Using

JAPE

for

“Introduction to Formal Proof”

Bernard Sufrin

18th May, 2015[*]

Abstract

Jape is not needed to work on the tutorial sheets associated with the course “Introduction to Formal Proof”; but it has helped intrepid users explore ways of discovering proofs more effectively than by just using pencil-and-paper. Here we give a brief introduction to the use of Jape to explore proofs in the logic presented in the course.[1]

[*](Draft 1.674)

[1]The Jape theory used in this introduction is the `IFPFRESH` variant, in which the scopes of invented variables are explicitly declared.

Contents

1 Introduction

INTRODUCTION TO FORMAL PROOF is based on our conviction that computer science students who have mastered some functional programming can easily be introduced to the ideas behind formal proof in propositional and first-order logics described by inference systems; can soon get used to making formal proofs of non-trivial conjectures in such logics; and can understand the connections between formal semantics and inference systems sufficiently well to understand notions of the soundness and completeness.

With this background it is easier to see how inference systems are used to formalise judgments made in other settings in computer science – for example logics of program correctness, type inference systems, and operational semantics.

JAPE is a configurable proof calculator designed to support interactive discovery of formal proofs in inference systems. It is distributed with several example configurations. These support proofs in several variants of the sequent calculus, natural deduction, functional programming, and operational semantics; they also support proof in a Hoare-style logic of program derivation and derivations in a polymorphic type inference system.

Recent *Jape* builds for 32 and 64 bit Linux, and for OS/X (10.6 upwards) are available at the distribution site:[1] [†]

```
http://japeforall.org.uk
```

Instructions for downloading and installing Jape are given there, and in what follows we will assume that you have done so.

The logic used here is the propositional and first-order logic explained in detail in the lecture notes for the course; we reproduce it *without explanation* in Appendix D. The Jape configuration files that support it are available from the course website, as a folder named `IFP`. You should also download this folder.

2 Proving $E, \neg E \vdash F$

2.1 Getting Started

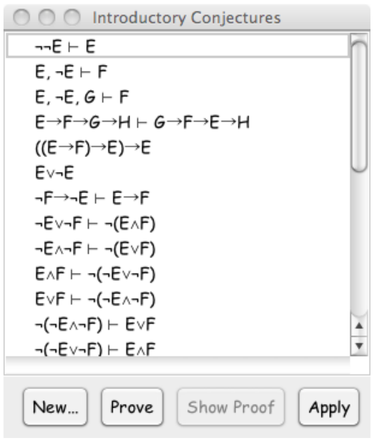
First start Jape running and install the IFP configuration.

In a normal OS/X installation Jape can be started with the shell command `open -a jape` or by clicking on the Jape app icon in the appropriate place; in a normal Linux installation Jape is started with the shell command `$HOME/bin/Jape`

When the Jape splash-screen comes up, use the Open new theory button on the File menu to import the IFP configuration file `IFP.jt`.[2]

The Introductory Conjectures window, should soon appear:

[†]Notes appear on page 23.

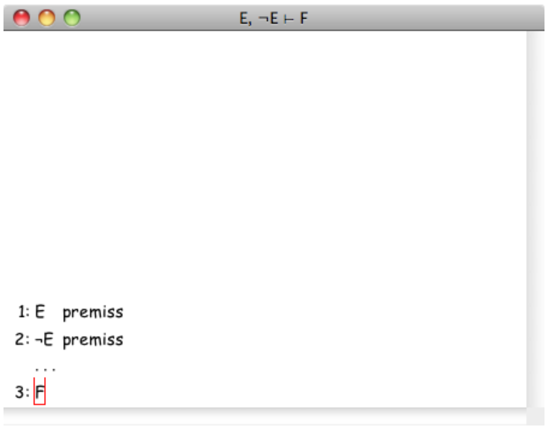

Introductory Conjectures

¬¬E ⊢ E
E, ¬E ⊢ F
E, ¬E, G ⊢ F
E→F→G→H ⊢ G→F→E→H
((E→F)→E)→E
E∨¬E
¬F→¬E ⊢ E→F
¬E∨¬F ⊢ ¬(E∧F)
¬E∧¬F ⊢ ¬(E∨F)
E∧F ⊢ ¬(¬E∨¬F)
E∨F ⊢ ¬(¬E∧¬F)
¬(¬E∧¬F) ⊢ E∨F
¬(¬E∨¬F) ⊢ E∧F

Introductory Conjectures Window

Select the conjecture $E, \neg E \vdash F$ from this window, and press the Prove button there.

A new window is opened that shows the conjecture – with an ellipsis $(\cdots)$ standing for the unfinished part of the proof.


E, ¬E ⊢ F

```
1: E   premiss
2: ¬E  premiss
    ...
3: F
```

Initial proof window for $E, \neg E \vdash F$

The open-ended red rectangle around the conclusion, F, indicates that Jape has selected it as the goal for us to work on next.[3] The two premisses that are in scope are called the *context*. The goal is described as *open* because we have yet to complete its proof.

You may already have decided that the ¬-elim rule will eventually be used in the proof; but right now it can't be used because its conclusion is $\bot$, not F. But we *can* make progress by invoking the ⊥-elim rule from one of the the Rules menus.

This succeeds: in line with the idea that a proof rule can be used "backwards" to transform a partial proof tree, it has transformed our partial proof from $\boxed{\begin{array}{c}\vdots\\ F\end{array}}$ to $\boxed{\dfrac{\begin{array}{c}\vdots\\ \bot\end{array}}{F}\ \bot\text{ elim}}$

The partial display in the proof window changes to reflect this:

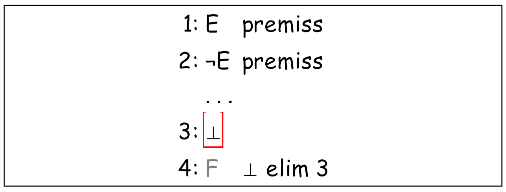
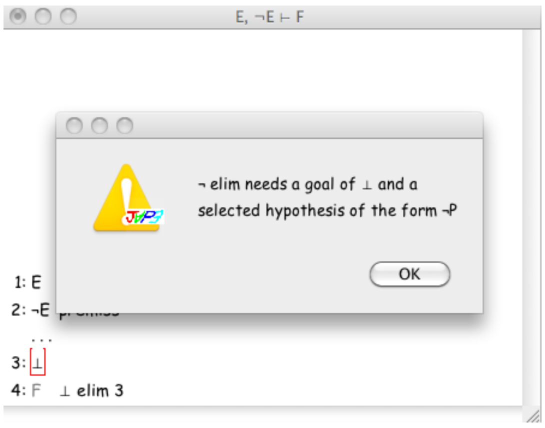

```
1: E   premiss
2: ¬E  premiss
    ...
3: ⊥
4: F   ⊥ elim 3
```

fig2

This shows that Jape has now selected $\perp$ as the goal and left the context the same.

The ellipsis between lines 2 and 3 signifies that the proof is still incomplete. The conditions are now right to invoke the ¬-elim rule, since the premisses and selected conclusion are just what the rule requires. Let's see what happens if we invoke it directly from a Rules menu:



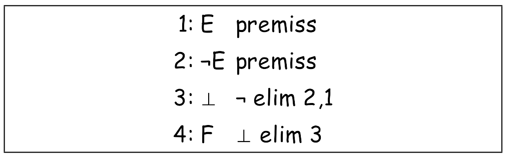
¬ elim needs a goal of ⊥ and a selected hypothesis of the form ¬P

```
1: E
2: ¬E  premiss
   . . .
3: ⊥
4: F   ⊥ elim 3
```

After invoking ¬-elim with no selection

What went wrong? We certainly chose the right rule, but interaction with IFP Jape has been designed so that if you are going to use that rule then you need to indicate which negation you want to eliminate.[4]

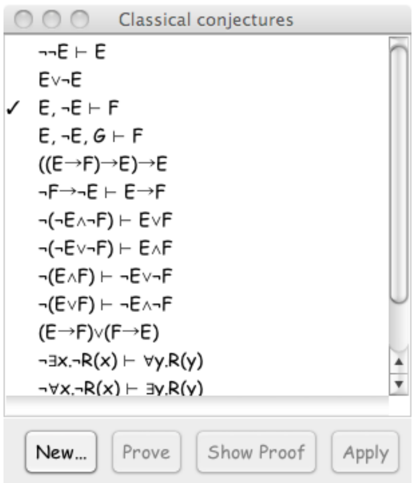
After acknowledging the error message by pressing its OK button you can invoke the rule successfully by selecting the $\neg E$ hypothesis (click on it); and then invoking the rule again. This is enough to complete the proof, and the display changes to:



```
1: E   premiss
2: ¬E  premiss
3: ⊥   ¬ elim 2,1
4: F   ⊥ elim 3
```

fig4

You can now record the proof by clicking the Done button on the Edit menu.[5] Once a theorem has been proven and recorded it can be used in the proofs of other theorems: this is signified by the $\surd$-mark beside it in the conjecture window.



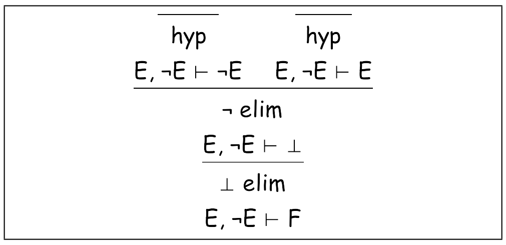
¬¬E ⊢ E
E∨¬E
✓ E, ¬E ⊢ F
E, ¬E, G ⊢ F
((E→F)→E)→E
¬F→¬E ⊢ E→F
¬(¬E∧¬F) ⊢ E∨F
¬(¬E∨¬F) ⊢ E∧F
¬(E∧F) ⊢ ¬E∨¬F
¬(E∨F) ⊢ ¬E∧¬F
(E→F)∨(F→E)
¬∃x.¬R(x) ⊢ ∀y.R(y)
¬∀x.¬R(x) ⊢ ∃y.R(y)

Proven conjectures with saved proofs have $\surd$-marks beside them

2.2 Jape's proof representation

In general you can display in a proof window the *sequent tree* that is Jape's underlying representation of a proof or partial proof. Just tick the View:Tree display radiobutton[6] to change the style of display in the proof window.

To see the sequent tree for the proof we just finished we first have to get it back into a proof window by selecting it in the conjecture window then pressing the Show Proof button. Changing to the tree display style yields the display:



```
   ───────      ───────
     hyp          hyp
  E, ¬E ⊢ ¬E   E, ¬E ⊢ E
  ──────────────────────
         ¬ elim
       E, ¬E ⊢ ⊥
       ─────────
         ⊥ elim
       E, ¬E ⊢ F
```

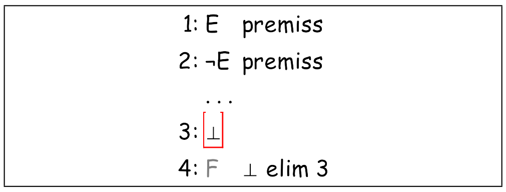
tree4

We explain sequent trees in detail in the second part of the course: here it is enough to know that the formulæ to the left of the ⊢ in each node are the hypotheses in scope for the proof of the formula to the right of the ⊢. In this proof they never changed from being the premisses of the theorem, because we never used a rule that introduced a new hypothetical (boxed) subproof.[7]

2.3 Proof by Pointing

Although some consider the invocation of proof rules from the menu to be an essential part of the discipline of learning about proofs, we prefer a less austere approach. To see what we mean let us first restart a proof of our original conjecture by selecting it in the conjecture window and pressing the Prove button again.[8]

Once again we take the proof as far as:

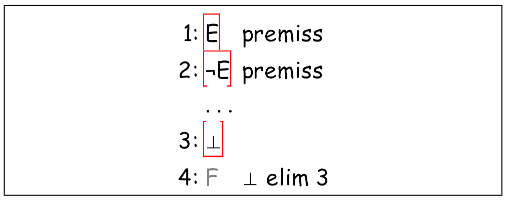


```
1: E   premiss
2: ¬E  premiss
   ...
3: ⊥
4: F   ⊥ elim 3
```

fig2

by invoking ⊥-elim from the menu.

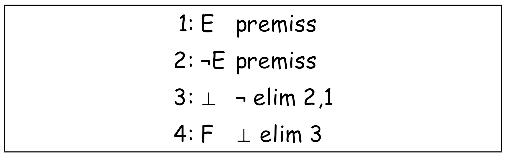
Now we select the premisses E and $\neg E$ by clicking on the respective formulæ (press the shift key while making the second selection – this tells Jape that we are selecting an *additional* premiss).



```
1: E   premiss
2: ¬E  premiss
   ...
3: ⊥
4: F   ⊥ elim 3
```

fig2s

Now we double-click on $\neg E$ and the elimination rule will be invoked, thereby closing the current goal, and with it the proof.

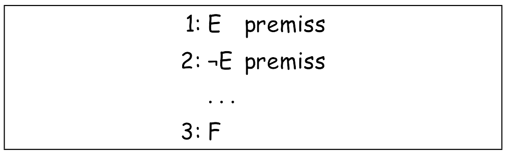


```
1: E   premiss
2: ¬E  premiss
3: ⊥   ¬ elim 2,1
4: F   ⊥ elim 3
```

fig4

IFP Jape isn't particularly intelligent, but tries to do the right thing when it can: and in this case it was obvious what the right thing was.

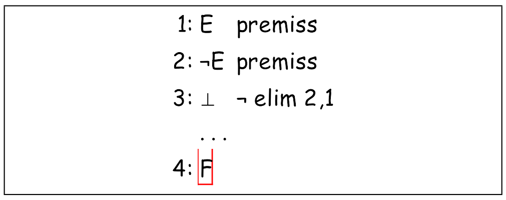
We can see a subtler demonstration of Jape doing the right thing by returning to the start of the proof (using Edit:Undo or its keyboard shortcut):



```
1: E   premiss
2: ¬E  premiss
    ...
3: F
```

fig1

Selecting F, E and $\neg E$ and double-clicking $\neg E$ now yields:



```
1: E   premiss
2: ¬E  premiss
3: ⊥   ¬ elim 2,1
    ...
4: F
```

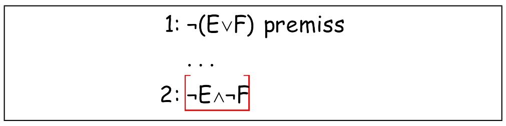
fig6

What happened here was that IFP Jape interaction has been designed to notice that when hypotheses of the form ϕ and $\neg\phi$ are selected from the context, and the goal is anything other than $\perp$, the proof can start by invoking $\neg$-elim, and then be closed with $\perp$-elim. The latter rule can be invoked by double-clicking on the $\perp$ or by invoking it from a Rules menu.[9]

In Appendix E we explain in detail what happens when a double click is made on a hypothesis or conclusion.

3 Proving $\neg(E \vee F) \vdash \neg E \wedge \neg F$

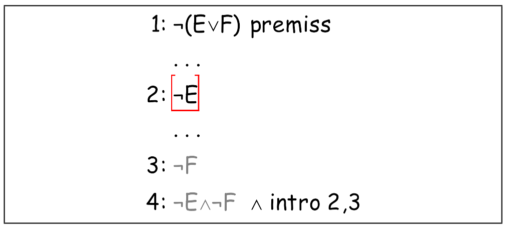
3.1 Getting Started



```
1: ¬(E∨F) premiss
    ...
2: ¬E∧¬F
```

p2fig1

The obvious first step here is with $\wedge$-intro, which can be invoked by double-clicking on the selected conclusion or from a Rules menu, yielding the display:

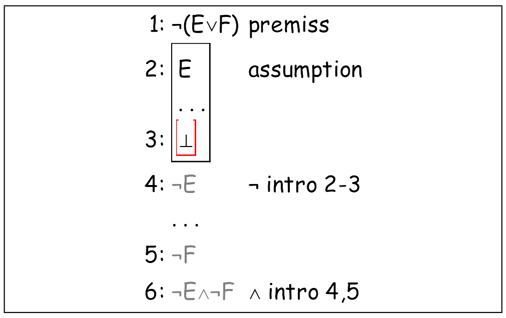


```
1: ¬(E∨F) premiss
    ...
2: ¬E
    ...
3: ¬F
4: ¬E∧¬F  ∧ intro 2,3
```

p2fig2

There are now two ellipses: this indicates that there are two subproofs to complete: one is the proof of the selected conclusion, $\neg E$; the second is the proof of the conclusion $\neg F$ – greyed-out because we are not (yet) working on it. If we wanted to work on it first then we would change the selection by clicking on $\neg F$.

Invoking ¬-intro transforms our goal into $\perp$, and introduces the assumption E inside a box that indicates its scope.

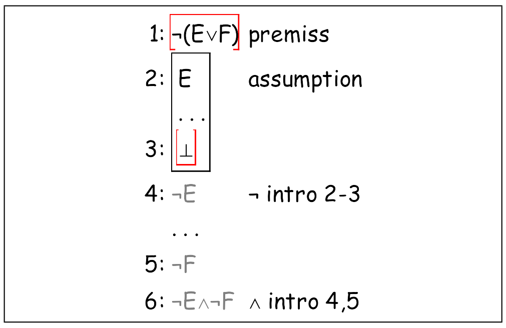


```
1: ¬(E∨F) premiss
2: E      assumption
   ...
3: ⊥
4: ¬E     ¬ intro 2-3
   ...
5: ¬F
6: ¬E∧¬F  ∧ intro 4,5
```

p2fig3

Now we notice that we could apply ¬-elim to generate the contradiction, *if we could establish* $E \vee F$.

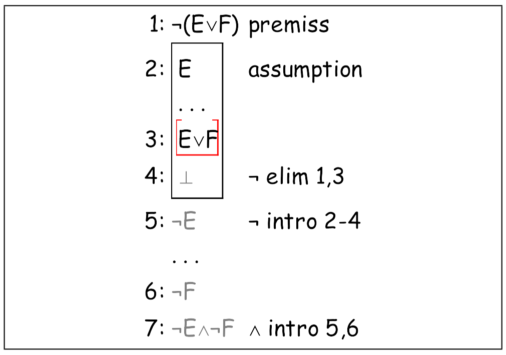
We set up a *backward* application of that rule by selecting the line 1 premiss and the goal:



```
1: ¬(E∨F) premiss
2: E      assumption
   ...
3: ⊥
4: ¬E     ¬ intro 2-3
   ...
5: ¬F
6: ¬E∧¬F  ∧ intro 4,5
```

p2fig4

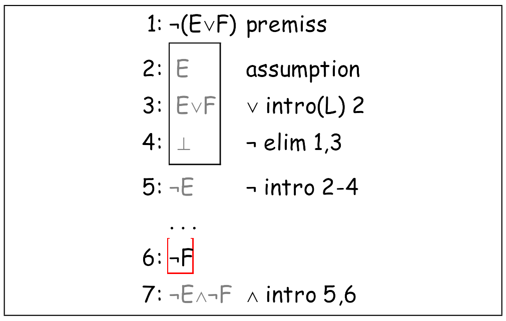
Then we invoke it from a Rules menu or by double-clicking on the selected premiss. This yields the proof display:



```
1: ¬(E∨F) premiss
2: E      assumption
   ...
3: E∨F
4: ⊥      ¬ elim 1,3
5: ¬E     ¬ intro 2-4
   ...
6: ¬F
7: ¬E∧¬F  ∧ intro 5,6
```

p2fig5

Now all that is needed to close this branch of the proof is to invoke ∨-intro(L). This can be done directly from a Rules menu, or by double-clicking on the goal.[10]

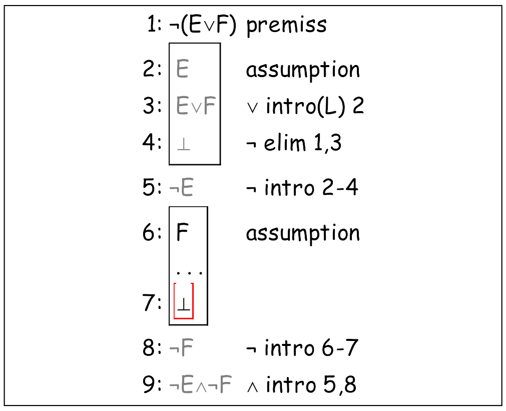


```
1: ¬(E∨F) premiss
2: | E      assumption
3: | E∨F    ∨ intro(L) 2
4: | ⊥      ¬ elim 1,3
5: ¬E       ¬ intro 2-4
   ...
6: ¬F
7: ¬E∧¬F   ∧ intro 5,6
```

p2fig6

Notice that the closed branch of the proof is greyed out. This is because the conclusion of the second, open, branch of the proof has been selected: it is intended to signify that nothing established by the first branch can be used in the second branch.

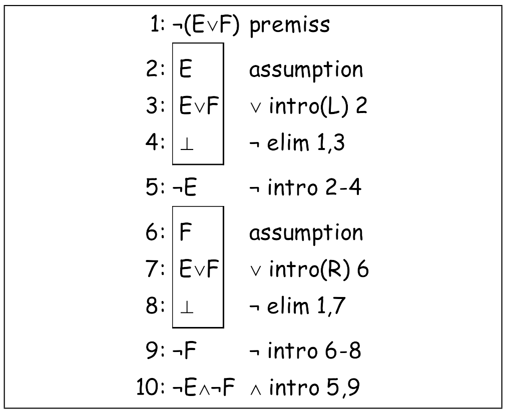
It is easy to see that the proof of the second branch can be completed in much the same way as the first. The first step in doing this is to invoke the ¬-intro rule, yielding:



```
1: ¬(E∨F) premiss
2: | E      assumption
3: | E∨F    ∨ intro(L) 2
4: | ⊥      ¬ elim 1,3
5: ¬E       ¬ intro 2-4
6: | F      assumption
   | ...
7: | ⊥
8: ¬F       ¬ intro 6-7
9: ¬E∧¬F   ∧ intro 5,8
```

p2fig6a

In fact the only difference in the sequence of rules used in the two subproofs is at the ∨-intro stage. The complete proof is:

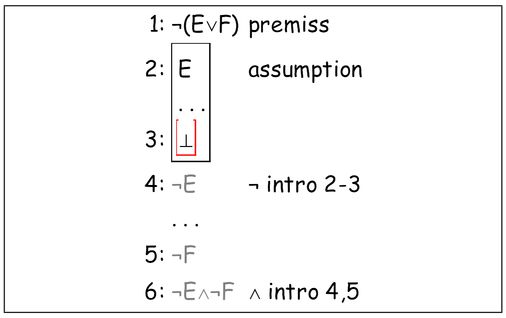


```
1: ¬(E∨F) premiss
2: | E      assumption
3: | E∨F    ∨ intro(L) 2
4: | ⊥      ¬ elim 1,3
5: ¬E       ¬ intro 2-4
6: | F      assumption
7: | E∨F    ∨ intro(R) 6
8: | ⊥      ¬ elim 1,7
9: ¬F       ¬ intro 6-8
10: ¬E∧¬F  ∧ intro 5,9
```

p2fig7

3.2 An alternative: forward use of ∨-intro

When the proof looked like:



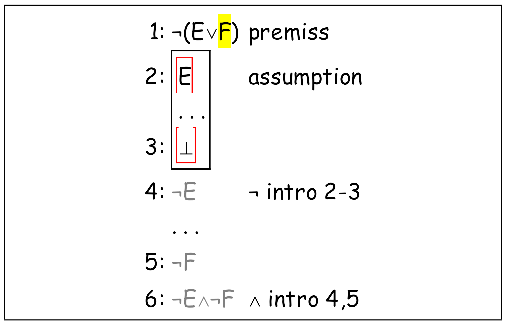
```
1: ¬(E∨F) premiss
2: E        assumption
   ...
3: ⊥
4: ¬E       ¬ intro 2-3
   ...
5: ¬F
6: ¬E∧¬F   ∧ intro 4,5
```

p2fig3

we noticed that we could apply ¬-elim to prove the contradiction, *if we could establish* $E \vee F$, and we saw that applying ¬-elim "backwards" established $E \vee F$ as an (easily closeable) goal because the hypothesis E was in its scope.

Next we will show how to invoke the two rules in opposite order: first "going forward" from E using ∨-intro(L) to establish $E \vee F$ in the scope of the goal $\perp$; then using the ¬-elim forwards, because both $E \vee F$ and its negation will be in the context.

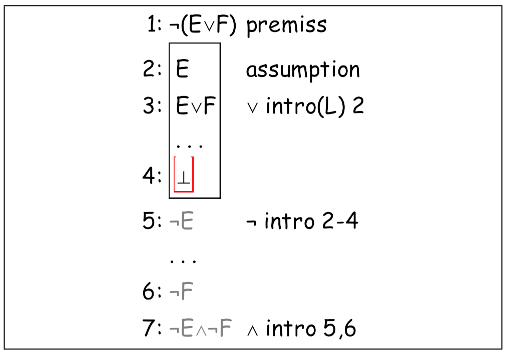
We select the hypothesis E, next we *text-select* F.[2] The text-selected formula is shown with a yellow background.



```
1: ¬(E∨F) premiss
2: E        assumption
   ...
3: ⊥
4: ¬E       ¬ intro 2-3
   ...
5: ¬F
6: ¬E∧¬F   ∧ intro 4,5
```

p2fig3ts

Now we invoke ∨-intro(L) (forward) from a Rules menu.[11] The proof transforms to:



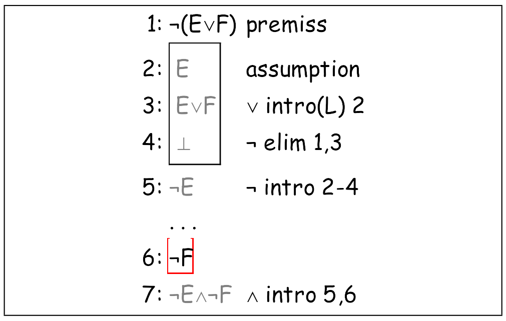
```
1: ¬(E∨F) premiss
2: E        assumption
3: E∨F      ∨ intro(L) 2
   ...
4: ⊥
5: ¬E       ¬ intro 2-4
   ...
6: ¬F
7: ¬E∧¬F   ∧ intro 5,6
```

p2fig4ts

What the boxed display is trying to show by placing the ellipsis below $E \vee F$ is that we have effectively added $E \vee F$ as an assumption to the context in which we are attempting to prove $\perp$. Selecting $E \vee F$ with a click makes this even clearer: for the open selection rectangle around it is shown with the opening below it.[12]

Closing this branch of the proof can now be done with the ¬-elim rule – which can be invoked from the menu or by double-clicking the negation.

[2]See Appendix F for operating-system-specific details of the mouse gestures used in conclusion-selection, hypothesis-selection, and text-selection.



```
1: ¬(E∨F) premiss
2:  E       assumption
3:  E∨F     ∨ intro(L) 2
4:  ⊥       ¬ elim 1,3
5: ¬E       ¬ intro 2-4
   ...
6: ¬F
7: ¬E∧¬F   ∧ intro 5,6
```

p2fig6

The box-style display of the resulting closed subproof is identical to the one we built the first time.[13] The remaining branch can be closed in the same way.

4 Exercises

At this point you might find it helpful to try proving the following conjectures from the introductory conjectures panel.

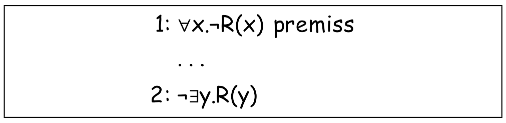
1. $\neg E \vee \neg F \vdash \neg(E \wedge F)$
2. $\neg E \wedge \neg F \vdash \neg(E \vee F)$
3. $E \wedge F \vdash \neg(\neg E \vee \neg F)$
4. $\neg F \rightarrow \neg E \vdash E \rightarrow F$
   Hint: start with the derived rule RAA.
5. $\vdash (\neg F \rightarrow \neg E) \rightarrow (E \rightarrow F)$
   Hint: use the previous theorem.

You may well now be able to complete the proofs of the introductory propositional conjectures. If you get reasonably far with them, then try the conjectures in the more "advanced" conjecture file, `IFP_conjectures.j`. This can be opened from the File: Open button, and it will add another conjectures window, labelled "Conjectures".[14]

5 Proving: $\forall x \cdot \neg R(x) \vdash \neg\exists y \cdot R(y)$

5.1 Straightforward approach

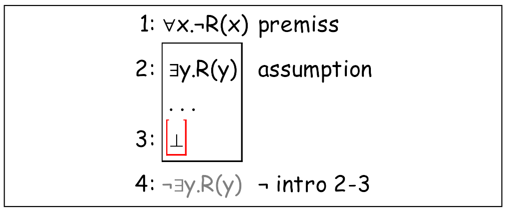
We start from



```
1: ∀x.¬R(x) premiss
   ...
2: ¬∃y.R(y)
```

p3fig1

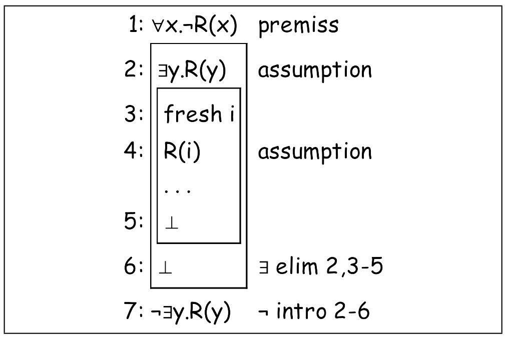
by invoking: ¬-intro (from the [Rules] menu, or by double-clicking on the negation), yielding:



```
1: ∀x.¬R(x)   premiss
2: ∃y.R(y)    assumption
   ...
3: ⊥
4: ¬∃y.R(y)   ¬ intro 2-3
```

p3fig2a

Now invoking ∃-elim with no text selection, causes IFP Jape to *invent* a variable with which to continue the proof. In this version of the Jape presentation of the logic the scope of the variable is made explicit by the boxed fresh i declaration on line 3.

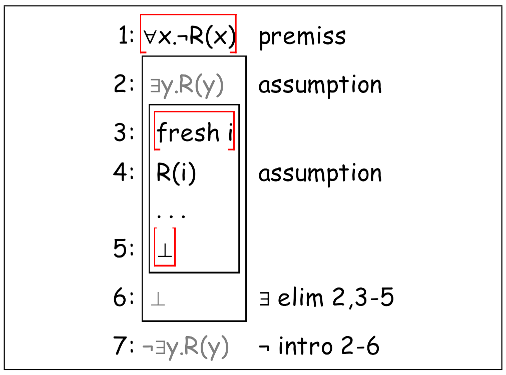


```
1: ∀x.¬R(x)   premiss
2: ∃y.R(y)    assumption
3: fresh i
4: R(i)       assumption
   ...
5: ⊥
6: ⊥          ∃ elim 2,3-5
7: ¬∃y.R(y)   ¬ intro 2-6
```

p3fig2b

At this stage Jape's highlighting of the in-scope hypotheses is very helpful. It is clear that we can make progress by using the first premiss to conclude $\neg R(i)$ and then using ¬-elim.

We now need to indicate the term that we want to substitute for x in the conclusion of the rule ∀-elim that we are now about to apply. We can do so by selecting the variable declaration on line 3. We also need to select the premiss on line 1, and to do that we need to make it an additional selection.[3]

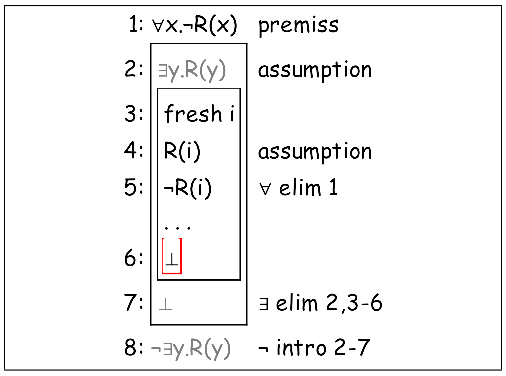


```
1: ∀x.¬R(x)   premiss
2: ∃y.R(y)    assumption
3: fresh i
4: R(i)       assumption
   ...
5: ⊥
6: ⊥          ∃ elim 2,3-5
7: ¬∃y.R(y)   ¬ intro 2-6
```

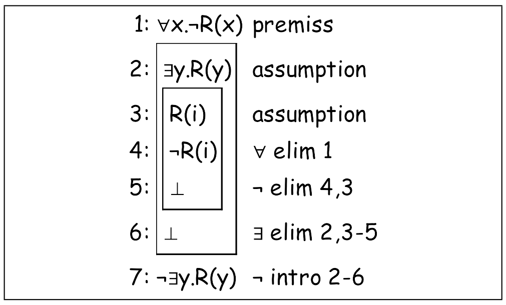
p3fig2b1

Applying the ∀-elim (forward) rule (by double-clicking on the ∀ hypothesis, or from the Rules menu) yields (if we didn't forget to indicate that we want to substitute the variable i for x):

[3]See appendix F for the details of gestures used to make selections, additional selections, and text selections. We could also text-select i so as to indicate the term to substitute for x.

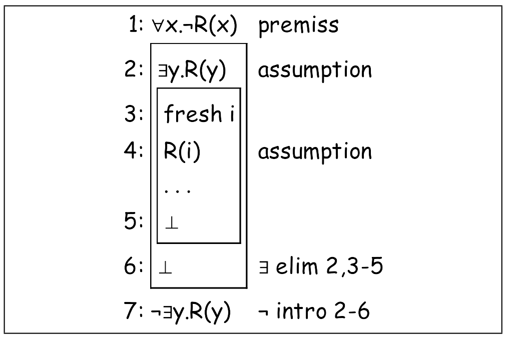

```
1: ∀x.¬R(x)    premiss
2: ∃y.R(y)     assumption
3: fresh i
4: R(i)        assumption
5: ¬R(i)       ∀ elim 1
   ...
6: ⊥
7: ⊥           ∃ elim 2,3-6
8: ¬∃y.R(y)    ¬ intro 2-7
```
p3fig2c

and an invocation of ¬-elim completes the proof.

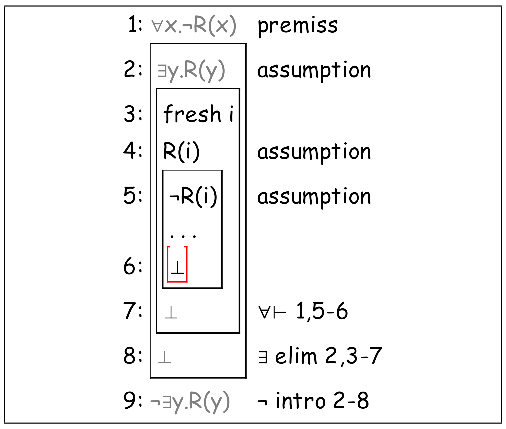

```
1: ∀x.¬R(x)  premiss
2: ∃y.R(y)   assumption
3: R(i)      assumption
4: ¬R(i)     ∀ elim 1
5: ⊥         ¬ elim 4,3
6: ⊥         ∃ elim 2,3-5
7: ¬∃y.R(y)  ¬ intro 2-6
```
p3fig2d

5.2 Using the derived rule ∀⊢

Now let us rewind the proof to the point where we applied ∀-elim, namely:


```
1: ∀x.¬R(x)    premiss
2: ∃y.R(y)     assumption
3: fresh i
4: R(i)        assumption
   ...
5: ⊥
6: ⊥           ∃ elim 2,3-5
7: ¬∃y.R(y)    ¬ intro 2-6
```
p3fig2b

and see what happens if we use the *derived* rule ∀ ⊢ with the same hypothesis, and i text-selected.[4]


```
1: ∀x.¬R(x)    premiss
2: ∃y.R(y)     assumption
3: fresh i
4: R(i)        assumption
5: ¬R(i)       assumption
   ...
6: ⊥
7: ⊥           ∀⊢ 1,5-6
8: ⊥           ∃ elim 2,3-7
9: ¬∃y.R(y)    ¬ intro 2-8
```
p3fig2cx

[4]If you forget to text-select the i then a proof variable will be invented: these are discussed in Appendix A.

Notice that the new hypothesis $\neg R(i)$ introduced into the context is shown within a nested box. The proof can be completed in the same way as before.

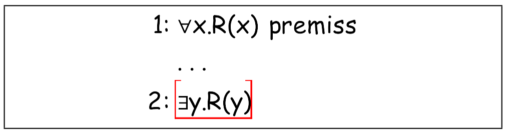
A Proof Variables

Under the hood Jape is equipped with a (very) powerful pattern matching engine[15] that allows rules to be applied "optimistically" (by deferring some decisions) in contexts which don't supply enough information. The machinery makes use of *proof variables* – placeholders for formulæ, terms, and variables that we know we will have to invent, but which we haven't yet invented.

Proof variables have names starting with an underscore: for example _P, _T, _v.

A.1 Example 1: attempting $\forall x \cdot R(x) \vdash \exists y \cdot R(y)$

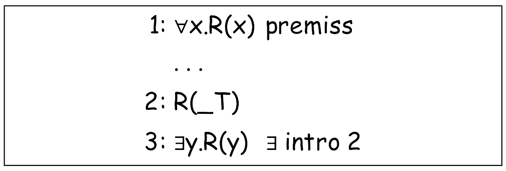
In starting the proof



```
1: ∀x.R(x) premiss
   ...
2: ∃y.R(y)
```

proofvars1

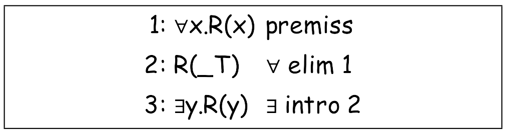
we are likely to want to use ∃-intro first. If we haven't yet thought of a term T for which $R(T)$ holds, we can still apply the rule, leaving it to Jape to invent a term proof variable for us. Here's the outcome: the proof variable is _T.



```
1: ∀x.R(x) premiss
   ...
2: R(_T)
3: ∃y.R(y)  ∃ intro 2
```

proofvars2

Now the obvious thing to do next is to use ∀-elim(forward) to close the proof by showing $R($_T$)$. Here's what happens if we use the rule with text-selection _T:[16]

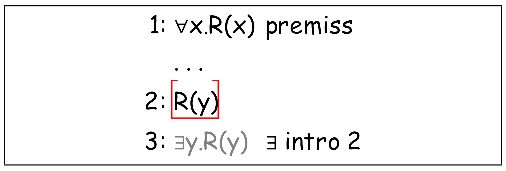


```
1: ∀x.R(x) premiss
2: R(_T)    ∀ elim 1
3: ∃y.R(y)  ∃ intro 2
```

proofvars4

What has been shown by this "proof"? That *by inventing a term to bind to* _T we could have a proper proof.[17]

We could have avoided the appearance of this proof variable by making a text selection just before invoking the ∃-intro rule; and again just before invoking ∀-elim. Here's what happens if we text-select y just before invoking ∃-intro.

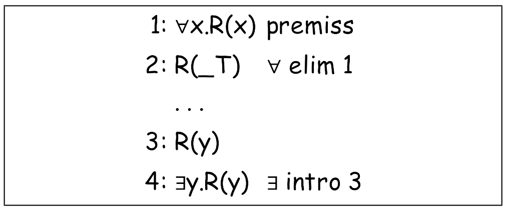


```
1: ∀x.R(x) premiss
   ...
2: R(y)
3: ∃y.R(y)  ∃ intro 2
```

proofvars5

Text-selecting y and then invoking ∀-elim(forward) again completes the proof.

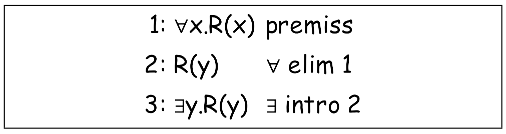
Forgetting to text-select y before invoking ∀-elim, and then letting Jape invent a proof variable, leads to:



```
1: ∀x.R(x) premiss
2: R(_T)    ∀ elim 1
   ...
3: R(y)
4: ∃y.R(y)  ∃ intro 3
```

proofvars6

The proof variable is still unbound; but we can bind it by invoking the hyp rule. Jape recognises that for the rule to succeed $R(\_T)$ and $R(y)$ would have to be the same formula. Since this can be forced to happen by binding y to _T, it does that forthwith, and the hyp rule closes the proof.



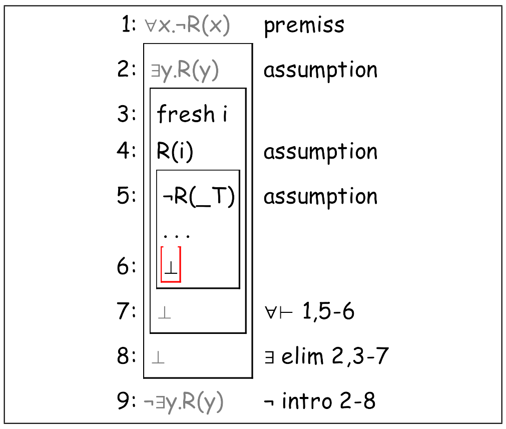
```
1: ∀x.R(x) premiss
2: R(y)     ∀ elim 1
3: ∃y.R(y)  ∃ intro 2
```

proofvars7

Remark: This theorem makes semantic sense only in a model with a nonempty universe. This is not problematic: recall that the IFP proof rules as a whole are sound (only) for nonempty universes.

A.2 Example 2: $\forall x \cdot \neg R(x) \vdash \neg\exists y \cdot R(y)$ again

Turning back to section 5.2, let us suppose we had forgotten to text-select i when we applied $\forall \vdash$. In that case the display would have changed to:



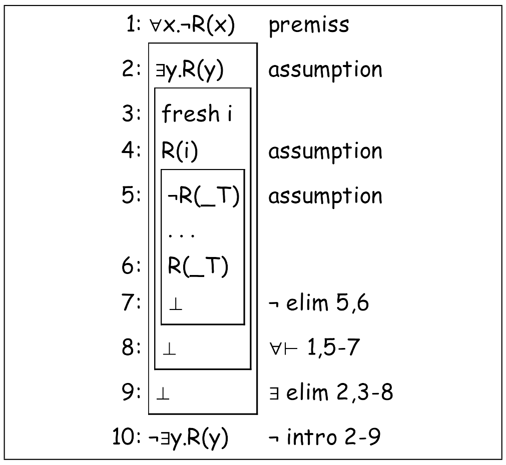
```
1: ∀x.¬R(x)      premiss
2: ∃y.R(y)       assumption
3: fresh i
4: R(i)          assumption
5: ¬R(_T)        assumption
   ...
6: ⊥
7: ⊥             ∀⊢ 1,5-6
8: ⊥             ∃ elim 2,3-7
9: ¬∃y.R(y)      ¬ intro 2-8
```

p3fig2cvar

IFP Jape has used a *proof variable* _T to stand for a term that we will, sooner-or-later, have to produce.

If we invoke ¬-elim after selecting the two hypotheses $R(i)$ and $\neg R(\_T)$ Jape recognises that for the rule to succeed $R(i)$ and $R(\_T)$ will have to be the same formula, then binds i to _T before transforming the proof tree. The proof is again complete.

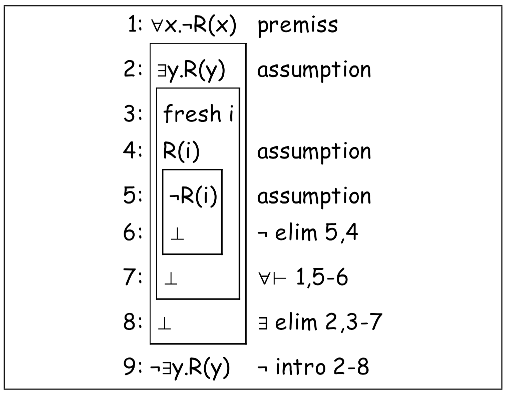
Returning to the use of ¬-elim, we recall that it can be invoked in several ways. If we had forgotten to select $R(i)$ before invoking it, then Jape would have assumed that we wanted to delay deciding on a binding for the proof variable, and transformed the proof to:



```
1: ∀x.¬R(x)    premiss
2: ∃y.R(y)     assumption
3: fresh i
4: R(i)        assumption
5: ¬R(_T)      assumption
   ...
6: R(_T)
7: ⊥           ¬ elim 5,6
8: ⊥           ∀⊢ 1,5-7
9: ⊥           ∃ elim 2,3-8
10: ¬∃y.R(y)   ¬ intro 2-9
```

p3fig2cvara

Invoking hyp at this point causes Jape to recognise that for the rule to succeed $R(i)$ and $R(\_\mathsf{T})$ will have to be the same formula. It makes the same binding as before, and the proof is, again, complete.[18]

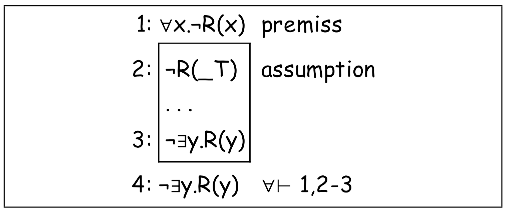


```
1: ∀x.¬R(x)   premiss
2: ∃y.R(y)    assumption
3: fresh i
4: R(i)       assumption
5: ¬R(i)      assumption
6: ⊥          ¬ elim 5,4
7: ⊥          ∀⊢ 1,5-6
8: ⊥          ∃ elim 2,3-7
9: ¬∃y.R(y)   ¬ intro 2-8
```

p3derived

A.3 Provisos

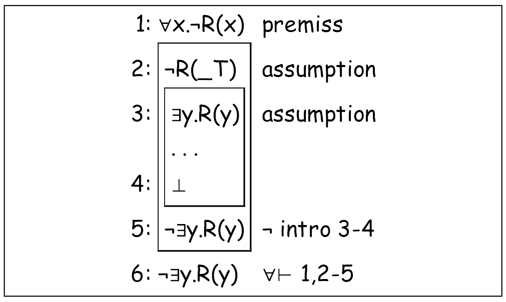
Now let us try to find an essentially different proof for our conjecture. We first invoke $\forall\vdash$ *without a text-selection*. The proof is transformed to one with an unbound proof variable, _T, in it:



```
1: ∀x.¬R(x)   premiss
2: ¬R(_T)     assumption
   ...
3: ¬∃y.R(y)
4: ¬∃y.R(y)   ∀⊢ 1,2-3
```

p3fig2

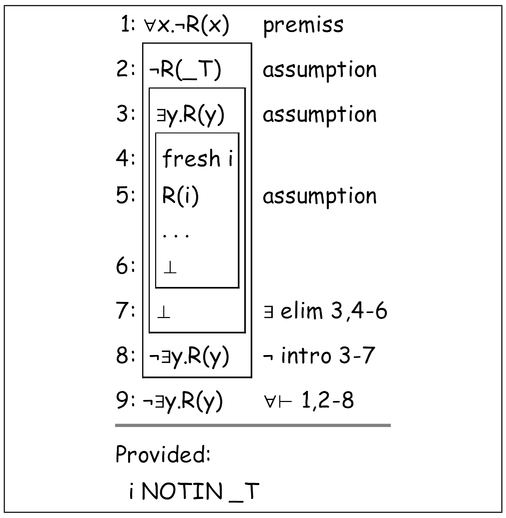
An obvious way of proceeding is to invoke ¬-intro (for the negation on line 3) yielding:



```
1: ∀x.¬R(x)   premiss
2: ¬R(_T)     assumption
3: ∃y.R(y)    assumption
   ...
4: ⊥
5: ¬∃y.R(y)   ¬ intro 3-4
6: ¬∃y.R(y)   ∀⊢ 1,2-5
```

p3fig3

The obvious next step is to invoke ∃-elim (for the ∃ hypothesis). We will let Jape choose the variable used in the hypothetical subproof. The resulting display shows that Jape has not just transformed the proof tree: it has also added a *proviso*.



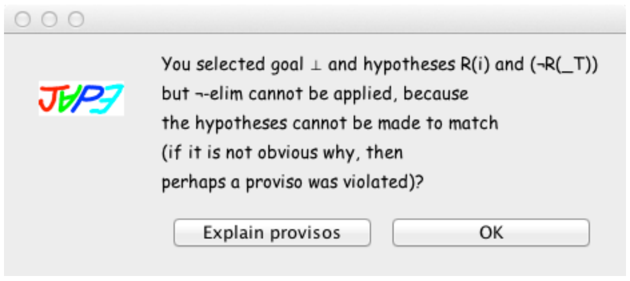
```
1: ∀x.¬R(x)      premiss
2: ¬R(_T)        assumption
3: ∃y.R(y)       assumption
4: fresh i
5: R(i)          assumption
   ...
6: ⊥
7: ⊥             ∃ elim 3,4-6
8: ¬∃y.R(y)      ¬ intro 3-7
9: ¬∃y.R(y)      ∀⊢ 1,2-8

Provided:
 i NOTIN _T
```

p3fig4

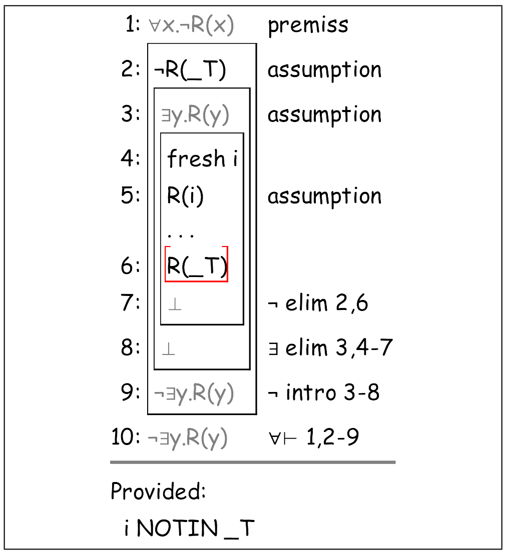
This proviso corresponds, concretely, to the side condition "v is fresh" of ∃-elim.

The proof cannot be completed from here. The obvious way to try is to invoke ¬-elim – expecting Jape to bind i to _T. But here is how IFP Jape responds if you invoke it by selecting the assumptions on lines 2 and 5 and double clicking on the negated one.



```
You selected goal ⊥ and hypotheses R(i) and (¬R(_T))
but ¬-elim cannot be applied, because
the hypotheses cannot be made to match
(if it is not obvious why, then
perhaps a proviso was violated)?
```

If, on the other hand, you double-click on the assumption $\neg R(\_(T)$ on line 2 without selecting $R(i)$ on line 4, the display changes to



```
1: ∀x.¬R(x)      premiss
2: ¬R(_T)        assumption
3: ∃y.R(y)       assumption
4: fresh i
5: R(i)          assumption
   ...
6: R(_T)
7: ⊥             ¬ elim 2,6
8: ⊥             ∃ elim 3,4-7
9: ¬∃y.R(y)      ¬ intro 3-8
10: ¬∃y.R(y)     ∀⊢ 1,2-9

Provided:
 i NOTIN _T
```

p3fig5

The proviso remains, and invoking hyp will fail: try it!

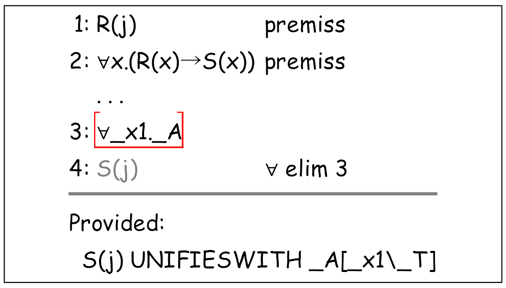
The problem here is not that Jape chose an inconvenient variable (i) while applying the ∃-elim rule: but that there is *no* choice of variable that will satisfy the proviso. A proof rooted at $\forall\vdash$ simply cannot succeed; and I hope you will not be surprised to find that neither can a proof rooted at ∀-elim.

A.4 Here be Dragons!

The optimistic application of rules by inventing proof variables can surprise the inexperienced Japeist. Here, for example, is the state of a proof of:

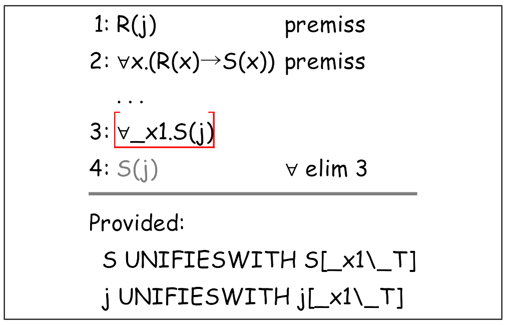
$$R(j), \forall x \bullet (R(x) \to S(x)) \vdash S(j)$$

after invoking $\dfrac{\Gamma \vdash \forall x \bullet \phi(x)}{\Gamma \vdash \phi(T)}$ ∀-elim


```
1: R(j)              premiss
2: ∀x.(R(x)→S(x))    premiss
   ...
3: ∀_x1._A
4: S(j)              ∀ elim 3
Provided:
 S(j) UNIFIESWITH _A[_x1\_T]
```
forallvar1

The attempt to match the consequent of the rule with the current proof was successful, matching Γ directly with $R(j), \forall x \bullet (R(x) \to S(x))$, and inventing proof variables _A, _T, and _x1 to be constrained by the given proviso. Since that proviso cannot be solved deterministically without further information being provided, Jape leaves the variables in the proof. It can be tempting to try to solve the constraints by hand (using the Edit: Unify button described in note 17), but this is not recommended, and can lead the inexperienced into a blind alley.

For example: here's the consequence of binding _A to $S(j)$ by hand.


```
1: R(j)              premiss
2: ∀x.(R(x)→S(x))    premiss
   ...
3: ∀_x1.S(j)
4: S(j)              ∀ elim 3
Provided:
 S UNIFIESWITH S[_x1\_T]
 j UNIFIESWITH j[_x1\_T]
```
forallvar3

The original constraint has been replaced by two others. Further hand-solving can eliminate the constraints completely; but at best (binding _x1 to a non-j variable, y say) line 3 will be transformed into $\forall y \bullet S(j)$ that is no more provable in the context than the original goal $S(j)$. At worst (binding _x1 to j) line 3 becomes $\forall j \bullet S(j)$ – which (take our word for it) is not provable in the given context.

A.5 Exercise

Prove $\exists x \bullet \forall y \bullet R(x, y) \vdash \forall y \bullet \exists x \bullet R(x, y)$ without text-selecting.

B Resources

Good background texts on logic and proof are *Proof and Disproof in Formal Logic* (Richard Bornat, OUP, 2005); and *Logic in Computer Science* (Huth and Ryan, CUP, 2008).

Richard Bornat's book is the natural place to go if you want to explore formal proofs in logics other than propositional and first-order logic, and (especially) if you want to understand proofs of program correctness in "Hoare Logic". It also contains an excellent introduction to the idea of "Disproof" and an account of how Japecan be used as a disproof-calculator.

The proof rules we use for "Introduction to Formal Proof" are essentially identical to those in one of the (unnamed) systems explained in Huth and Ryan.

C Acknowledgement

Richard Bornat and I first designed and implemented Jape as a joint endeavour starting in 1991; but Richard has been responsible for the evolution of the design and its implementation since just before the turn of the millenium. Since then I have been little more than a challenging logic designer – occasionally fixing bugs that Richard can't be bothered with.

Despite the fact that its development has all-but stopped Richard and I are both sufficiently committed to the ideas Jape embodies that we continue to maintain and distribute it.

D IFP Logic

D.1 Natural Deduction rules for the logical connectives

$$\frac{\phi \qquad \psi}{\phi \wedge \psi}\ \wedge\text{-intro} \qquad \frac{\phi \wedge \psi}{\phi}\ \wedge\text{-elim-L} \qquad \frac{\phi \wedge \psi}{\psi}\ \wedge\text{-elim-R}$$

$$\frac{\phi}{\phi \vee \psi}\ \vee\text{-intro-L} \qquad \frac{\psi}{\phi \vee \psi}\ \vee\text{-intro-R} \qquad \frac{(\phi \vee \psi) \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \kappa\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ \kappa\end{array}}}{\kappa}\ \vee\text{-elim}$$

$$\frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \to \psi}\ \to\text{-intro} \qquad \frac{\phi \quad \phi \to \psi}{\psi}\ \to\text{-elim}$$

$$\frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\text{-intro} \qquad \frac{\phi \quad \neg\phi}{\bot}\ \neg\text{-elim} \qquad \frac{\neg\neg\phi}{\phi}\ \neg\neg\text{-elim}$$

$$\frac{\bot}{\phi}\ \bot\text{-elim}$$

$$\frac{\phi \to \psi \quad \psi \to \phi}{\phi \leftrightarrow \psi}\ \leftrightarrow\text{-intro} \qquad \frac{\phi \leftrightarrow \psi}{\phi \to \psi}\ \leftrightarrow\text{-elim-R} \qquad \frac{\phi \leftrightarrow \psi}{\psi \to \phi}\ \leftrightarrow\text{-elim-L}$$

D.2 Sequent rules for the quantifiers

$$\frac{\Gamma \vdash \forall x \bullet \phi(x)}{\Gamma \vdash \phi(T)}\ \forall\text{-elim}\ (T \text{ free for } x \text{ in } \phi(x)) \qquad \frac{\Gamma \vdash \phi(v)}{\Gamma \vdash \forall x \bullet \phi(x)}\ \forall\text{-intro}\ \ (v \text{ is fresh})$$

$$\frac{\Gamma \vdash \phi(T)}{\Gamma \vdash \exists x \bullet \phi(x)}\ \exists\text{-intro}\ (T \text{ free for } x \text{ in } \phi(x)) \qquad \frac{\Gamma, \phi(v) \vdash \kappa}{\Gamma, \exists x \bullet \phi(x) \vdash \kappa}\ \exists\text{-elim}\ \ (v \text{ is fresh})$$

D.3 Structural Rules

$$\frac{}{\Gamma,\ \phi\ \vdash\ \phi}\ \text{hyp} \qquad \frac{\Gamma\ \vdash\ \phi \qquad \Gamma,\ \phi\ \vdash\ \psi}{\Gamma\ \vdash\ \psi}\ \text{cut} \qquad \frac{\Gamma,\ \phi,\ \phi\ \vdash\ \psi}{\Gamma,\ \phi\ \vdash\ \psi}\ \text{thin}$$

$$\frac{\phi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\psi}\ \text{cut}$$

D.4 Some Derived Rules

$$\frac{\Gamma,\ \neg\phi\ \vdash\ \bot}{\Gamma\ \vdash\ \phi}\ \textsf{RAA}$$

$$\frac{\Gamma, \neg\phi \vdash \phi}{\Gamma, \neg\phi \vdash\ \kappa}\ \neg\vdash$$

$$\frac{\Gamma,\ \phi,\ \psi\ \vdash\ \kappa}{\Gamma,\ \phi\wedge\psi\ \vdash\ \kappa}\ \wedge\vdash$$

$$\frac{\Gamma,\ \phi(T)\ \vdash\ \kappa}{\Gamma,\ \forall x \bullet \phi(x)\ \vdash\ \kappa}\ \forall\vdash \quad (T \text{ free for } x \text{ in } \phi(x))$$

E Proof by pointing: the meaning of a double-click

IFP Jape tries to do the right thing in response to a double-click on a conclusion formula: it will usually apply the appropriate introduction rule for the topmost connective of the formula. The only special cases are:

$\Phi \vee \Psi$	Attempt to close the current goal by $\vee$-elim(L) or $\vee$-elim(R) if one of them is immediately applicable.
$\forall x \bullet \Phi$	Invoke $\forall$-intro, inventing a variable if none has been text-selected.
$\exists x \bullet \Phi$	Invoke $\exists$-intro, inventing a term if none has been text-selected.

In response to double-clicks on a hypothesis formula, IFP Jape will decide what to do based on the topmost connective of the formula:

$\bot$	Invoke $\bot$-elim
$\Phi \wedge \Psi$	Attempt to close the current goal by $\wedge$-elim(L) or $\wedge$-elim(R) if one of them is immediately applicable; otherwise introduce both Φ and Ψ into the context.
$\Phi \rightarrow \Psi$	Attempt to introduce Ψ into the context.
$\Phi \vee \Psi$	Invoke $\vee$-elim.
$\neg\neg\Phi$	Invoke $\neg\neg$-elim if the goal is Φ; otherwise, if the goal is (say) Ψ then cut the current proof tree with Φ and invoke the rule, leaving Ψ to be established from Φ.
$\neg\Phi$	Invoke $\neg$-elim if the selected conclusion is $\bot$.
$\neg\Phi$	If hypothesis Φ is also selected, and the conclusion κ is not $\bot$ then (as described in Section 2.3) cut the current proof tree with $\bot$, leaving the invocation of $\bot$-elim to be done by hand.
$\forall x \bullet \Phi(x)$	Invoke $\forall$-elim, inventing a term if none has been text-selected.
$\exists x \bullet \Phi(x)$	Invoke $\exists$-elim, inventing a variable if none has been text-selected.

F Commonly used Mouse Gestures and Keyboard Shortcuts

Linux and Windows	
Hypothesis selection	click on the hypothesis with Button-1
Additional Hypothesis selection	CTRL-click on the hypothesis with Button-1
Conclusion selection	click on the conclusion with Button-1
Remove formula selections	click off-formula with Button-1
Text selection	Click with Button-2 on the main connective of the formula or subformula to be text-selected.
Extend text selection	SHIFT-drag with Button-2.
Remove text selections	click off-text with Button-2
Undo / Re-do	CTRL-Z / SHIFT-CTRL-Z
Preserve finished proof	CTRL-D

OS/X	
Hypothesis selection	click on the hypothesis with Button-1
Additional Hypothesis selection	CMD-click on the hypothesis with Button 1
Conclusion selection	click on the conclusion with Button-1
Remove formula selections	click off-formula with Butto- 1
Text selection	Click with Button-2 or ALT-Button-1 on the main connective of the formula or subformula to be text-selected.
Extend text selection	SHIFT-drag with Button-2 or ALT-Button-1.
Remove text selections	click off-text with Button-2 or ALT-Button-1
Undo / Re-do	CMD-Z / SHIFT-CMD-Z
Preserve finished proof	CMD-D

Notes

[1] A Windows (Win32) version is usually available at the same site, though it usually lags behind the other two.

[2] In a some installations you can click on the `IFP.jt` file to start Jape. This can also be done in Windows, and in an OS/X installation in which the file extension `.jt` has been associated with Jape.

You can import the IFP configuration directly from the command line in OS/X, with

`open -a jape` *path-to-theory/*`IFP.jt` (OS/X)

and in a normal Linux installation with the shell command

`$HOME/bin/Jape` *path-to-theory/*`IFP.jt` (Linux)

[3] At this point in this proof there is little choice about what to do next, because there is only a single unproved conclusion.

[4] Yes, Virginia, there is indeed only one negation present in the proof and an "intelligent" tool would have spotted that – but IFP Jape users are learning about formal proofs, so we designed the interaction to eschew intelligence.

[5] This can also be done by typing the keyboard shortcut shown beside the menu button.

[6] *i.e.* the $\boxed{\text{Tree display}}$ radiobutton in the $\boxed{\text{View}}$ menu.

[7] You might wonder why the uses of the hyp rule are not shown in the box display of the proof: this is because they add lines to the presentation that are just duplicates of the premiss lines.

You can see this for yourself by deselecting the $\boxed{\text{View:Hide duplicated hyp lines in box display}}$ checkbox.

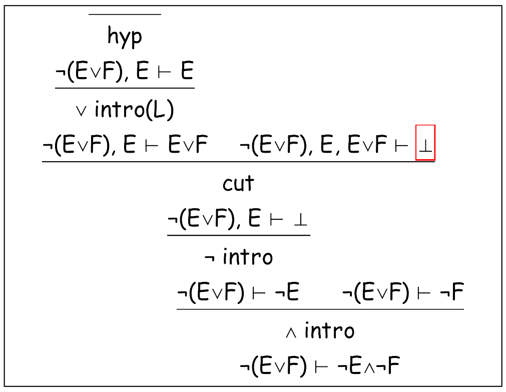
[8] Jape is likely to want us to agree that we really want to make another proof and that we don't mind an additional window with the same proof in it.

[9] A glance at the sequent tree representation of the resulting proof will show that in order to preserve the illusion that we have used the elimination rule to reason forwards from E and $\neg E$ we have quietly used the cut rule described in the second part of the course. Richard Bornat went a long way further to preserve the illusion of forward proof: the details are given in the Jape documentation.

[10] When an open $\vee$ conclusion is double-clicked IFP Jape sees whether it can close it from a hypothesis and one of the the $\vee$ introduction rules.

[11] The hypothesis and the text-selection were both necessary because we needed to tell IFP Jape about both the left-, and right-subformulæ of the $\vee$ formula we wanted to introduce.

[12] A glance at the sequent tree display shows this more clearly:



p2fig4tstree

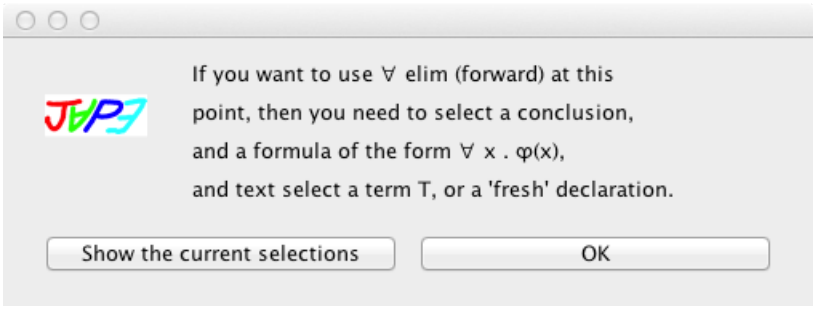
Here IFP Jape shows that the forward use of the elimination rule was simulated by using the cut rule (which is treated in detail in the course) to join the obvious proof of $\neg(E \vee F), E \vdash E \vee F$, to the proof of $\neg(E \vee F), E \vee F \vdash \bot$ that we must finish. In IFP Jape, the Box display style normally hides applications of the cut rule: presenting only its right branch.

[13] The previous note makes it clear why the underlying sequent tree representation is different.

[14] If you are really intrepid you should try working on a few derived proof rules by opening the file `IFP_derived.j`. Finishing most proofs will require a double-click on a "given" (an antecedent of the derived rule listed in the "Givens" list below the proof).

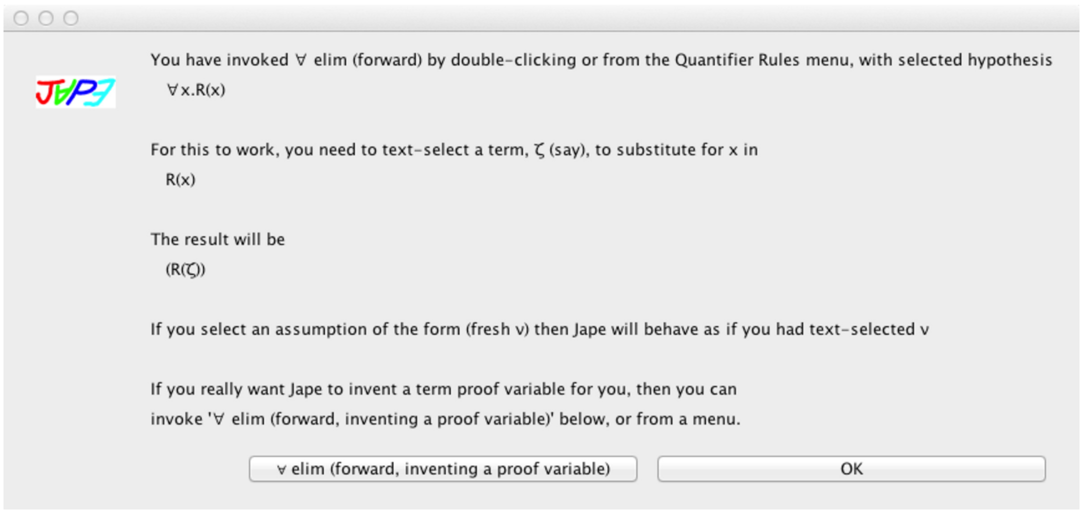
[15] Actually it is a *unification engine*

[16] Using the rule directly from the menu with no text-selection and without selecting the premiss yields the warning



If you want to use ∀ elim (forward) at this point, then you need to select a conclusion, and a formula of the form ∀ x . φ(x), and text select a term T, or a 'fresh' declaration.

Using it by double-clicking on the premiss (or from the menu with no text selection and the premiss selected) yields the warning



You have invoked ∀ elim (forward) by double-clicking or from the Quantifier Rules menu, with selected hypothesis
∀x.R(x)

For this to work, you need to text-select a term, ζ (say), to substitute for x in
R(x)

The result will be
(R(ζ))

If you select an assumption of the form (fresh v) then Jape will behave as if you had text-selected v

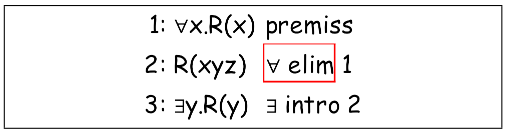
If you really want Jape to invent a term proof variable for you, then you can invoke '∀ elim (forward, inventing a proof variable)' below, or from a menu.

In this case you may be able to see how make progress by permitting Jape to invent a variable for you, and then invoking the hypothesis rule.

[17] The binding of a term to the proof variable can be done straight away *without restarting the proof*: for example by using the Edit: Unify button after text-selecting both the variable x and the proof variable _T.

If we find text-selection clumsy at this point, or if we don't see a term in the display that we'd like to text-select from, then it can also be done by invoking Edit: Apply tactic, and typing `UNIFY` _T *term* (for the appropriate term).

Direct use of Unify or of Apply tactic are *not the preferred way of proceeding*; but here's the proof completed using variable xyz as the term.



1: ∀x.R(x) premiss
2: R(xyz) ∀ elim 1
3: ∃y.R(y) ∃ intro 2

proofvars4u

[18] Notice the additional layer of box.