Eindhoven, 1998.