Improving Jape

Richard Bornat

School of Computing, Middlesex University, UK richard@bornat.me.uk

Abstract. Some recent improvements in the design of Jape, an interactive formal 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. It deals with Kripke trees to permit calculation of model-based disproof in constructive Natural Deduction. Its display is capable of dealing with very long formulae, and it is simple to fork off subproofs as lemmas, to permit working on practical small examples in Hoare logic. Changes to its internal workings, to its tactic language and to user interaction are described.

1 Background

In 1998 the proof calculator Jape [5, 7–9] was evaluated by the educationalist James Aczel, as part of a project in which he was looking at whether and how novices use visualisation in formal proof development [1]. Jape had been in existence for about eight years and for the whole of that time I had had it under constant development and review with the needs of novice provers in mind. Nevertheless, the evaluation was salutary – as evaluations so often are – and led to several significant changes in Jape's external and internal machinery.

I realised, after using the new Jape for a year or so to underpin a course in formal proof and disproof, that my students needed mechanical help with the disproof part of the course. I was teaching them how to draw diagrams to express counter-examples in a Kripke model of Natural Deduction. They had difficulty in following the rules of the diagrams – monotonicity, acyclicity – and still more difficulty in evaluating the diagrams they had drawn, or examples that I presented to them. Jape could clearly be modified to draw lawful diagrams and to evaluate them correctly on request. That would take from students the burden of accuracy in disproof which it had already removed in formal proof, and leave them with the more useful educational burden of understanding what is going on.

Finally, I wanted to teach them Hoare logic. Hoare logic is the reason that most practical computer scientists are, or ought to be, interested in formal proof. The formal rules of the logic are trivial and with an underpinning of Natural Deduction, students ought to be able to make almost completely formal program verifications, fudging only the arithmetic (I dreamed of linking Jape to a finite-arithmetic oracle, making completely formal if incompletely expressed verifications, but that hasn't happened so far). Then I decided to write a book

[10] about formal proof and disproof, and I knew that I had to tackle the problems of proof display and proof management which large verifications clearly require.

This paper describes some of the work that these realisations entailed.

2 The evaluation

Aczel's evaluation threw up various points, four of which I deal with in this paper.

2.1 Incomplete steps and unknowns

It became clear that formal-proof novices, unlike experts, don't appreciate the freedom which Jape offers its users to defer choices during the development of a proof. Exploration is quite difficult enough when restricted to choices about which rule to apply and when. Jape uses 'unknowns' – place-holding metavariables which can be unified with concrete formulae – to defer the specialisation of a logical step, allowing the unknown to be resolved later when it becomes clear what it ought to be. But unknowns aren't part of a novice's understanding of logic, so this is a piece of meta-machinery which can easily derail their fragile understanding. Couple this with an awkward use of subtly-different kinds of mouse gestures to smuggle additional rule-step arguments past Jape's point-and-click single-argument primitivism, and there was so much confusion that few novices 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 $\forall$ elimination step to produce figure 1.[1] She hadn't said what variable to use in place of x in $\forall \cdot (P(x) \rightarrow R(x))$, and Jape's attempt to be helpful had obscured rather than clarified her situation.

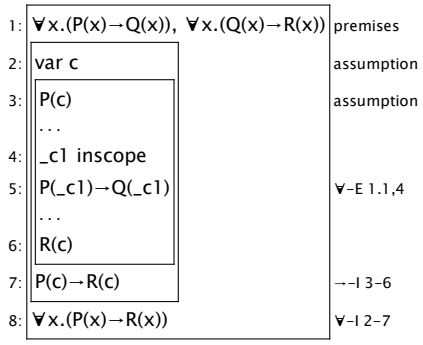

Fig. 1. An incomplete $\forall$ elimination step

[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 much nastier.

So much meta-logical internal machinery was showing that it is no wonder she despaired. Not only had Jape invented an unknown `_c1`, written in a notation which she had never encountered (names that start with an *underscore*?), but my encoding of Natural Deduction had led it to use the word 'inscope' on line 4, a word completely unexplained in the logic course that she was attending. There was an explanation of inscope buried in the manual I distributed to her and her friends, but to understand it you had to understand what was going on in figure 1: a circularity, so far as she was concerned. An entire section of the same manual described procedures for eliminating unknowns once they had appeared, and a further subsection was dedicated to strategies for avoiding introducing them in the first place. In a case like figure 1, the manual suggested that you should use multiple text selections and unification to replace `_c1` by `c`, or text-select `c` to avoid the introduction of`_c1` (text selection is alt+press-and-drag, quite an intricate gesture, and not at all something that I would like to try to explain to a tearful student). Either of these tricks, for reasons which the manual left unexplained, also hid line 4 and thus eliminated the nasty inscope word from the display.

In a separate course on GUI design I taught students that help text is a symptom of failure of design: well-designed GUIs should be so straightforward to use (technically, should provide so much affordance in the appearance of their controls) that 'help' isn't necessary. Poorly-designed ones, by contrast, need an online manual to tell you how to do anything. Long explanations in printed manuals that nobody ever reads of how to avoid meta-errors are just as bad. The problem, Aczel persuaded me, is that novices could make incomplete proof steps, which they find extremely confusing because to understand them they have to step outside the realm of formal logic into a world of proof search which they are only unconsciously exploring.

The solution was to allow only complete steps. I decided to force my novice users to select both the $\forall x \cdot (P(x) \rightarrow Q(x))$ formula on line 1 of figure 1 and the var c formula on line 2. To do that I had to implement multiple antecedent selection in Jape's GUI module, interpret multiple selections in Jape's tactic language, and modify the logic encoding to provide appropriate error messages if there was an incomplete selection. Once I had done it, proof-search strategies became explicable in terms of the logic, and the meta-logical machinery was entirely hidden.

2.2 Error message content

It was found that my advice to novices to read all Jape's error messages as "bang!!" or "whoops!"wasn't working. They tried to understand what Jape said to them, and were understandably demoralised when they couldn't. Many error messages, after all, are provoked by simple slips such as, for example, when we ask for an introduction step in a situation when we should have asked for a similarly-named elimination step.

In a videotaped session a pair of novices, who had made just that slip by asking for $\wedge$ introduction instead of $\wedge$ elimination, were confronted with figure

2. The error message used language outside their experience 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 comfortable in the sequent calculus – Jape's internal representation of proofs and proofs in development – and able to envisage the sequent-tree workings behind Jape's box-and-line proof display. Worst of all, it didn't even hint that they might try an elimination step instead. The problem was caused by the fact that most of Jape's error messages were generic and had nothing to do with the logic encoding that was in use.

The solution was clear: devise and deploy error messages particular to the logic encoding, so that Jape talks to its users in a language that they understand about proof objects that they can see. 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. I also had to make changes to the logic encoding, particularly to the treatment of negation and contradiction, to remove some unnecessary causes of error.

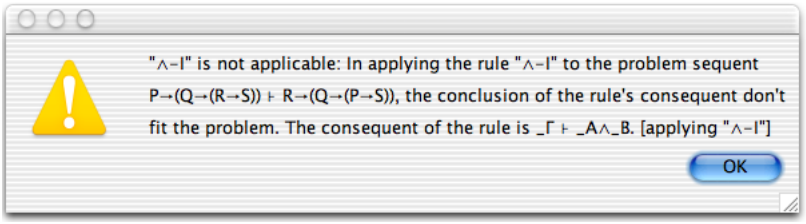

Fig. 2. An inappropriate error message

2.3 True box-and-line display

Eliminating incomplete steps and improving error reporting addressed the needs of novices learning a particular logic. They took a considerable amount of focussed effort. Unfortunately, other logic encodings couldn't immediately benefit without similar expenditure of effort, because Jape's tactic language doesn't allows much code reuse. Other encoders simply have to follow the path I trod: Jape is not as easy to customise as Bernard Sufrin and I initially hoped it would be.

The third significant problem, though, affected almost every user, because it was about the correct display of box-and-line proofs (Jape can do sequent-tree displays, but they are hardly ever used for all kinds of reason: see section 3). Simply, Jape's box-and-line display didn't show a true box-and-line proof development, because you couldn't always call on reachable lines higher up the proof when making a deduction. There were certain rules of thumb which an

encoding expert could call upon novices to use in an attempt 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 on the prover. My Jape publicity claimed that you could always choose what step to make and where to make 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 in detail below.

2.4 Displaying the effect of a selection

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 different 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.

3 Rendering sequent-tree Proofs in box-and-line style

Jape's basic box-and-line proof rendering mechanisms are dealt with elsewhere [3, 7], 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 in the box-and-line style as figure 5. Jape's rendering algorithm converts the tree into the compact box-and-line display in four stages.

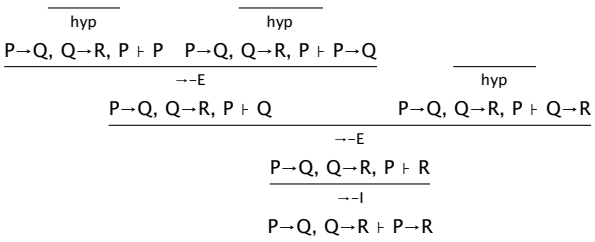

Fig. 3. A wide tree proof

3.1 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

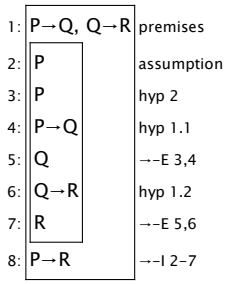

Fig. 4. A linearised proof

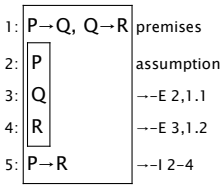

Fig. 5. A compact box proof

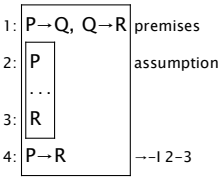

Fig. 6. First step of the proof

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 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 with premises on line 1 of figure 4, followed by the rendering of its subtree in lines 2-7, followed on line 8 by the root formula labelled with →-I. The root's subtree is also rendered as a box: an assumption on line 2 followed by the left subtree (lines 3-5), the right subtree (line 6) and line 7 labelled with →-E. Each consequent line refers to its 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.

3.2 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.

3.3 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 covedrs 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 $P \to Q$ – and invoking an appropriate rule.

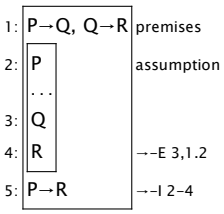

Fig. 7. A backward step from figure 6

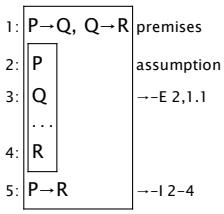

Fig. 8. A forward step from figure 6

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 the sequent tree. 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.

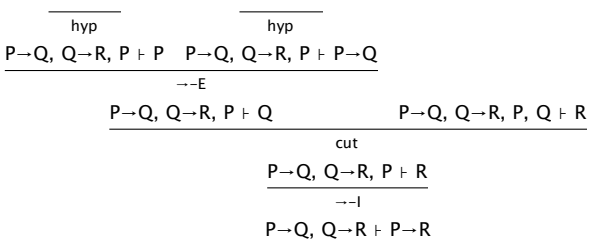

Fig. 9. A tree with a forward step

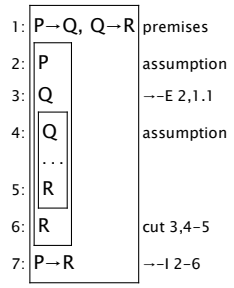

Fig. 10. A linearised cut step

3.4 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 $A = B \Rightarrow B = A$ like logical identity and treat transitivity $(A = B; B = C) \Rightarrow A = C$ like cut, generating a sequence of lines $A =$, $= B$, $= C$, ..., $= Z$. Jape's rendering mechanism can do this, using a similar mechanism to the one that deals with cuts and identities, but I don't discuss it in this paper.

3.5 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 (Directed Acyclic Graph), not a tree. In a tree, by contrast, 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 which follow. 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 figures 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.

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

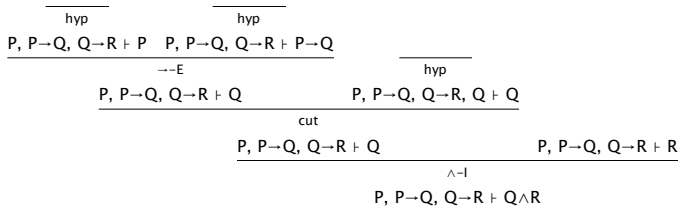

Fig. 11. Intro then elim

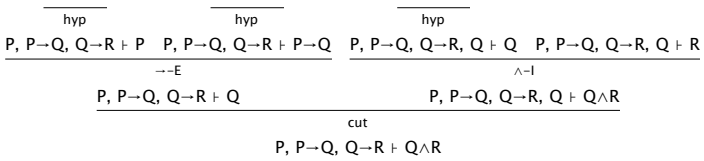

Fig. 12. Elim then intro

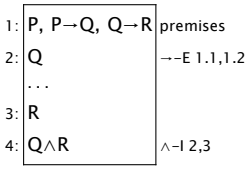

Fig. 13. An ambiguous display

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 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 perhaps 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 with 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.

4 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

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

[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.

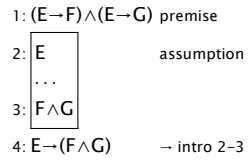

Fig. 14. Before forward step

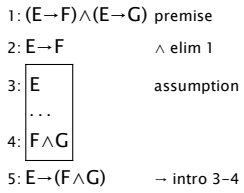

Fig. 15. After forward step

```
                    (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.

The complication is navigation: Jape tactics navigate by storing tree paths (`LETGOALPATH` and `GOALPATH`, for example, in the tactics above), which are inval-

[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.

idated 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.)

5 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 $line$ = *tree*A 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!

6 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 (`LETCONC`) selections. 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. But 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, since it could be used potentially as either.

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 clicks on the top half of an ambiguously-

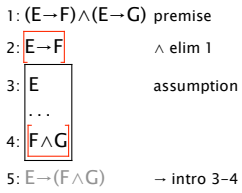

Fig. 16. Directed selection boxes

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.

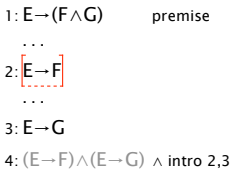

Fig. 17. A cut-conclusion

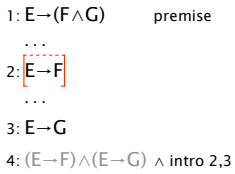

Fig. 18. A cut-hypothesis

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.

7 Assessment of the box-and-line display improvements

The various display, tactic and pointing mechanisms which permit true forward steps 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 evaluated, and so it's difficult to assess their eventual impact. The selection-gesture-feedback improvement (different boxes for different kinds of selections) seems like a no-brainer. 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 [2, 16, 4, 17, 6], 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.

7.1 Where next?

Jape was designed as a sequent-tree calculator, because Bernard Sufrin and I understood the syntax and semantics of sequent 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 possible way forward is temptingly clear. 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.

A 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 [15] and its cousin separation logic [13], for example, have trees in their left contexts, which is hard to mimic in Jape let alone implement efficiently. Jape could usefully retreat from our 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. 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 [9] is my swansong so far as major reimplementation is concerned.

8 Disproof

The most fruitful way, I imagine, of incorporating model theory into a course on proof is to treat it as a means of disproof. If you can show a counter-example in a model, and if the logic is sound (Natural Deduction certainly is, but proof of its soundness is not a part of the course) then a formal proof is impossible. You can even use a failed search for a constructive proof to guide the search for a constructive counter-example. You can have fun, in those cases where a

[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.

classical proof and a constructive counter-example exist, comparing proof and counter-example of contentious examples.

All of this was done, for one year in the course that I taught, on blackboard and on paper. The contrast between the ease with which most students learnt from Jape-supported proof search, where the machine was accepted fairly readily as an accurate and impartial arbiter in difficult cases, and the pain they experienced with paper-and-pencil counter-examples, where the only arbiters were fallible teachers, was stark. Constructive counter-models are graphical constructs with very simple rules: it was plain that machine support might be provided, and so I attempted it.

9 A scant introduction to Kripke semantics

Classical logic is based on the notion of truth. A properly formed assertion is either true or false, independently of our understanding of it. The corresponding 'method of truth tables', enumerating all possible states of the support for an assertion, is familiar to most computer scientists, at least for the propositional case and for the purposes of designing hardware. Kripke's constructive model using possible worlds [14, 11], though graphical and very accessible, is less well known. I therefore give a very brief introduction, at the level of engineering mathematics rather than foundation.

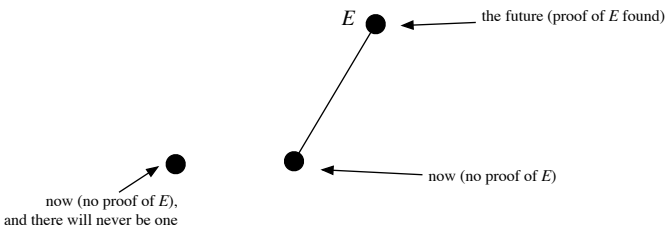

Fig. 19. Two alternative views of the present

Constructive logic is based on the notion of proof. A properly formed assertion can be proved or disproved or there can be evidence neither way.[5] In contrast with classical logic, then, it's reasonable to consider a distinction between 'now' and 'the future': we might have no proof of assertion E now, but one might be found tomorrow, as recently was the case for Fermat's Last Theorem, and as we might hope for Goldbach's Conjecture. In the semantics the two possible situations can be shown by alternative diagrams, as in figure 19. The first (left-hand) diagram describes a situation in which there is no proof of E, and none will ever be found. The second (right-hand) world describes a situation in which

[5] It's not a three-valued classical logic – it's odder than that [11].

there is no proof of E now, but one will be found. We don't know, when we don't have a proof of E, which of these worlds we are living in, so it might well be the second. Since constructivists interpret $\neg E$ as 'there will never be a proof of E', the second world denies both E and $\neg E$, and thus the classical law of excluded middle. On the other hand, a diagram with just a single world and no future development, like the left-hand one above, corresponds to the classical model.

Jape supports the drawing of these possible-world diagrams. The natural deduction encoding described here uses the definitions of figure 20 where $w \geq w'$ means you can travel from w to w' by following upward lines in a diagram. The glory of this collection is the treatment of $\rightarrow$, which captures beautifully the notion that I am forced to accept $A \rightarrow B$ if and only if, however things might develop, whenever I'm forced to accept A then I'm also forced to accept B.

$w \vDash A \wedge B$ iff $w \vDash A$ and $w \vDash B$
$w \vDash A \vee B$ iff $w \vDash A$ or $w \vDash B$
$w \vDash A \rightarrow B$ iff for every $w' \geq w$, if $w' \vDash A$ then $w' \vDash B$
$w \vDash \neg A$ iff for no $w' \geq w$, $w' \vDash A$
$w \vDash \forall x.P(x)$ iff for every $w' \geq w$ and every i, if $w' \vDash$ actual i then $w' \vDash P(i)$
$w \vDash \exists x.P(x)$ iff for some i, $w \vDash$ actual i and $w \vDash P(i)$

Fig. 20. Constructive forcing semantics of natural deduction

10 Drawing counter-examples

The assertion $(E \rightarrow F) \vee (F \rightarrow E)$ has no constructive proof. An attempt to prove it using constructive rules in Jape gets stuck quite quickly, as illustrated in figure 21. There is still a constructive rule which is applicable – from contradiction conclude F – but it's pointless to try it because you can't generate a contradiction from E. The expert recognises from the structure of the stuck proof that the counter-example will probably involve a world at which there is a proof of E but no proof of F.

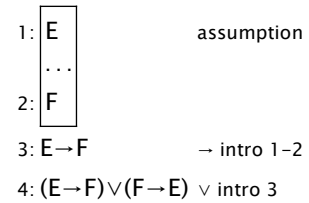

Fig. 21. A stuck constructive proof

Choosing Disprove in Jape's menus splits the window into two panes with the proof attempt in the lower pane and a minimal semantical situation in the upper pane, as illustrated in figure 22. The red-ringed blob is the currently-selected root world of the currently-selected situation; subformulae in the assertion in the lower pane are coloured to show their status in that situation. Atomic assertions E and F are black, to show that they are not forced (because E and F don't occur in the root world). The arrow in $E \rightarrow F$ and the brackets which surround it are coloured violet to show that the subformula is forced – trivially so in this case, just because E appears nowhere in the situation. The arrow and brackets of $F \rightarrow E$ are coloured similarly, for similar reasons. The disjunction connective, between the brackets, is violet because at least one of $E \rightarrow F$ and $F \rightarrow E$ is forced, and the whole conclusion is underlined in violet to make it particularly clear that it is forced. A counter-example demonstrates that an assertion does not always hold by showing a situation in which all the premises are forced but the conclusion is not. This situation is not yet a counter-example.[6]

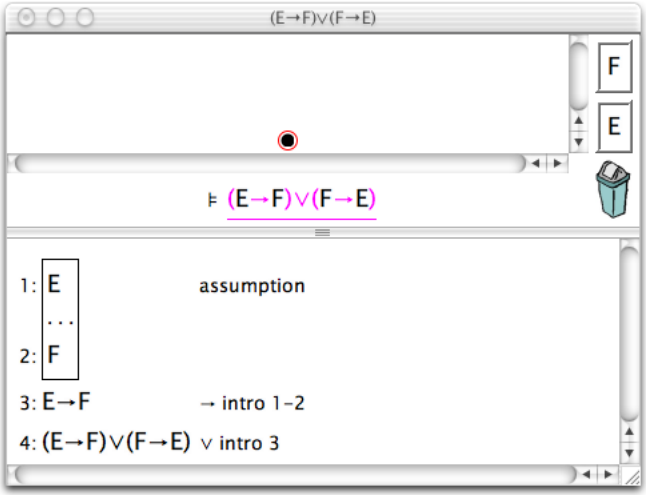

Fig. 22. The beginning of a disproof attempt

A new world can be created by option-dragging the base world, producing figure 23. Note that the new world makes no change in the colouring, because it introduces no future developments. Then E can be forced at the new world by dragging the E tile from the right-hand side of the window and dropping it on the new world, producing figure 24. Now the colouring changes: $E \rightarrow F$ is no longer forced, because there is a reachable world at which E is forced and F is

[6] In typically sneaky mathematical style, a situation in which there are no premises, as in this example, is one in which all the premises are forced. It is also one in which they are all unforced: see the discussion of green sheep in [10].

not. But $F \rightarrow E$ is still forced for the same reasons as before, and therefore the disjunction is still forced.

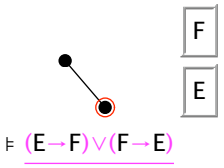

Fig. 23. An extra world

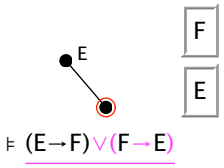

Fig. 24. A labelled extra world

At this point the expert knows just what to do to complete the disproof: do for F what was just done for E. That's justified because the first proof step picked out the left-hand half of the disjunction, and we might as easily have picked the right, giving figure 25. Previously, to make a new world I dragged a new one out of the base world. That is the right thing to do again, but it is no longer the only thing to do, so I illustrate the effect of alternative choices.

First, the easy way and a correct move: dragging a new world out of the base and dropping F onto it completes a disproof, signalled in figure 26 by the fact that the assertion is now *not* underlined.

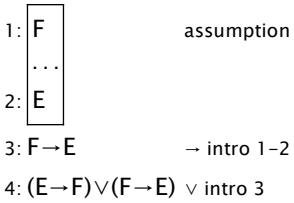

Fig. 25. Stuck again

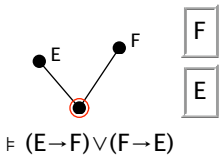

Fig. 26. The completed disproof

The other, more laboured, route to a counter-example starts by extending the E world as in figure 27. Jape enforces monotonicity when the new world is created by including E in the new world. Note that the colouring and underlining of the assertion is unchanged from the single-E situation. Then when F is dropped on the new world to give figure 28 we find that $F \rightarrow E$ is still forced, because in the top world, the only world in which F is forced, so is E. This situation isn't a counter-example.

It's possible to go forward in at least two ways. One way is to undo back to the two-world situation and do the right thing from there (Jape applies undo and redo to the last pane – proof or disproof – that you clicked in, rather than using an overall interaction history). Alternatively, you can modify the current situation by deleting parts of it. Monotonicity stopped us making a world with

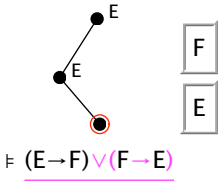

Fig. 27. Extending the upper world

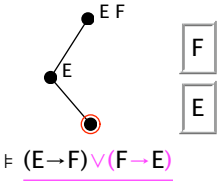

Fig. 28. Not a counter-example

just F in it, so first I decide to get rid of the link I just made (but not the world): press on the link from the E world to the E, F world and drag it to the swing-bin in the bottom right of the disproof pane (the line stays attached to its endpoints and drags rubber-band fashion till it is dropped in the bin, which lights up, as it should, to receive the undesired object – but it's hard to capture that moment!), giving figure 29. Option-dragging the base world onto the isolated E, F world makes a new link, giving figure 30. Finally, throwing the undesired E in the bin takes us back to the counter-example of figure 26.

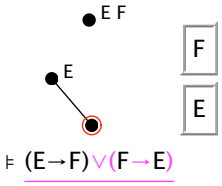

Fig. 29. An isolated world

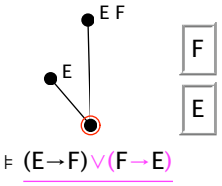

Fig. 30. A different connection

11 Exploring diagrams

The definition of $A \rightarrow B$ – where A is forced, B must be forced as well – is the glory of Kripke's model. It's also quite hard to grasp at first. I give some assistance to novices by allowing them to experiment with different situations, and more by allowing them to explore a situation – counter-example or not – to see where particular subformulae are forced.

Figure 31, for example, is the result of clicking on one of the worlds of figure 28. The colouring of the assertion is now not as it was when the base world was selected. Occurrences of E are now violet, but F is still black (atomic assertions are forced if they are present, not forced if absent). The black arrow shows that $E \rightarrow F$ is not forced, and the colouring of the basic assertions shows that that's because E is forced and F isn't; similarly, it shows why $F \rightarrow E$ is forced. The disjunction, which is the overall assertion, is still forced because one of its components is forced.

That situation shows the novice why $E \rightarrow F$ isn't forced in the base world – it must hold everywhere above the base world, not just in one place, and the intermediate world contradicts it. Clicking on the top world makes the same point more strongly, producing the classical single-world evaluation of figure 32. Now everything is forced: both the atomic assertions, both the implications and the disjunction. Implication, in this semantics, is a promise rather than a static property, and a promise broken in one world is broken in all the worlds below it. Although $E \rightarrow F$ holds in the top world, and it would hold in the root world if it didn't have descendants, it doesn't hold in the intermediate world and that's enough to say that it doesn't hold in any world which has that one as a descendant.

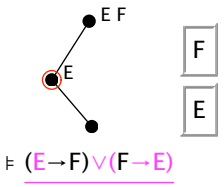

Fig. 31. Another evaluation

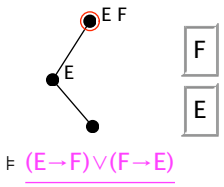

Fig. 32. A classical evaluation

Dragging and dropping is extensively supported in Jape's treatment of constructive disproof. You can drag worlds onto worlds or lines, and lines onto worlds. You can drag anything – worlds, labels, lines – into the swing bin (it's the most straightforward mechanism for deletion, vastly easier to use than a "select; delete" technique). Using drag and drop, plus an occasional focus-click, for every action in the proof pane is a worthwhile simplification, a reasonably quiet interface.

12 Working with quantifications

The forcing-semantics of quantification relies on named individuals inhabiting a universe. Adding individuals to the logic, via the pseudo-assertion "actual <*name*>", makes it possible to give a much more coherent account of Jape's proof capabilities. Using individuals in disproof requires a little care, and Jape is not yet completely developed in this direction.

Consider, for example, the assertion

$$\text{actual}\, i,\ \forall x.\,(R(x) \vee \neg R(x))\,,\ \neg\forall y.\neg R(y)\ \vDash\ \exists z.R(z)$$

It looks unexceptionable: in a non-empty universe (there is at least the individual i), where R is a decidable property, not false everywhere, there must be some individual with property R. But it's one of the entries in Jape's "Classical conjectures" panel, so even the novice can guess that it must have a constructive disproof as well as a classical proof.

The classical proof, shown in figure 33, is easy once you realise that the decidability hypothesis was inserted as a sop to the constructivists, and even the non-emptiness of the universe is nothing to do with anything. But the constructive attempt, though it uses all the premises and explores more avenues, gets stuck as in figure 34 because it can't make the same first move.

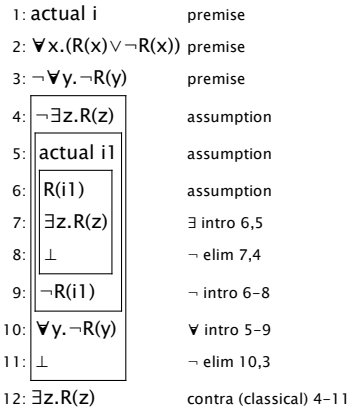

Fig. 33. A classical success

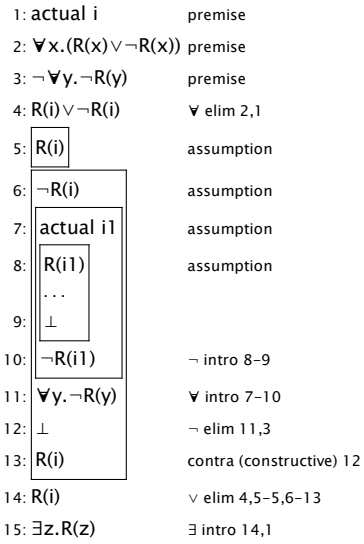

Fig. 34. A constructive failure

Never mind! The stuck proof tells us what to do: make a world with an individual i (line 1), another with *i1* (line 7), don't have $R(i)$ (line 6), but do have $R(i1)$ (line 8). Jape's starting point for a disproof, shown in figure 35, colours actual i black, because it doesn't occur at the root world, underlines the second premise, which is forced trivially because there are no individuals in the diagram, and doesn't underline the third for similar reasons. The conclusion then isn't forced because there is no individual to stand witness. The tiles on the right allow us to make worlds involving the only individual (i) and the only predicate (R) in the assertion. They won't be enough, but we can make a start.

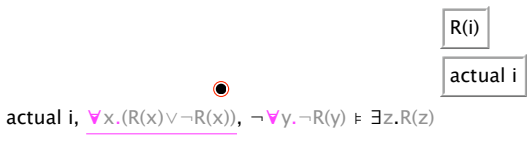

Fig. 35. The start of a constructive disproof

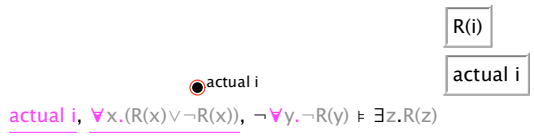

Fig. 36. An individual added

Recall that to produce a counter-example we must make a situation in which all the premises are forced and the conclusion is not. So the first step must be to force actual i in the root world (figure 36). Observe that actual i is now forced, and that the $\forall$ is still forced because we don't force $R(i)$. The $\exists$ is still black, just because we don't force $R(i)$) for the only individual (i) available. Notice, too, that the formulae inside the quantifications are grey: a new kind of colouring.[7] Greyness signifies that a formula isn't strictly relevant to the status of the assertion, and that's true in this case, because $R(x)$ and $R(y)$ aren't interesting in a universe containing an individual i: only $R(i)$ matters. I shall return to the issue of evaluation and colouring of quantified formulae below.

We can press on. The stuck proof suggests (because of its box structure) that we should make a new world containing an individual *i1*. It's easy to make a new world by option-dragging the root world, but we don't have a tile with actual *i1*. Jape lets us add individuals to the model by double-clicking an 'actual' tile. This solves our immediate problem, and we can drag and drop the result onto the upper world (figure 37).

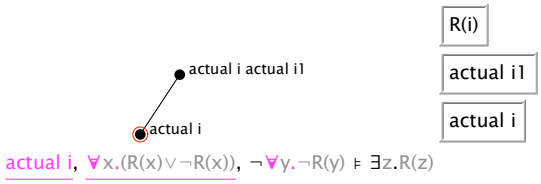

Fig. 37. A new world and a new individual

We still don't have $R(i1)$ to drop onto the model, but double-clicking a predicate tile gets you a novel instance built up from the available individuals (if there is more than one possibility, a choice dialogue lets you pick the one you want). That new instance can be dropped onto the diagram and the disproof is complete (figure 38): all premises forced (shown by underlining) and the conclusion unforced.

This is all very well: the stuck proof shows the expert how to proceed, and the interface supports all the necessary moves. But the display doesn't help

[7] For those of you reading in black and white, as they almost used to say on UK television, the grey bits look just like the violet bits. Sorry!

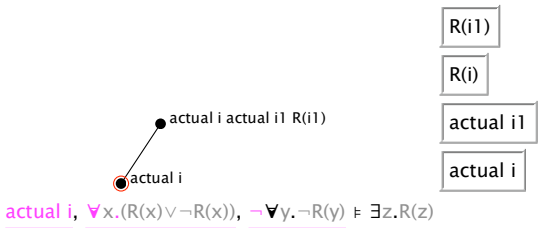

Fig. 38. A completed disproof

the novice to understand *why* it's colouring the assertion as it does. Most of the syntax colouring is grey, telling us that the literal subformulae aren't used, and that there is more behind the scenes to be explained. Exploring the worlds doesn't help much at all, as figure 39 shows.

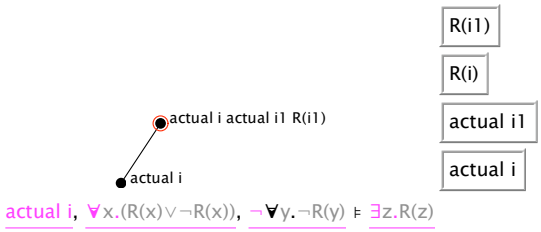

Fig. 39. A lot of colour but little illumination

The solution I provide is to allow the user to click on a quantified formula, and for Jape to colour the worlds and individual occurrences which support that assertion, as illustrated in figure 40 (only i in the upper world would support it, but that means that the quantification is forced there), figure 41 (only $i1$ in the upper world, but because i doesn't agree, the quantification isn't forced anywhere) and figure 42 (forced by everybody and nobody, everywhere).

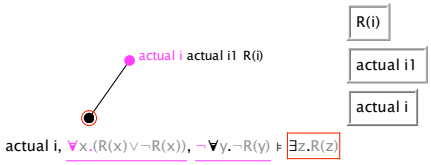

Fig. 40. Coloured illumination (1)

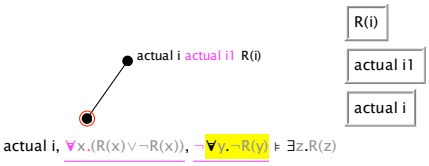

Fig. 41. Coloured illumination (2)

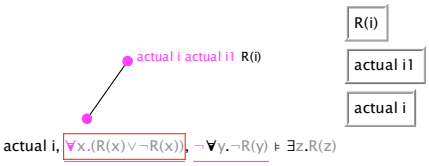

Fig. 42. Coloured illumination (3)

13 Deficiencies of the disproof mechanism

Layout of the labels decorating a world is primitive: they are shown as a single line of text extending right from the world. In a large diagram those text lines confusingly overlap other objects.

When drawing complex structures a single-history undo is sometimes irksome. It would be worth experimenting with an object-focussed undo, though I'm conscious of the need to preserve a quiet interface.

14 Evaluation of the disproof mechanism

This mechanism hasn't been independently evaluated. My informal assessment is that students find Jape's proof mechanisms relatively easy to use, and formal proof correspondingly relatively easy to learn. They find disproof harder to understand and in practice they struggle with the disproof interface.

On the other hand many of them do learn about models and disproof using Jape, and all appear to be grateful for the unbiased verdict of a calculator on their attempts to answer exercises. The interaction with the disproof calculator is much more like what many of them would hope to have with a proof calculator: they can put in a copy of what they have written on paper, and receive a judgement, provided that their pencil and paper version is not too outlandish.

The quantifier evaluations of figures 40, 41 and 42 have not, so far as I'm aware, been presented to any novices anywhere.

15 Hoare logic

Hoare logic [12] is a marvellous invention. It turns difficult semantic arguments about program states into simple formal calculations about program texts, backed

up by some invention in the case of loops and a good deal of hand waving in the arithmetic that so often underlies array-element aliasing. Whatever its deficiencies, it is one of the glories of computer science. It is *the* reason why I took up the teaching of logic and the development of Jape. I wanted, and still want, my students to be able to play with Hoare logic.

One of the things that Bernard Sufrin and I realised early on in the development of Jape is that little things matter. Using the correct symbols, using the same proof layout as the lecture notes, using the same rule names, were all crucial both to the initial acceptance of Jape by novices and to their continuing reliance on it when proofs got tough. The discussion of box-and-line displays above shows how important it is to hide irrelevant details (indirect `hyp` steps, and all `cut` steps, for example) and to focus on the logic.

Hoare logic offers a new challenge because the formulae, even in very small examples, are very very long. Jape's box-and-line display is capable of splitting a line of assumptions or premises into two or more shorter lines if it is too long to fit within the width of the proof window, but it wasn't at first capable of folding long formulae. It is now: see figure 43, for example.

It is all, I'm sorry to say, a bit of a hack. Line 10 in figure 43 splits nicely into three – precondition, program, post-condition – but line 3 is a little less elegant. Line 6 is worse. Essentially, Jape tries to treat a line as a juxtaposition and to split it between juxtaposed elements, as on lines 3, 7, 8, 9 and 10, but if all else fails it will split it at a minimum-waste point as on line 6, trying to keep the fragment-lines as nearly equal length as it can.

The other long-awaited development which the treatment of Hoare logic forced upon me is the easy devolution of lemmas. The proof of figure 43 in [10] is made with six lemmas: essentially, each of the components of figure 43 is separately proved. To hive off a lemma, you point to a conclusion line and whatever hypothesis formulae you wish, as in figure 44 in which the lemma $\neg\forall y \cdot \neg R(y) \vdash \exists z \cdot R(z)$ is proposed; then you choose "Make lemma ..." from a menu, choose a name for the lemma and a conjecture panel to put it in. You can't, of course, cite the lemma in your proof until you have proved it!

Acknowledgements

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

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

. . .

1: {n≥2}(i:=2){2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)}

. . .

2: 2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)→i≠0

. . .

3: {2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧n mod i≠0}
(i:=i+1){2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)}

. . .

4: 2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧n mod i≠0→_M>0

5: integer Km — assumption

. . .

6: {2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)
∧n mod i≠0∧_M=Km}(i:=i+1){_M<Km}

{2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)}

7: while n mod i≠0 do i:=i+1 od — while 2,3,4,5-6

{2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧¬(n mod i≠0)}

. . .

8: 2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧¬(n mod i≠0)
→2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧n mod i=0

{2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)}

9: while n mod i≠0 do i:=i+1 od — consequence(R) 7,8

{2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧n mod i=0}

{n≥2}(i:=2){2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)}

10: while n mod i≠0 do i:=i+1 od — Ntuple 1,9

{2≤i∧i≤n∧∀x.(2≤x∧x<i→n mod x≠0)∧n mod i=0}

Provided:
DISTINCT i, n

Fig. 43. A proof in Hoare logic with many folded lines

1: actual i, ∀x.(R(x)∨¬R(x)), ¬∀y.¬R(y) premises

. . .

2: ∃z.R(z)

Fig. 44. Selection describing a lemma

his insights into the causes of students' difficulties, inspired many of the developments discussed above. The anonymous students who agreed to be videotaped did him and me and their successors a great service.

David Pym persuaded me to take Kripke semantics seriously as an educational target. The hundred and eighty students who suffered the first time I tried to teach it without mechanical support provided the evidence which underpinned the case for a disproof calculator.

Paul Taylor, Jules Bean, Mike Samuels and Graham White (who invented the 'scrabble tile' layout) advised me and supported me whilst I built the disproof parts of Jape. Paul's ideas on explanation were particularly influential.

David Pym encouraged me to write a book on formal proof and disproof, and so encouraged me to develop Jape to the point where it could withstand widespread scrutiny.

The design and implementation mistakes in Jape and the complexity of its implementation are nowadays all my fault. I really do hope that this is the end.

References

1. J.C. Aczel, P. Fung, R. Bornat, M. Oliver, T. O'Shea, and B.A. 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*, 1999.
2. R. Bornat. User interface principles for theorem provers. invited talk at UITP 95, University of Glasgow, 1995.
3. R. Bornat. Displaying sequent-calculus proofs in natural-deduction style: experience with the Jape proof calculator. Presented at International Workshop on Proof Transformation and Presentation, Schloss Dagstuhl, 1997.
4. R. Bornat and B.A. Sufrin. Jape's quiet interface. Presented at UITP 96, University of York, 1996.
5. R. Bornat and B.A. Sufrin. Jape: A calculator for animating proof-on-paper. In *CADE-14*, volume 1249 of *Lecture Notes in Computer Science*, pages 412–415. Springer-Verlag, 1997.
6. R. Bornat and B.A. Sufrin. Using gestures to disambiguate search and unification. Presented at UITP 98, University of Eindhoven, 1998.
7. R. Bornat and B.A. Sufrin. Animating formal proof at the surface: the Jape proof calculator. *Computer Journal*, 42(3):177–192, 1999.
8. R. Bornat and B.A. Sufrin. A minimal graphical user interface for the Jape proof calculator. *Formal Aspects of Computing*, 11:244–271, 1999.
9. Richard Bornat. Jape software. `http://www.jape.org.uk`.
10. Richard Bornat. *Proof and Disproof in Formal Logic. An Introduction for programmers.* OUP, 2005.
11. M. Dummett. *Elements of Intuitionism.* Oxford Logic Guides. Clarendon Press, 1977.
12. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
13. Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Symposium on Principles of Programming Languages*, pages 14–26, 2001.

14. S. A. Kripke. Semantical analysis of intuitionistic logic. In *Formal Systems and Recursive Functions*, pages 92—-130. North-Holland, 1965.
15. D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.
16. B.A. Sufrin and R. Bornat. User interfaces for generic proof assistants (part I: interpreting gestures). Presented at UITP 96, University of York, 1996.
17. B.A. Sufrin and R. Bornat. User interfaces for generic proof assistants (part II: Displaying proofs). Presented at UITP 98, University of Eindhoven, 1998.