Natural Deduction Proof and Disproof in Jape

Richard Bornat (richard@bornat.me.uk)

February 21, 2017

Preface

This manual is a how-to for Jape and my Natural Deduction encoding. Earlier versions contained a good deal of logic teaching. That's been dropped, since now there's a book (Logic for Programmers, OUP) which says it all a good deal better. You will learn about logic by playing with Jape, so I suppose there's still teaching in there.

This document was written on a Mac, and the illustrations are all taken from the MacOS X version of Jape. All Japes, on Linux, Solaris, Windows, MacOS X or whereever, run exactly the same code but the interfaces can look slightly different. I haven't produced different versions of the manual because I think — rather, I hope — that those differences in fonts, in window controls, in the placing of menus and so on, don't really matter. If I'm wrong then I hope somebody will tell me.

Jape or NDJape?

This manual is really about how Jape, which is capable of working with many different logics, has been made to deal with Natural Deduction as described in my book. Instead of talking about what 'Jape' does when dealing with $\rightarrow$ formulae, I should really talk about what 'Jape loaded with the I2L encoding' does. But that's such a mouthful that I just talk as if there is only one Jape, and all it does is Natural Deduction.

If you're interested there's a manual on the website (http://www.japeforall.org.uk) which tells you how to roll your own logic encodings. Best of luck.

Differences from the book

- Jape writes its premises on a single line, separated by commas, instead of using a separate line for each premise;

- Jape writes quantifications as $\forall x.P(x)$ and $\exists y.Q(y)$ — with a dot between bound variable and predicate — instead of $\forall x(P(x))$ and $\exists y(Q(y))$

So far as I know, these are the only significant differences.

Any comments?

If Jape doesn't work for you, it isn't working, and if it isn't working I'd like to hear about it.

Please send any comments, thoughts, complaints or whatever to me (see title page for email address). I'll do my best to reply quickly and if your message is of general interest I'll put it up on the website (with your permission, and hiding your email address from the spambots at least).

4

Acknowledgements

Contents

Chapter 1

Basics

1.1 Getting started

If you don't already have Jape, download it from http://www.japeforall.org.uk. Install it, following the instructions carefully.

You should have a directory containing the Jape application and a subdirectory called examples. Double-click Jape, and you will see a window like figure 1.1.[1]

Using the Open New Theory command in the File menu, open examples/natural_deduction/I2L.jt. You should see several windows containing logical conjectures (claims to be proved), called *panels* in Jape. The Conjectures panel should look something like figure 1.2 (you will probably have to resize the window to make it look like this). Double-click any line to begin, or select a line and press the Prove button at the bottom of the panel. You will see a *proof window* like figure 1.3, and off you go!

---

[1] You may see minor differences. The illustrations in this manual are taken from MacOS X. On Windows and Linux you will see a menu bar in the window and the window title will be Jape. But those differences don't matter much, so I won't refer to them again.
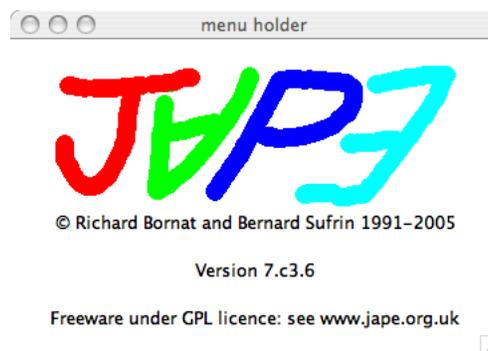

Figure 1.1: The Jape splash screen
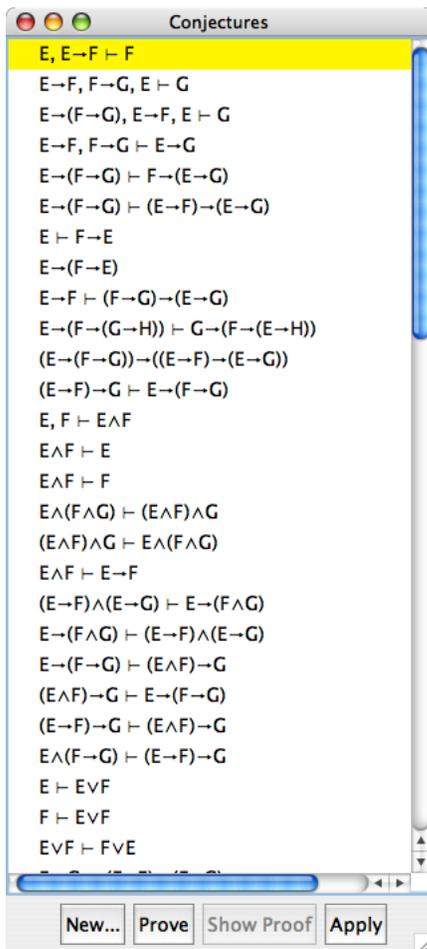
Figure 1.2: The Conjectures panel
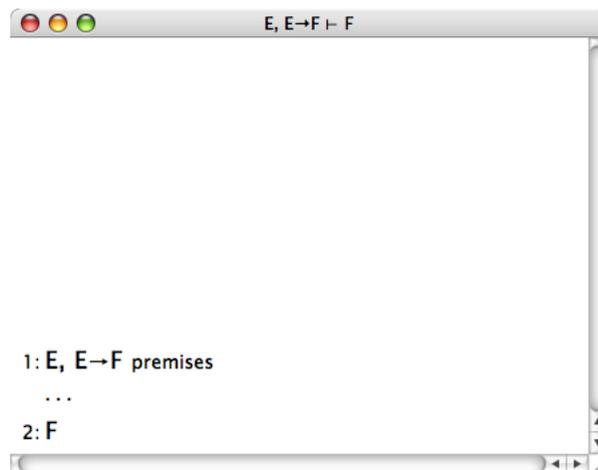
Figure 1.3: A proof window
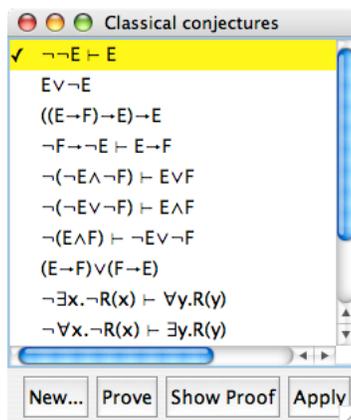
Figure 1.4: A conjecture panel with a proved
conjecture
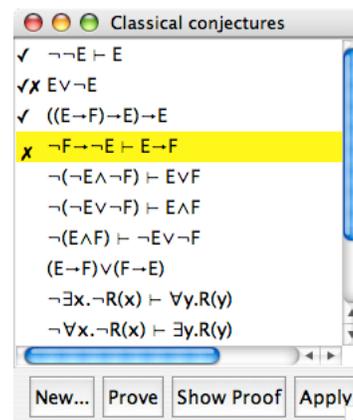
Figure 1.5: A conjecture panel with proved
and disproved conjectures

1.2   Finishing a proof

When you have made a proof of a conjecture — no more lines of dots in the proof window — you can save it: pull down the Edit menu and select Done. The proof window closes, and Jape records the fact that the conjecture is proved, marking its entry in the conjectures panel with a tick in the margin, as illustrated in figure 1.4. If you disprove a conjecture (see chapter 7) then you can record that too, and you get a cross in the margin. Because classical proof and constructive disproof overlap, you can even get both marks against the same conjecture, as illustrated in figure 1.5.

Proof of a conjecture makes it a theorem. You can use theorems in your proofs as if they were additional Natural Deduction rules (select the theorem in the panel, press the Apply button), and you can review their proofs using the Show Proof button. See chapter 8 for more information.

1.3   Saving and restoring your work

Jape offers to record your proofs — saved and unsaved — when you quit or when you choose "Save Proofs" or "Save Proofs As ..." from the File menu. It will reload saved proofs using "Open ...", also from the File menu.

1.4   Printing and Exporting

You can print a proof using the Print Proof command on the Edit menu. If you have a disproof on the go (see chapter 7) you can print it using the Print Disproof command, or proof and disproof together using Print.

To make a pdf or ps copy of a proof in a file, use Export Proof, Export Disproof or Export, all on the Edit menu.

1.5   Making Jape work for you

1.5.1   Only reflect

Jape is designed to be easy to use, which means that the mouse and menu and window stuff don't get in the way of the logic. It's so easy to use that you can have great fun clicking away, 'solving' lots of problems without always knowing exactly what you're doing. That's ok, because you can learn while you're having fun, and you can do things for yourself without asking for help. But it's obviously not the whole story.

Because Jape is easy to use it brings you quickly to a point where you can ask interesting and important questions. The kind of question you are supposed to ask is "is the logic really supposed to work like *that*?" If there are experts around you can ask them, but if you are on your own you can still ask yourself. That isn't a daft thing to do: educationalists call it *reflection*, and it's one of the best ways to learn.

Some logical proofs are hard to believe at first. Some single logical steps are pretty surprising. I hope that you will always read through finished Jape proofs to see if you can believe them. If there is a surprise, ask yourself: where does the surprise come from? By undoing and redoing steps you can watch the surprise emerge and explain to yourself why it's necessary.

Jape's a machine, and that has disadvantages as well as advantages. The big advantage is that a machine can do formal calculation — proof and disproof — perfectly, without mistake. The big disadvantage is that it can't understand anything about what it's doing. Jape doesn't know the difference between a nice proof

and a nasty one. Sometimes reflection will show you that there is a shorter or prettier proof than the one you have made. You can always undo your work and try again!

1.5.2   Be brave

We read proofs top-to-bottom most of the time. Novices, reasonably but mistakenly, imagine that proofs are constructed top-to-bottom too: start at line 1 and work forwards. Well, sometimes it is done that way, but at least as often it's done the other way round, bottom-to-top, starting with the last line and working backwards. Be brave and try it! If you stick to forward proof you make life very difficult for yourself, so bravery pays dividends.

Bravery is needed, too, when learning how proof steps work. It's reasonable when playing with an interactive program to first try only steps that make small changes, but eventually you have to try everything. It's just like riding a bike: once you've plucked up courage you can't remember what it felt like to be scared of ∨ elim or ∧ intro or whatever. Just do it!

1.5.3   Never guess an assumption

A proof is really a *structure of deductions*, not a sequence of lines, and its assumption boxes show that structure. Reading a proof from top to bottom, every assumption that is introduced must also be *discharged* by the use of a rule which makes use of the box. Jape guarantees correct use of assumptions by combining introduction and discharge into a single step. Assumptions are introduced (and discharged) by using rules, and the rules that do it — some forward, some backward — are labelled in the menus. Jape helps you, in every case, by calculating the assumption that you need: there is *never* a need to guess an assumption.

Chapter 2

Gestures: mouse clicks, presses and drags

Like every other interactive program, Jape accepts instructions through the mouse and the keyboard. Mostly you will use the mouse to *select* a formula or a subformula in the proof window and then to *command* what to do with your selection by choosing a command from a menu. The ways that Jape uses the mouse are pretty standard.

2.1 Formula selection

Formula selection is made with a single click (left-click on a multi-button mouse). You click on a formula — a *hypothesis* above the line of dots or a *conclusion* below the dots — and Jape highlights your selection with an enclosing red box. If you click on a hypothesis you get a downward-facing box as in figure 2.1(a); if you click on a conclusion you get an upward-facing box as in figure 2.1(b). You can select both a hypothesis and a conclusion, as in figure 2.1(c).

Clicking on the background — a white part of the proof window — cancels all your selections.

Simple clicks will let you select one conclusion and one hypothesis at a time. Normally a second hypothesis click cancels the first, but if you hold down the Shift key while you click you can select more than one hypothesis, as shown in figure 2.1(d). And then, also using the Shift key whilst clicking, you can cancel individual formula selections.

There is no way of selecting more than one conclusion at a time.

2.1.1 Ambiguous formulae click both ways

In figure 2.2(a) there are open conclusions on lines 3 and 4, each preceded by a line of dots to show that there's work to be done. Lines 1 and 2 aren't conclusions, and can only be used as hypotheses to prove lines

(a) hypothesis selection   (b) conclusion selection   (c) hypothesis and conclusion   (d) multi-hypothesis selection
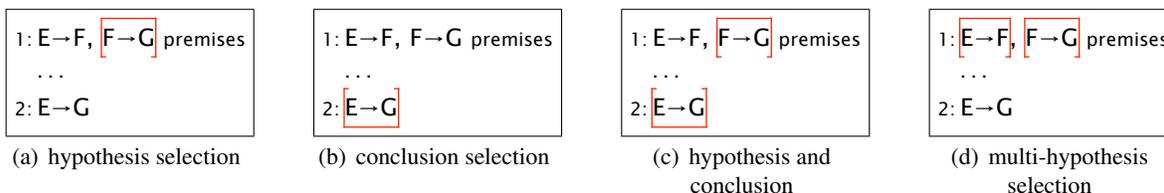
Figure 2.1: Formula selection by mouse click

(a) Proof attempt with intermediate hypothesis/conclusion formula

(b) top-half click selects as conclusion

(c) bottom-half click selects as hypothesis
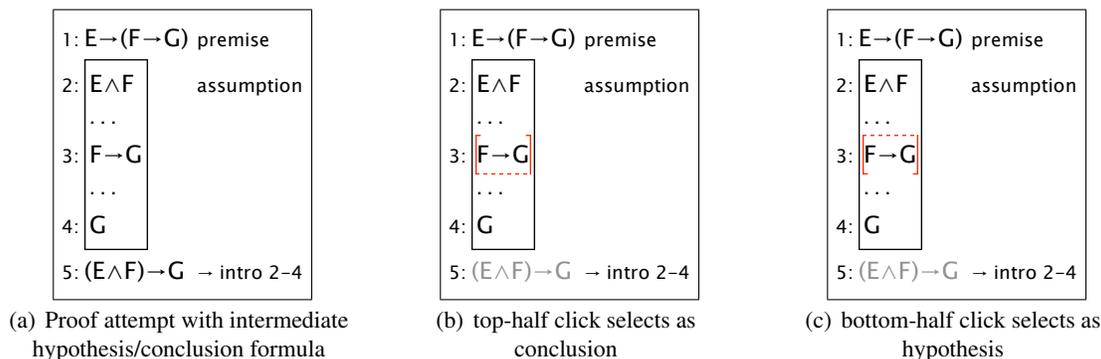
Figure 2.2: Ambiguous conclusion/hypothesis selection

(a) subformula selection

(b) multiple-subformula selection
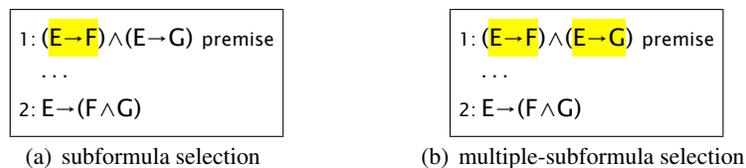
Figure 2.3: Subformula selection by option/alt/middle-press-and-drag

3 and 4. Line 5 can't be used at all: it's a proved conclusion. Line 4 can only be a conclusion and not a hypothesis, because there's nothing below it in the box. But line 3 is ambiguous: it has to be proved as a conclusion, perhaps using lines 1 and 2, and it can be used as a hypothesis to prove line 4.

Ambiguous formulae like $F \rightarrow G$ on line 3 of figure 2.2(a) can be selected in two ways. If you click on the *top half* of the formula you get a box open at the top — i.e. a conclusion selection — with a dotted line across the bottom, as shown in figure 2.2(b). If you click on the *bottom half* you get a box open at the bottom — i.e. a hypothesis selection — with a dotted line across the top, as shown in figure 2.2(c).

2.1.2 Greying-out

When you select a hypothesis you can only make a step towards conclusions below you and in the same box as the hypothesis you clicked. When you select a conclusion you can only make a backward step towards hypotheses above you and in the same box or enclosing boxes. Jape greys out all the formulae you can't use, to help you see what's going on, as shown by line 5 in figures 2.2(b) and 2.2(c).

If you click on a greyed-out formula Jape cancels your current selection(s). If you click on a conclusion that's already been used up (one which isn't immediately below a line of dots) then Jape greys it out and cancels all your selections.[1]

2.2 Subformula selection

Occasionally you need to tell Jape to focus on part of a formula. You do this by a press-and-drag gesture:

- with a three-button mouse, middle-press-and-drag;

- otherwise by holding the Alt shift (sometimes labelled 'option') during a press-and-drag.

---

[1] This was once a bug. Now it's become a feature, because I rather like it.

Jape highlights the subformula you've selected by changing the background colour to yellow, as shown in figure 2.3(a).

If you hold down the add-a-selection key (ctrl on Windows and Linux, command on MacOS X) you can make more than one subformula selection, as shown in figure 2.3(b). Using the same key combination, you can cancel and/or modify subformula selections you've already made.

Jape restricts subformula selections to well-formed subformulae, as you can discover by experiment. That means that if you option/alt/middle-click on a connective or a quantifier, a whole subformula is highlighted. If you option/alt/middle-click on a name, only that name is highlighted.

Option/alt/middle-clicking or pressing on a new subformula cancels all other subformula selections, unless you hold down the add-a-selection key (Command on MacOS, Ctrl on other systems). Option/alt/middle-clicking on the background — a white part of the proof window — cancels all your subformula selections.

Formula and subformula selections are independent: you can have one without the other, or both, or neither. Cancelling one kind of selection doesn't cancel the other.

2.3   Dragging

In the disproof pane — see chapter 7 — you can drag formulae, tiles, worlds and lines around to make a diagram. You do it by press-and-drag. See chapter 7 for details.

Chapter 3

Backward and forward steps

Because we read proofs forward, top-to-bottom, forward steps are easiest to understand and most proof novices prefer working forwards. But lots of proofs are difficult working forwards, and some are almost impossibly difficult. To make proofs in Jape you need to be able to make backward steps as well, and I've gone to some trouble to force you to recognise the fact. In the Backward menu, shown in figure 3.1, the first group of steps above the line work best backwards and the group below that line can be made to work backward if you try hard. The Forward menu, in figure 3.2, similarly shows steps that work well forward before ones that work forward only with difficulty. 'hyp' is hard to classify, so it appears in both menus.

To make a step you select one or more formulae and choose a step from the Backward or Forward menu. The selections you need to make depend on the step you choose, and are detailed in the descriptions of the steps in later chapters, but in general for a forward step you must choose a hypothesis and for a backward step an open (unproved) conclusion.

3.1 Making a forward step

Before you make a forward step you must always select a hypothesis formula. Depending on the kind of step, you may have to select more than one hypothesis: for details, see the description of the step later in this manual — or just try it and see what happens!
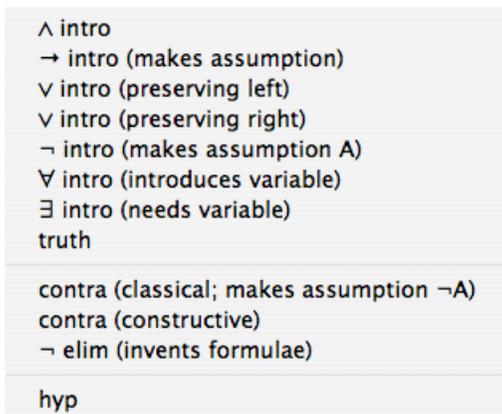
∧ intro
→ intro (makes assumption)
∨ intro (preserving left)
∨ intro (preserving right)
¬ intro (makes assumption A)
∀ intro (introduces variable)
∃ intro (needs variable)
truth
contra (classical; makes assumption ¬A)
contra (constructive)
¬ elim (invents formulae)
hyp

∧ elim (preserving left)
∧ elim (preserving right)
→ elim
∨ elim (makes assumptions)
¬ elim
∀ elim (needs variable)
∃ elim (assumption & variable)
contra (constructive)
∧ intro
∨ intro (invents right)
∨ intro (invents left)
hyp

Figure 3.1: Backward menu   Figure 3.2: Forward menu

(a) before

(b) after

Figure 3.3: A sample forward step

(a) hypothesis only selected

(b) consequent below hypothesis

Figure 3.4: A forward step without a target conclusion

You can also select a conclusion as well, if you want to. Some steps require a conclusion selection: you can look up the step in this manual or you can try it out and see what happens.

Once you have made your selection(s), choose your step from the Forward menu.

Jape writes the result of the step — the consequent deduced from the antecedent(s) you selected — just below your selection(s). For example, figure 3.3 shows an → elim step with two hypothesis selections. The consequent is line 4 in the 'after' picture. Notice that the justification of the step is written against the consequent.

(a) hypothesis and conclusion selected

(b) consequent above conclusion

Figure 3.5: A forward step with a target conclusion

1:	(E∨F)∧(E∨G)	premise
2:	E∨F	∧ elim 1
3:	E∨G	∧ elim 1
4:	E	assumption
5:	E∨(F∧G)	∨ intro 4
6:	F	assumption
7:	E	assumption
	. . .	
8:	E∨(F∧G)	
9:	G	assumption
	. . .	
10:	E∨(F∧G)	
11:	E∨(F∧G)	∨ elim 3,7–8,9–10
12:	E∨(F∧G)	∨ elim 2,4–5,6–11

1:	(E∨F)∧(E∨G)	premise
2:	E∨F	∧ elim 1
3:	E∨G	∧ elim 1
4:	E	assumption
5:	E∨(F∧G)	∨ intro 4
6:	F	assumption
	. . .	
7:	E∨(F∧G)	
8:	E∨(F∧G)	∨ elim 2,4–5,6–7

(a) before: hypothesis and conclusion selected   (b) after: argument by cases outlined

Figure 3.6: ∨ elim needs a target conclusion

3.1.1 Selecting a target conclusion

Normally Jape writes the result of a forward step just after the hypothesis selection. Sometimes that may not be convenient, and it may be tidier to write it lower down the proof, just above a conclusion you are working towards.

In figure 3.4(a) only the hypothesis on line 1 is selected. A forward step — in this case "∧ elim (ignoring right)" — puts its consequent just below the selected hypothesis, as shown in figure 3.4(b).

If you select a target conclusion as well, as in figure 3.5(a), the same step will put its consequent before the line of dots above the selected conclusion, as shown in figure 3.5(b).

3.1.2 What can go wrong with a forward step?

When Jape can't make a forward step it's for one of the following reasons. In all cases you'll get an error message which explains the problem.

1. *No selected hypothesis*.
   If you don't select a hypothesis formula, you can't make a forward step.

2. *Not enough selections*.
   Some steps need more than a single hypothesis selection.

3. *Wrong hypothesis shape*.
   Most forward steps apply to a particular shape of hypothesis formula. If you select the wrong shape, you can't make the step.

4. *No target conclusion*.
   Two forward steps — ∨ elim, ∃ elim — need a conclusion selection as well as a hypothesis, because

```
1: (E∧F)∧G premise
   . . .
2: E∧(F∧G)
```
(a) conclusion selected

```
1: (E∧F)∧G premise
   . . .
2: E
   . . .
3: F∧G
4: E∧(F∧G)  ∧ intro 2,3
```
(b) two new conclusion antecedents
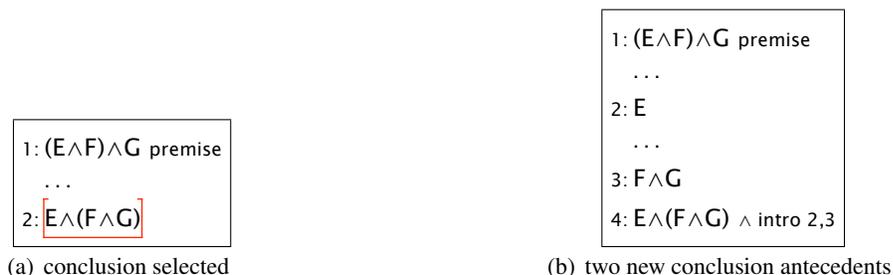
Figure 3.7: A sample backward step

```
1: (E∧F)∧G premise
2: E∧F        ∧ elim 1
3: E          ∧ elim 2
4: F          ∧ elim 2
5: G          ∧ elim 1
   . . .
6: E∧(F∧G)
```
(a) conclusion selected

```
1: (E∧F)∧G premise
2: E∧F        ∧ elim 1
3: E          ∧ elim 2
4: F          ∧ elim 2
5: G          ∧ elim 1
   . . .
6: F∧G
7: E∧(F∧G)  ∧ intro 3,6
```
(b) one hypothesis, one conclusion antecedent
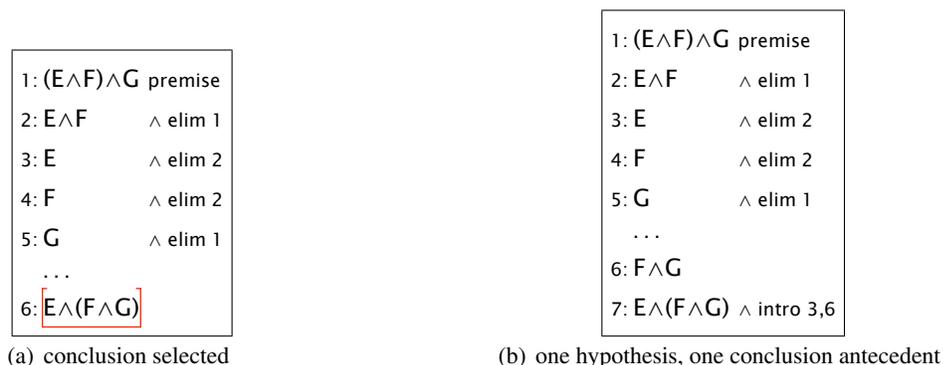
Figure 3.8: A sample backward step with a relevant hypothesis

the consequent can't be calculated from the antecedents. Jape writes the result of the step just above the target conclusion, and writes the justification next to the selected consequent. For example, see the ∨ elim step in figure 3.6. The result of the step (there's quite a lot of it!) is written above line 7 of the 'before' state, to form lines 7 to 10 of the 'after' state.

3.2   Making a backward step

Backward steps work on an open conclusion — a line without a justification, written just below a line of three dots. They may prove it completely, if Jape can find hypotheses to match the antecedents, or the antecedents may become unproved conclusions. For example, figure 3.7(a) shows an open conclusion selection, and figure 3.7(b) shows the effect of a backward ∧ intro step, where each antecedent has become a new open conclusion. Figure 3.8(a) shows the same conclusion selected when there are more hypothesis formulae available, and in figure 3.8(b) only one open conclusion is generated because line 3 matches the antecedent of the ∧ intro step.

In every case the justification of the step is written next to the selected conclusion, the consequent of the step.

3.2.1   What can go wrong with a backward step?

1. *Wrong conclusion shape*.

```
1: (E∧F)∨(E∧G) premise
   . . .
2: E
   . . .
3: F∨G
4: E∧(F∨G)        ∧ intro 2,3
```

(a) E is the consequent

```
1: (E∧F)∨(E∧G) premise
   . . .
2: E
   . . .
3: F∨G
4: E∧(F∨G)        ∧ intro 2,3
```

(b) $F \vee G$ is the consequent

Figure 3.9: Conclusion selection to tell Jape where to work

Each backward step, except for contra, applies to a particular shape of consequent formula. If you select the wrong shape, you can't make the step.

2. *No selected consequent.*
   If there is more than one unproved conclusion, Jape doesn't try to choose between them. In figure 3.7(b), for example, Jape wouldn't know where to apply a backward step. Selecting an unproved conclusion, as in figure 3.9, resolves the ambiguity. (Notice in figure 3.9(a) that line 2 is an ambiguous hypothesis/conclusion formula, selected as a conclusion by clicking on it's top half.)

3. *Hypothesis selected.*
   Backward steps (except for ∃ intro) don't need and can't make use of a selected hypothesis formula.

3.3 Steps which don't need a selection

If there is only one formula in the proof that can be selected as a hypothesis, it would be annoying to be told to select it before you can make a forward step. If there's only one unproved conclusion in the proof, it would be annoying to be told to select that to make a backward step. If the only possible target conclusion is on the next line to the hypothesis, it would be annoying to be told to select it when a step needs a target. So in all those cases Jape lets you get away without selection, and does the obvious thing.

Chapter 4

Rules of thumb for proof search

Rules of thumb[1] are guesswork, approximate guides that don't always work. Proof search is much easier if you recognise some simple principles.

1. Almost all the rules work on the *shape* of a hypothesis or consequent formula. Use the shape as a guide to help you choose a rule.

2. Use rules that introduce assumptions into the proof *as early as possible*. Those rules are:

   - $\rightarrow$ intro backwards (for the assumption);
   - $\vee$ elim forwards (for the assumptions);
   - $\neg$ intro backwards (for the assumption);
   - $\forall$ intro backwards (for the variable);
   - $\exists$ elim forwards (for the variable and the assumption).

3. $\vee$ intro works backwards better than it does forwards; but since it throws away half the conclusion you apply it to, use it very carefully and as late as possible.

4. $\neg$ elim works better forwards than it does backwards, once the contradictory formulae have been revealed.

5. If you use $\neg$ elim backwards, look for a hypothesis to unify with the unproved conclusion $\neg\_B$ (a negated unknown).

6. Classical contra — 'proof by contradiction' — can be used as a last resort in any situation. It introduces an assumption, and doesn't mind what shape the consequent is. (Constructive contra is just as applicable, but since it doesn't introduce an assumption, it isn't usually much help.)

---

[1] An inch was originally defined as the width of an adult male thumb, so a 'rule of thumb' is an approximate measure, then (by punning 'rule'=measurer) an approximate guide.

There is a Greek word 'heuristic' for rule-of-thumb, but I prefer the old English.

Some people object to the English phrase because there was once a folk tradition in England that a man could legally beat his wife with a stick no thicker than his thumb, and it was popularly called rule of thumb. (There's a reference, for example, in E.P. Thompson's *Customs in Common*.) The tradition was repulsive and mistaken, but (a) it's been forgotten, except by historians, and (b) the 'approximate guide' reading predates it.

7. If Jape won't let you make the proof, you're *doing it wrong*. There aren't any bugs in Jape's treatment of Natural Deduction, and all the proofs in the Conjectures and Classical conjectures panels are possible — I've done them all. Similarly, all the ones in the Invalid conjectures panel are impossible (they are included so that you can disprove them — see chapter 7).

8. Don't be afraid to Undo and Redo to search alternative routes to proof. Jape permits multiple Undos and corresponding Redos.

4.1   Look at the shape of the formula

There is a general principle: logical steps in Natural Deduction almost always simplify a formula, removing a connective ($\rightarrow$, $\land$, $\lor$ or $\neg$) or a quantifier ($\forall$ or $\exists$) and breaking the formula into its constituent parts. Persuasion (intro) rules do that working backwards, and use (elim) rules do it forwards. To choose a rule, look for the main connective (or the quantifier) in an unproved conclusion or a hypothesis, and use the rule which works on that connective (or quantifier).

The way that rules match formulae is so nearly mechanical that I could have set Jape up to choose the relevant rule when you merely double-click on a formula. Because I want you to learn about Natural Deduction and not just the use of the mouse, I've been grandad-ish and set it up so that you have to choose the rules for yourself.

Chapter 5

The steps summarised

5.1 ∧ steps

You use ∧ intro backwards by selecting an $A \land B$ conclusion and choosing "∧ intro" from the Backward menu. It generates new conclusion lines from the antecedents of the rule, as shown in figures 3.7 and 3.8 on page 18. I *strongly recommend* using ∧ intro backwards rather than forwards.

You use ∧ elim forwards by selecting an $A \to B$ hypothesis and choosing one of the two versions of the step from the Forward menu: "∧ elim (preserving left)" deduces A, and "∧ elim (preserving left)" deduces B. The effect is illustrated in figures 3.4 and 3.5 on page 16.

∧ intro forward is also possible: select two hypothesis formulae to be the antecedents and use "→ intro" from the Forward menu. But it's easier backwards, really it is. The problem in figure 5.1, for example, is easy if the first step is ∧ intro backwards — you have two things to prove and you have to prove them separately — and impossible almost any other way.

∧ elim backwards is possible, but for some reason or other I didn't include it (and I'm sure I had a good reason, so I'm not going to add it now).

5.2 → steps

You use → intro backward by selecting a $A \to B$ conclusion and then choosing "→ intro" from the Backward menu. See figure 5.2, for example. Notice that the step introduces a new assumption and therefore a new box (and that's why it should be used early).

You can use → elim forward by selecting a A hypothesis and a $A \to B$ hypothesis, as shown by the trivial example in figure 5.3. (Shift-click to select the second hypothesis: if you select the wrong thing either cancel it with a shift-click or click on the background to cancel all your selections, and start again.)

You can also use → elim half-forward, half-backward if you select a $A \to B$ hypothesis and a target conclusion and apply "→ elim" from the Forward menu. Jape writes a new conclusion A followed by a

1: $E \to (F \land G)$ premise

. . .

2: $(E \to F) \land (E \to G)$

Figure 5.1: ∧ intro backwards needed

(a) $E \to (F \to G)$ selected

(b) hypothetical argument outlined

Figure 5.2: $\to$ intro backward

(a) E and $E \to F$ selected

(b) consequent F deduced

Figure 5.3: $\to$ elim forward

(a) $F \to G$ and target conclusion selected

(b) consequent G deduced from new open conclusion F
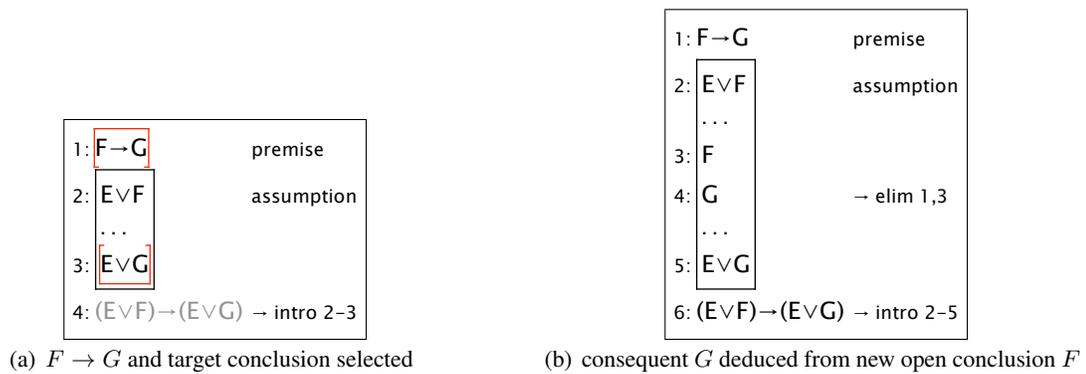
Figure 5.4: $\to$ elim half backward, half forward

```
1: ¬(¬E∨¬F) premise

2: ┌ ¬E        assumption
   │
   │ . . .
   │
3: │ ⎡¬E∨¬F⎤
   │
4: └ ⊥          ¬ elim 3,1

5: E            contra (classical) 2–4

   . . .

6: F

7: E∧F          ∧ intro 5,6
```

(a) before: conclusion selected

```
1: ¬(¬E∨¬F) premise

2: ┌ ¬E        assumption
   │
3: │ ¬E∨¬F     ∨ intro 2
   │
4: └ ⊥          ¬ elim 3,1

5: E            contra (classical) 2–4

   . . .

6: F

7: E∧F          ∧ intro 5,6
```

(b) after: uncertainty resolved
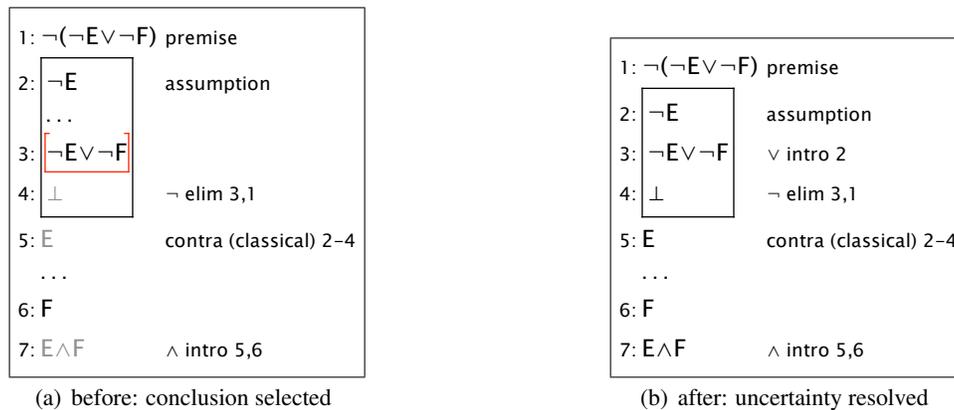
Figure 5.5: ∨ intro backwards is easy

```
1: ¬(¬E∨¬F) premise

2: ┌ ⎡¬E⎤       assumption
   │ . . .
   │
3: └ ⊥

4: E            contra (classical) 2–3

   . . .

5: F

6: E∧F          ∧ intro 4,5
```

(a) before: hypothesis selected

```
1: ¬(¬E∨¬F) premise

2: ┌ ¬E        assumption
   │
3: │ ⎡¬E∨_B1⎤   ∨ intro 2
   │ . . .
   │
4: └ ⊥

5: E            contra (classical) 2–4

   . . .

6: F

7: E∧F          ∧ intro 5,6
```

(b) after: 'unknown' formula introduced
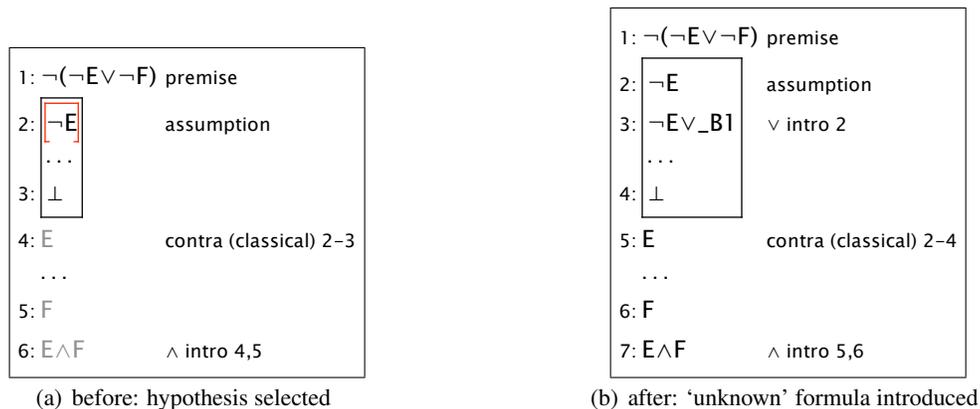
Figure 5.6: ∨ intro forwards generates an unknown

consequent B, as shown in figure 5.4. It's a half-backward step because it introduces a new conclusion.

$\to$ intro forward is just daft, and Jape won't attempt it. $\to$ elim backward is possible, but it introduces unknowns and there's no real advantage, so I don't let Jape try.

5.3 ∨ steps

Because ∨ elim forwards implements argument by cases, and because the consequent C can't be deduced from the $A \vee B$ antecedent, it generates big proof changes from a small gesture. That frightens novices, but be brave, because ∨ elim is one of the rules that generates assumptions, so you have to use it as early as possible in a proof.

∨ intro backwards is destructive — it throws away half a conclusion — so you use it as *late* as possible, even though it's very easy to use.

You use ∨ elim forwards by selecting a hypothesis which fits $A \vee B$, an open conclusion C (if there's only one available conclusion, and if its on the line below the hypothesis, Jape will let you off the conclusion selection) and "∨ elim" from the Forward menu. See figure 3.6 on page 17, for example. Note that the boxes representing the A leads to C and B leads to C arguments (lines 7-8 and 9-10 in figure 3.6(a), for example) are written just above the selected conclusion C, and the justification is (of course) written against

(a) before: hypothesis ¬($E \lor F$) and target conclusion selected

(b) after: conclusion $E \lor F$ generated
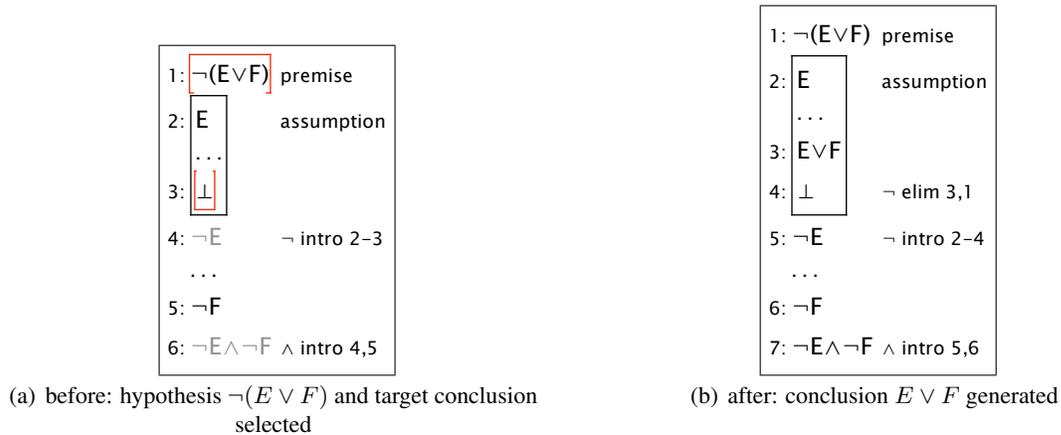
Figure 5.7: ¬ elim forwards is easy

(a) before: conclusion ¬F selected

(b) after: contradiction argument outlined
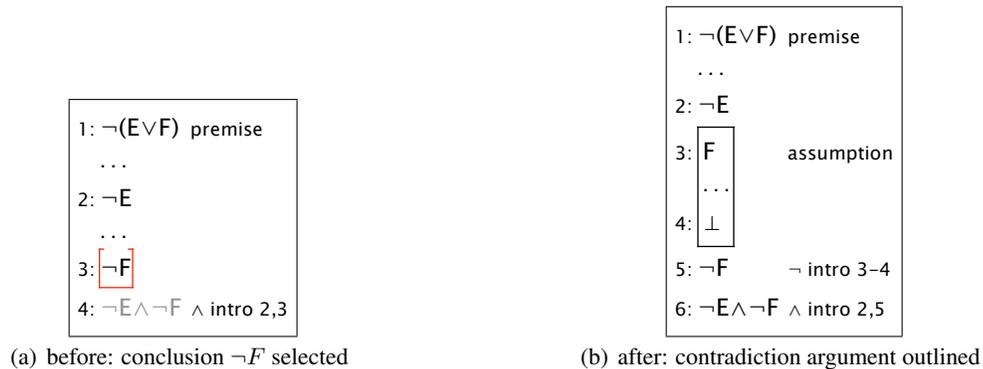
Figure 5.8: ¬ intro backwards is very easy

the selected conclusion.

∨ intro is easy to use backwards: select an open conclusion, decide which half to keep and which to throw away, and apply the corresponding step ("∨ intro (preserving left)" or "∨ intro (preserving right)") from the Backward menu. See figure 5.5, for example.

For some reason that I've forgotten, I was persuaded to allow ∨ intro forward. I rather regret it, because this isn't a step for novices to use. But, if you must: you select a hypothesis, decide which half of the consequent it has to be, and apply the corresponding step ("∨ intro (inventing left)" or "∨ intro (inventing right)") from the Forward menu. The step always invents an unknown, as illustrated in figure 5.6 by an "inventing right" step, and you have to deal with the unknown somehow (see chapter 6 for suggestions).

If unknowns frighten you, don't use ∨ intro forwards (the easy way to solve the proof problem in figure 5.6, for example, is ¬ elim forwards from line 1 with target conclusion line 4, producing figure 5.5(a); then ∨ intro backwards preserving left produces figure 5.5(b)).

∨ elim backwards would be possible, but not for novices, so I didn't allow it.

## 5.4   ¬ steps

¬ elim works best forwards, ¬ intro backwards.

1:	¬(E∨F)	premise
2:	E	assumption
	. . .	
3:	⊥	
4:	¬E	¬ intro 2–3
	. . .	
5:	¬F	
6:	¬E∧¬F	∧ intro 4,5

(a) before: conclusion ⊥ selected

1:	¬(E∨F)	premise
2:	E	assumption
	. . .	
3:	_B1	
	. . .	
4:	¬_B1	
5:	⊥	¬ elim 3,4
6:	¬E	¬ intro 2–5
	. . .	
7:	¬F	
8:	¬E∧¬F	∧ intro 6,7

(b) after: unknown _B1 invented, and two conclusions generated
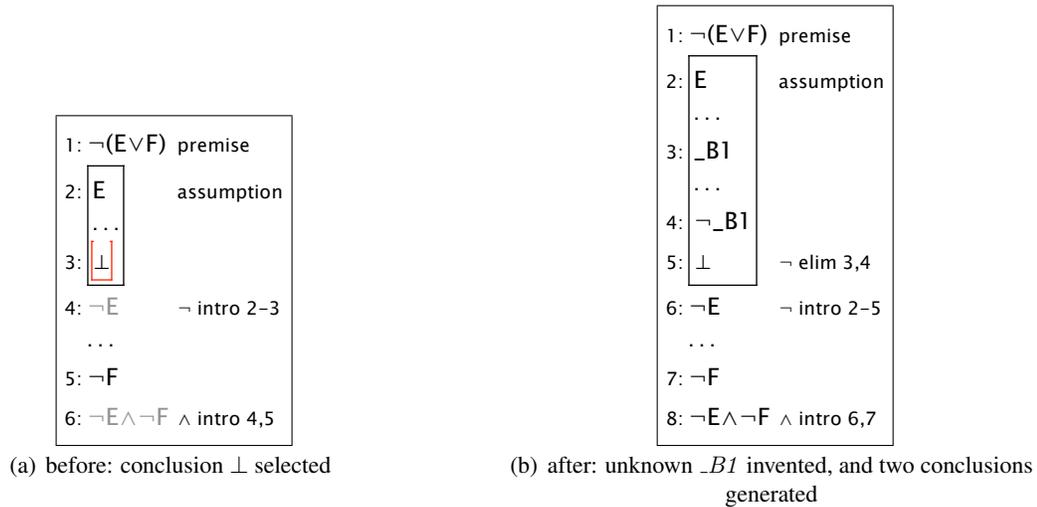
Figure 5.9: ¬ elim backwards is harder

	. . .
1:	E∨¬E

(a) before: conclusion $E \vee \neg E$ selected

1:	¬(E∨¬E)	assumption
	. . .	
2:	⊥	
3:	E∨¬E	contra (classical) 1–2

(b) after: contradiction argument outlined
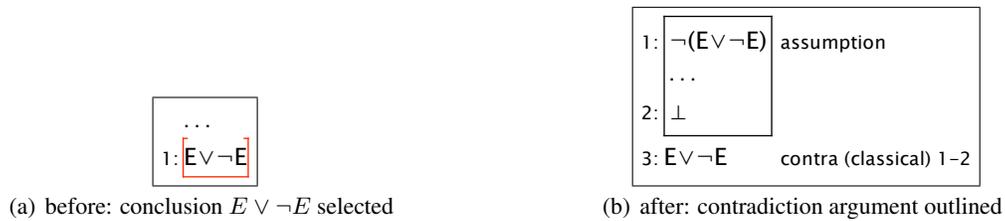
Figure 5.10: A sample classical contradiction step

To use ¬ elim forwards, select a hypothesis ¬A, or two hypotheses ¬A and A, and choose "¬ elim" from the Forward menu. You can select a target conclusion too, if you like. The step generates a consequent ⊥, (plus a conclusion A if you only select one hypothesis). See figure 5.7, for example.

To use ¬ intro backwards, select an open conclusion ¬B and "¬ intro" from the Backward menu. See figure 5.8, for example.

It's possible to use ¬ elim backwards, if you select an open conclusion ⊥. It generates an unknown _B and two new open conclusions ¬_B and ¬_B, as illustrated in figure 5.9. It's usually easier to do it forwards, though.

¬ intro forwards doesn't make much sense, so I didn't allow it.

1:	E∧¬E	premise
2:	¬E	∧ elim 1
	. . .	
3:	E	
4:	⊥	¬ elim 3,2
	. . .	
5:	F	

(a) before: hypothesis ⊥ and target conclusion selected

1:	E∧¬E	premise
2:	¬E	∧ elim 1
	. . .	
3:	E	
4:	⊥	¬ elim 3,2
5:	F	contra (constructive) 4

(b) after: conclusion 'proved'
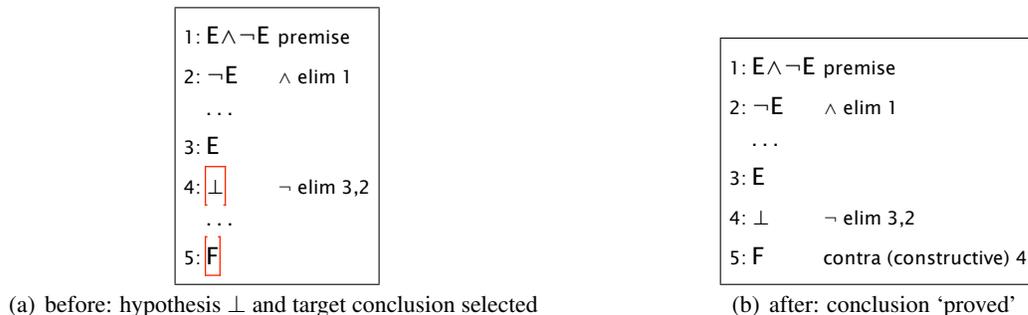
Figure 5.11: A sample constructive contradiction step

(a) before: $\forall x.\neg R(x)$ conclusion selected

(b) after: generalised proof outline introduced, with private variable i
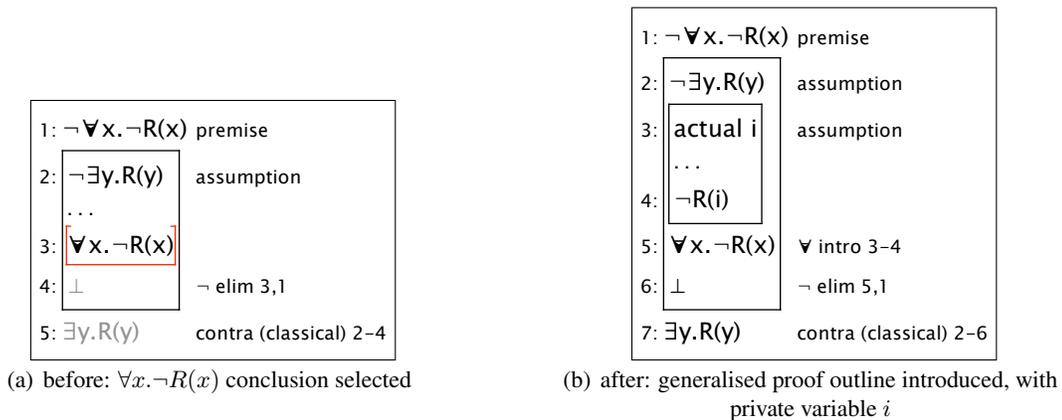
Figure 5.12: A $\forall$ intro step backwards

5.5 ⊥ (contradiction) steps

Classical contra is a hard rule to use (there ought to be a blues song about that), but you know you are going to have to use it for some problems. It only works backwards. Constructive contra is easier if you use it forwards.

To use classical contra, select an open conclusion A and choose "contra (classical)" from the Backward menu. It creates a hypothetical contradiction argument with assumption $\neg A$, as shown in figure 5.10.

To use constructive contra forwards, select a ⊥ hypothesis and an open conclusion, and choose "contra (constructive)" from the Forward menu. See figure 5.11, for example.

Constructive contra backwards is destructive: it throws away whatever conclusion it is applied to. But you can do it if you want to. Classical contra forwards would be absurd.

5.6 ⊤ (truth) step

You can always prove ⊤ as an open conclusion: select it and apply "truth" from the Backward menu. And that's all you can do with ⊤: no forward step is possible.

5.7 ∀ steps

$\forall$ intro works backwards, $\forall$ elim forwards. $\forall$ intro is worth using early, because it introduces a variable. The privacy condition on the $\forall$ intro step won't usually bother you, because Jape always invents a new variable (i, $i1$, $i2$, and so on).

To use $\forall$ intro backwards, select an open conclusion $\forall x.P(x)$ and choose "$\forall$ intro" from the Backward menu. Jape builds the outline of the generalised proof, as illustrated in figure 5.12.

To use $\forall$ elim forwards, you must select a hypothesis $\forall x.P(x)$ and also $\mathrm{actual}\ j$, then choose "$\forall$ elim" from the Forward menu. The effect is illustrated in figure 5.13.
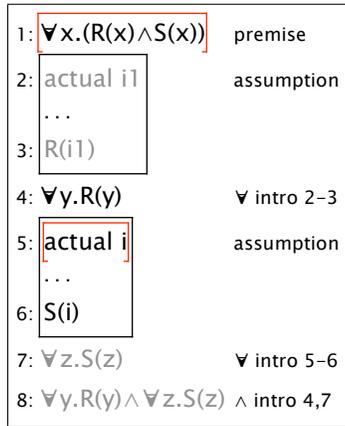
(a) before: ∀x.(R(x) ∧ S(x)) and actual i hypotheses selected
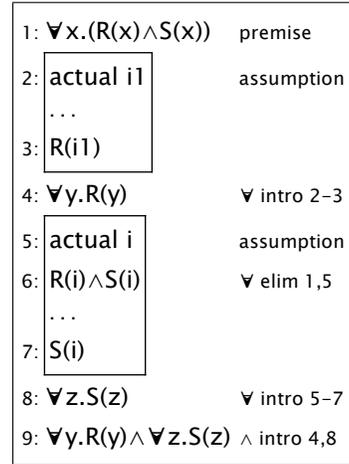
(b) after: R(i) ∧ S(i) consequent deduced

Figure 5.13: A ∀ elim step forwards
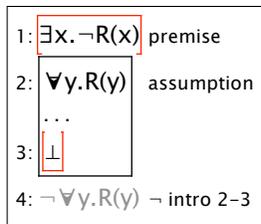
(a) before: ∃x.¬R(x) hypothesis and target conclusion ⊥ selected
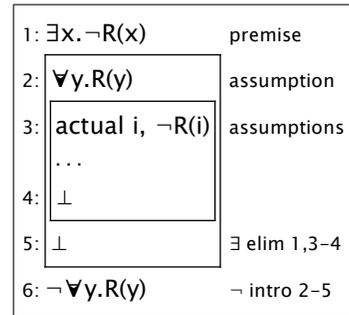
(b) after: generalised proof outline introduced, with private variable i

Figure 5.14: An ∃ elim step forwards
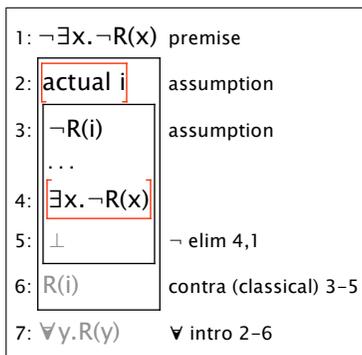
(a) before: ∃x.¬R(x) conclusion and actual i hypothesis selected
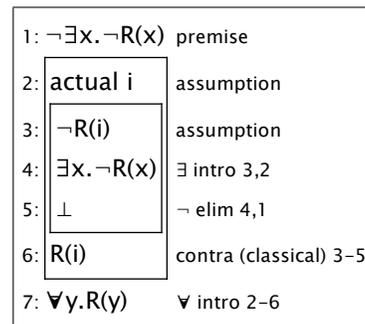
(b) after: ¬R(i) antecedent deduced

Figure 5.15: An ∃ intro step backwards

5.8   ∃ **steps**

∃ intro works backwards, ∃ elim forwards. ∃ elim is worth using early, because it introduces a variable. The privacy condition on the ∃ elim step won't usually bother you, because Jape always invents a new variable (i, $i1$, $i2$, and so on).

To use ∃ elim forwards, you must select a hypothesis $\exists x.P(x)$ and also a target conclusion, and then choose "∃ elim" from the Forward menu. The justification is written next to the target conclusion, and a generalised proof outline is introduced, as illustrated in figure 5.14.

To use ∃ intro backwards, you must select a conclusion $\exists x.P(x)$ and also a hypothesis $\text{actual } j$, and then choose "∃ intro" from the Backward menu. The effect is illustrated in figure 5.15.

Chapter 6

Unknowns, hyp and Unify

When James Aczel looked at novices learning logic with Jape, he pointed out that they found *incomplete steps* quite disconcerting. Incomplete steps are ones that let you leave out important information so that you can fill it in later, when you've discovered what it ought to be. You might be allowed to leave out the variable in an ∃ intro step, for example, and fill it in later.

Jape uses *unknowns* — names starting with an underscore, like _B1 — to stand for formulae which can be filled in later. Even though James persuaded me to eliminate almost all incomplete steps from my treatment of natural deduction, my users pleaded with me to allow some. So it is possible to introduce unknowns into a Jape proof, and because of that it's necessary to know how to get rid of them again.

To understand what follows you have to recognise the distinction between *formula selection* (red box round a conclusion or a hypothesis formula) and *subformula selection* (yellow background behind part or all of a conclusion or hypothesis formula). Note, in particular, that a subformula selection that encompasses a complete formula is *not* the same thing as a formula selection.

6.1 Introducing an unknown

∨ elim backwards takes a formula with an ∨ connective and throws away half of it, because from A you can prove $A \vee B$, and from B you can prove $A \vee B$. Some obsessively-forward reasoners wanted me to allow ∨ intro forwards, and just to spite them I did: it's there in the Forward menu, under the line. Figure 6.1 shows how you can deduce $A \vee {\_B}$ from A by choosing "∨ intro (inventing right)" from the Forward menu.

¬ elim backwards is another good way to get an unknown, illustrated in figure 6.2. There's only one unknown in the proof, but it occurs twice.

These are by no means the only way to introduce unknowns into a proof. One very interesting way is to select a conclusion, choose "Text command" from the File menu, type "apply cut" and press return. You
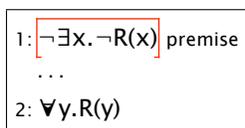
1: ¬∃x.¬R(x) premise
...
2: ∀y.R(y)

(a) before: $\exists x.\neg R(x)$ hypothesis selected
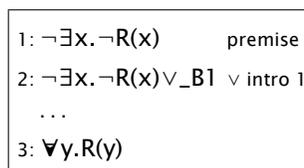
1: ¬∃x.¬R(x)   premise
2: ¬∃x.¬R(x)∨_B1 ∨ intro 1
...
3: ∀y.R(y)

(b) after: unknown _B1 introduced in consequent

Figure 6.1: An ∨ intro step forwards

```
1: ¬∃x.¬R(x) premise                              1: ¬∃x.¬R(x) premise

   . . .                                             . . .
2: ⊥                                              2: _B1

3: ∀y.R(y)       contra (constructive) 2            . . .
                                                  3: ¬_B1

                                                  4: ⊥                ¬ elim 2,3
                                                  5: ∀y.R(y)     contra (constructive) 4
```

(a) before: conclusion is ⊥   (b) after: unknown _B1 introduced in antecedents
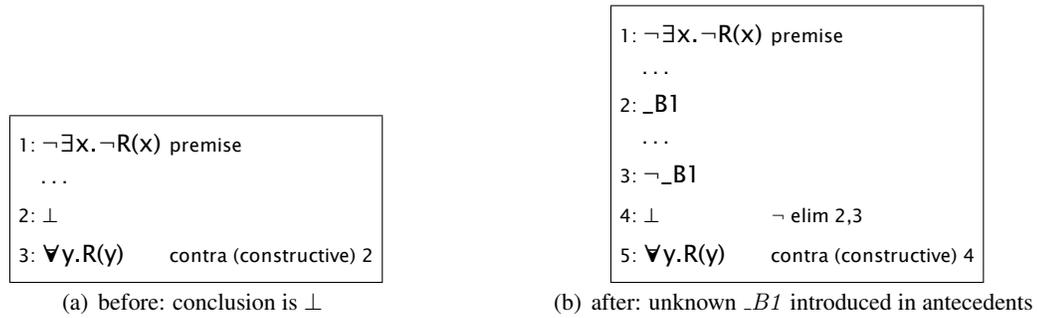
Figure 6.2: A ¬ elim step backwards

```
                                                  1: ¬∃x.¬R(x) premise

                                                     . . .
                                                  2: ¬R(x)

1: ¬∃x.¬R(x) premise                                 . . .
                                                  3: ¬¬R(x)
   . . .
2: ⊥                                              4: ⊥                ¬ elim 2,3
3: ∀y.R(y)       contra (constructive) 2          5: ∀y.R(y)     contra (constructive) 4
```

(a) before: conclusion ⊥ selected, subformula ¬*R*(*x*)   (b) after: ¬*R*(*x*) used in antecedents of ¬ elim
            selected
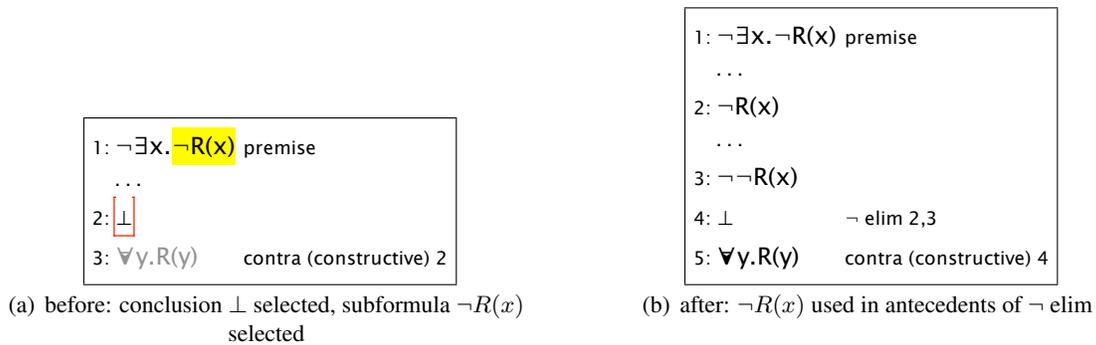
Figure 6.3: A ¬ elim step backwards with subformula selection

will get an unknown, intermediate between the hypotheses and your conclusion. I leave you to work out how useful that can be.

6.2   Avoiding unknowns by subformula selection

If you know beforehand what should go in place of the unknown, you can tell Jape what to use at the time you make an ∨ intro or ¬ elim step, by subformula selection (alt/option/middle-press-and-drag over the formula or subformula you want to use). Figure 6.3 shows an example: note that the hypothesis is *not* selected, but ¬*R*(*x*) is subformula-selected.

```
1: ¬∃x.¬R(x) premise

   . . .
2: _B1                                            1: ¬∃x.¬R(x) premise

   . . .                                             . . .
3: ¬_B1                                           2: ∃x.¬R(x)

4: ⊥         ¬ elim 2,3                            3: ⊥                ¬ elim 2,1
5: ∀y.R(y)     contra (constructive) 4            4: ∀y.R(y)     contra (constructive) 3
```

(a) before: subformulae ∃*x*.¬*R*(*x*) and _B1 selected   (b) after: instances of _B1 replaced by ∃*x*.¬*R*(*x*)
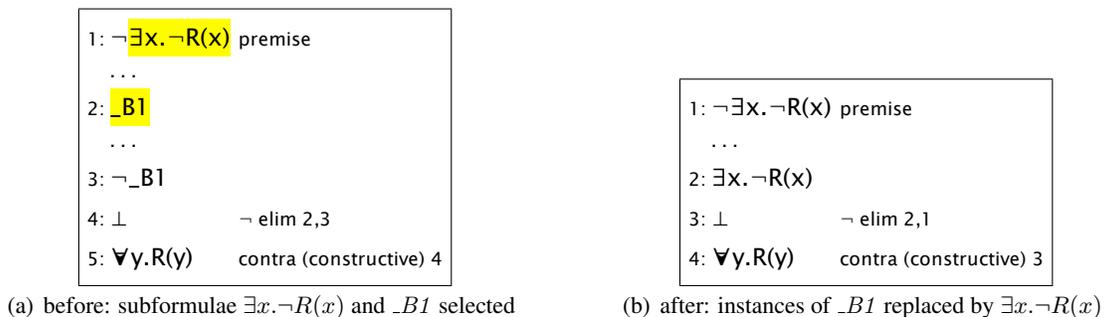
Figure 6.4: Effect of a Unify command

(a) before: hypothesis $\neg\exists x.\neg R(x)$ and conclusion $\neg\_B1$ selected

(b) after: instances of $\_B1$ replaced by $\exists x.\neg R(x)$

Figure 6.5: Effect of `hyp` on an unknown
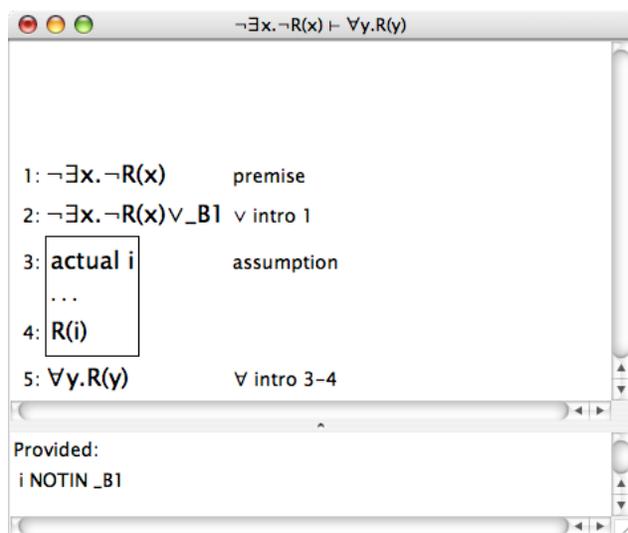

Figure 6.6: Proof with proviso

Figure 6.7: Preparation for proviso-violating unification

6.3   Eliminating an unknown with Unify

If you have an unknown in your proof, and a subformula somewhere in the proof which you want to use to replace the unknown, then subformula selection and the "Unify" command from the Edit menu will do the job. You subformula-select two or more subformulae which you want to make the same by replacing unknowns (command/control-alt/option/middle-press-and-drag is the gesture you need to make more than one subformula selection). Figure 6.4 shows an example. Note that there are no formula selections in figure 6.4(a), only subformula selections.

6.4   Eliminating an unknown with `hyp`

The `hyp` step is really just a cloak for the Unify command: make this selected conclusion the same as this selected hypothesis (or these selected hypotheses). For example see figure 6.5. Note that making $\neg\exists x.\neg R(x)$ the same as $\neg\_B1$ means making $\exists x.\neg R(x)$ the same as $\_B1$.
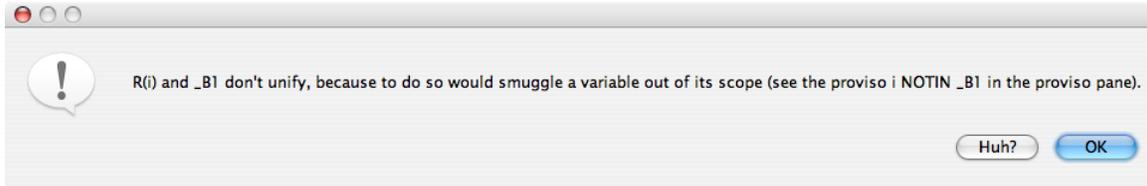
Figure 6.8: Jape's reason for rejecting unification step

6.5   Provisos and the privacy condition

Recall that the $\forall$ intro and $\exists$ elim steps introduce a variable — i, $i1$, $i2$, ... — for private use within a generalised proof. If a proof attempt doesn't include unknowns, Jape can enforce the privacy condition easily: simply use a variable that isn't in use already, anywhere in the proof. If there are unknowns about, things aren't so simple: Jape has to ensure that you don't unify that unknown with a formula that uses the private variable, because that could violate the privacy condition.

Figure 6.6 shows what happens when Jape has to invent a variable and there is an unknown in the proof. It must prohibit the possibility that the formula _B1 includes the variable i: it does this with the *proviso* "i NOTIN _B1" in the *proviso pane* below the proof. Figure 6.7 shows preparation for an attempt to violate the proviso with a Unify command (making _B1, which appears outside the box, the same as $R(i)$ inside the box). Figure 6.8 shows the error message that Jape generates when you try the unification.

You get the same situation if you try the steps in the other order ($\forall$ intro backwards, $\vee$ intro forwards selecting only the hypothesis).

Chapter 7

Disproof

My encoding of natural deduction in Jape deals with disproof by allowing you to make Kripke diagrams which Jape will check against a sequent to see if they are examples, or counter-examples, or neither. You do it by manipulating blobs, lines and atomic formulae in a *disproof pane* above the proof.

Jape shows you what is going on by underlining and colouring formulae and subformulae in the sequent and by colouring blobs on the screen. This chapter should, therefore, be read in colour rather than in monochrome.[1]

7.1 Getting started

At any point during a proof attempt you can invoke 'Disprove' from the Edit menu.[2] Jape splits the proof window and shows you a Kripke diagram with a single empty world and the proof-attempt sequent (you can select hypotheses and conclusions to construct an alternative sequent: see below) in a *disproof pane* above the proof pane. The sequent may already be underlined and coloured as in figure 7.1.

The main things to notice in this illustration are

- the blob in the diagram is ringed in red;

- some of the names and connectives in the sequent are coloured magenta, some are grey, and some are black;

- some of the formulae are underlined and some are not;

- the sequent uses the semantic turnstile ⊨;

- there's a little green waste bin on the right;

- above the waste bin there are some tiles, each containing an atomic formula.

---

[1] I am aware that use of colour means that some people can't use Jape as easily as others. This is only the tip of a nasty iceberg: Jape uses a visual presentation to make it easier to understand proof and disproof, and that excludes many people with reduced visual acuity from using it at all.

Contrary to some propaganda, these problems aren't easy to overcome. But I'd love to try, and I'd love to hear from anybody who has suggestions to make, advice to offer, technical information to share, guinea-piggery to volunteer, whatever. Messages to me at the title page address, as usual.

[2] There isn't a Disprove button in the conjecture panels. Perhaps there should be! You have to hit Prove to start a proof attempt and then choose Edit:Disprove.
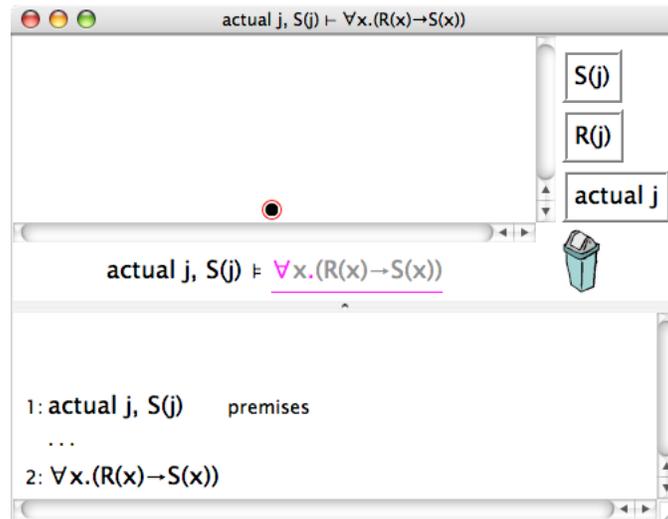
Figure 7.1: Proof attempt with proof pane below and disproof pane above
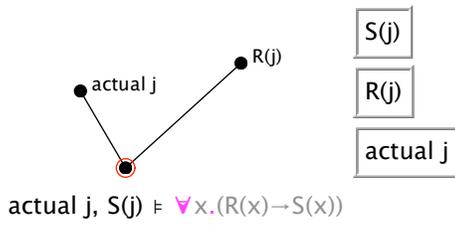

Figure 7.2: A multi-world situation
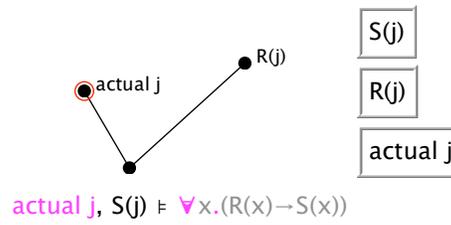
Figure 7.3: A single-world situation

In general magenta means 'forced', black means 'unforced' and grey means 'irrelevant'.

Because of operating system variations, your window may not look exactly like the illustration, but it will have the same parts. In other illustrations I show the important bits of the proof and/or disproof panes, without the waste bin and the scroll bars and stuff.

7.2   Alternative sequents

If you select a conclusion in the proof pane (or, if Jape doesn't want to let you do that, the reason next to the conclusion) and then hit Edit:Disprove, Jape will use the disproof sequent consisting of the conclusion as consequent and all the hypotheses above it as antecedent formulae. If you select some of the hypotheses as well as the conclusion, they will be used as the antecedents.

You can choose a new disproof sequent at any time, even in the middle of a disproof attempt.

If you want to set up an entirely new challenge, enter it into one of the conjectures panels (New button), hit Prove and then Edit:Disprove in the proof window.

7.2.1   Selecting a situation

There is always one selected world in a diagram, ringed in red. The selected world is the root world of the situation evaluated by Jape. For example, in figure 7.2 the situation is the whole diagram; in figure 7.3 it's
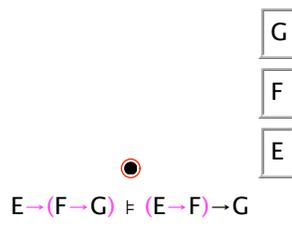
Figure 7.4: A simple counter-example
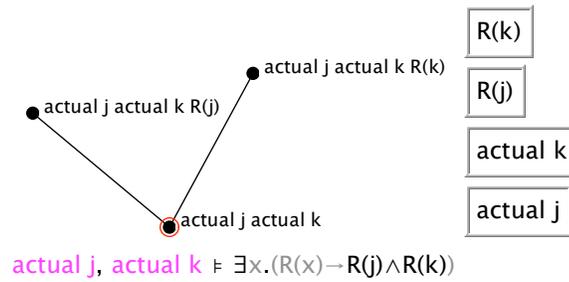
Figure 7.5: An elaborate counter-example

the single world at top left.

7.2.2 What's forced ? — colouring, greying, underlining

Jape evaluates all the formulae and sub-formulae of the sequent in the situation with the selected world as its root, and colours all those formulae and sub-formulae.

- The default colour is black, and it means *not forced*.

- If a (sub-)formula's value is never called on — usually this is true of quantified predicates, for example — then it's grey, meaning *strictly irrrelevant*.

- An atomic (sub-)formula which is *forced* is coloured magenta (this covers atoms like E, atomic predicates like $S(j)$ and individuals like $\mathrm{actual}\,k$).

- When a non-atomic (sub-)formula is forced, its connective is coloured magenta.

- When an entire sequent premise or conclusion is forced, it is underlined in magenta.

So: when all the premises are magenta-underlined and the conclusion is not, you have a counter-example; when everything is underlined you have an example; otherwise you have nothing in particular.

Figure 7.4 shows a sample counter-example, and the colouring gives a hint of why it's a counter-example: E isn't forced anywhere, so the premise implication is forced (magenta arrow, underlining); and then, because $E \to F$ is forced but G isn't, the conclusion isn't forced (black arrow, no underlining). Figure 7.5 shows an elaborate counter-example: the premises are obviously forced, but there are no individuals to force the $\exists$ quantification in the conclusion.

The disproof pane in figure 7.1 shows neither a counter-example nor an example. The premises aren't forced because, as atomic formulae, they aren't present at the root world; the conclusion is forced because every individual in the situation (there aren't any ... remember the green sheep) forces a version of $R(x) \to S(x)$. Similar remarks apply to figures 7.2 and 7.3.

7.3 Making diagrams

To begin your disproof Jape presents you with the simplest diagram: the isolated empty world. You add components to your diagram — worlds, formulae and lines — using drag-and-drop mouse gestures. You can delete components by dragging them to the waste bin. You can add tiles with double-clicks. You can Undo your actions to any degree that you like, and Redo likewise.[3]

To *move* a component of a diagram you put the mouse pointer over it, press (left button on a multi-button mouse), hold a moment, then — still pressing — move to a new position, and finally release. Jape shows you a transparent image of the thing you are dragging, so you can see what you are doing.

To *duplicate* a component you hold down the alt/option key throughout the gesture (or use the middle button on a multi-button mouse) and you drag a new copy of the thing you pressed on.

Sometimes the purpose of the drag is to *drop* one component onto, or into, another. You drag the first component until the mouse pointer is over the one you want to drop onto/into. The recipient will change colour if it's prepared to accept the drop, and you simply let go while it's lit up. If the drop fails (you aren't over a component, or it won't light up) then the dropped component flies back to base.

7.3.1   Dragging worlds

You can move-drag worlds about the place to make your diagram look nicer. You press on the blob, and you drag the blob plus any formulae attached to it. Lines connected to the world stay connected, and you can drop it anywhere you like. To preserve the 'connections only upwards' principle, Jape will delete a line if you drag a world below its parent.

By duplicate-dragging a blob, you drag a copy of the world — the blob plus any formulae attached to it — with a line attaching it to the world it came from. The new world stays where you put it. If it's above the world you dragged it from, they will be connected by a line. If it's level or below, then no line.

You can drag a world onto a line (which lights up to show you are over it) or another world (which lights up ditto) and Jape does the obvious thing, adding whatever formulae are necessary to whatever worlds in order to maintain monotonicity.

If you drag a world to the waste bin it's deleted, along with all the formulae attached to it and all the lines connected to it. The bin lights up when it will accept the world; it won't accept the currently-selected world.

If you drop the world onto another the two worlds are merged. Jape does the correct monotonicity adjustments to the diagram by adding formulae to the children of the destination.

7.3.2   Dragging lines

You can drag lines and drop them onto worlds or into the waste bin, as you wish. Jape does the obvious thing in each case (if you can't guess what that is, just try it!).

If you want to make a line between world A and world B, where A is below B, duplicate-drag A to B and drop the new world onto B. Jape makes the monotonicity adjustments, adding formulae to B and its children as necessary. Dragging from B to A will have some not-very-useful effect (but there's always Undo!).

7.3.3   Dragging formulae

You can drop a formula on a world, or drag it away from a world, in one of several ways. If a formula is added to a world then, to preserve monotonicity, it is added to the children of that world as well. If a formula is deleted from a world then it is deleted from all ancestors of that world.

You can drag a formula from a tile and drop it onto a world (the world lights up, of course, to show that it's eager to accept; worlds won't accept a second copy of a formula they already have). The formula isn't deleted from the tile, whichever kind of drag gesture you use, because tiles are infinite sources of formula-copies.

---

[3] Undo and Redo apply to the last pane you used the mouse in, either proof or disproof.
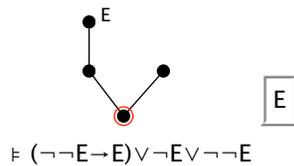
Figure 7.6: A counter-example
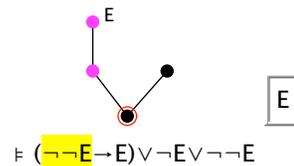
Figure 7.7: $\neg\neg E$ forced here
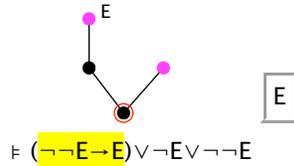
Figure 7.8: $\neg\neg E \rightarrow E$ forced here
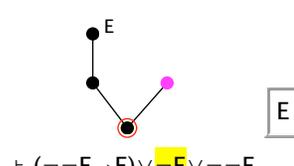
Figure 7.9: $\neg E$ forced here
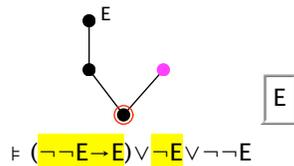
Figure 7.10: Multiple forcing
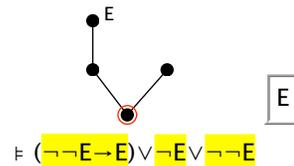
Figure 7.11: Too much forcing

You can move-drag a formula from one world onto another; it is added to the destination world and deleted from the source world. If you duplicate-drag it instead, it is simply added to the destination world.

7.4 Making individuals and predicate instances

If a sequent mentions an individual by including a hypothesis like $\text{actual}\, j$, then there will be a tile for that individual (see figures 7.1, 7.2, 7.3 and 7.5, for example). If there are quantifiers but no individuals you will get a free $\text{actual}\, i$ tile.

If you need more individuals, double-click any "actual ..." tile. Jape uses an obvious numbering algorithm to name the new individual — $\text{actual}\, j$, for example, generates first $\text{actual}\, j1$ then $\text{actual}\, j2$, and so on.[4]

If you want to make a new predicate instance, double-click a predicate-instance tile. Jape looks through your tiles for unused individuals and uses them to give you a new one. For example if you have a tile for $S(j)$ and tiles for $\text{actual}\, j$ and $\text{actual}\, j1$, then double-clicking $S(j)$ gets you $S(j1)$, as you might expect. If there's more than one instance that could be made, Jape shows you the alternatives and asks you to choose. If you don't have enough individuals to make a new instance, it tells you so.

7.5 Exploring reasons

The hardest thing to explain to a novice is *why* a particular formula is forced in a particular situation. Figure 7.6 shows a particularly ludicrous formula and its counter-example, which really needs explanation. Jape provides mechanisms which can be some help.

---

[4] It would be nicer if it went i, j, k, $i1$, $j1$, $k1$, $i2$, ... One day I'll make it do that.
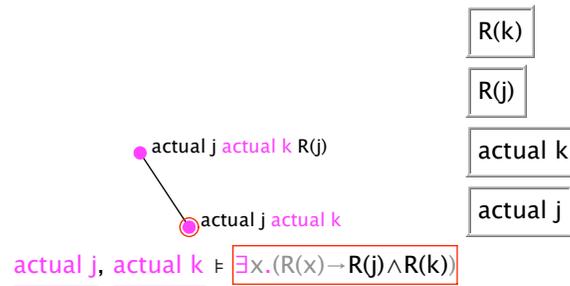
Figure 7.12: Support for a quantifier

The colouring of atomic (sub-)formulae and connectives is the first mechanism. But with the negative connectives — $\rightarrow$ and $\neg$ — and with the quantifiers, we often need a bit more help.

Jape allows you to select (click or left-click) or subformula-select (alt/option/middle-press-and-drag) in the sequent. It will colour magenta all the worlds which force the selected (sub)formula. For example, figure 7.7 shows where $\neg\neg E$ is forced in the difficult example. Figures 7.8 and 7.9 show information about some other subformulae.

If you choose more than one formula, Jape colours the worlds that force them all: see figures 7.10 and 7.11.

If you choose a quantifier formula in the sequent, Jape colours magenta not only the worlds which support it, but also the presence markers which support that formula: i.e. individuals at worlds which which generate a forced formula if you instantiate the quantified formula with them. In figure 7.12, for example, worlds which force the selected quantifier are magenta; individuals which support it are magenta too. You can see that $R(k) \rightarrow R(j) \wedge R(k)$ at each of the worlds in this example.

7.6 Completing a disproof

When you have a disproof (all the premises underlined, the conclusion not underlined), you can register it with Jape using Edit:Done. Disproved conjectures are marked with a cross; proved conjectures get a tick. Because you can prove some conjectures classically and also disprove them constructively, it's possible to get both a tick and a cross together. See figure 1.5 on page 8 for examples.

7.7 Printing disproofs

You can print or export an image of the disproof pane, using Edit:Print Disproof or Edit:Export Disproof.

Chapter 8

Using theorems and stating conjectures

8.1 Using theorems

A theorem is a proved conjecture, one with a tick next to it in a conjecture panel. Jape lets you use the theorem

$$A_1, A_2, \ldots, A_n \vdash B$$

as if it was the rule

$$\frac{A_1 \quad A_2 \quad \ldots \quad A_n}{B}$$

You can apply a theorem backwards or forwards using the Apply button in its conjecture panel.

To work backwards, just select an open conclusion in the proof window, select the theorem in its panel, and press the Apply button. Jape will apply the theorem like any rule, instantiating it to fit the conclusion, making antecedents of its premises and entering them as hypotheses or linking to existing hypotheses if possible. See figure 8.1, for example.

Because forward steps take a lot of careful use of behind-the-scenes technology, theorems work best backwards. But sometimes you must work forwards, and you can do that in two ways.

If your theorem has exactly one premise, you can select a hypothesis to match that premise and apply the theorem like a forward rule, as illustrated in figure 8.2.

If your theorem has no premises or more than one premise, don't select a hypothesis but do select a conclusion and apply the theorem. It sounds daft but it works: Jape makes a version of the theorem in which all the variable and formula names are replaced by unknowns and inserts it into the proof, as illustrated in figure 8.3. You have to get rid of the unknowns using `hyp` steps or the Unify command, as described in chapter 6.

(a) before: conclusion $E \vee \neg E$ selected   (b) after: hypothesis $\neg(\neg E \wedge \neg\neg E)$ generated
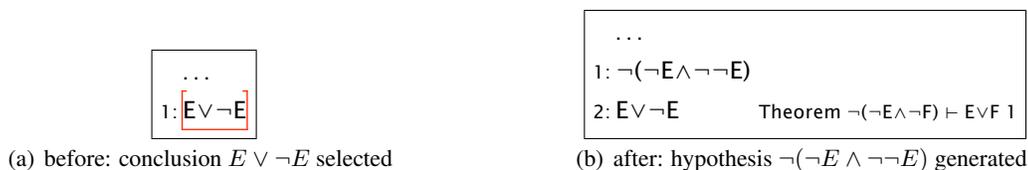
Figure 8.1: Theorem applied backward

(a) before: hypothesis $(E \to F) \to E$ selected   (b) after: consequent $(E \wedge F) \to E$ generated
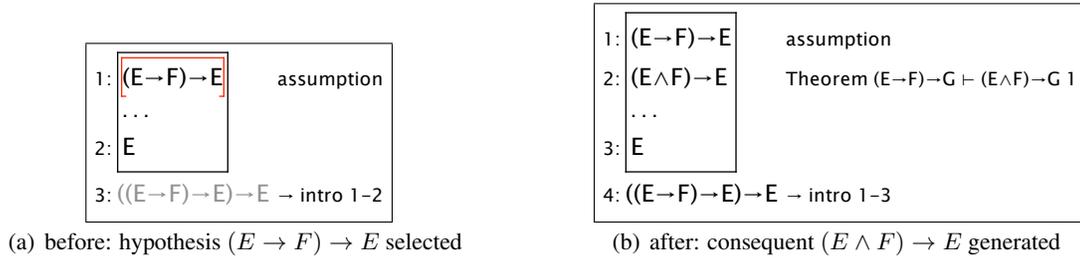
Figure 8.2: Single-premise theorem applied forward

(a) before: conclusion selected   (b) after: theorem inserted, with unknowns
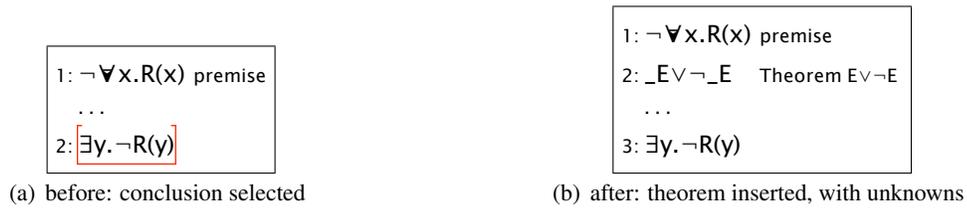
Figure 8.3: Theorem applied to non-matching conclusion
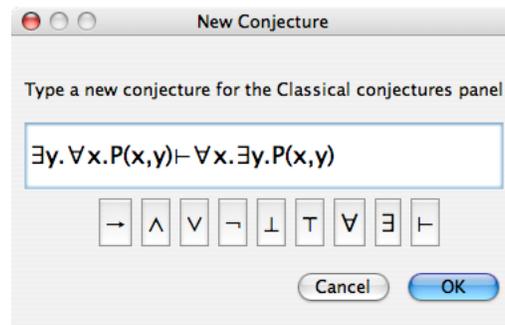

Figure 8.4: The New Conjecture window

8.2 Stating your own conjectures

Press the "New..." button on any conjectures panel and you see a window like figure 8.4. You type your conjecture using the keyboard in the normal way, and you can use the buttons provided to get the fancy symbols that aren't on your keyboard.

Names in a conjecture have to obey certain rules:

- *variables* must start with x, y, z, i, j or k and can continue with any letter or digit;

- *formula* or *predicate* names must start with A, B, C, D, E, F, G, H, P, R, S or T and can continue with any letter or digit.

If you make a mistake, like leaving out a dot or a bracket in a crucial spot, or typing a name it can't recognise, Jape will complain. If you make more than one mistake, Jape only spots the first (reading left to right). Sometimes the error messages will be hard to understand: I'm sorry for that, but it's really hard to improve them.

Chapter 9

Troubleshooting

9.1 Problems getting started

On MacOS / OS X your security settings may stop you running Jape, saying it comes from a unknown developer. There's a way round this: ctrl-click Jape, choose Open from the little menu that appears, and say Open on the alert window that comes up. After that it can be double-clicked as normal.

On Windows and Linux Bernard distributes an 'installation jar' – a bundle of stuff that distributes itself round your filestore. From his website:

> It is never appropriate for you to unpack the .jar file, but it turns out that certain file-archiving software on these operating systems tell the operating system that .jar files are archives, and that opening means unpacking. On a Linux machine the way round this is to java -jar `Install...jape.jar` from a terminal window. On a Windows machine right-click on the jar file, and select the `Open with ... java ...` menu entry.

The Jape app can be put anywhere you like. The examples can be anywhere you like.

If you installed Jape and it won't start properly, throw it all away and install it again. This time follow the instructions *precisely*. Then it will work.

9.2 What if a proof step goes wrong?

When you try to apply a rule one of two things can happen. Either the rule applies, and the step goes through, or it doesn't, and Jape shows you an error message.

Even though a rule does apply and the proof step does go through, it may not turn out to be the right thing to do. Sometimes an apparently successful step can lead to a dead end. Sometimes a step works, but not in the way that you expected — perhaps lots of unknowns suddenly appear in the proof, or there are lots of extra lines that you didn't expect, or lots of lines suddenly disappear.

Whenever something happens that isn't what you expected, the first stage of a cure is to use the Undo command from the Edit menu. Undo takes you back one step in the proof, two Undos take you back two steps, and so on. Using several Undos can move you back from a dead end to an earlier position from which you can move forward in a different direction.

You can even recover from Undo! The Redo command (also in the Edit menu) reverses the effect of Undo, two Redos reverse two Undos, and so on. So if you decide, after Undoing, that you really did want to make

the step after all, Redo will make it again. (If you Undo and make a new proof step, then the one you Undid is gone for ever, like it or not.)

The Undo command allows you to explore if you don't know what rule or theorem to apply in a proof: you can experiment with different rules and theorems from the menus until you find one that works. That can be a bad thing, if you just try things at random until you find one that happens to work, and don't reflect on what you are doing. It's also the slowest way of finding proofs. But sometimes we all need to search and experiment (aka "thrash about"), and then Undo and Redo are invaluable. I hope that when you do search and find a surprising avenue that happens to work,. you will pause to ask yourself *why* it works. Jape is designed to support 'reflective exploration' — it helps with the exploring part, and you learn by reflecting on the results.