Using I2L Jape

(*Fourth Unix/X edition, Jape version 3.5*)

Richard Bornat, Department of Computer Science, QMW

October 1999

Contents

Preface to Unix edition

This document was prepared in the first instance for the MacOS implementation of Jape. I should like to produce a version which explained exactly how to use Jape on Unix/Linux/Solaris with X, but X is so variable – from choice of window manager to configuration of options – that it would be impossible to do so. Instead I have produced a simplified version of the MacOS document, without any mention of how windows are manipulated and managed.

The illustrations are from the MacOS version as well: I still haven't found out how, in the X world, to copy a proof from a Jape session into a word-processed document. But I don't think it matters, because apart from some minor differences in the way that the Konstanz font is displayed on the Mac and under X, they should be recognisable to any user of Jape.

Using I2L Jape (summary)

You ***select*** a formula by left-clicking it.

You ***text-select*** part of a formula by middle-button press-and-drag.

A ***hypothesis formula*** is a premise, an assumption, or the result of a forward step. When you select a hypothesis, you get a box open at the bottom, to show you that you can move downwards (forward) from that formula.

Conclusions are all the non-hypothesis formulæ. When you select a conclusion, you get a box open at the top, to show you that you can move upwards (backwards) from that formula.

Unproved conclusions are formulæ written just under a line of dots: they don't have justifications written next to them. ***Proved conclusions*** are conclusions which do have justifications written next to them.

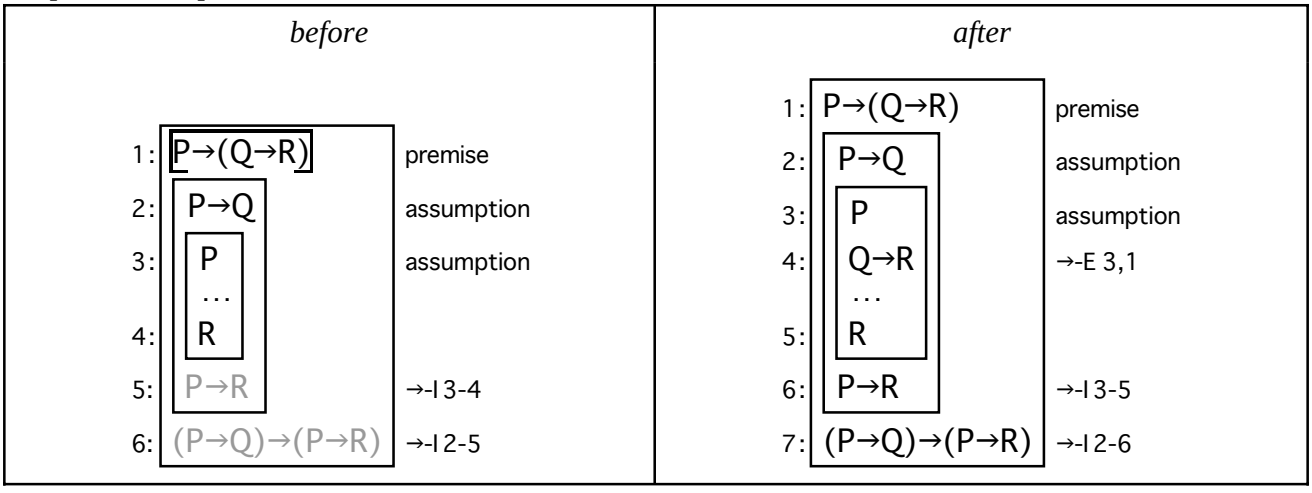
The ***current conclusion*** is either the only unproved conclusion (if there's only one) or it's the selected unproved conclusion (if there's more than one). When Jape makes a forward step it writes the result just before the line of dots above the current conclusion. When it makes a backward step, it writes the result between the line of dots and the current conclusion.

Some rules must be given an ***argument*** by text-selection; some rules may be given an argument. Argument information is additional to hypothesis and/or formula selection.

1. Making a forward step

All the forward steps require you to select a hypothesis formula. Choosing a step from the Forward menu then makes the step.

A forward step always writes its result just *before* the line of dots above an unproved conclusion, marking a gap in the proof. The unproved conclusion below the dots can then make use of the result of the forward step. For example, this is the effect of the →-E rule:



Going wrong (1) no selected hypothesis

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

Going wrong (2) wrong shape of hypothesis

Each forward step, except for ∨-I, ∧-I and ∃-I, applies to a particular shape of hypothesis formula. If you select the wrong shape, you can't make the step.

Going wrong (3) too many unproved conclusions

When you select a hypothesis, Jape greys-out all conclusions except the ones that can make use of the hypothesis. If there is more than one non-grey conclusion showing, then Jape doesn't know where to write the result of the step. In the left-hand proof below, for example, there are unproved conclusions on lines 3 and 4, so Jape can't make a forward step from line 2. In the right-hand proof, selecting line 4 resolves the ambiguity.

bad

1:	∃x.(P(x)∧Q(x))	premise
2:	var c, P(c)∧Q(c)	assumptions
	…	
3:	∃x.P(x)	
	…	
4:	∃x.Q(x)	
5:	∃x.P(x)∧∃x.Q(x)	∧-I 3,4
6:	∃x.P(x)∧∃x.Q(x)	∃-E 1,2-5

good

1:	∃x.(P(x)∧Q(x))	premise
2:	var c, P(c)∧Q(c)	assumptions
	. . .	
3:	∃x.P(x)	
	. . .	
4:	∃x.Q(x)	
5:	∃x.P(x)∧∃x.Q(x)	∧-I 3,4
6:	∃x.P(x)∧∃x.Q(x)	∃-E 1,2-5

Going wrong (4) no unproved conclusion

When you select a hypothesis, Jape greys-out all conclusions except the ones that can make use of the hypothesis. If there are no available unproved conclusions, then the forward step can't take place. In the following proof, for example, there are no unproved conclusions which can make use of line 2 (there is an unproved conclusion on line 8, but it is in another box):

1:	(P∧Q)∨(P∧R)	premise
2:	P∧Q	assumption
3:	P	∧-E 2
4:	Q	∧-E 2
5:	Q∨R	∨-I 4
6:	P∧(Q∨R)	∧-I 3,5
7:	P∧R	assumption
	. . .	
8:	P∧(Q∨R)	
9:	P∧(Q∨R)	∨-E 1,2-6,7-8

2. Making a backward step

Backward steps work on the current conclusion and give it a justification. They may prove it completely, or they may throw up more conclusions to prove. For example, this is the effect of the ∧-I rule:

before

1: ¬(¬P∨¬Q) premise
…
2: P∧Q

after

1: ¬(¬P∨¬Q) premise
…
2: P
…
3: Q
4: P∧Q ∧-I 2,3

Going wrong (1) wrong shape of conclusion

Each backward step, except for ¬-E, applies to a particular shape of conclusion formula. If you select the wrong shape, you can't make the step.

Going wrong (2) too many unproved conclusions.

If there is more than one unproved conclusion, Jape doesn't know which is the current conclusion, so it doesn't know where to make a step. In the left-hand proof below, for example, Jape wouldn't know where to apply the ∨-I rule. In the right-hand proof, selecting an unproved conclusion resolves the ambiguity.

bad

1: P∨(Q∧R) premise
…
2: P∨Q
…
3: P∨R
4: (P∨Q)∧(P∨R) ∧-I 2,3

good

1: P∨(Q∧R) premise
…
2: P∨Q
…
3: P∨R
4: (P∨Q)∧(P∨R) ∧-I 2,3

Going wrong (2) selecting a proved conclusion

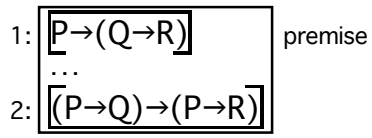
If you select a proved conclusion Jape can't make a proof step, because that line has already been used up and included in the proof[1]. For example, Jape can't make any kind of step in this situation:

1: P∨(Q∧R) premise
…
2: P∨Q
…
3: P∨R
4: (P∨Q)∧(P∨R) ∧-I 2,3

[1] You sometimes select a proved conclusion because you want to Backtrack or Prune the proof, which is why Jape lets you make the selection.

Going wrong (3) selecting a hypothesis

Backward steps don't need and can't make use of a selected hypothesis formula. For example, Jape will be confused if you ask it to make a →-I step in the following situation:



3. Giving an argument to a step

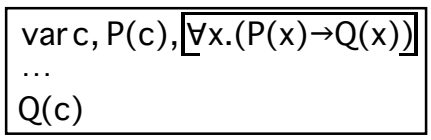
Some steps – ∀-E going forward, ∃-I going backward, require an argument. Some steps – ∨-I, ∧-I going forward, ¬-I going backward – can accept an argument.

To use ∃-I going forward, see appendix H.

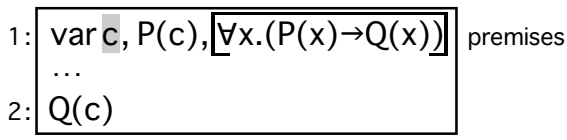
Going wrong (1) no selected argument

The ∀-E rule going forward, like the ∃-I going backward, requires you to text-select an in-scope variable before you can make the step. For example, in the left-hand proof below Jape doesn't know which variable to use in a ∀-E step. In the right-hand proof a text-selection tells it to use variable *c*.

bad



good

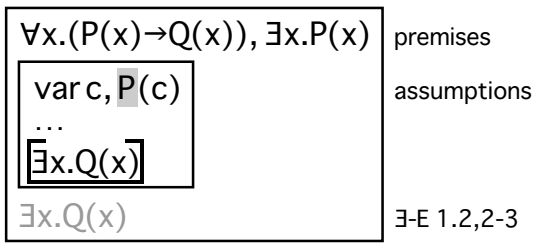


The ∃-I rule, going forward, requires you to select the variable instances you want to quantify *within* the formula you are going to quantify. See appendix H for more details.

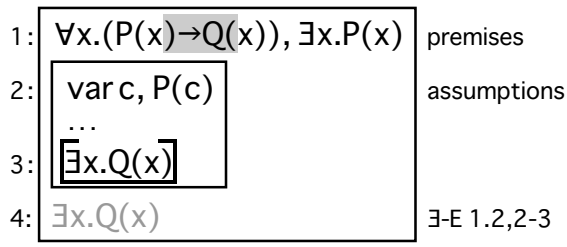
Going wrong (2) bad argument

If you select text which isn't the kind of argument which a rule will accept, or which doesn't make sense as a formula, Jape won't make the step. For example, it won't make a ∃-I step in either of the situations below:

wrong kind of argument

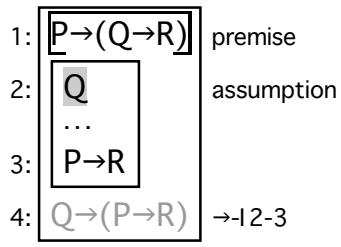


not an argument at all



Going wrong (3) unnecessary argument selection

Most steps don't want and can't use an argument (text-selection) as well as a hypothesis formula. For example, Jape would be confused if you asked it to make a →-E step in the following situation:



4. Some rules of thumb

Rules of thumb[1] – general principles that don't always work – can be very useful as a guide. You never know your luck – try these.

1. Use rules that introduce structure into the proof *as early as possible*. Those rules are:
   - →-I backwards (for the assumption);
   - ∨-E forwards (for the assumptions);
   - ¬-I backwards (for the assumption);
   - ∀-I backwards (for the variable);
   - ∃-E forwards (for the variable and the assumption).
2. Use ∀-E and ∃-I only when there are variables around for them to use.
3. ∧-I, ∨-I and ∃-I are much easier to use backwards than forwards, so *use them backwards*.
4. ∨-I backwards throws away half the formula, so use it very carefully, late on in a proof.
4. As a last resort, try proof by contradiction: ¬-E, followed by ¬-I, followed by ∧-I and then perhaps by hyp, all backwards.
5. If it 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 panel are possible – I've done them all[2], except for the ones at the end labelled NOT, which I guarantee are unprovable in Jape and on paper as well.

---

[1] Some people believe that 'rule of thumb' refers to an ancient English law which governed the thickness of sticks which could be used by a husband in chastising a wife, and they say that we shouldn't use the phrase for that reason. That belief has some historical justification (see below), but it isn't where the phrase 'rule of thumb' comes from.
The phrase derives from the fact that an inch is approximately (originally by legal definition was exactly) the width of a human adult male thumb. So you can use your *thumb* as an *rule* (nowadays we would say *ruler*) if you are prepared to put up with a bit of inaccuracy: hence, by punning the meanings of 'rule', we get 'rule of thumb' for any procedure which sometimes works and sometimes doesn't.
Belief in the existence of a law which allows husbands to beat their wives with sticks of less than a particular thickness is very ancient. E.P. Thompson, in "Customs in Common", has a reference from the time of the enclosures in England. It was indeed described as 'the rule of thumb'. The name was, I suppose, a folk double pun.

[2] Some of the negation proofs are really quite tricky. It isn't Jape's fault: it's the fault of the logic. Proofs involving negation are rarely easy, but it's possible to make them a lot simpler by using slightly different negation rules, without changing the meaning of the logic. So don't blame Jape, blame the logic I had to encode.

Using I2L Jape

1. What is I2L Jape?

When I first came across Natural Deduction I found its rules seductively persuasive, but I found it hard to construct a proof. It wasn't until I was introduced to a mechanical proof assistant that I began to be able to make proofs for myself, and not until I got used to the box presentation of proofs in Jape that I began to be totally relaxed about 'natural' deduction.

So I've been there, I know how it feels, and I would like to save you the confusion that I suffered from. *Jape* is an acronym for 'just another proof editor'. Jape handles the details of the application of rules in a formal proof, leaving you to think about which rules to apply, to what, and in which order. I hope that practice with I2L Jape will help you to understand the rules of Natural Deduction and to become so familiar with them that you can then make proofs on your own account without I2L Jape to help you.

When you are using I2L Jape I hope that you will quickly come to realise that Natural Deduction is mainly about the *simplification* of formulas by the application of rules. To find a Natural Deduction proof using I2L Jape you use rules to consume operators in the premises, assumptions and the conclusions of a proof in progress. You have to choose the assumption, premise or conclusion to work on, you have to choose the rule to apply, but I2L Jape does all the book-keeping, making sure that you have chosen a rule that applies to the things you have selected and showing you a nice tidy version of the proof as you build it.

Jape is a general proof editor, capable of handling many different logics. If you already know how to run Jape, you know how to run I2L Jape and, to a certain extent, vice-versa – but I expect that most readers of this document will be starting from scratch.

Learning by reflection

Jape is a calculator, really: it lets you apply the rules of logic correctly, but stops you applying them wrongly. It doesn't give you much help, beyond error messages when you try to do something wrong. It will let you go up blind alleys if you ask it to, because it doesn't know the difference between a blind alley and a highway.

Despite that, Jape is an educational tool. You learn from Jape by *reflecting* on what it has helped you to do. When you finish a proof, or when you make a step which surprises you, you will learn if you read what is on the screen and think about what it means.

I hope that you will find I2L Jape easy to use, and I hope that it will stimulate you to think about the structures of proofs, about why the rules are the way they are, and about how Natural Deduction works.

Forward or backward proof?

I2L Jape was developed in order to help you learn by experience that most Natural Deduction proofs are so natural as to be almost automatic, with a structure dictated by the structure of the premises, assumptions and conclusion. In order to learn that lesson it is sometimes necessary to make use of backward reasoning.

It would be as wrong to say that backward proof is better than forward as it would be to say the opposite. One of the things that I hope you will learn from I2L Jape is that you can't choose in advance whether to make a proof by working forwards from the premises or backwards from the conclusions. The proof state

– the collection of premises, assumptions and conclusions, plus the theorems and rules you can use – should dictate what you must do, not some orthodoxy.

Forward proof works well when you have a step which you can make entirely from the premises and assumptions which you already have available. It is attractive, to the novice, because it makes it seem that the proof is being developed line-by-line in the same order as the completed proof can be read. Backward proof works best when the structure of the proof is determined by the shape of the conclusion.

You will find when making a proof with I2L Jape that sometimes the first move you make must be a backward step. This isn't 'wrong' or in any sense cheating: it is the way that proofs actually get developed. In reality it is the line-by-line 'forward' reading of a completed proof that is deceitful, because it hides the history of proof development, and it misleads the novice by pretending that a proof is a sequence of steps, rather than a structure of deductions.

There is no best way nor even a 'right way' to develop a proof; nor is there a 'wrong way' – although you can decide, once a proof is completed, that it might have been done better, and then Jape will let you undo your work and try again. Forward reasoning isn't better, or righter – or wronger – than backward reasoning: each has its place, and in order to make a *natural* Natural Deduction proof it is usually necessary to employ both styles.

Making and discharging assumptions

Forward proof stops working well when you run out of things to do with the premises. When this happens, provers blinded by the notion of forward progress will want to invent an assumption to relieve their difficulties. That's a mistake: a proof is really a *structure of deductions*, not a sequence of lines, and assumption boxes show that structure. Every assumption that is introduced must also be *discharged* by the use of a rule. Jape guarantees correct use of assumptions by combining introduction and discharge into a single step, which is how it's really done in logic. So 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.

Premise, assumption, hypothesis, conclusion: what's in a word?

Most proofs start with a premise formula and a conclusion formula. Backward steps generate new conclusions and/or assumptions; forward steps generate new formulæ to work forward from and/or new assumptions.

When you work forward you can do so from three kinds of formula: premises, assumptions, and the results of previous forward steps. In this manual, and in the I2L Jape error messages, I've used the word *hypothesis* to describe that kind of formula.

When you work backward you do so from unproved conclusions: formulæ written just below a line of dots

Differences from the lecture notes

Jape isn't really a Natural Deduction engine, and its internal mechanism is based on inference trees, not the sort of line-and-box proofs that I2L Jape displays. Most of the time I2L Jape does a really good job of pretending that it is a Natural Deduction engine and that it really does work with lines and boxes, but sometimes the internal workings show through. You may notice the following:

- Jape doesn't always allow you to appeal to an earlier line of the proof, even when the box structure would let you;
- Jape's implementation of 'scope boxes' associated with the ∃-E and ∀-I rules uses pseudo-assumptions 'var *x*', rather than annotated boxes;
- Jape's syntax for quantification is ∀*variable.formula* and ∃*variable.formula* – in each case the full stop (dot, period) is essential, and *formula* has to be bracketed unless it's just a predicate application.

So far as I know, those are the only significant differences between Jape's treatment and the course notes, and only the first one should cause you any significant difficulty. I think I know how to fix it, but it won't be done in this academic year.

Any comments?

If you have any comments about I2L Jape that you would like me or anybody else to hear, there's a dcs.jape newsgroup at QMW, or you can email me (richard@dcs.qmw.ac.uk). I'm especially interested to hear from people who find any difficulty, no matter how slight, in using the program to do their work: I'll try to improve Jape if I get to hear of your problems. So if you see something wrong, tell me, put a message in the newsgroup or email me. If you think Jape is wonderful, please let me know that!

2. Getting started

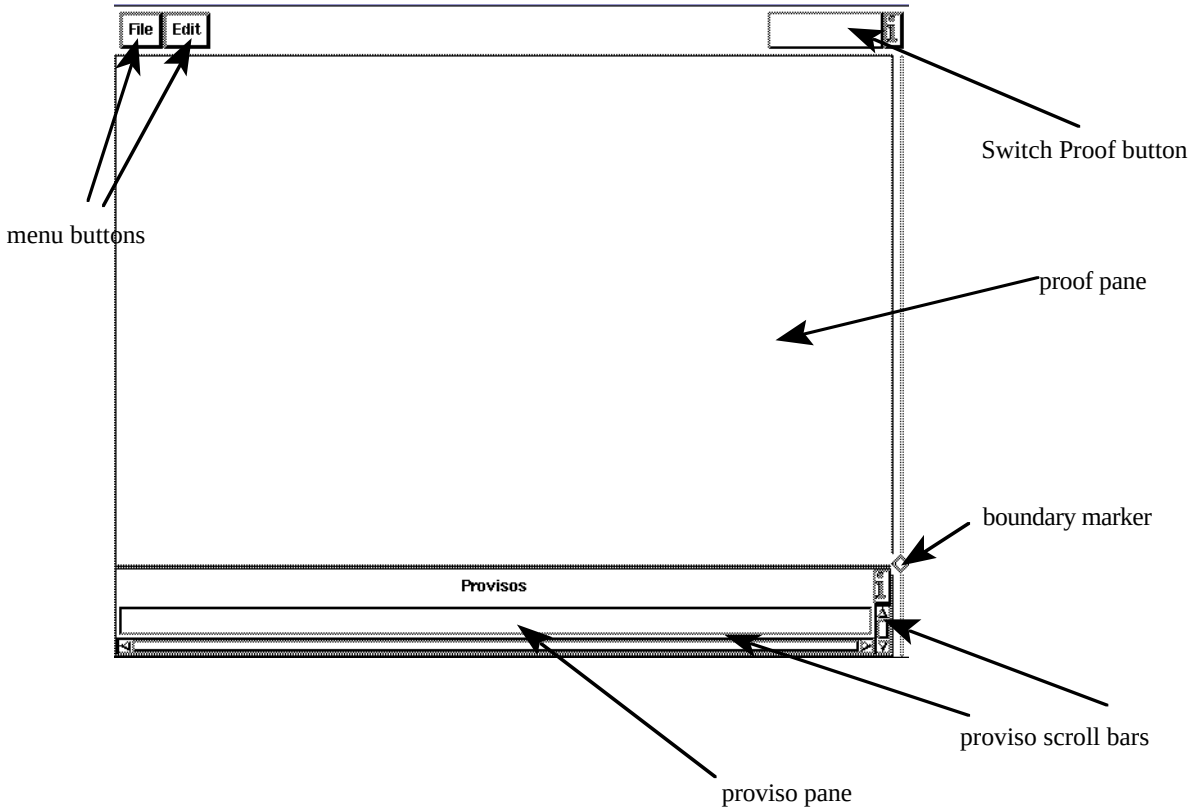
Type the following to an terminal window:

```
/import/jape/i2ljape&
```

Jape then starts and creates two windows:

- Jape conjectures;
- Jape session for theory “I2L”.

The session window looks something like this:



Most of the work goes on in the proof pane. The proviso pane shows provisos (sometimes called side-conditions on a proof) which can arise as a result of using ∀-I and ∃-E steps: most of the time it’s blank. You can alter the boundary between the panes by dragging the boundary marker up and down. You scroll around in the proviso pane with the proviso scroll bars; you can drag a proof in the proof pane by middle-button pressing on a blank part of the pane and dragging the proof to the position you want.

If you have some saved proofs, now is the time to load them (use ‘Load a file of proofs’ from the File menu).

If you are running Jape for the first time, read on.

Doing things in the session window

You make your proofs in the proof pane. You can move a proof around in its pane by moving the cursor to a blank part of the background, holding down the middle button of the mouse and dragging the proof around. If you lose the proof by dragging too fast or too far, press the Home key on the keyboard, and it should pop up somewhere visible.

You can have more than one proof on the go at the same time. Only one of them will be *active*, and shown in the session window, but each of them will be listed in the menu under Switch Proof (top right of the session window). The active one has a highlighted check-box in that menu. By clicking on the entry

of an inactive proof in the Switch Proof menu, you make it active and it takes over the session window. You can abandon the active proof at any time by clicking Abandon Proof in the Switch Proof menu.

Selecting and choosing

X Japе has various different ways of ‘gesturing’ at a proof with the mouse, introduced in the discussion below (see appendix A for a complete list). When this manual uses one of the words ‘choose’, ‘select’ or ‘click’, you are supposed to click something with the left mouse button unless the context makes it clear that something else is intended.

Buttons in the Conjectures panel

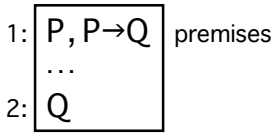
The Conjectures panel has various parts (which I’d love to illustrate with a screen snapshot, but I don’t know how). Most important is the list of conjectures – a collection of proof problems which we think you might like to experiment with. The list is scrollable, using the scroll bar to its right. If you want to see more, or less, of the list you can alter the size of the window using whatever gesture or gestures your window manager provides.

The Conjectures window is described in this document as a *panel*, because it carries buttons and a list of things to choose from. The buttons in the panel each have their particular use, discussed below. For the moment we shall concentrate on the Prove button, which is used to start a proof. If you select an entry in the panel (left-click on it) and then press the Prove button, a display of that conjecture, ready to prove, will appear in the proof pane.

3. Two proofs by forward reasoning using →- elimination

If you haven't already started the proof of the first conjecture in the Conjectures panel, do it now: select the first entry in the list of conjectures and press the Prove button. Jape will display the problem to you in the proof pane of the session window. (If you already have a proof of the first conjecture on the go, Jape will tell you and you can then choose whether to start another. If you have already proved and registered the proof of that conjecture, Jape will tell you and you can decide whether you want to attempt a new proof.)

If I showed a picture of a session window every time I wanted to include an example, this manual would be far too large. From now on I shall save space by showing only the contents of the proof pane like this:



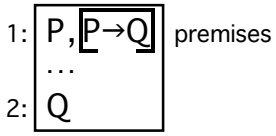
You are asked to prove from premises P and P→Q the conclusion Q. The premises are listed on a single line, separated by a comma. Jape uses one line rather than two so as to save screen space, which is a precious resource, as you will see if ever the proof becomes so long that it exceeds the height of the proof pane[1]. The line of dots between lines 1 and 2 indicates that there is something to prove – a 'gap' in the proof so far. The lines of the proof are already numbered, but those numbers can be expected to change as extra lines are added to the proof.

Making a proof step

You should already have an idea how to solve this problem: you use the →-E rule to derive Q from premises P and P→Q (if you don't know, you are about to learn, and if you've forgotten about the →E rule, see the appendix on the → rules). In Jape's language you simplify the P→Q premise using the →-E rule: you don't have to tell it about the P premise because it will find it automatically.

The rules of Natural Deduction are in two menus: Backward for backward steps, Forward for forward steps. (As well as the rules described in lectures, there's an entry for 'hyp' at the bottom of each menu, whose use is discussed later.)

To make the proof step you must 'select' the P→Q premise and then 'apply' the →-E rule from the Forward menu. You select the premise by clicking once on it with the left mouse button (move the mouse pointer over the premise, press the left mouse button and immediately release). A little box, open at the bottom to show you that you can move forward from this formula, will appear round the premise you have selected:

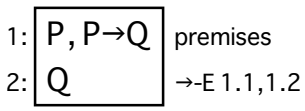


If you have selected the wrong premise, don't panic: left-click on the one you really wanted to choose and Jape will obey. If you clicked on the conclusion by accident, just left-click on some blank part of the proof pane and Jape will cancel all your selections; then you can start again.

---

[1] You can reduce the size of a proof somewhat by setting the font size, using the Set Font sizes command from the Files menu. X fonts are always a bit unreadable at small sizes, so it's a good idea to experiment carefully.

Now pull down the Forward menu and click →-E. Jape makes the step, and shows a justification for it. In this case the result of the step is the conclusion we are trying to prove, so the proof is completed and the line of dots disappears:



I hope you did get this effect: if not, use Undo (in the Edit menu, or use the keyboard Undo key) and try again.

Notice the way that I2L Jape numbers justifications which refer to premises or assumptions: the step which leads to line 2 depends on the first formula on line 1 (1.1 means P) and the second formula on line 1 (1.2 means P→Q).

That's not the only way to make this proof, but it will do for now.

Registering your proof with Jape

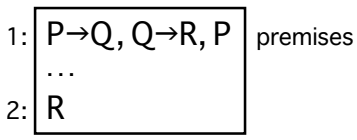
Now that you have made a proof of a conjecture you can save it : pull down the Edit menu and select Done. The proof disappears from the session window, and Jape records the fact that the conjecture is now proved, marking its entry in the conjectures panel with 'THM' in the margin.

Proof of the conjecture has made it a theorem. You can use theorems in your proofs as if they were additional Natural Deduction rules by using the Conjectures panel's Apply button, and you can review their proofs using Show Proof. But more of that later.

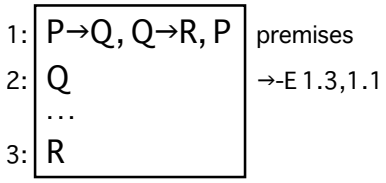
4. Three different ways to make the same proof

There are often several equally attractive ways to make the same proof. It's worth trying different ways to see how Jape can help you make a proof in the way you like best.

Select the second entry in the Conjectures panel and click Prove. You should see a proof pane containing

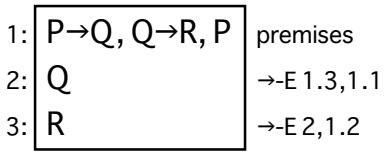


I shall show you three different ways to make the same proof (and there are still other ways in which it can be done). The first starts by forward proof from P and P→Q, which are both premises, to make the intermediate conclusion Q. Select (left-click) P→Q and apply →-E from the Forward menu. The proof becomes



Notice how the line numbering has changed: the conclusion is now line 3, and line 2 is the result of the →-E step. Notice also how the justification of line 2 identifies the premises it has used: P (1.3) and P→Q (1.1).

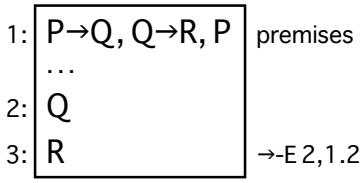
Now on line 2 we have Q and on line 1 we have Q→R: clearly we can derive R using the →-E rule. Select Q→R , apply →-E, and the proof is completed:



That's one way of doing it. What are the others? To find out, apply Undo (from the Edit menu) twice, to get back to the starting point..

Why the →-E step only needs a single formula

Back at the start of the proof, select Q→R and apply →-E. You will see



This shows the difference between Jape's notion of a proof step and what you might expect. The →-E rule works on a proof of A→B and a proof of A. By writing Q→R in the premises, you are stating an assumption that there is a proof of that formula, so you have half of what the proof step requires. If there isn't a proof of Q around – and there isn't at the moment – you can still make the step, but you are required to *make* a proof of Q to complete it – that's what line 2 is saying. The rule takes a proof of A→B and a proof of A and makes a proof of B: in this case proofs of Q→R and Q make a proof of R. But R was the conclusion you were trying to prove, so Jape can tie up your proof step with the conclusion – and that's what line 3 is saying.

This single example shows you just why Jape doesn't make you point to two formulæ to use the →-E rule[1]: if it did, you would be forced to do your proof in a fixed order, and sometimes that isn't helpful. If you want to break down Q→R before you have a proof of Q, that's no problem: but you'll have to make the proof of Q eventually.

Now we've got an premise P→Q a premise P, and we have to make a proof of Q, so we can proceed as in the first proof. Select P→Q and apply →-E, and the proof is once more completed:

1:	P→Q, Q→R, P	premises
2:	Q	→-E 1.3,1.1
3:	R	→-E 2,1.2

The same proof, a different way of deriving it with I2L Jape. That's one of Jape's strengths: it doesn't force you to apply rules in just one order. At least not always!

Using a theorem

Look again at the start of the proof (undo, undo will take you there):

1:	P→Q, Q→R, P	assumptions
	…	
2:	R	

You have already proved a theorem P→Q, P ⊢ Q. That theorem applies in this situation: we do have an premise P and an premise P→Q, and, just as when you apply a rule, it doesn't matter what order they appear in, how many times they appear, or what other assumptions we have.

We want to apply the theorem to the premises P and P→Q. Select either of them (it doesn't matter which); then select the theorem (it's the first entry in the Conjectures panel) and press Apply[2]. The proof pane changes to show the effect of applying the theorem:

1:	P→Q, Q→R, P	premises
2:	Q	Theorem P, P→Q ⊢ Q 1.3,1.1
	…	
3:	R	

Now the proof can be completed as before, using the →-E rule, or it can be completed using the same theorem again. We now have Q (line 2) and Q→R (line 1, premise 2), and we want to prove R. The theorem covers that possibility. Apply it once more[3] and the proof is completed once again:

1:	P→Q, Q→R, P	premises
2:	Q	Theorem P, P→Q ⊢ Q 1.3,1.1
3:	R	Theorem P, P→Q ⊢ Q 2,1.2

Choose Done (Edit menu) to register the proof with Japе.

---

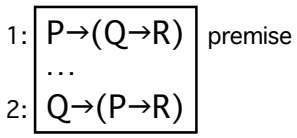
1 It doesn't explain why Jape doesn't *let* you point to two formulæ. That's a harder question to answer, and this isn't the place. I'm sorry, but it just doesn't.

2 If you press Prove by mistake, you will be offered the opportunity to start a new proof of P→Q, P ⊢ Q: just click No in the dialogue box and then press Apply. If you press Show by mistake the proof you have already made of P→Q, P ⊢ Q will be displayed in the session window: just choose Abandon Proof from the Switch Proof menu, and then press Apply.

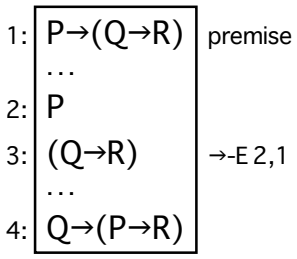
3 You don't need to select a hypothesis, because Jape can work out which hypotheses are to be used from the theorem itself and the conclusion Q.

5. A proof which *needs* backward reasoning

Choose P→(Q→R) ⊢ Q→(P→R) from the Conjectures panel (it's the fifth entry). You will see a proof pane containing



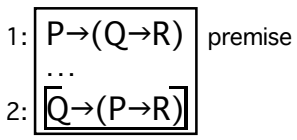
This time you can't use forward reasoning to simplify the premise. If you don't see why not, try it:



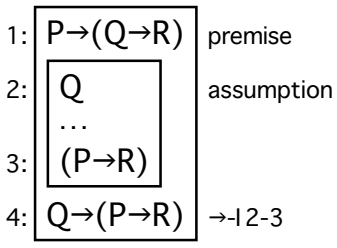
Line 2, with the three dots above it, is an invitation to prove P (read 'why the →-E step only needs a single formula' above, to explain what's going on). The proof is stuck: you can't make a proof of an arbitrary formula P out of thin air!. Undo back to the beginning.

It's at this point that a rigid-thinking forward reasoner would begin to bleat about the need to invent an assumption. Jape can help, but it does so by using a rule, it does it *safely*, and you don't even have to invent the assumption. And (rigid-thinkers have to bend at this point) it's a bit of *backward* reasoning from the conclusion.

In the Backward menu the →I rule is labelled '(makes assumption)'. Observe that the conclusion fits the rule: anything of the form A→B might be introduced by the →I rule. So select the conclusion:



(this time you get a selection open at the top, to show you that you can work backward from this formula), apply the →I rule from the Backward menu, and you should see

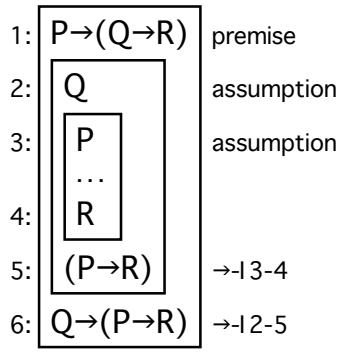


What I2L Jape is done is to proceed *as if* the last line was produced by using the →I rule: that would have to be because there is a proof of P→R from Q, and I2L Jape has drawn the corresponding box structure with a gap which shows that you still have to make the subsidiary proof. What's nice about this step is that the assumption, which is introduced on line 2, is discharged on line 4 – forward reasoners can't do that so easily.

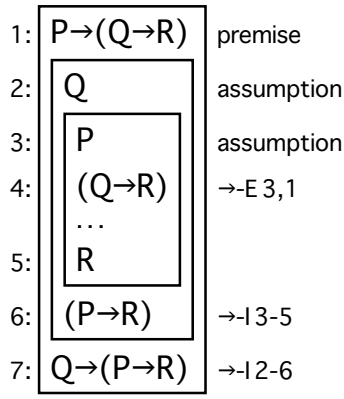
This kind of step shows you that a proof isn't a sequence of lines, it's a *structure of deductions*. The deduction of P→R from Q, which will fill in the box, is a complete part of the proof. It has to start with Q and end with P→R, or it wouldn't fit. The association between lines 2, 3 and 4 is really important, and it's the most important part of the proof so far.

The other thing about this step is the assumption that's been introduced. A forward-prover would probably try to introduce the assumption P, but that *wouldn't work*, because it couldn't be discharged[1]. The assumption Q is the only one that works, and it works because of the shape of the conclusion. In order to complete the proof you have to deal with that fact: the assumption really comes from the conclusion, inventing an assumption isn't helpful.

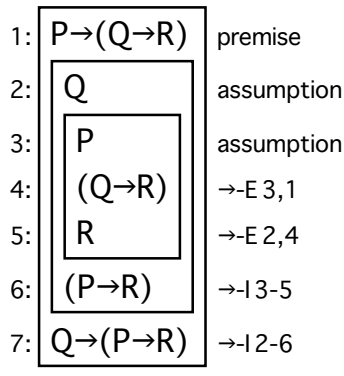
Now just because we know Q we don't necessarily know P, so we still can't use →E on line 1. But we can work backwards from line 3. If you apply the →I rule to that line, you will see:



We have to prove R from the three assumptions P→(Q→R), Q and P: that's pretty easy using forward reasoning and the →-E rule. Select the premise on line 1 and apply →-E. You will see



Now choose line 4 and apply →-E again:



The proof is complete.

This example illustrates the general principle that *backward* reasoning with an introduction (-I) rule essentially eliminates connectives from conclusions, just as the earlier examples showed that *forward* reasoning with an elimination (-E) rule eliminates connectives from premises and assumptions. In fact the

---

[1] You could have a completely irrelevant part of the proof which assumed P, deduced something or other, discharged the assumption, and then came back to the original problem. But what would be the point of that?

forward/backward process, and the way that it applies to logical connectives, is so mechanical that I can set I2L Jape up so that it chooses the rule itself when you double-click on a formula. Because we want you to learn about Natural Deduction and not just the use of the mouse, I've been heartless and I2L Jape is set up so that you have to choose the rules for yourself.

6. Some rules go both ways

So far I've shown you how to use -E rules forwards from hypotheses, and -I rules backwards from unproved conclusions.

Some -I rules simply don't make sense going forwards, because like →-I they introduce assumption or scope boxes into the proof: they are →-I, ¬-I and ∀-I. The rest make some sense going forward, but they are much easier to use backwards. I've put the ∨-I, ∧-I and ∃-I rules into the Forward menu, but you really ought to be using them backwards almost all of the time.

When you use the ∨-I rule in the Backward menu, or the ∧-E rule in the Forward menu, you will see that there are two entries in each case: one labelled 'preserving left' and one labelled 'preserving right'. That's because these rules break a formula in half and only keep one of the halves, so you have to tell Jape what you want it to do. In the Forward menu ∨-I and ∧-I each have two entries, distinguished by 'invents right' and 'invents left'. What happens when you use those menu entries is discussed in the next section.

One -E rule has to be used backward some of the time: for proof by contradiction using the rules in the lecture course, you have to start with a backward step of ¬-E (see the appendices for more on proof by contradiction). So that's in the Backward menu as well as the Forward menu.

Surprisingly, *all* the other -E rules make sense working backwards, but following research on last year's undergraduate users of Jape, I've decided not to allow it because it seems to cause too much confusion. So the other -E rules are in the Forward menu only (and I hope I've deleted the bits of this manual that used to describe how to use them backwards!).

7. Unknowns, unification and the *hyp* rule

Sometimes you will see *unknowns* in your proof. They look like formula names, but they start with an underscore: _A, _B, that sort of thing. An unknown is a placeholder in a proof – a formula whose identity hasn't been decided. When you can see what the formula in that place should be, there are at least two different ways to tell Jape what to do.

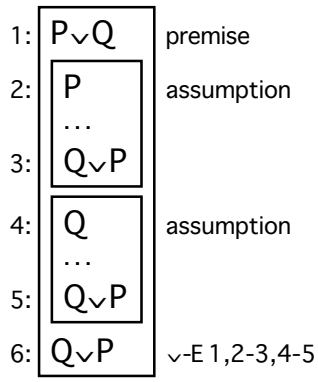
There are various different ways in which unknowns can turn up in your proof. One is to use rules which work better backward, but which you are allowed to use forward – ∨-I, ∧-I or ∃-I. Another is to use the ¬-I rule backwards.

If you make a forward step with ∨-I or ∧-I you can give an argument by text-selection to stop the unknown appearing. If you make a forward step with ∃-I you have to make a text-selection and you still get an unknown. When you use ¬-I you almost always introduce unknowns, because you are genuinely searching for a contradiction and you don't know what it is yet, but you can give an argument if you like.

> I can't stress strongly enough that it is hard work to use the ∨-I, ∧-I and ∃-I rules forward, easy to use them backward. ∃-I is particularly difficult to manage, going forward, but dead easy going backward.
>
> But if you must, you must. Details of how to do it are in the appendices; I only give one illustration here.

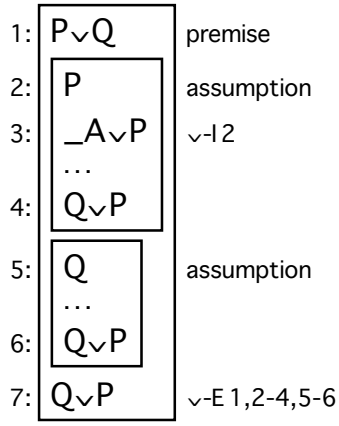
How to get an unknown into your proof

The simplest way to get an unknown is with the ∨-I rule used forward. Suppose you have the following:



You want to get to Q∨P (on line 3) from P on line 2. The easiest thing is to go backward from line 3: select line 3, choose "∨-I (preserving right)" from the Backward menu, and the job's done.

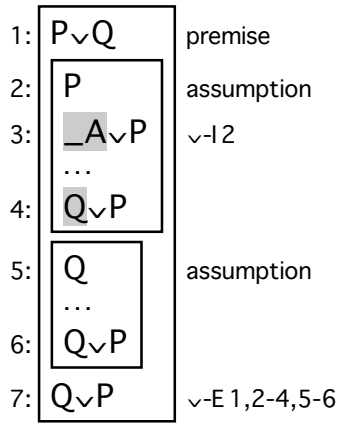
If you must go forward, select P on line 2 and choose "∨-I (invents left)" from the Forward menu. This is what you see:



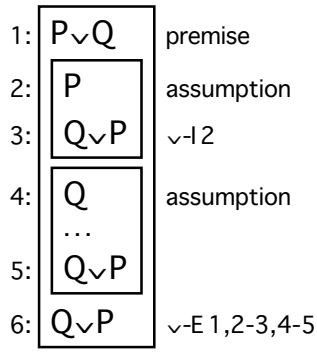
_A is an unknown, a placeholder that Jape has had to put in the proof because it doesn't guess what ought to be there. It's not too hard to see that the unknown _A should be replaced by Q, and then the box from line 2 to line 4 can be completed.

How to eliminate an unknown: (1) by unification

Since we know that _A should be replaced by Q, we can tell Jape to *unify* those formulæ. Unify means 'make them the same, by replacing unknowns with actual formulæ'. What you do is to text-select two formulæ at the same time[1]:



and then use the Unify command from the Edit menu to make them the same. When you do so, Jape sees that lines 3 and 4 have become identical, and gets rid of line 4 entirely:



You will be delighted, I'm sure, to know that there are easier ways to get the same result (working backwards from line 3 is the easiest, but there are still two others).

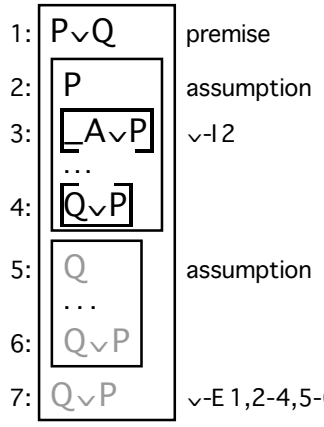
How to eliminate an unknown: (2) with a hyp step

Instead of unifying by text selection, we can often tell Jape to unify a hypothesis formula with a conclusion formula, using the hyp step which is on both Forward and Backward menus.

---

1 It's a bit tricky to text-select the _A: make sure you get the underscore as well as the letter.

You select line 2 (a hypothesis) and line 3 (a conclusion):

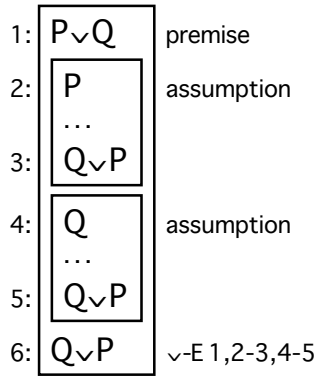


and then you use hyp from the Forward or the Backward menu. The effect is to make the two lines the same, and then to make line 4 disappear, just as before.

How to avoid unknowns by text selection (sometimes)

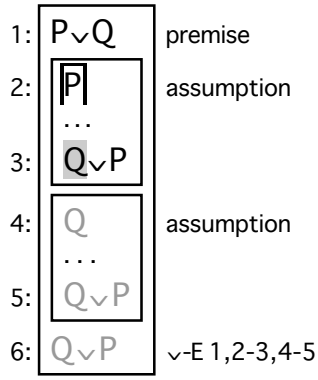
The ∨-I, ∧-I rules going forward will each accept an argument. So will the ¬-I rule going backwards. In every case you give an argument by text-selection, which is used instead of the unknown that Jape would normally introduce.

Consider the same problem as before:

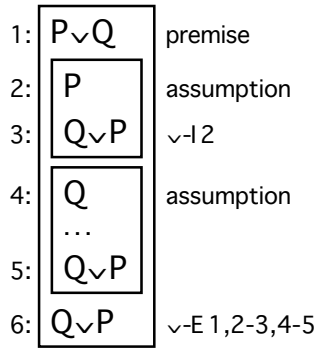


If you are still convinced that you want to work forward (the alternative is to select line 3 and apply "∨-I (preserving right)" from the Backward menu, which is *so* much easier), you can provide the forward ∨-I step with an argument to replace _A before it ever appears. You can always do this when you know what the argument should be (if you don't know what it should be then you really ought to be using the rule backwards).

Select P on line 2, and text-select Q somewhere in the proof:



Then apply "∨-I (inventing left)" from the Forward menu. Because you told it to use Q rather than _A, it makes a step to Q∨P, and that part of the proof is complete:



But it's a lot of work, and it's *so much easier* to go backwards ...

8. Scope boxing in Jape

The ∀-I rule of the lecture course looks something like this:

$$\begin{array}{lll} & \overset{c}{\boxed{\begin{array}{l}\ldots\\ A(c)\end{array}}} & \\ i: & & \ldots \\ \ldots & & \\ j: & \forall x.A(x) & \forall - I\ i \end{array}$$

where the little *c* in the margin next to the box ending on line *i* shows that it's a 'scope box'. Scope boxing imposes a condition on the proof that the name *c* doesn't appear outside the box. That condition stops you proving nonsense like ∀x.P(x) ⊢ ∃x.P(x) – it's not a theorem because an premise that *P*(*x*) holds universally doesn't prove that it holds at any particular position (the universe of possible values of *x* might be empty, and then the left-hand side would be trivially true while the right-hand side would be false).

Jape can't use scope-boxing directly, but it can imitate the mechanism reasonably well. It uses ordinary boxing with a pseudo-assumption 'var *c*' that means '*c* can be used in this box'. To use it you have to be a bit agile with the mouse, which is a pity but there it is.

Jape's versions of the ∀-I and ∃-E rules are[1]

$$\begin{array}{lll} i: & \boxed{\begin{array}{l}\text{var } c\\ \ldots\\ A(c)\end{array}} & \ldots \\ \ldots & & \\ k: & \forall x.A(x) & \forall - I\ i–j \end{array}$$

$$\begin{array}{lll} i: & \exists x.A(x) & \ldots \\ \ldots & & \\ j: & \boxed{\begin{array}{l}\text{var } c, A(c)\\ \ldots\\ B\end{array}} & \text{assumptions} \\ k: & & \ldots \\ \ldots & & \\ l: & B & \exists - E\ i, j..k \end{array}$$

The box in these diagrams are ordinary boxes. The "var *c*" pseudo-assumption is part of Jape's treatment of scope boxing: you can't appeal to var *c* outside the box in which it is introduced[2].

When you apply the ∀-E and ∃-I rules you have to make use of the variables that are introduced by the other quantifier rules. Here's an illustration. The starting point is:

$$\begin{array}{ll} 1: & \boxed{\begin{array}{l}\forall x.(P(x)\rightarrow Q(x))\\ \ldots\\ \forall x.P(x)\rightarrow\forall x.Q(x)\end{array}} \quad \text{premise} \\ 2: & \end{array}$$

First I break down the conclusion with →-I:

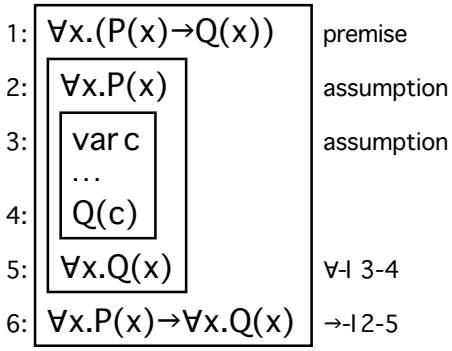
$$\begin{array}{ll} 1: & \boxed{\begin{array}{l}\forall x.(P(x)\rightarrow Q(x))\\ \boxed{\begin{array}{l}\forall x.P(x)\\ \ldots\\ \forall x.Q(x)\end{array}}\\ \forall x.P(x)\rightarrow\forall x.Q(x)\end{array}} \quad \begin{array}{l}\text{premise}\\ \text{assumption}\\ \\ \rightarrow\text{-I } 2\text{-}3\end{array} \\ 2: & \\ 3: & \\ 4: & \end{array}$$

---

1 Behind the scenes Jape doesn't use scope boxing and it doesn't even use predicate notation. Those wo are interested may read more of the gory details in appendix I. With luck, you will never notice the difference.

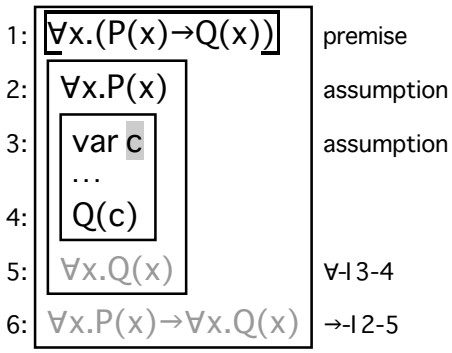
2 If necessary, Jape uses a proviso to ensure that *c* is a 'fresh individual', to stop you using *c* outside the box. The proviso looks like <variable> NOTIN <unknown>, and you will very rarely see it. When it is there, it's in the proviso pane.

Next I use ∀-I on line 3:

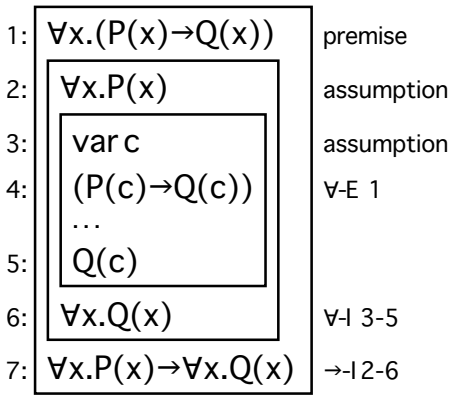


Line 3 is the first line of a scope box (it's labelled 'assumption' but never mind).

Because there's a variable on line 3, I can now use ∀-E. I'm going to have to do it twice, once with line 1 and once with line 2. It doesn't matter which I do first , so I choose line 1. Before I make the step I text-select *c* on line 3, because that's the variable I want to use[1]:



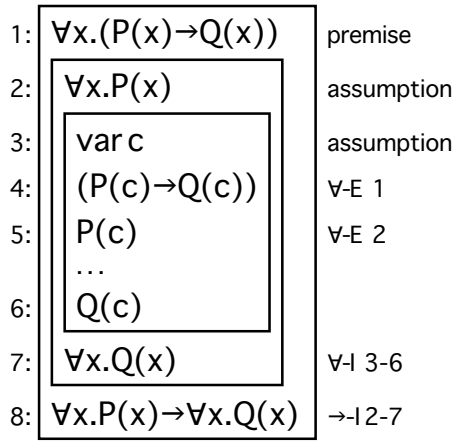
and then I apply the rule



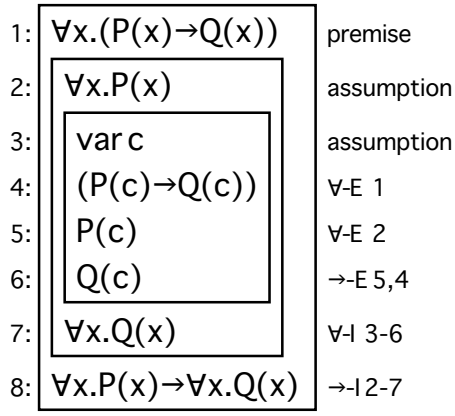
---

1 It's the only variable there is, so I have to use it. But Jape won't try to guess what I mean if I don't select it – it will just stop me making the step.

If I do the same thing again with line 2 (text-select *c* somewhere, select line 2, apply ∀-E) then the proof moves one step farther forward:

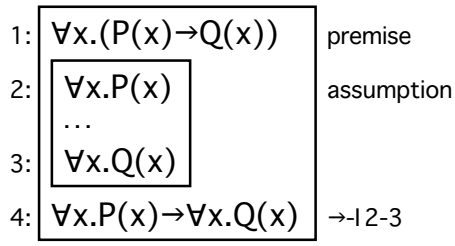


And finally a step of →-E finishes it off:



Enforcing the scope box condition

Suppose that I had done things in a slightly different order. Suppose that I try to make a ∀-E step before the ∀-I step:



There isn't a variable that I can select which is in scope, so I can't make the step. You can try to confuse Jape by text-selecting the variable *x* somewhere on the screen, but as you will see if you try it, it's ready for you.

9. Using theorems and defining conjectures

The Apply button in the Conjectures panel lets you use theorems in proofs – 'sequent substitution' is the phrase that is used in the Introduction to Logic course to describe this action. I illustrated it above, in the last of the three alternative proofs of $P \to Q$, $Q \to R$, $P \vdash R$. In that proof I used the theorem P, $P \to Q \vdash Q$ twice: once to prove P, $P \to Q \vdash Q$, and once to prove Q, $Q \to R \vdash R$. Each of those sequents is a 'substitution instance' of the theorem; only the first happens to be identical to the theorem.

I2L Jape makes substitution instances of a theorem by replacing some or all of the names in its sequent. But only some names are replaceable, and it uses a simple lexical convention to help it to decide. Names in sequents fall into one of three classes:

- *variables*: any name starting with *x*, *y*, *z* or *c*;
- *formulas*: any name starting with *A*, *B*, *C*, *P*, *Q*, *R* or *S*.

Variable names in a theorem sequent can be replaced by any variable, formula names by any formula, constant names by any constant. This means, for example, that $P \to (Q \to R)$, $P \vdash Q \to R$, and $(P1 \wedge P2) \to (Q1 \vee Q2), P1 \wedge P1 \vdash Q1 \vee Q2$ are each substitution instances of $P \to Q, P \vdash Q$.

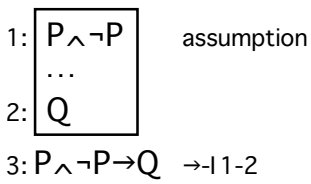
I use formula names *A*, *B* and *C* in rules, *P*, *Q*, *R* and *S* in theorems.

Stating your own conjectures

The (New) button at the bottom of the Conjectures panel lets you define your own conjectures. It brings up a dialogue box (no illustration, I'm sorry to have to say again). You type the conjectured sequent into the box labelled CONJECTURE. (Usually that is all you need to do, but if the conjecture has any provisos you type them into the box labelled PROVISOS; if it has any parameter names you type them into the box labelled (...).) The keypad buttons at the bottom of the box are there so that you don't have to learn how to type the special symbols that appear in logical formulæ and sequents. When you have finished, press the State Conjecture button (you need also to press Cancel in the Conjecture Entry dialogue box, because it doesn't automatically disappear as it should): Jape adds your conjecture to the Conjectures panel and selects it, so that you can press Prove to start a proof of it.

When you are stating a conjecture, don't forget to distinguish between the various classes of name (see above). If you use a name which is outside either class, Jape will refuse to accept the conjecture. If you state a conjecture which has the wrong kind of name in it – one which contains a formula like $\forall P.Q$, for example, which has a formula name where a variable name is required – Jape may accept it but you will find that the rules don't actually apply.

Never mind all those complications! It's actually very easy to define a conjecture and then to start a proof of it. For example, I typed $P \wedge \neg P \to Q$ into the CONJECTURE box, pressed State Conjecture in the dialogue window and then Prove in the Conjectures panel. Here's the first step in the proof of that wild conjecture – it's provable, but I'm not going to tell you how.



10. Reviewing and altering proofs of theorems

If you select a theorem in the Conjectures panel and press the Show button, you will be shown the last-registered proof of that theorem – only its final stage, not the intermediate steps. If you don't like what you see you can change it if you like, and register the changed proof with Done. Because Jape doesn't register the intermediate steps of a proof, you can't use Backtrack or Undo to trace through its development, but you can 'prune' parts of it and do them differently.

For example, consider the following proof of $P \wedge (Q \wedge R) \vdash (P \wedge Q) \wedge R$:

1:	$P \wedge (Q \wedge R)$	premise
2:	P	$\wedge$-E(L) 1
3:	$(Q \wedge R)$	$\wedge$-E(R) 1
4:	Q	$\wedge$-E(L) 3
5:	$(P \wedge Q)$	$\wedge$-I 2,4
6:	$(Q \wedge R)$	$\wedge$-E(R) 1
7:	R	$\wedge$-E(R) 6
8:	$(P \wedge Q) \wedge R$	$\wedge$-I 5,7

Because of the order in which I developed the proof (first several $\wedge$-Is, backwards, to reduce the conclusion, then $\wedge$-Es , forwards, to simplify the premise), I've used $\wedge$-E twice to get two lines which state $Q \wedge R$ – lines 3 and 6. I could have done things in another order. To show what I mean, I click on the conclusion in line 8 and choose Prune from the Edit menu, to see

1:	$P \wedge (Q \wedge R)$	assumption
	…	
2:	$(P \wedge Q) \wedge R$	

Then I can use $\wedge$-E four times to reduce the premise:

1:	$P \wedge (Q \wedge R)$	assumption
2:	P	$\wedge$-E(L) 1
3:	$Q \wedge R$	$\wedge$-E(R) 1
4:	Q	$\wedge$-E(L) 3
5:	R	$\wedge$-E(R) 3
	…	
6:	$(P \wedge Q) \wedge R$	

and then $\wedge$-I twice to complete the proof:

1:	$P \wedge (Q \wedge R)$	premise
2:	P	$\wedge$-E(L) 1
3:	$(Q \wedge R)$	$\wedge$-E(R) 1
4:	Q	$\wedge$-E(L) 3
5:	R	$\wedge$-E(R) 3
6:	$(P \wedge Q)$	$\wedge$-I 2,4
7:	$(P \wedge Q) \wedge R$	$\wedge$-I 6,5

This version is one line shorter. It's not much of a difference, but I think it's an improvement. In other cases pruning and reproving can be much more effective.

Prove starts a new proof

If you select a theorem – a conjecture which you've already proved and registered with Jape – in the Conjectures panel and press Prove, Jape will start a new proof of that theorem, after first checking that is what you really want to do.

Beware circular arguments!

Jape checks your proofs for circularity. It's possible to make a proof of theorem A, then prove theorem B using theorem A, then go back and re-'prove' theorem A using theorem B. The two proofs are now nonsense– A depends on B depends on A depends on B[1] ... – because you have made a 'circular argument'.

Jape doesn't check your proof as you make it, but when you think you've finished and try to register the new proof, it will refuse, telling you just how the proof is circular. Change the proof, or use Abandon Proof to forget all about it!

---

[1] Well, sometimes complicated meta-argument using induction on the structure of your proofs can make this kind of circularity acceptable. It's simplest to say that the proofs are nonsense.

11. Printing a proof (File menu)

If you choose Print from the File menu, you will be presented with a dialogue box with some strange graphics on it, asking you to choose how your proof should be printed. Just press the Write File button if – as normally – you don't want to rotate or re-size your proof before printing[1]. The proof will be sent to your default printer (the one named in your PRINTER environment variable, the same one that output is sent to when you use lpr to print a file from an terminal window). At the time of writing the proof that is printed doesn't have a title.

You may want to do more complicated things with your proof, and it's possible to save your proof as an EPS (Encapsulated PostScript) file (if you don't know what that is, relax and skip to the next section). This is what you do. Choose the "Output Proof as Scaled Postscript" entry in the File menu. A dialogue box is shown to you, called "select a file for the postscript proof", which for reasons cited above I unfortunately can't illustrate here. In that dialogue box you can enter a file name (next to File: at the bottom), or you can just press OK and it will choose a file name for you.

Next you will see another dialogue box – the same one as in the Print dialogue – which invites you to re-size and/or rotate the printed proof. You can do wonderful things with this dialogue box – shrinking, growing, rotating the image – but almost certainly you will just need to press Write File.

Now you have to go to a terminal window, and type a UNIX command. If necessary you will have to create one – you're on your own here. Type

```
jlpr filename
```

or, if you know how to choose a printer to send your output to, type

```
jlpr -Pprinter filename
```

The filenames that Jape uses will all be of the form `ItLjapenn.jps`. Printers in use in the ITL at QMW are called itlaser1, itlaser3.

Good luck. It's been so long since I used UNIX much that I'd forgotten how ghastly it all is.

---

1 If you do want to rotate or resize, I think the controls in the dialogue box are straightforward. Anyway, you can experiment.

12. Saving and restoring your proofs (the File menu)

On the File menu there are a collection of commands to open, close and save files of logic descriptions and proofs. You won't need, and shouldn't use, the first, second and fourth commands (Select a top-level theory file, Add a theory file to the current theory, load the file ./J): if you do use them by accident, be sure to press Cancel in the file selection window or I shan't be liable for the consequences.

Save Proofs and Save As… will save your completed proofs, and any proofs in progress in the session window, in a file of your choice, using a file-selection dialogue (it's best to save them in files called *somethingorother*.jp). Load a file of proofs will load them back again (and it will automatically look for files called *whatsit*.jp, which is why it's best to save proofs in a file with that sort of name). If you have any unsaved proofs when you quit, or any proofs in progress in the session window, Jape will ask you if you want to save them.

If you want to build up a collection of proofs in a file, make sure you reload your old proofs before you start adding to the collection. If you start a new collection and then save it to the same file, X Jape will – without checking that it is what you want to do – overwrite the old collection.

Close seems to have the same effect as Quit Jape. Why do you need two identical commands on the same window? Search me!

The "Select a top-level theory file" command

The "Select a top-level theory file" command on the File menu lets you erase the logic loaded into Jape – both its rules and any proofs which you might have made – and load the rules of another logic. The dialogue is a little confusing and, in particular, once you have pressed OK in the file selection window, pressing Cancel in later windows may not have the effect you expect.

There are some interesting logics encoded in Japeish, which can be found in the `/import/jape/examples_3.5` directory. If you look at them then you are on your own: I take no responsibility for any confusion which any of those logics may cause!

Appendix A: Using the mouse

I2L Jape on X understands various mouse gestures in the proof pane:

- a single *left-button click* selects a hypothesis or conclusion formula to work on;
- a single *middle-button click* to select a character as an argument to a rule;
- *press and drag* with the *middle button* down to select a sequence of characters as argument to a rule;
- *press and drag* with the *middle button* down over a blank portion of the pane to move the proof within the pane.

In addition, various mouse gestures cancel the effect of some of the above:

- a *left-button click* or *middle-button click* in a blank area of the screen cancels all formula and text selections;
- a *middle-button click* over a text selection cancels it (and, currently, any other text selections in the same formula, but not any in other formulæ);

Single-left-click

A box is drawn round the formula you clicked on, open at the top (conclusion formula) or at the bottom (hypothesis formula), and it becomes a focus of attention for the command you apply. You can select both a hypothesis and a conclusion formula, but no more than one of each kind. Clicking on a new hypothesis formula cancels any previous hypothesis selection, and similarly for conclusions.

When you select a conclusion, all the other conclusions are greyed-out to show that you can't use them. Only the hypotheses appropriate to that conclusion are still non-grey.

When you select a hypothesis, all but the unproved conclusions which are associated with it are greyed-out.

If you select only a hypothesis, there may be many unproved conclusions with which it is associated. In order to make a proof step, Jape needs to know which conclusion you want to work towards. It will ask you to select a conclusion if necessary.

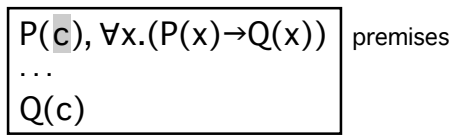
Left-clicking in a blank area of the pane cancels all formula and text selections.

Double-left-click

Jape allows the logic designer to define the effect of a double-click on an assumption or conclusion, so that rules can be 'automatically' applied. We have decided not to use that facility in I2L Jape, because we want you to think about, and learn about, the choice of rules.

Middle-click and middle-press-and-drag

If you click the middle button over a character it is text-selected; if you press the middle button over a character and move the mouse left or right, that character and the others you drag over are text-selected; in either case the selected text is highlighted. It looks like this:



When you next apply a rule, the highlighted text is given as an argument to that rule: that's mainly useful in the rules which have to do with ∀ and ∃ (see below). If the text won't do – for example because it isn't a proper logical formula, or because it is the wrong kind of formula – Jape will complain.

You can cancel a text selection that you have already made either by double-middle clicking it, or by double-middle clicking in a blank area of the proof pane.

Appendix B – Troubleshooting

Problems with windows

I didn't write the graphical interface to X Japе, so I can't say much about what to do if it won't start, or if the windows don't perform as described. Ask a demonstrator or a friend where all your windows have gone before panicking. Mail me (richard@dcs.qmw.ac.uk) if you have real difficulty, and I'll do my best to help.

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 Japе 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 a successful step now can lead to a dead end later. 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 intended, the first stage of a cure is to use the Undo command (choose it from the Edit menu). Undo takes you back one step in the proof, two Undos take you back two steps, and so on. Using Undo can reverse an unintended step; using several 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 (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 that surprising 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 indeed such a search usually turns out to be a very slow way to make proofs. But sometimes we all need to search for a proof, 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. Japе is designed to support that kind of 'reflective exploration' – it helps with the exploring part, and you learn by reflecting on the results.

Appendix C: The → rules

Normally you will use the → *elimination* rule to make a forward step from an assumption (or from a conclusion already proved by a forward step) and you will use the → *introduction* rule to make a backward step from a conclusion. In either case the assumption or conclusion should be of the form <something> → <something>, and in either case the effect is to *simplify* the assumption or conclusion you selected by splitting it into two parts and eliminating the → connective.

Forward step with →-E

i: A ...
...
j: $A \rightarrow B$ ...
...
k: B $\rightarrow -E\ i,j$

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A & A \rightarrow B\end{array}}{B} \rightarrow -E$$

– from a proof of A and a proof of $A \rightarrow B$, make a proof of B.

Select a hypothesis formula which has an arrow you wish to eliminate, select the conclusion if necessary, and apply the rule. For example:

before	*after*
1: P→(Q→R) premise … 2: Q→(P→R)	1: P→(Q→R) premise … 2: P 3: (Q→R) →-E 2,1 … 4: Q→(P→R)

Backward step with →-I

i : A assumption
...
j : B ...
...
k : $A \rightarrow B$ $\rightarrow -I\ i..j$

$$\frac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \rightarrow B} \rightarrow -I$$

– from a proof of B, within which you are allowed to assume a proof of A, construct a proof of $A \rightarrow B$.

Select an unproved conclusion which includes an arrow you want to eliminate, and apply →-I. For example:

before	*after*
1: P→(Q→R) premise … 2: Q→(P→R)	1: P→(Q→R) premise 2: Q assumption … 3: (P→R) 4: Q→(P→R) →-I 2-3

Appendix D: the ∧ rules

Normally you will use the ∧ *elimination* rule to make a forward step from an a hypothesis and you will use the ∧ *introduction* rule to make a backward step from an unproved conclusion. In either case the formula should be of the form <something> ∧ <something>.

You can use the ∧ introduction rule forwards, and in that case it makes sense to give it an argument – see section 7 above.

Forward step with ∧-E

The two forms of ∧-E are:

$$\begin{array}{lll} i: & A \wedge B & \ldots \\ & \ldots & \\ j: & A & \wedge -E\ i \end{array} \qquad \begin{array}{c} \vdots \\ \dfrac{A \wedge B}{A}\ \wedge -E \end{array}$$

$$\begin{array}{lll} i: & A \wedge B & \ldots \\ & \ldots & \\ j: & B & \wedge -E\ i \end{array} \qquad \begin{array}{c} \vdots \\ \dfrac{A \wedge B}{B}\ \wedge -E \end{array}$$

– from a proof of $A \wedge B$ you can make a proof of A or a proof of B, as you wish. Because there are two alternative forms there are two entries in the Forward menu, one preserving the left-hand half of the formula (A), and the other preserving the right-hand half (B).

Select a hypothesis formula which has a conjunction which you want to eliminate, and apply the relevant version of the rule. For example:

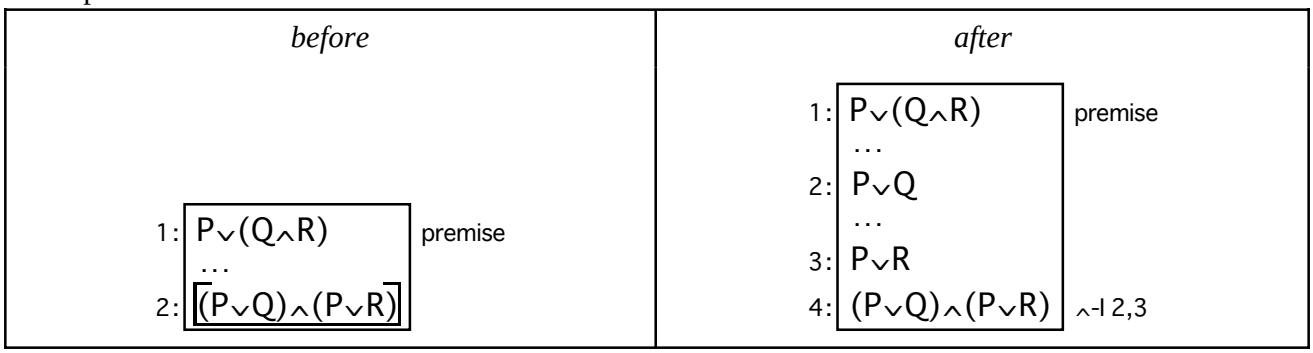
before	*after*
1: [(P∨Q)∧(P∨R)] premise … 2: [P∨(Q∧R)]	1: [(P∨Q)∧(P∨R)] premise 2: [P∨Q] ∧-E 1 … 3: [P∨(Q∧R)]

Backward step with ∧-I

$$\begin{array}{lll} i: & A & \ldots \\ & \ldots & \\ j: & B & \ldots \\ & \ldots & \\ k: & A \wedge B & \wedge -I\ i, j \end{array} \qquad \begin{array}{c} \vdots \quad \vdots \\ \dfrac{A \quad B}{A \wedge B}\ \wedge -I \end{array}$$

– from a proof of A and a proof of B make a proof of $A \wedge B$.

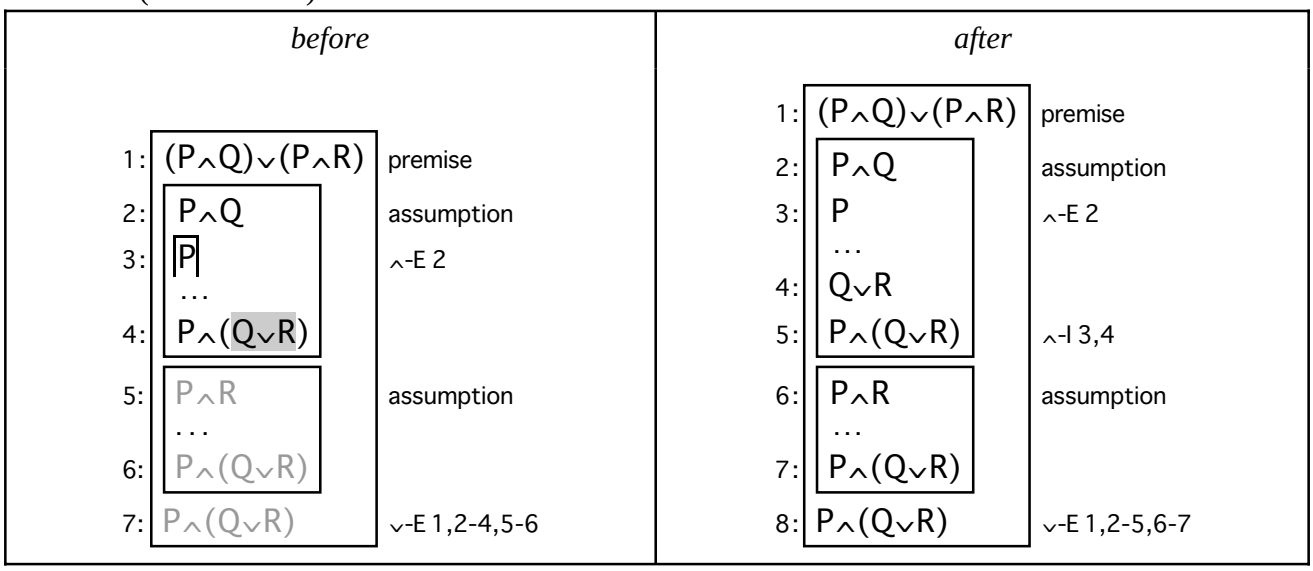
Select an unproved conclusion which is a conjunction that you want to simplify, and apply the rule. For example:



Forward step with ∧-I (not recommended)

> Although it is possible to use the ∧-I rule forwards, it's so much eaiser to use it backwards that I strongly recommend you not to use it forwards.

Because it is possible to form *something*∧*hypothesis* or *hypothesis*∧*something*, there are two entries in the menu. Select a hypothesis formula which is to form half of the result, and choose the relevant form of the rule. It's best to select an argument formula which will replace *something*, or else Jape will use an unknown (see section 7).



– but since you get *exactly* the same effect just by using ∧-I backwards at line 4, the forward step is a lot of unnecessary work.

Appendix E: the ∨ rules

Normally you will use the ∨ *elimination* rule to make a forward step from an assumption (or from a conclusion already proved by a forward step) and you will use one of the ∨ *introduction* rules to make a backward step from a conclusion. In either case the assumption or conclusion should be of the form <something> ∨ <something>, and in either case the effect is to *simplify* the assumption or conclusion you selected.

You can use the ∨ introduction rules to reason forwards, and in that case it usually makes sense to give the rule an argument – see below.

Forward step with ∨-E

i : $A \vee B$ ...
...
j: A assumption
...
k : C ...
...
l : B assumption
...
m : C ...
...
n : C $\vee - E\ i, j..k, l..m$

$$\frac{A \vee B \quad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \quad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C} \vee E$$

– from a proof of $A \vee B$, a proof of C (which can assume a proof of A) and another proof of the same C (which this time can assume a proof of B), make a proof of C.

Select a hypothesis formula which includes a disjunction which you wish to eliminate. For example:

before

1: (P∨Q)∧(P∨R) premise
2: P∨Q ∧-E 1
...
3: P∨(Q∧R)

after

1: (P∨Q)∧(P∨R) premise
2: P∨Q ∧-E 1
3: P assumption
...
4: P∨(Q∧R)
5: Q assumption
...
6: P∨(Q∧R)
7: P∨(Q∧R) ∨-E 2,3-4,5-6

Note lines 4, 6 and 7 are the same formula, the conclusion which the step was working towards.

Backward step with ∨-I

The two forms of ∨-I are

i : A ...
...
j : $A \vee B$ $\vee - I(L)\ i$

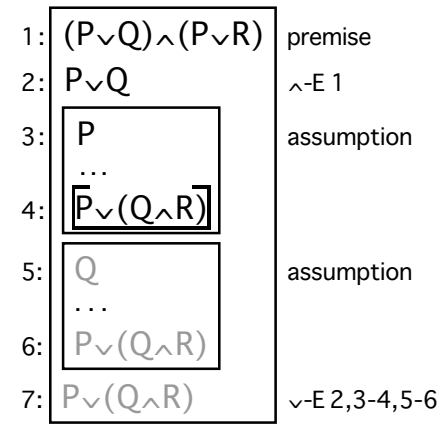
$$\frac{\begin{array}{c}\vdots\\ A\end{array}}{A \vee B} \vee -I(L)$$

$$\begin{array}{l} i:\ B \quad \ldots \\ \quad \ldots \\ j:\ A \vee B \quad \vee -I(R)\ i \end{array} \qquad\qquad \dfrac{\begin{array}{c}\vdots\\ B\end{array}}{A \vee B}\ \vee -I(R)$$

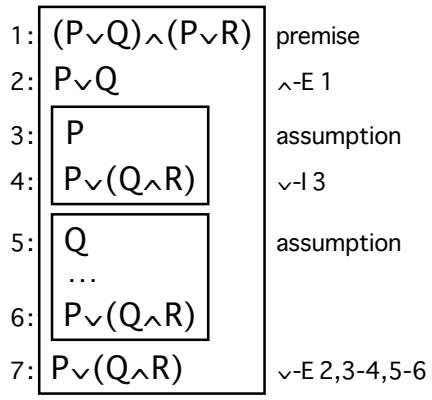
– from a proof of *A* or from a proof of *B* you can make a proof of *A*∨*B*.

Select an unproved conclusion which is a disjunction which you want to simplify, and apply the relevant version of the rule. For example:

before



after

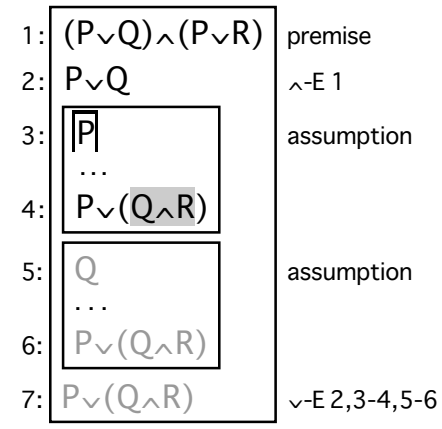


Forward step with ∨-I (not recommended)

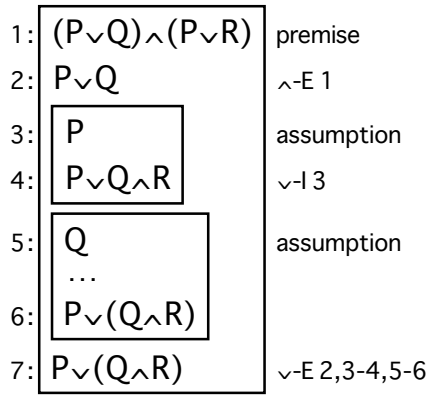
> Although it is possible to use the ∨-I rule forwards, it's so much easier to use it backwards that I strongly recommend you not to use it forwards.

Because it is possible to form *something*∨*hypothesis* or *hypothesis*∨*something*, there are two entries in the menu. Select a hypothesis formula which is to form half of the result, and choose the relevant form of the rule. It's best to select an argument formula which will replace *something*, or else Jape will use an unknown (see section 7).

before



after



– but since you get *exactly* the same effect just by using ∨-I backwards at line 4, the forward step is a lot of unnecessary work.

Appendix F: the ¬ rules[1]

You can use the ¬ elimination rule forward, and the ¬ introduction rule backward, like any of the other rules. You can also use the ¬ elimination rule backwards, and you often do so as the first step of a 'proof by contradiction'.

Forward step with ¬-E

$$\begin{array}{ll} i: \neg\neg A & \ldots \\ \quad \ldots & \\ j: A & \neg - E\ i \end{array} \qquad\qquad \begin{array}{c} \vdots \\ \neg\neg A \\ \hline A \end{array} \neg - E$$

– from a proof of $\neg\neg A$ you can conclude A.

If you have an assumption of the form $\neg\neg A$ then you can select it and apply the ¬-E rule. For example:

before	*after*
1: ¬¬P premise … 2: Q→P	1: ¬¬P premise 2: P ¬-E 1 … 3: Q→P

Backward step with ¬-I

$$\begin{array}{ll} i: & \boxed{\begin{array}{l} A \\ \ldots \\ B \wedge \neg B \end{array}} \quad \text{assumption} \\ \quad \ldots & \ldots \\ k: \neg A & \neg - I\ i..j \end{array} \qquad\qquad \begin{array}{c} [A] \\ \vdots \\ B \wedge \neg B \\ \hline \neg A \end{array} \neg - I$$

– if from assumption A you can prove the contradiction $B \wedge \neg B$, then A can't be valid and you have made a proof of $\neg A$. The formula B is arbitrary, and if you apply this rule without providing an argument Jape will introduce an unknown (see section 7).

> You will often find that it is not helpful to give an argument: better to explore the proof a little farther to see what turns up before you decide what the unknown formula *B* should be.

Select a conclusion which has a negation which you wish to get rid of, and apply the rule. For example:

before	*after*
1: P→Q premise … 2: ¬(P∧¬Q)	1: P→Q premise 2: (P∧¬Q) assumption … 3: _B∧¬_B 4: ¬(P∧¬Q) ¬-I 2-3

We can let the unknown _B ride along while we explore to see what the contradiction is going to be.

---

[1] I find the rules for ¬, as given in the I2L course, very *un*natural. But you have to be able to use them just as they are described in the course, and therefore I2L Jape has to include them as taught. It would be much simpler if the rules used an explicit symbol to mean 'contradiction', but they don't.

Backward step with ¬-E

Select an unproved conclusion and apply the rule. For example:

before	*after*
... 1: P∨¬P	... 1: ¬¬(P∨¬P) 2: P∨¬P ¬-E 1

'Proof by contradiction': backward steps with ¬-E,¬-I, ∧-I (and possibly hyp)

If ¬*C* is ridiculous – if it leads to a contradiction – then *C* must hold. In the rules of the lecture course, you have to use a long-winded strategy: if ¬*C* leads to a contradiction, then ¬¬*C* holds (by ¬-I), and then *C* holds (by ¬-E). And since ¬-I only works backwards, you do this part of the proof backwards.

The process has three steps, or sometimes four. Here is an example:

step 1: apply ¬-E	*step 2: apply ¬-I*	*step 3: apply ∧-I*
... 1: P∨¬P	... 1: ¬¬(P∨¬P) 2: P∨¬P ¬-E 1	1: ¬(P∨¬P) assumption ... 2: _B∧¬_B 3: ¬¬(P∨¬P) ¬-I 1-2 4: P∨¬P ¬-E 3

step 4: apply hyp	*after*
1: ¬(P∨¬P) assumption ... 2: _B ... 3: ¬_B 4: _B∧¬_B ∧-I 2,3 5: ¬¬(P∨¬P) ¬-I 1-4 6: P∨¬P ¬-E 5	1: ¬(P∨¬P) assumption ... 2: P∨¬P 3: (P∨¬P)∧¬(P∨¬P) ∧-I 2,1 4: ¬¬(P∨¬P) ¬-I 1-3 5: P∨¬P ¬-E 4

Step 4 is the one which is optional. In that step you look for a suitable negated formula in the hypotheses to replace ¬_B, and if you find one, you use *hyp* to get rid of the unknown (see section 7). If there is more than one negated hypothesis formula, you may have to try one and if that doesn't work, Undo and try another, and so on. (Proof by contradiction is rarely straightforward, so don't despair, just keep experimenting.)

Appendix G: the ∀ rules

Jape can't use exactly the same syntax as the lecture notes for ∀ formulæ: it requires a dot (full stop, period) after the bound variable, and the body of the formula doesn't need to be bracketed. Otherwise the treatment is identical. You need to bracket the body of the formula if it isn't a predicate or a negation, so in Jape you write ∀x.(P(x)∧Q(x)).

Jape can't use the scope-boxing mechanism of the lecture notes. Instead it uses a pseudo-assumption 'var <variable>' to imitate variable scoping.

Normally you will use the ∀ *elimination* rule to make a forward step from a hypothesis and you will use the ∀ *introduction* rule to make a backward step from a conclusion. In either case the formula must be ∀ <variable> . <something>, and the effect is to *simplify* the formula you selected.

You must give an argument to the elimination rule by text-selecting a variable, so the order in which you use quantification rules is important.

Forward step with ∀-E

The ∀-E rule in box form is

$$\begin{array}{lll} i: & \text{var } c & \ldots \\ & \ldots & \\ j: & \forall x.A(x) & \ldots \\ & \ldots & \\ k: & A(c) & \forall - E\ j \end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \forall x.A(x)\end{array}}{A(c)}\ (c \text{ in scope}) \forall - E$$

A is some predicate: then provided there is a declaration of variable c in a box enclosing line k; from a proof of $\forall x.A(x)$ you can make a proof of $A(c)$.

Select a hypothesis which is a universal which you want to specialise, text-select an instance of a variable whose scope includes the current conclusion, and apply the ∀-E rule. For example:

before

1: var c, P(c), ∀x.(P(x)→Q(x)) premises
…
2: Q(c)

after

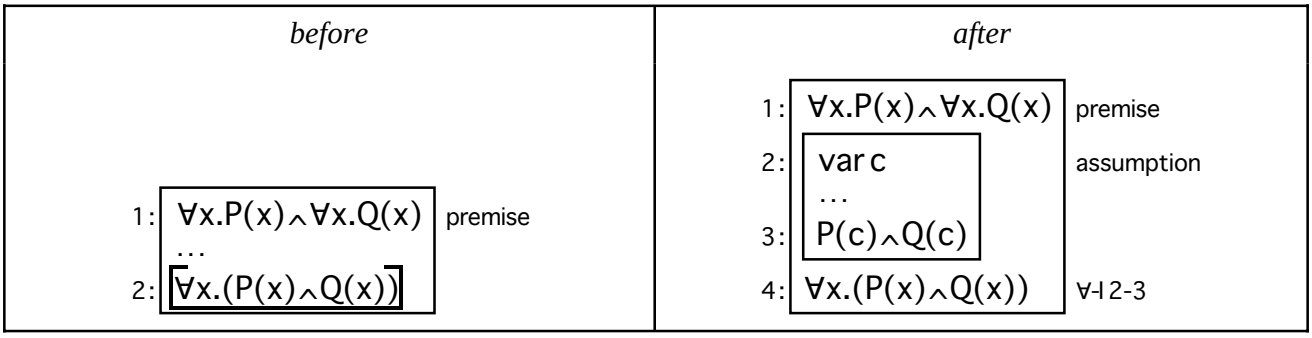
1: var c, P(c), ∀x.(P(x)→Q(x)) premises
2: P(c)→Q(c) ∀-E 1.3
…
3: Q(c)

Backward step with ∀-I

$$\begin{array}{lll} i: & \text{var } c & \\ & \ldots & \\ j: & A(c) & \ldots \\ & \ldots & \\ k: & \forall x.A(x) & \rightarrow -\forall - I\ i{-}j \end{array} \qquad \dfrac{\begin{array}{c}[\text{var } c]\\ \vdots\\ A(c)\end{array}}{\forall x.A(x)}\ (\text{FRESH } c)\ \forall - I$$

A is some predicate, and c is a 'fresh variable' – a name you haven't made any assumptions about outside the box which proves $A(c)$. The *var c* pseudo-assumption is Jape's way of imitating scope-boxing.

Select a conclusion which is a universal which you want to specialise, and apply the ∀-I rule. For example:


before
1: ∀x.P(x)∧∀x.Q(x) premise
...
2: ∀x.(P(x)∧Q(x))
after
1: ∀x.P(x)∧∀x.Q(x) premise
2: var c assumption
...
3: P(c)∧Q(c)
4: ∀x.(P(x)∧Q(x)) ∀-I 2-3

Appendix H: the ∃ rules

Jape can't use exactly the same syntax as the lecture notes for ∀ formulæ: it requires a dot (full stop, period) after the bound variable, and the body of the formula doesn't need to be bracketed. Otherwise the treatment is identical. You need to bracket the body of the formula if it isn't a predicate or a negation, so in Jape you write ∃x.(P(x)∧Q(x)).

Jape can't use the scope-boxing mechanism of the lecture notes. Instead it uses a pseudo-assumption 'var <variable>' to imitate variable scoping.

Normally you will use the ∃ *elimination* rule to make a forward step from a hypothesis and you will use the ∃ *introduction* rule to make a backward step from a conclusion. In either case the formula should be ∃ <variable> . <something>.

You must give an argument to the ∃-I rule by text-selecting some variable, so the order in which you use quantification rules is important.

Forward step with ∃-E

i:	$\exists x.A(x)$	...
	...	
j:	var $c, A(c)$	assumptions
	...	
k:	B	...
	...	
l:	B	$\exists - E\ i, j..k$

$$\frac{\exists x.A(x) \qquad \begin{array}{c}[\text{var } c, A(c)] \\ \vdots \\ B\end{array}}{B}\ (\text{FRESH } c)\ \exists - E$$

A is some predicate, and *c* is a 'fresh variable' – a name you haven't made any assumptions about outside the box which assumes *A*(*c*) to prove *B*. The *var c* pseudo-assumption is Jape's way of imitating scope-boxing.

Select a hypothesis which is an existential that you wish to reason with, and apply the ∃-E rule. For example:

before

```
1: ∃x.(P(x)∧Q(x))        premise
   ...
2: ∃x.P(x)∧∃x.Q(x)
```

after

```
1: ∃x.(P(x)∧Q(x))        premise
2:   var c, (P(c)∧Q(c))  assumptions
     ...
3:   ∃x.P(x)∧∃x.Q(x)
4: ∃x.P(x)∧∃x.Q(x)       ∃-E 1,2-3
```

Note that lines 3 and 4 are the same formula, the conclusion which the step was working towards (cf. ∨-E above).

Backward step with ∃-I

i: var *c* ...
...
j: *A*(*c*) ...
...
k: ∃*x*.*A*(*x*) ∃–*I j*

$$\frac{\begin{array}{c}\vdots\\ A(c)\end{array}}{\exists x.A(x)}\ (c \text{ in scope})\ \exists - I$$

A is some predicate: then provided there is a declaration of variable *c* in a box enclosing line *k*; from a proof of *A*(*c*) you can make a proof of ∃*x*.*A*(*x*).

Select an unproved conclusion which is an existential which you want to establish, text-select an instance of a variable whose scope includes the current conclusion and apply the ∃-I rule. For example:

before

1:	∀x.(P(x)→Q(x)), ∃x.P(x)	premises
2:	var c, P(c) …	assumptions
3:	∃x.Q(x)	
4:	∃x.Q(x)	∃-E 1.2,2-3

after

1:	∀x.(P(x)→Q(x)), ∃x.P(x)	premises
2:	var c, P(c) …	assumptions
3:	Q(c)	
4:	∃x.Q(x)	∃-I 3
5:	∃x.Q(x)	∃-E 1.2,2-4

Forward reasoning with ∃-I

> Although it is possible to use the ∃-I rule forwards, it's a real pain. I strongly recommend you *not* to use it forwards.

Select a hypothesis which you want to use as the basis of an existential, text-select the instance(s) of variables inside the hypothesis which should be replaced by the bound variable, and use the ∃-I rule. If you've done it right, you will get an existential with an unknown bound variable, and some nasty-looking provisos. Almost always you can tidy things up by unifying the new existential with an unproved conclusion ***but in that case you could have got the same result much more easily with a backward step.*** For example:

step 1: apply ∃-I

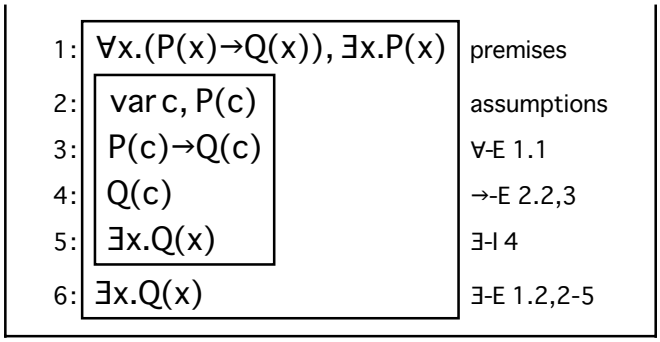
1:	∀x.(P(x)→Q(x)), ∃x.P(x)	premises
2:	var c, P(c)	assumptions
3:	P(c)→Q(c)	∀-E 1.1
4:	Q(c) …	→-E 2.2,3
5:	∃x.Q(x)	
6:	∃x.Q(x)	∃-E 1.2,2-5

step 2: apply hyp

1:	∀x.(P(x)→Q(x)), ∃x.P(x)	premises
2:	var c, P(c)	assumptions
3:	P(c)→Q(c)	∀-E 1.1
4:	Q(c)	→-E 2.2,3
5:	∃_x6.Q(_x6) …	∃-I 4
6:	∃x.Q(x)	
7:	∃x.Q(x)	∃-E 1.2,2-6

_x6 NOTIN Q
_x6 NOTIN c

after



You could have produced that effect much more easily, just by applying ∃-I backwards at line 5. The forward step is a waste of time and effort.