Hoare Logic in JAPE

Robert Keller

8 October 2007

Natural deduction is only one type of proof that can be constructed in JAPE. Hoare logic is another example, and several interesting tutorial programs have been proved totally correct using this feature. Open theory hoare.jt from the theory menu. Five windows open:

- Variable programs
- Comparison
- Useful Lemmas
- Array programs
- Indexing

For now we will be concerned with the first three. In the Variable Programs menu are several programs that can be proved. Click the first:

{i = 2} (i := i+1) {i = 3}

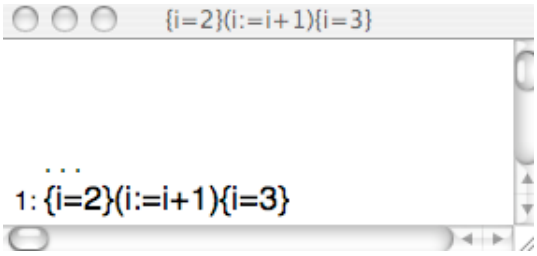
You will recognize this as almost being an instance of the assignment rule:

{ Q[x/E] } x := E { Q }

where Q[x/E] means Q with any free occurrences of x replaced with E. View the menu bar. In addition to Backward and Forward options we had for Natural Deduction, we also have **Programs** and **Extras**, which give us proof rules that will be needed for programs.

This addition doesn't have a full set of axioms for the integers, etc. So the idea is to work the unproved parts down to "obvious" assertions, then use the **obviously** option from the **Extras** menu to indicate that any further proof of that assertion needs to be done out of line. A good way to do this might be to put all uses of obviously into the Useful Lemmas window, rather than in the program proof.

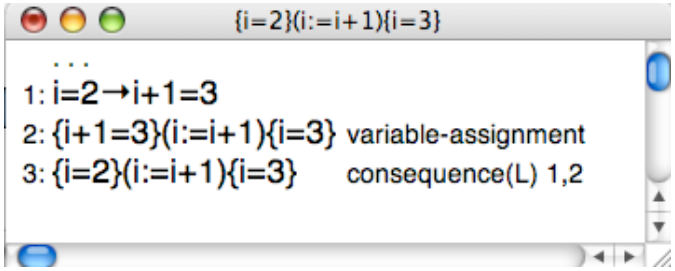
Here is the window for this program before the proof is begun.



The available rules in the Programs menu are as follows:

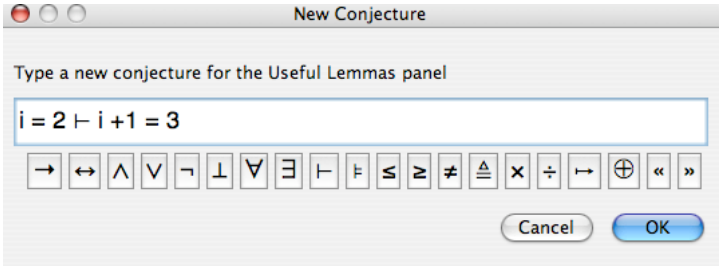
skip	The rule for a skip ("no op") statement. Skip is sometimes needed to provide a one-branch conditional or a loop with an empty body.
tilt	
sequence	Deals with 2 or more statements in a sequence.
Ntuple	Deals with a sequence of statements with assertions interposed between statements. This is needed for dealing with the invariants required for **while** statements.
variable-assignment	
array-element-assignment	
choice	Deals with conditional (**if** … **then** … **else** … **fi**) statements.
while	Deals with iterative (**while** … **do** … **od**) statements.
consequence(L)	Applies logical consequence on the left-hand side of the statement.
consequence(R)	Applies logical consequence on the right-hand side of the statement.

For the first example, we use the **variable-assignment** rule, giving:

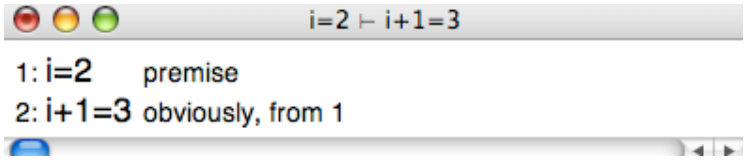


Note that JAPE has automatically introduced a consequence, indicating that if we want to use the variable-assignment rule for this conclusion, we will have no option but to prove the logical assertion on line numbered 1. This assertion is pure logic, not a programming language statement.

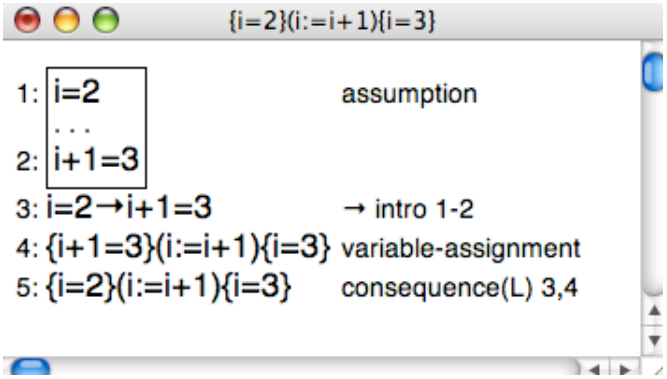
While we could justify this statement using **obviously**, selected from the **Extras** menu, it might look nicer to make it a lemma, so we'll try that. Click **New…** in the **Useful Lemmas** window. Enter a lemma as shown below:



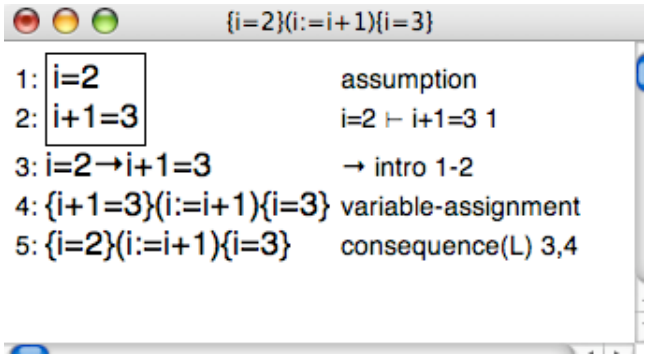
Then "prove" this lemma, by selecting the premise and conclusion using the obviously rule:



This way, when we are all done, we will have all assumptions of this nature showing in the Useful Lemmas, rather than having the sprinkled throughout the program. To apply the lemma to the unproved assertion in the program window, we'll use → **introduction**, which opens a box:



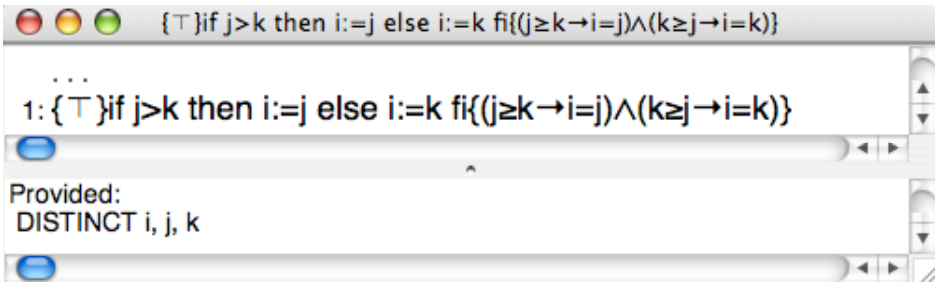
We can then close the box by selecting its conclusion, then clicking **Apply** in the Useful Lemmas window, assuming the lemma we just created is still selected there.



This completes the proof of the first little program.

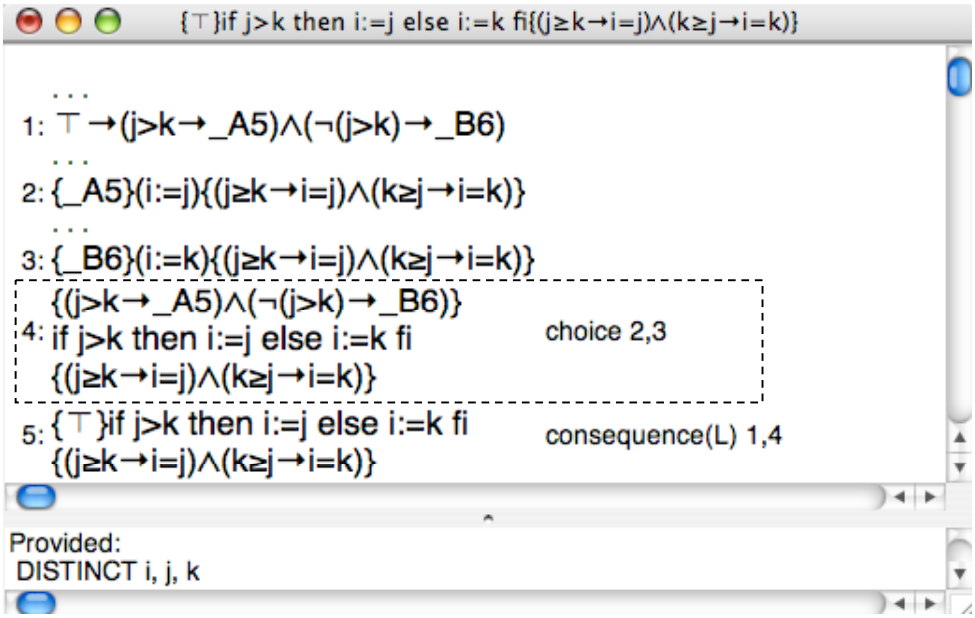
Save your work often, as JAPE sometimes gets wedged.

Let's try the conditional example shown below:



The DISTINCT annotation is a way of indicating the assumption that variables are not aliased to each other.

Obviously the correct rule choice use here is the **choice** rule:



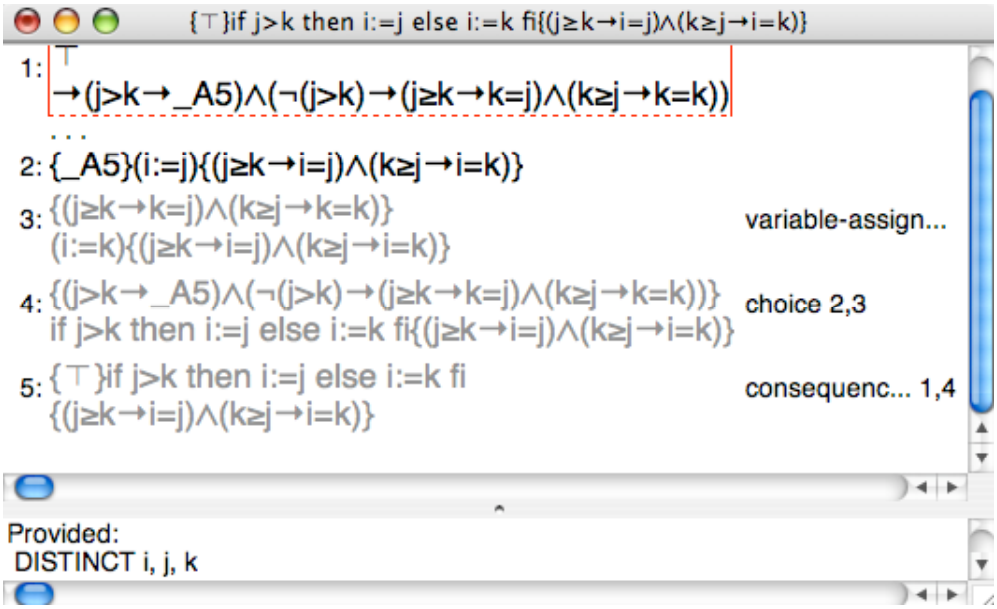
Because the formulas can have a run-on appearance, we have drawn a dashed box around the result of the choice rule above. Applying choice, JAPE has once again introduced a consequence for us. JAPE's **choice** rule slightly different than the standard one in Hoare logic:

$$\frac{\vdots \qquad \vdots}{\{A\}\ prog1\ \{C\} \qquad \{B\}\ prog2\ \{C\}}\ choice$$
$$\{(E\ computes) \wedge (E \rightarrow A) \wedge (\neg E \rightarrow B)\}\ \text{if}\ E\ \text{then}\ prog1\ \text{else}\ prog2\ \text{fi}\ \{C\}$$

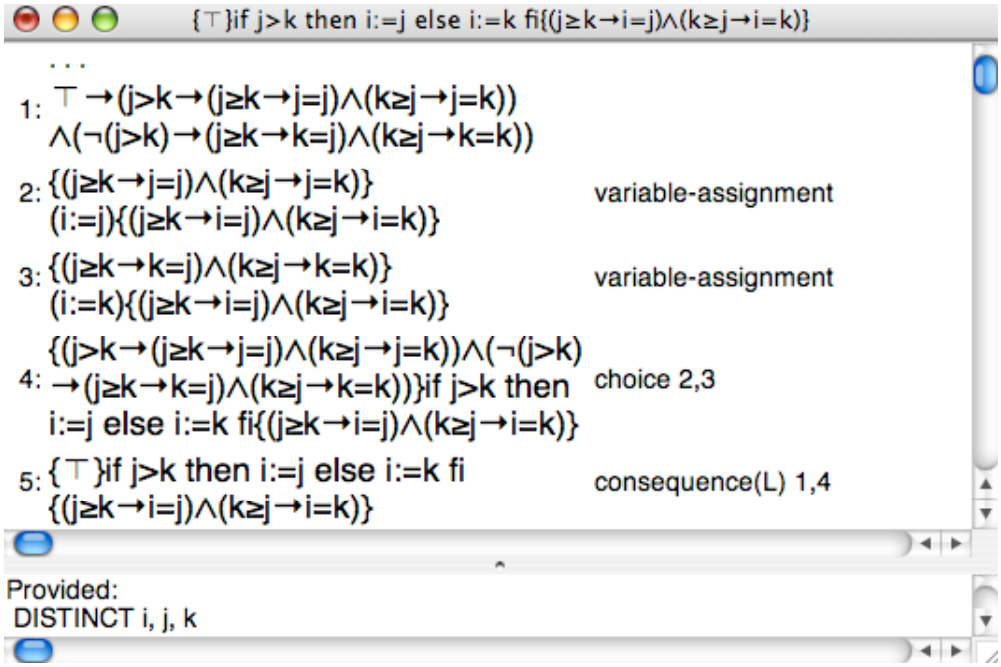
Ignoring the "E computes" conjunct for the moment, this rule describes sub-proofs that need to be done in order to establish the post-condition C after a conditional statement. The dots indicate that these triples have their own sub-proofs. It is up to us to choose

what A and B are (_A5 and _B6 in the above example), and this will be done either automatically by JAPE, or by choosing **Unify** from the **Edit** menu.

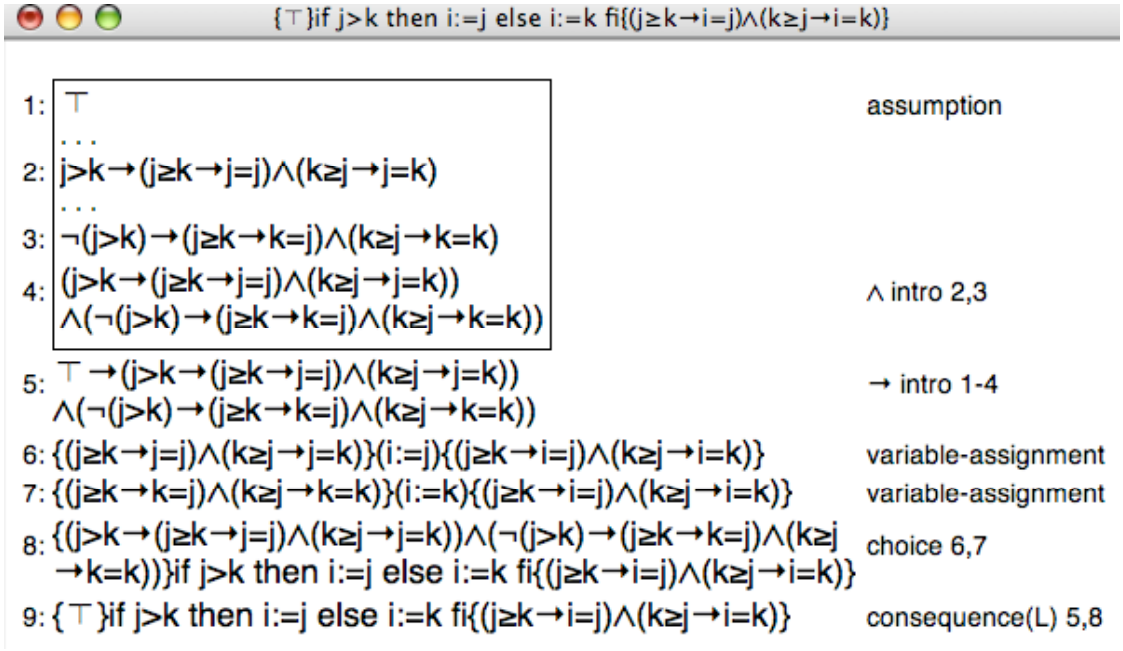
Let's click variable assignment for line 3 above. Notice that JAPE does the unification for us:



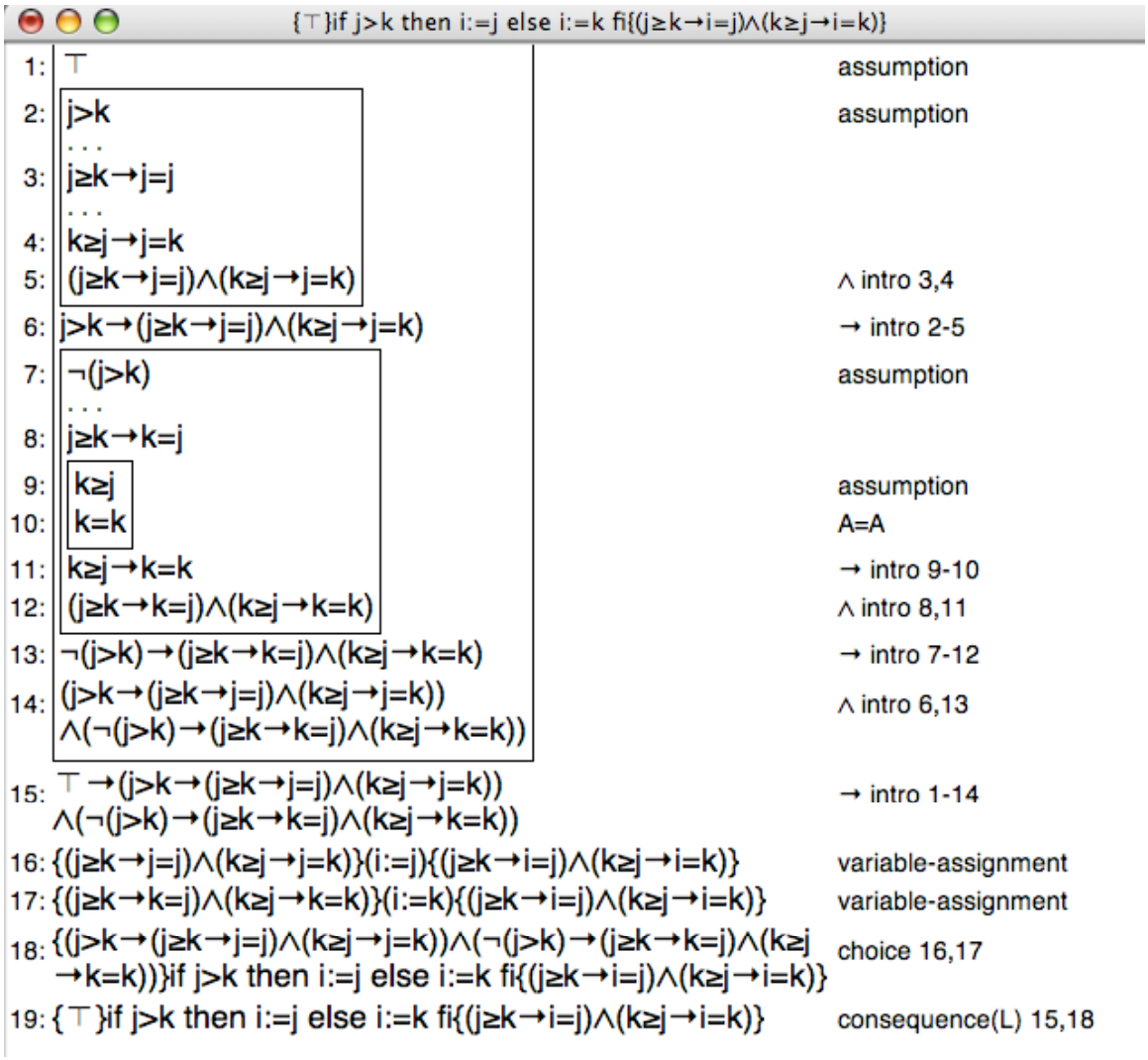
This is because of its ability to construct the weakest pre-condition from the post condition for an assignment statement. In the same step, JAPE has substituted for _B6 in line 1. Similarly we can use the same rule on the other assignment statement:



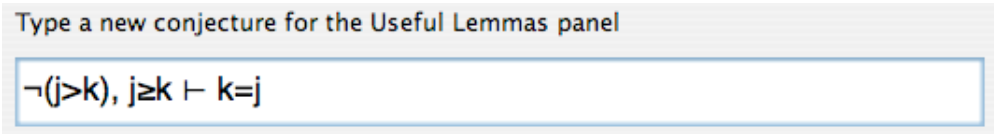
Now we are left with just a logical assertion to prove. This will use an → introduction rule, then backward ∧ introduction rule.



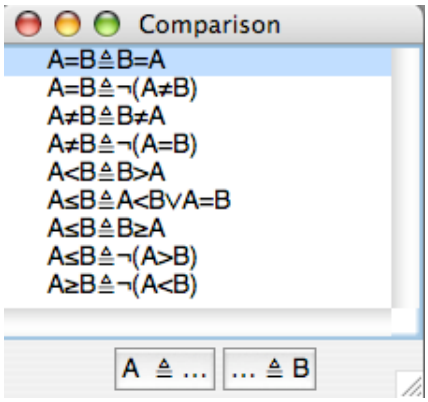
We keep breaking these formulas down using the appropriate logic rules, simplifying the conclusions until we arrive at statements that can be proved. For example, k=k on line 11 below is a consequence of the **A=A** rule in the **Extras** menu.



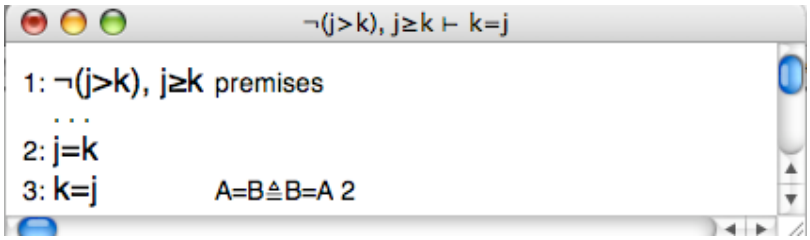
When we get to the point of proving k = j, we have hypotheses j ≥ k and ¬(j>k) with which to work. We can create a lemma for this sub-proof:



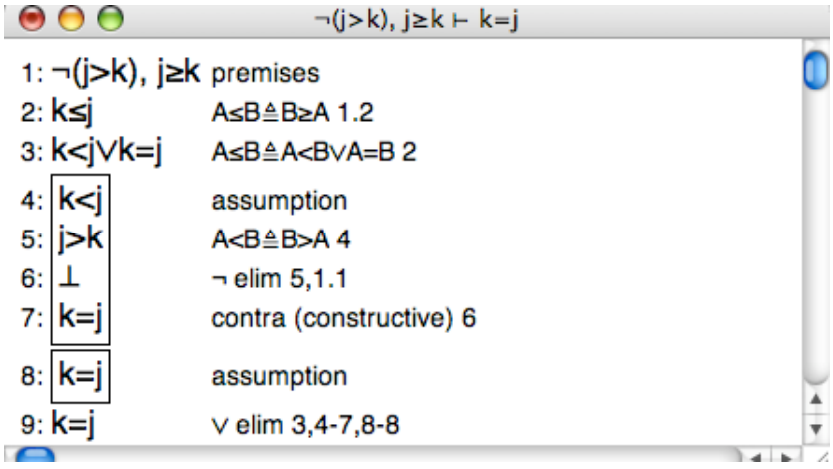
We then prove this lemma using rules available from the Comparison menu. Note that these rules are bi-directional, and the direction of application is chosen by clicking the appropriate button at the bottom:



The next display shows use of the rule indicated, in the right-to-left direction, because we applied the rule to the conclusion k = j to get the hypothesis j = k.



At this point, the remainder seems to be an exercise in finding rules that pattern match against the available hypotheses and conclusions. At any time, we might stop and declare these to be obvious, to get on with the more germane issues. But we persist. Realizing that the rule above yielding line 2 was the wrong choice, we back up and eventually construct the proof using other rules from Comparison, as shown.



Then we record this proof to Useful Lemmas.

Now we can return to our main proof and apply the lemma, to get line 9: