

Interactive Formal Verification

8: Operational Semantics

Tjark Weber
(Slides: Lawrence C Paulson)
Computer Laboratory
University of Cambridge

Overview

Overview

- The operational semantics of programming languages can be given *inductively*.
 - Type checking
 - Expression evaluation
 - Command execution, including concurrency

Overview

- The operational semantics of programming languages can be given *inductively*.
 - Type checking
 - Expression evaluation
 - Command execution, including concurrency
- Properties of the semantics are frequently proved by induction.

Overview

- The operational semantics of programming languages can be given *inductively*.
 - Type checking
 - Expression evaluation
 - Command execution, including concurrency
- Properties of the semantics are frequently proved by induction.
- Running example: an abstract language with WHILE

Language Syntax

```
typedec1 loc -- "an unspecified type of locations"
```

```
type_synonym val    = nat -- "values"
```

```
type_synonym state = "loc => val"
```

```
type_synonym aexp  = "state => val"
```

```
type_synonym bexp  = "state => bool"    -- "functions on states"
```

```
datatype
```

```
  com = SKIP
```

```
    | Assign loc aexp          ( "_ ::= _ " 60 )
    | Semi    com com          ( "_; _"    [60, 60] 10 )
    | Cond    bexp com com     ( "IF _ THEN _ ELSE _" 60 )
    | While   bexp com         ( "WHILE _ DO _" 60 )
```

Language Syntax

```
typedec1 loc -- "an unspecified type of locations"
```

```
type_synonym val    = nat -- "values"
```

```
type_synonym state = "loc => val"
```

```
type_synonym aexp  = "state => val"
```

```
type_synonym bexp  = "state => bool" -- "functions on states"
```

```
datatype
```

```
  com = SKIP
```

```
    | Assign loc aexp
```

```
    | Semi    com com
```

```
    | Cond    bexp com com
```

```
    | While   bexp com
```

Arithmetic & boolean expressions
are *functions* over the state

```
( "_ ::= _ " 60 )
```

```
( "_; _ " [60, 60] 10 )
```

```
( "IF _ THEN _ ELSE _" 60 )
```

```
( "WHILE _ DO _" 60 )
```

A “Big-Step” Semantics

A “Big-Step” Semantics

$$\langle \text{skip}, s \rangle \rightarrow s$$

A “Big-Step” Semantics

$$\langle \mathbf{skip}, s \rangle \rightarrow s$$

$$\langle x := a, s \rangle \rightarrow s[x := a \ s]$$

A “Big-Step” Semantics

$$\langle \mathbf{skip}, s \rangle \rightarrow s$$

$$\langle x := a, s \rangle \rightarrow s[x := a \ s]$$

$$\frac{\langle c_0, s \rangle \rightarrow s'' \quad \langle c_1, s'' \rangle \rightarrow s'}{\langle c_0; c_1, s \rangle \rightarrow s'}$$

A “Big-Step” Semantics

$$\langle \mathbf{skip}, s \rangle \rightarrow s$$

$$\langle x := a, s \rangle \rightarrow s[x := a \ s]$$

$$\frac{\langle c_0, s \rangle \rightarrow s'' \quad \langle c_1, s'' \rangle \rightarrow s'}{\langle c_0; c_1, s \rangle \rightarrow s'}$$

$$\frac{b \ s \quad \langle c_0, s \rangle \rightarrow s'}{\langle \mathbf{if} \ b \ \mathbf{then} \ c_0 \ \mathbf{else} \ c_1, s \rangle \rightarrow s'}$$

$$\frac{\neg b \ s \quad \langle c_1, s \rangle \rightarrow s'}{\langle \mathbf{if} \ b \ \mathbf{then} \ c_0 \ \mathbf{else} \ c_1, s \rangle \rightarrow s'}$$

A “Big-Step” Semantics

$$\langle \mathbf{skip}, s \rangle \rightarrow s$$

$$\langle x := a, s \rangle \rightarrow s[x := a \ s]$$

$$\frac{\langle c_0, s \rangle \rightarrow s'' \quad \langle c_1, s'' \rangle \rightarrow s'}{\langle c_0; c_1, s \rangle \rightarrow s'}$$

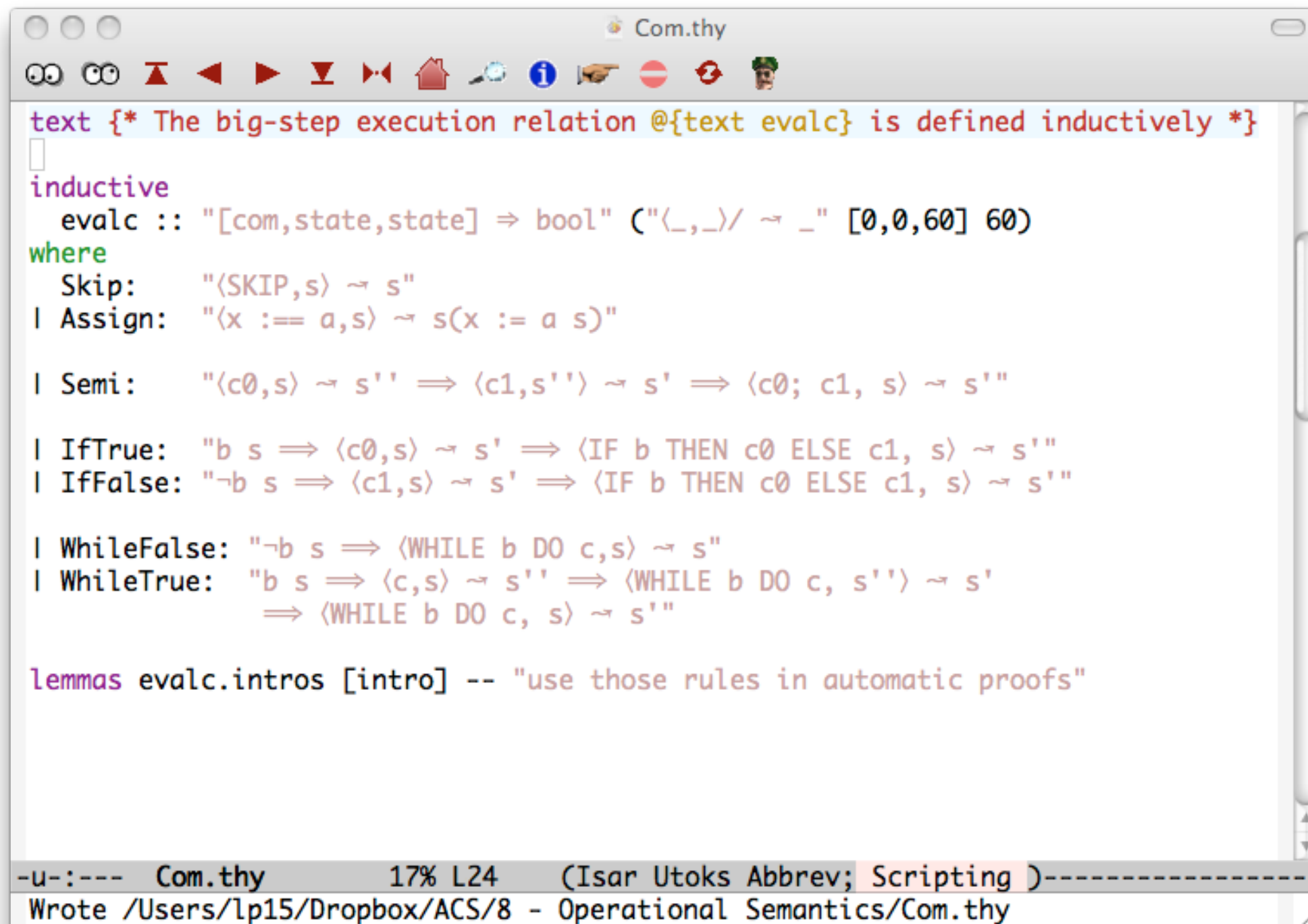
$$\frac{b \ s \quad \langle c_0, s \rangle \rightarrow s'}{\langle \mathbf{if} \ b \ \mathbf{then} \ c_0 \ \mathbf{else} \ c_1, s \rangle \rightarrow s'}$$

$$\frac{\neg b \ s \quad \langle c_1, s \rangle \rightarrow s'}{\langle \mathbf{if} \ b \ \mathbf{then} \ c_0 \ \mathbf{else} \ c_1, s \rangle \rightarrow s'}$$

$$\frac{\neg b \ s}{\langle \mathbf{while} \ b \ \mathbf{do} \ c, s \rangle \rightarrow s}$$

$$\frac{b \ s \quad \langle c, s \rangle \rightarrow s'' \quad \langle \mathbf{while} \ b \ \mathbf{do} \ c, s'' \rangle \rightarrow s'}{\langle \mathbf{while} \ b \ \mathbf{do} \ c, s \rangle \rightarrow s'}$$

Formalised Language Semantics



```
Com.thy
text {* The big-step execution relation @{text evalc} is defined inductively *}
inductive
  evalc :: "[com,state,state]  $\Rightarrow$  bool" ("<_,_>/  $\leadsto$  _" [0,0,60] 60)
where
  Skip:      "<SKIP,s>  $\leadsto$  s"
| Assign:    "<x := a,s>  $\leadsto$  s(x := a s)"
| Semi:      "<c0,s>  $\leadsto$  s'  $\Rightarrow$  <c1,s'>  $\leadsto$  s'  $\Rightarrow$  <c0; c1, s>  $\leadsto$  s'"
| IfTrue:    "<b s  $\Rightarrow$  <c0,s>  $\leadsto$  s'  $\Rightarrow$  <IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
| IfFalse:   "< $\neg$ b s  $\Rightarrow$  <c1,s>  $\leadsto$  s'  $\Rightarrow$  <IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
| WhileFalse: "< $\neg$ b s  $\Rightarrow$  <WHILE b DO c,s>  $\leadsto$  s"
| WhileTrue:  "<b s  $\Rightarrow$  <c,s>  $\leadsto$  s'  $\Rightarrow$  <WHILE b DO c, s'>  $\leadsto$  s'
                $\Rightarrow$  <WHILE b DO c, s>  $\leadsto$  s'"
lemmas evalc.intros [intro] -- "use those rules in automatic proofs"

-u:--- Com.thy      17% L24   (Isar Utoks Abbrev; Scripting )-----
Wrote /Users/lp15/Dropbox/ACS/8 - Operational Semantics/Com.thy
```

Formalised Language Semantics

an inductive *predicate*
with special syntax

```
text {* The big-step execution relation @{text evalc} is defined inductively *}
inductive
  evalc :: "[com,state,state]  $\Rightarrow$  bool" ("<_,_>/  $\leadsto$  _" [0,0,60] 60)
where
  Skip:    "<SKIP,s>  $\leadsto$  s"
| Assign:  "<x := a,s>  $\leadsto$  s(x := a s)"
| Semi:    "<c0,s>  $\leadsto$  s'  $\Rightarrow$  <c1,s'>  $\leadsto$  s'  $\Rightarrow$  <c0; c1, s>  $\leadsto$  s'"
| IfTrue:  "b s  $\Rightarrow$  <c0,s>  $\leadsto$  s'  $\Rightarrow$  <IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
| IfFalse: " $\neg$ b s  $\Rightarrow$  <c1,s>  $\leadsto$  s'  $\Rightarrow$  <IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
| WhileFalse: " $\neg$ b s  $\Rightarrow$  <WHILE b DO c,s>  $\leadsto$  s"
| WhileTrue:  "b s  $\Rightarrow$  <c,s>  $\leadsto$  s'  $\Rightarrow$  <WHILE b DO c, s'>  $\leadsto$  s'
                $\Rightarrow$  <WHILE b DO c, s>  $\leadsto$  s'"

lemmas evalc.intros [intro] -- "use those rules in automatic proofs"
```

-u-:--- Com.thy 17% L24 (Isar Utoks Abbrev; Scripting)-----
Wrote /Users/lp15/Dropbox/ACS/8 - Operational Semantics/Com.thy

Formalised Language Semantics

```
text {* The big-step execution relation @{text evalc} is defined inductively *}
inductive
  evalc :: "[com,state,state]  $\Rightarrow$  bool" ("<_,_>/  $\leadsto$  _" [0,0,60] 60)
where
  Skip:    "<SKIP,s>  $\leadsto$  s"
  Assign:  "<x := a,s>  $\leadsto$  s(x := a s)"
  Semi:    "<c0,s>  $\leadsto$  s'  $\Rightarrow$  <c1,s'>  $\leadsto$  s'  $\Rightarrow$  <c0; c1, s>  $\leadsto$  s'"
  IfTrue:  "b s  $\Rightarrow$  <c0,s>  $\leadsto$  s'  $\Rightarrow$  <IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
  IfFalse: " $\neg$ b s  $\Rightarrow$  <c1,s>  $\leadsto$  s'  $\Rightarrow$  <IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
  WhileFalse: " $\neg$ b s  $\Rightarrow$  <WHILE b DO c,s>  $\leadsto$  s"
  WhileTrue:  "b s  $\Rightarrow$  <c,s>  $\leadsto$  s'  $\Rightarrow$  <WHILE b DO c, s'>  $\leadsto$  s'
              $\Rightarrow$  <WHILE b DO c, s>  $\leadsto$  s'"

lemmas evalc.intros [intro] -- "use those rules in automatic proofs"
```

an inductive *predicate* with special syntax

declare as *introduction* rules for auto and blast

Rule Inversion

Rule Inversion

- When $\langle \mathbf{skip}, s \rangle \rightarrow s'$ we know $s = s'$

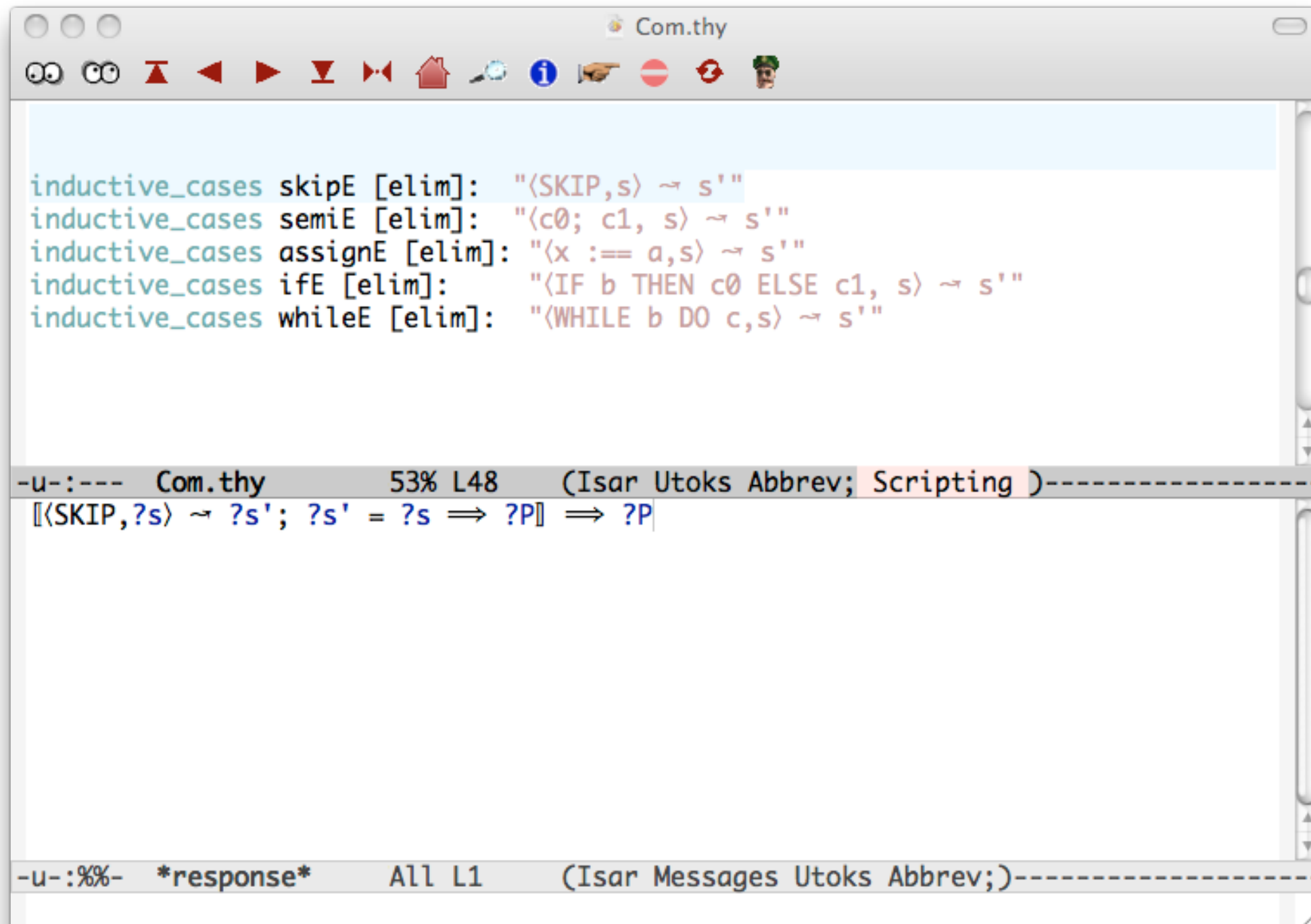
Rule Inversion

- When $\langle \mathbf{skip}, s \rangle \rightarrow s'$ we know $s = s'$
- When $\langle \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, s \rangle \rightarrow s'$ we know
 - b and $\langle c_0, s \rangle \rightarrow s'$, or...
 - $\neg b$ and $\langle c_1, s \rangle \rightarrow s'$

Rule Inversion

- When $\langle \mathbf{skip}, s \rangle \rightarrow s'$ we know $s = s'$
- When $\langle \mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1, s \rangle \rightarrow s'$ we know
 - b and $\langle c_0, s \rangle \rightarrow s'$, or...
 - $\neg b$ and $\langle c_1, s \rangle \rightarrow s'$
- This sort of case analysis is easy in Isabelle.

Rule Inversion in Isabelle



Rule Inversion in Isabelle

name of the new lemma

```
inductive_cases skipE [elim]: "<SKIP,s>  $\leadsto$  s'"
inductive_cases semiE [elim]: "<c0; c1, s>  $\leadsto$  s'"
inductive_cases assignE [elim]: "<x := a,s>  $\leadsto$  s'"
inductive_cases ifE [elim]: "<IF b THEN c0 ELSE c1, s>  $\leadsto$  s'"
inductive_cases whileE [elim]: "<WHILE b DO c,s>  $\leadsto$  s'"

```

```
-u-:--- Com.thy          53% L48    (Isar Utoks Abbrev; Scripting )-----
[[<SKIP,?s>  $\leadsto$  ?s'; ?s' = ?s  $\Rightarrow$  ?P]]  $\Rightarrow$  ?P

```

```
-u-:%%- *response*      All L1    (Isar Messages Utoks Abbrev;)-----

```

Rule Inversion in Isabelle

name of the new lemma

declared as an *elimination*
rule to auto and blast

```
inductive_cases skipE [elim]: "<SKIP,s>  $\rightsquigarrow$  s'"
inductive_cases semiE [elim]: "<c0; c1, s>  $\rightsquigarrow$  s'"
inductive_cases assignE [elim]: "<x := a,s>  $\rightsquigarrow$  s'"
inductive_cases ifE [elim]: "<IF b THEN c0 ELSE c1, s>  $\rightsquigarrow$  s'"
inductive_cases whileE [elim]: "<WHILE b DO c,s>  $\rightsquigarrow$  s'"
```

```
-u-:--- Com.thy          53% L48    (Isar Utoks Abbrev; Scripting )-----
[[<SKIP,?s>  $\rightsquigarrow$  ?s'; ?s' = ?s  $\Rightarrow$  ?P]]  $\Rightarrow$  ?P
```

```
-u-:%%- *response*      All L1    (Isar Messages Utoks Abbrev;)-----
```

Rule Inversion in Isabelle

name of the new lemma

declared as an *elimination*
rule to auto and blast

```
inductive_cases skipE [elim]: "<SKIP, s> ~ s'"
inductive_cases semiE [elim]: "<c0; c1, s> ~ s'"
inductive_cases assignE [elim]: "<x := a, s> ~ s'"
inductive_cases ifE [elim]: "<IF b THEN c0 ELSE c1, s> ~ s'"
inductive_cases whileE [elim]: "<WHILE b DO c, s> ~ s'"
```

$\langle \text{skip}, s \rangle \rightarrow s'$ implies $s = s'$

```
-u-:--- Com.thy 53% 48 (Isar Utoks Abbrev; Scripting )-----
[[<SKIP, ?s> ~ ?s'; ?s' = ?s ==> ?P]] ==> ?P
```

```
-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)-----
```

Rule Inversion in Isabelle

name of the new lemma

declared as an *elimination rule* to auto and blast

```
inductive_cases skipE [elim]: "<SKIP, s> ~ s'"  
inductive_cases semiE [elim]: "<c0; c1, s> ~ s'"  
inductive_cases assignE [elim]: "<x := a, s> ~ s'"  
inductive_cases ifE [elim]: "<IF b THEN c0 ELSE c1, s> ~ s'"  
inductive_cases whileE [elim]: "<WHILE b DO c, s> ~ s'"
```

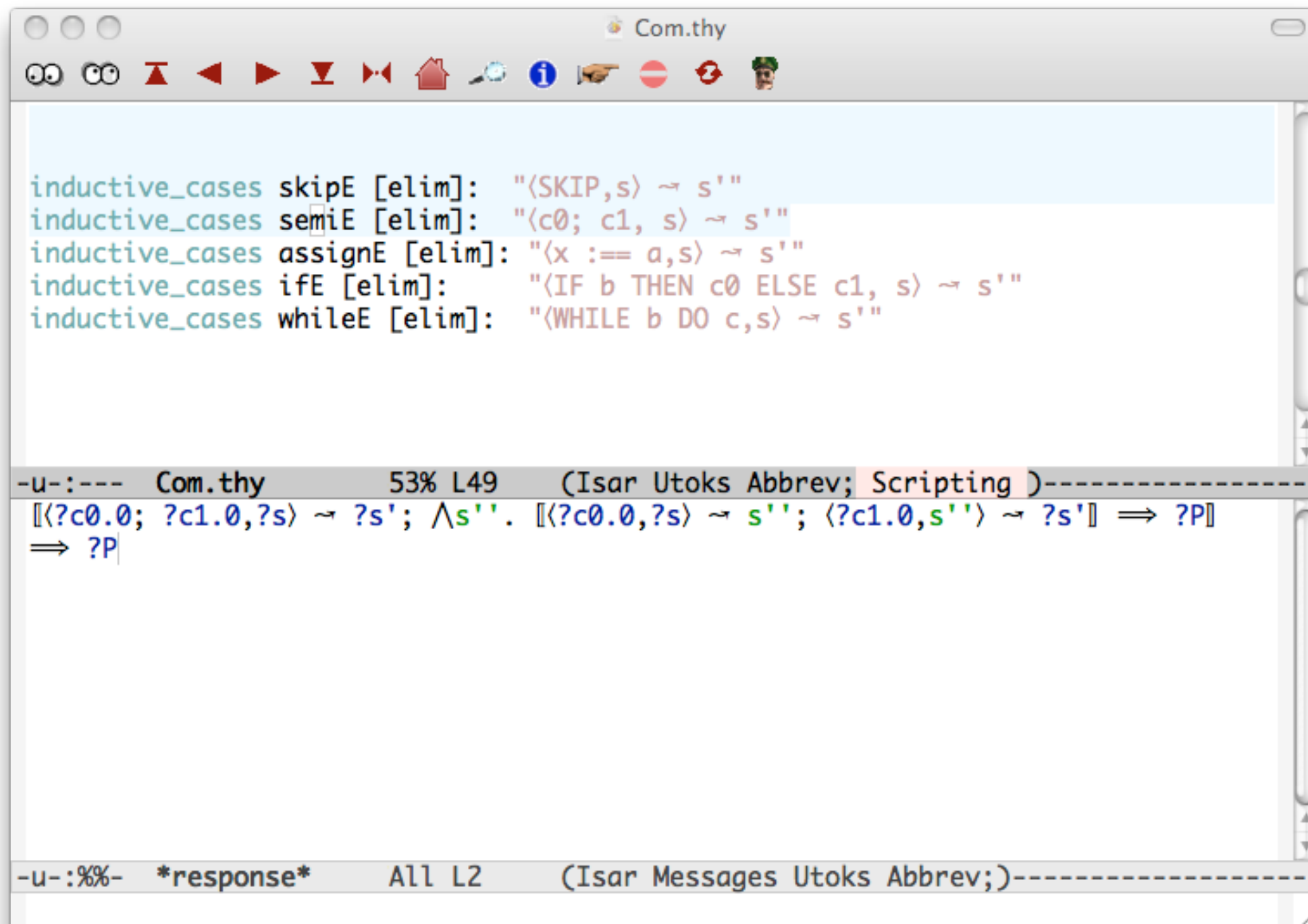
$\langle \text{skip}, s \rangle \rightarrow s'$ implies $s = s'$

```
-u-:--- Com.thy          53% 448 (Isar Utoks Abbrev; Scripting )-----  
[[<SKIP, ?s> ~ ?s'; ?s' = ?s ==> ?P]] ==> ?P
```

the typical format of an
elimination rule

```
-u-:%%- *response*      All L1 (Isar Messages Utoks Abbrev;)-----
```

Rule Inversion Again



The screenshot shows a window titled "Com.thy" with a toolbar at the top. The main text area contains five inductive cases for a relation $\leadsto$:

```
inductive_cases skipE [elim]: "<SKIP,s> ~ s'"
inductive_cases semiE [elim]: "<c0; c1, s> ~ s'"
inductive_cases assignE [elim]: "<x := a,s> ~ s'"
inductive_cases ifE [elim]: "<IF b THEN c0 ELSE c1, s> ~ s'"
inductive_cases whileE [elim]: "<WHILE b DO c,s> ~ s'"
```

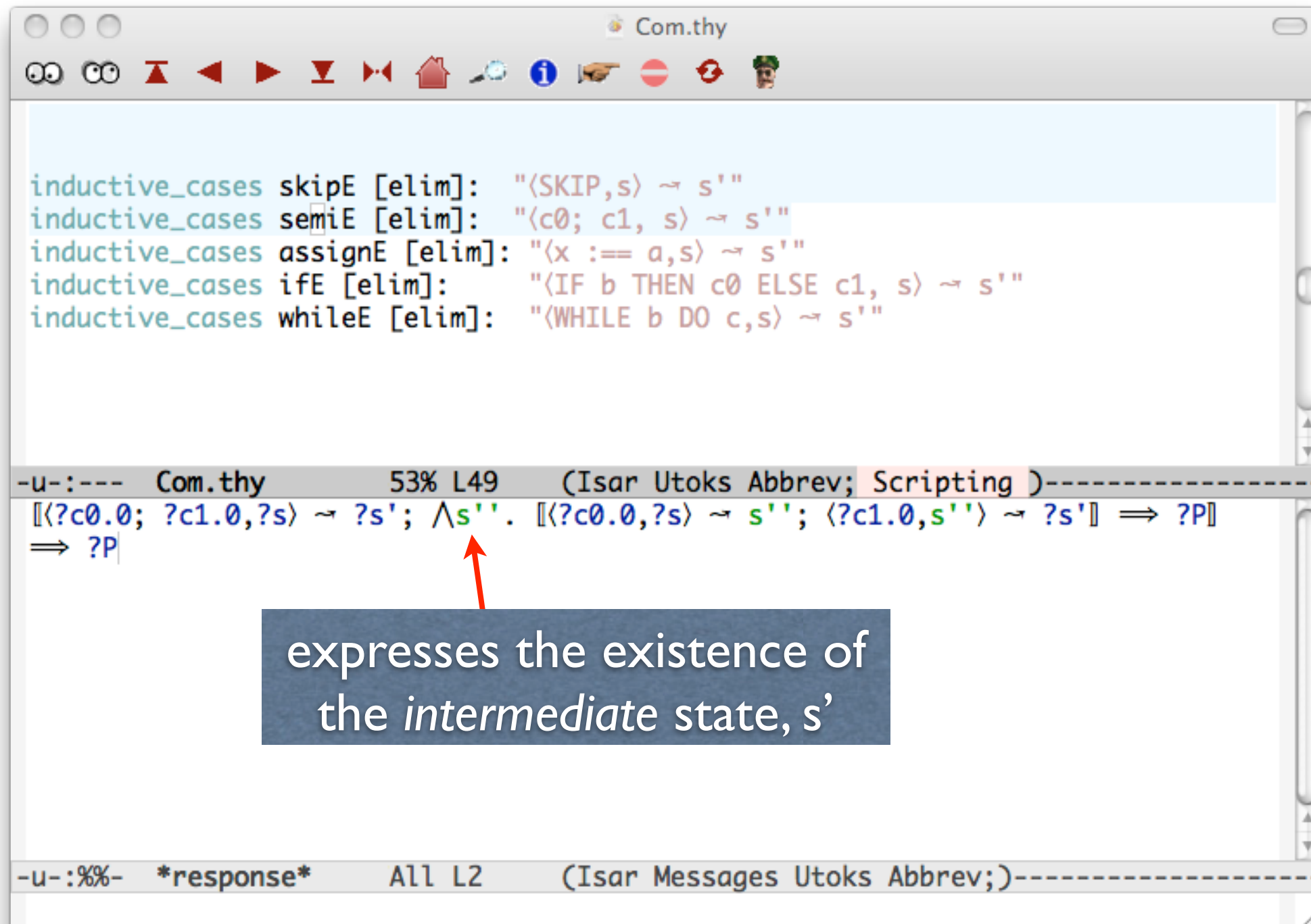
Below the text area is a status bar with the text: "-u-:--- Com.thy 53% L49 (Isar Utoks Abbrev; Scripting)-----".

The bottom panel shows a goal to be proved:

```
[[<?c0.0; ?c1.0,?s> ~ ?s'; ^s''. [[<?c0.0,?s> ~ s''; <?c1.0,s''> ~ ?s']] => ?P]]
=> ?P
```

At the very bottom, another status bar reads: "-u-:%%- *response* All L2 (Isar Messages Utoks Abbrev;)-----".

Rule Inversion Again



A Non-Termination Proof

$$\langle \text{while true do } c, s \rangle \not\rightarrow s'$$

The inductive version considers
all possible commands

A Non-Termination Proof

$$\langle \text{while true do } c, s \rangle \not\rightarrow s'$$

This formula is not provable by induction!

The inductive version considers
all possible commands

A Non-Termination Proof

$$\langle \text{while true do } c, s \rangle \not\rightarrow s'$$

This formula is not provable by induction!

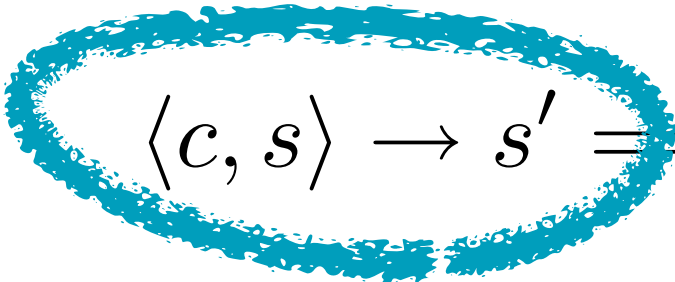
$$\langle c, s \rangle \rightarrow s' \implies \forall c'. c \neq (\text{while true do } c')$$

The inductive version considers
all possible commands

A Non-Termination Proof

$$\langle \text{while true do } c, s \rangle \not\rightarrow s'$$

This formula is not provable by induction!


$$\langle c, s \rangle \rightarrow s' \Rightarrow \forall c'. c \neq (\text{while true do } c')$$

The inductive version considers
all possible commands

Non-Termination in Isabelle

```

Isabelle Proof General: Com.thy

lemma while_never: "<c, s> ~ u ==> c ≠ WHILE (λs. True) DO c1"
  apply (induct rule: evalc.induct)
  apply auto
  -u-:***- Com.thy          51% L60 (Isar Utoks Abbrev; Scripting )-----
goal (7 subgoals):
1. ∧s. SKIP ≠ WHILE λs. True DO c1
2. ∧x a s. x ::= a ≠ WHILE λs. True DO c1
3. ∧c0 s s' c1a s'.
   [[<c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1; <c1a,s'> ~ s';
    c1a ≠ WHILE λs. True DO c1]]
   ==> (c0; c1a) ≠ WHILE λs. True DO c1
4. ∧b s c0 s' c1a.
   [[b s; <c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1]]
   ==> IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
5. ∧b s c1a s' c0.
   [[¬ b s; <c1a,s> ~ s'; c1a ≠ WHILE λs. True DO c1]]
   ==> IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
6. ∧b s c. ¬ b s ==> WHILE b DO c ≠ WHILE λs. True DO c1
7. ∧b s c s' s'.
   [[b s; <c,s> ~ s'; c ≠ WHILE λs. True DO c1; <WHILE b DO c,s'> ~ s';
    WHILE b DO c ≠ WHILE λs. True DO c1]]
   ==> WHILE b DO c ≠ WHILE λs. True DO c1
  -u-:%%- *goals*          2% L4 (Isar Proofstate Utoks Abbrev;)-----

```

Non-Termination in Isabelle

7 subgoals, one for each rule of the definition

```
Isabelle Proof General: Com.thy
[Icons: zoom, undo, redo, find, etc.]

lemma while_never: "<c, s> ~ u => c ≠ WHILE (λs. True) DO c1"
apply (induct rule: evalc.induct)
apply auto
-u-:***- Com.thy 51% L60 (Isar Utoks Abbrev; Scripting )-----
goal (7 subgoals):
1. ∧s. SKIP ≠ WHILE λs. True DO c1
2. ∧x a s. x ::= a ≠ WHILE λs. True DO c1
3. ∧c0 s s' c1a s'.
   [[<c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1; <c1a,s'> ~ s';
    c1a ≠ WHILE λs. True DO c1]]
   => (c0; c1a) ≠ WHILE λs. True DO c1
4. ∧b s c0 s' c1a.
   [[b s; <c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1]]
   => IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
5. ∧b s c1a s' c0.
   [[¬ b s; <c1a,s> ~ s'; c1a ≠ WHILE λs. True DO c1]]
   => IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
6. ∧b s c. ¬ b s => WHILE b DO c ≠ WHILE λs. True DO c1
7. ∧b s c s' s'.
   [[b s; <c,s> ~ s'; c ≠ WHILE λs. True DO c1; <WHILE b DO c,s'> ~ s';
    WHILE b DO c ≠ WHILE λs. True DO c1]]
   => WHILE b DO c ≠ WHILE λs. True DO c1
-u-:%%- *goals* 2% L4 (Isar Proofstate Utoks Abbrev;)-----
```

Non-Termination in Isabelle

```
Isabelle Proof General: Com.thy
[Icons]
lemma while_never: "<c, s> ~ u => c ≠ WHILE (λs. True) DO c1"
apply (induct rule: evalc.induct)
apply auto
-u-:***- Com.thy 51% L60 (Isar Utoks Abbrev; Scripting)
goal (7 subgoals):
1. ∧s. SKIP ≠ WHILE λs. True DO c1
2. ∧x a s. x ::= a ≠ WHILE λs. True DO c1
3. ∧c0 s s' c1a s'.
   [[<c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1; <c1a,s'> ~ s';
    c1a ≠ WHILE λs. True DO c1]]
   => (c0; c1a) ≠ WHILE λs. True DO c1
4. ∧b s c0 s' c1a.
   [[b s; <c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1]]
   => IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
5. ∧b s c1a s' c0.
   [[¬ b s; <c1a,s> ~ s'; c1a ≠ WHILE λs. True DO c1]]
   => IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
6. ∧b s c. ¬ b s => WHILE b DO c ≠ WHILE λs. True DO c1
7. ∧b s c s' s'.
   [[b s; <c,s> ~ s'; c ≠ WHILE λs. True DO c1; <WHILE b DO c,s'> ~ s';
    WHILE b DO c ≠ WHILE λs. True DO c1]]
   => WHILE b DO c ≠ WHILE λs. True DO c1
-u-:%%- *goals* 2% L4 (Isar Proofstate Utoks Abbrev;)------
```

7 subgoals, one for each rule of the definition

Most are trivial, by distinctness

Non-Termination in Isabelle

```
Isabelle Proof General: Com.thy
[Icons: zoom, undo, redo, find, etc.]

lemma while_never: "<c, s> ~ u => c ≠ WHILE (λs. True) DO c1"
  apply (induct rule: evalc.induct)
  apply auto
-u-:***- Com.thy 51% L60 (Isar Utoks Abbrev; Scripting)

goal (7 subgoals):
1. ∧s. SKIP ≠ WHILE λs. True DO c1
2. ∧x a s. x ::= a ≠ WHILE λs. True DO c1
3. ∧c0 s s' c1a s'.
   [[<c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1; <c1a,s'> ~ s';
    c1a ≠ WHILE λs. True DO c1]]
   => (c0; c1a) ≠ WHILE λs. True DO c1
4. ∧b s c0 s' c1a.
   [[b s; <c0,s> ~ s'; c0 ≠ WHILE λs. True DO c1]]
   => IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
5. ∧b s c1a s' c0.
   [[¬ b s; <c1a,s> ~ s'; c1a ≠ WHILE λs. True DO c1]]
   => IF b THEN c0 ELSE c1a ≠ WHILE λs. True DO c1
6. ∧b s c. ¬ b s => WHILE b DO c ≠ WHILE λs. True DO c1
7. ∧b s c s' s'.
   [[b s; <c,s> ~ s'; c ≠ WHILE λs. True DO c1; <WHILE b DO c,s'> ~ s';
    WHILE b DO c ≠ WHILE λs. True DO c1]]
   => WHILE b DO c ≠ WHILE λs. True DO c1

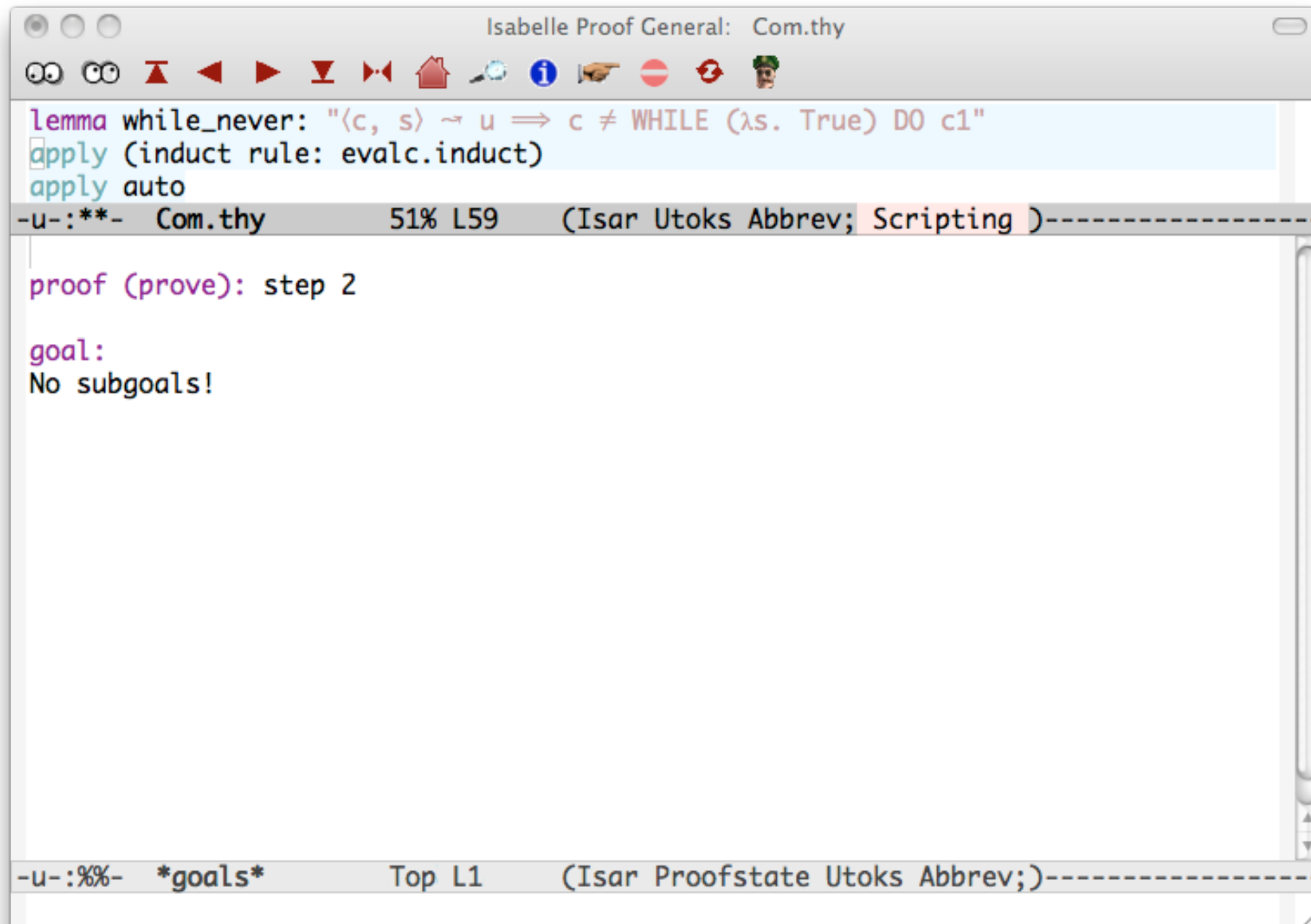
-u-:%%- *goals* 2% L4 (Isar Proofstate Utoks Abbrev;)------
```

7 subgoals, one for each rule of the definition

Most are trivial, by distinctness

trivial for another reason

Done!



The screenshot shows the Isabelle Proof General window for a file named 'Com.thy'. The window has a title bar with standard macOS window controls and a toolbar with various icons for navigation and editing. The main text area contains the following code:

```
lemma while_never: "<c, s> ~> u => c ≠ WHILE (λs. True) DO c1"
  apply (induct rule: evalc.induct)
  apply auto
```

Below the code, a status bar indicates the current position: '-u-:**- Com.thy 51% L59 (Isar Utoks Abbrev; Scripting)-----'. The main proof area shows the result of the proof:

```
proof (prove): step 2

goal:
No subgoals!
```

At the bottom, another status bar shows the goal state: '-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----'.

Determinacy

$$\frac{\langle c, s \rangle \rightarrow t \quad \langle c, s \rangle \rightarrow u}{t = u}$$

If a command is executed in a given state, and it terminates, then this final state is *unique*.

Determinacy in Isabelle

```

Com.thy
theorem com_det: "<c,s> ~ t ==> <c,s> ~ u ==> u = t"
apply (induct arbitrary: u rule: evalc.induct)
apply blast+
-u-:***- Com.thy 60% L62 (Isar Utoks Abbrev; Scripting )-----
1.  $\wedge s\ u. \langle \text{SKIP}, s \rangle \sim u \implies u = s$ 
2.  $\wedge x\ a\ s\ u. \langle x := a, s \rangle \sim u \implies u = s(x := a\ s)$ 
3.  $\wedge c0\ s\ s'\ c1\ s'\ u. \\ \llbracket \langle c0, s \rangle \sim s'; \wedge u. \langle c0, s \rangle \sim u \implies u = s'; \langle c1, s' \rangle \sim s'; \\ \wedge u. \langle c1, s' \rangle \sim u \implies u = s'; \langle c0; c1, s \rangle \sim u \rrbracket \\ \implies u = s'$ 
4.  $\wedge b\ s\ c0\ s'\ c1\ u. \\ \llbracket b\ s; \langle c0, s \rangle \sim s'; \wedge u. \langle c0, s \rangle \sim u \implies u = s'; \\ \langle \text{IF } b \text{ THEN } c0 \text{ ELSE } c1, s \rangle \sim u \rrbracket \\ \implies u = s'$ 
5.  $\wedge b\ s\ c1\ s'\ c0\ u. \\ \llbracket \neg b\ s; \langle c1, s \rangle \sim s'; \wedge u. \langle c1, s \rangle \sim u \implies u = s'; \\ \langle \text{IF } b \text{ THEN } c0 \text{ ELSE } c1, s \rangle \sim u \rrbracket \\ \implies u = s'$ 
6.  $\wedge b\ s\ c\ u. \llbracket \neg b\ s; \langle \text{WHILE } b \text{ DO } c, s \rangle \sim u \rrbracket \implies u = s$ 
7.  $\wedge b\ s\ c\ s'\ s'\ u. \\ \llbracket b\ s; \langle c, s \rangle \sim s'; \wedge u. \langle c, s \rangle \sim u \implies u = s'; \langle \text{WHILE } b \text{ DO } c, s' \rangle \sim s'; \\ \wedge u. \langle \text{WHILE } b \text{ DO } c, s' \rangle \sim u \implies u = s'; \langle \text{WHILE } b \text{ DO } c, s \rangle \sim u \rrbracket \\ \implies u = s'$ 
-u-:%%- *goals* 3% L5 (Isar Proofstate Utoks Abbrev;)-----

```

Determinacy in Isabelle

allow the other state to vary

```
Com.thy
theorem com_det: "<c,s> ~ t => <c,s> ~ u => u = t"
apply (induct arbitrary: u rule: evalc.induct)
apply blast+
-u-:**- Com.thy 60% L62 (Isar Utoks Abbrev; Scripting )-----
1.  $\wedge s\ u. \langle \text{SKIP}, s \rangle \sim u \Rightarrow u = s$ 
2.  $\wedge x\ a\ s\ u. \langle x := a, s \rangle \sim u \Rightarrow u = s(x := a\ s)$ 
3.  $\wedge c0\ s\ s'\ c1\ s'\ u. \llbracket \langle c0, s \rangle \sim s'; \wedge u. \langle c0, s \rangle \sim u \Rightarrow u = s'; \langle c1, s' \rangle \sim s'; \wedge u. \langle c1, s' \rangle \sim u \Rightarrow u = s'; \langle c0; c1, s \rangle \sim u \rrbracket \Rightarrow u = s'$ 
4.  $\wedge b\ s\ c0\ s'\ c1\ u. \llbracket b\ s; \langle c0, s \rangle \sim s'; \wedge u. \langle c0, s \rangle \sim u \Rightarrow u = s'; \langle \text{IF } b \text{ THEN } c0 \text{ ELSE } c1, s \rangle \sim u \rrbracket \Rightarrow u = s'$ 
5.  $\wedge b\ s\ c1\ s'\ c0\ u. \llbracket \neg b\ s; \langle c1, s \rangle \sim s'; \wedge u. \langle c1, s \rangle \sim u \Rightarrow u = s'; \langle \text{IF } b \text{ THEN } c0 \text{ ELSE } c1, s \rangle \sim u \rrbracket \Rightarrow u = s'$ 
6.  $\wedge b\ s\ c\ u. \llbracket \neg b\ s; \langle \text{WHILE } b \text{ DO } c, s \rangle \sim u \rrbracket \Rightarrow u = s$ 
7.  $\wedge b\ s\ c\ s'\ s'\ u. \llbracket b\ s; \langle c, s \rangle \sim s'; \wedge u. \langle c, s \rangle \sim u \Rightarrow u = s'; \langle \text{WHILE } b \text{ DO } c, s' \rangle \sim s'; \wedge u. \langle \text{WHILE } b \text{ DO } c, s' \rangle \sim u \Rightarrow u = s'; \langle \text{WHILE } b \text{ DO } c, s \rangle \sim u \rrbracket \Rightarrow u = s'$ 
-u-:%%- *goals* 3% L5 (Isar Proofstate Utoks Abbrev;)-----
```

Determinacy in Isabelle

```
theorem com_det: "<c,s> ~ t => <c,s> ~ u => u = t"
apply (induct arbitrary: u rule: evalc.induct)
apply blast+

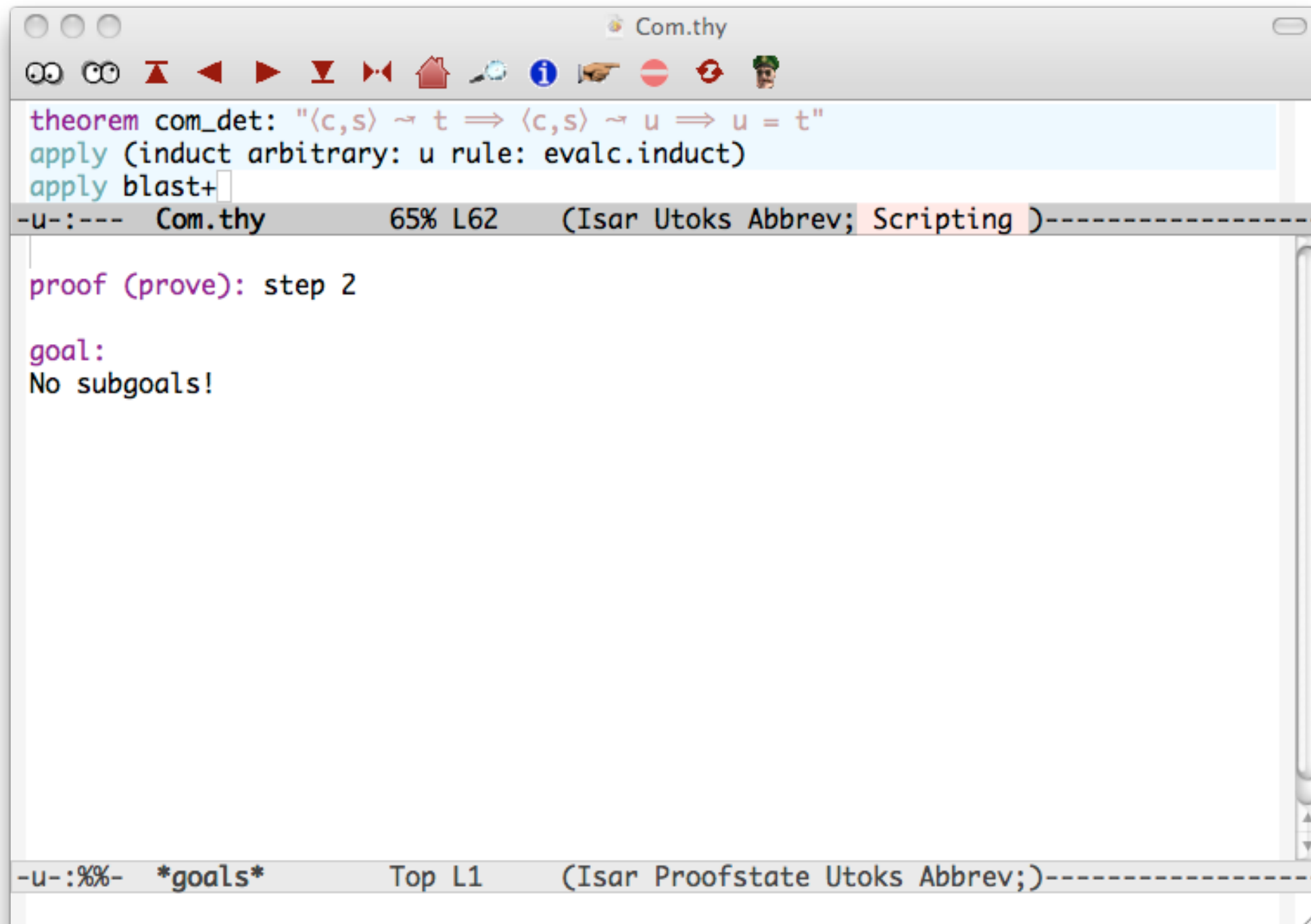
-u-:**- Com.thy 60% L62 (Isar Utoks Abbrev; Scripting )-----
1.  $\wedge s\ u. \langle \text{SKIP}, s \rangle \leadsto u \Rightarrow u = s$ 
2.  $\wedge x\ a\ s\ u. \langle x := a, s \rangle \leadsto u \Rightarrow u = s(x := a\ s)$ 
3.  $\wedge c_0\ s\ s'\ c_1\ s'\ u. \llbracket \langle c_0, s \rangle \leadsto s'; \wedge u. \langle c_0, s \rangle \leadsto u \Rightarrow u = s'; \langle c_1, s' \rangle \leadsto s'; \wedge u. \langle c_1, s' \rangle \leadsto u \Rightarrow u = s'; \langle c_0; c_1, s \rangle \leadsto u \rrbracket \Rightarrow u = s'$ 
4.  $\wedge b\ s\ c_0\ s'\ c_1\ u. \llbracket b\ s; \langle c_0, s \rangle \leadsto s'; \wedge u. \langle c_0, s \rangle \leadsto u \Rightarrow u = s'; \langle \text{IF } b \text{ THEN } c_0 \text{ ELSE } c_1, s \rangle \leadsto u \rrbracket \Rightarrow u = s'$ 
5.  $\wedge b\ s\ c_1\ s'\ c_0\ u. \llbracket \neg b\ s; \langle c_1, s \rangle \leadsto s'; \wedge u. \langle c_1, s \rangle \leadsto u \Rightarrow u = s'; \langle \text{IF } b \text{ THEN } c_0 \text{ ELSE } c_1, s \rangle \leadsto u \rrbracket \Rightarrow u = s'$ 
6.  $\wedge b\ s\ c\ u. \llbracket \neg b\ s; \langle \text{WHILE } b \text{ DO } c, s \rangle \leadsto u \rrbracket \Rightarrow u = s$ 
7.  $\wedge b\ s\ c\ s'\ s'\ u. \llbracket b\ s; \langle c, s \rangle \leadsto s'; \wedge u. \langle c, s \rangle \leadsto u \Rightarrow u = s'; \langle \text{WHILE } b \text{ DO } c, s' \rangle \leadsto s'; \wedge u. \langle \text{WHILE } b \text{ DO } c, s' \rangle \leadsto u \Rightarrow u = s'; \langle \text{WHILE } b \text{ DO } c, s \rangle \leadsto u \rrbracket \Rightarrow u = s'$ 

-u-:%%- *goals* 3% L5 (Isar Proofstate Utoks Abbrev;)-----
```

allow the other state to vary

trivial by rule inversion

Proved by Rule Inversion



The screenshot shows a theorem prover interface with a title bar 'Com.thy'. The main text area contains the following code:

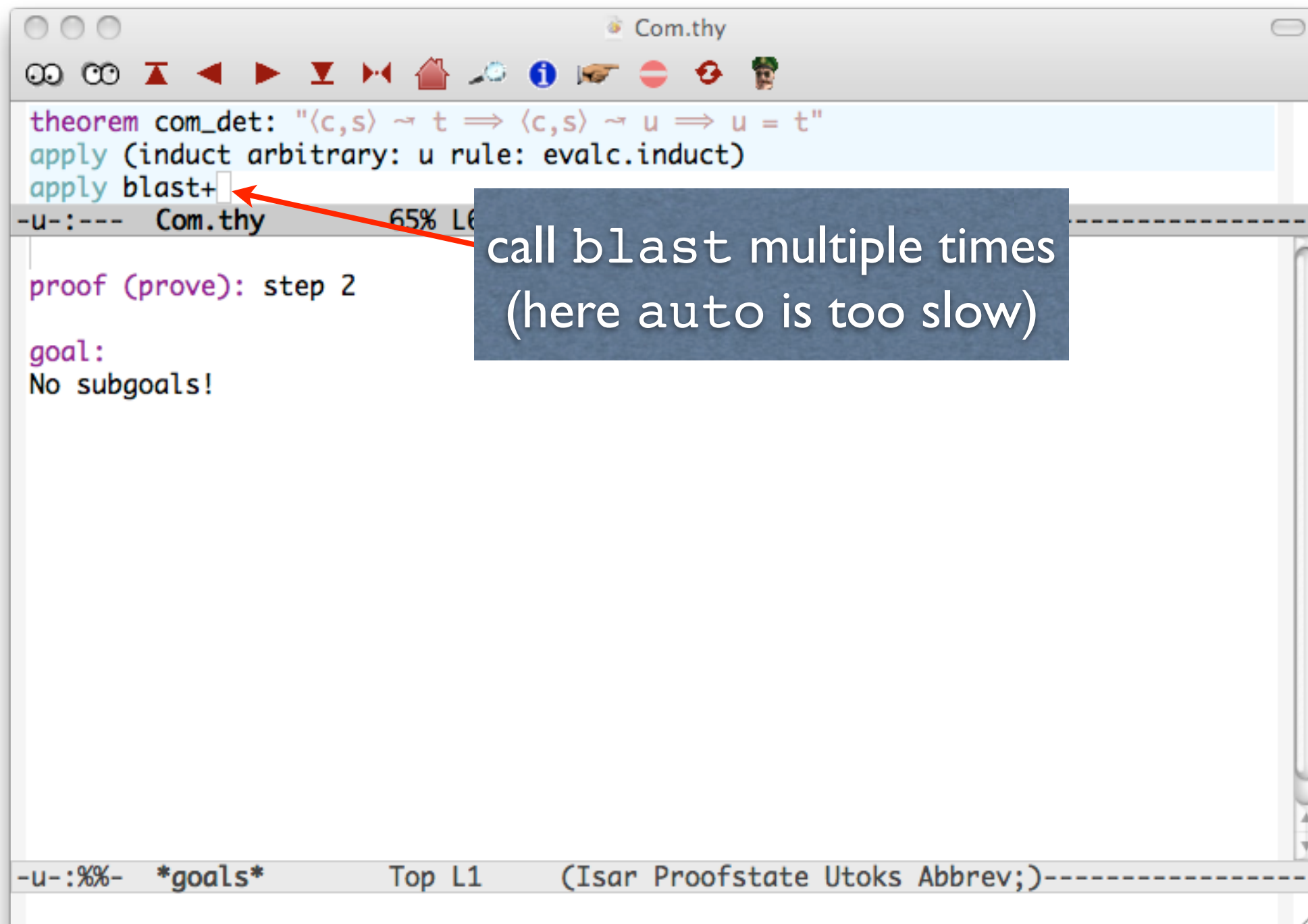
```
theorem com_det: "<c,s> ~ t => <c,s> ~ u => u = t"  
apply (induct arbitrary: u rule: evalc.induct)  
apply blast+
```

Below the code, a status bar indicates the current position: '-u-:--- Com.thy 65% L62 (Isar Utoks Abbrev; Scripting)-----'. The main text area also contains the following text:

```
proof (prove): step 2  
goal:  
No subgoals!
```

At the bottom, another status bar indicates the current goals: '-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----'.

Proved by Rule Inversion



The screenshot shows a theorem prover interface with a window titled 'Com.thy'. The main text area contains the following code:

```
theorem com_det: "<c,s> ~ t => <c,s> ~ u => u = t"  
  apply (induct arbitrary: u rule: evalc.induct)  
  apply blast+
```

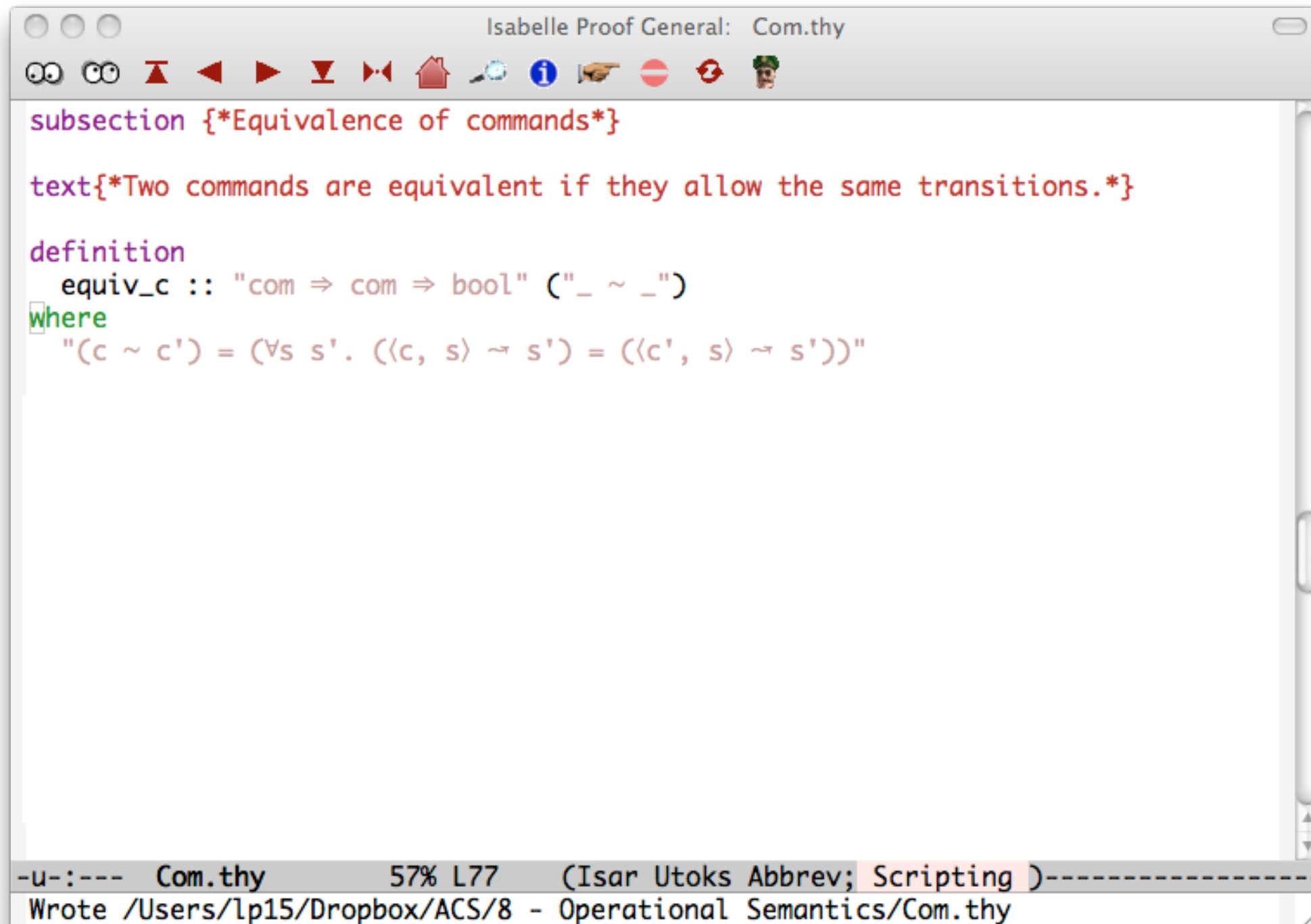
A red arrow points from the 'blast+' command to a blue callout box that says: "call blast multiple times (here auto is too slow)".

Below the main text area, the interface shows the progress of the proof:

```
-u-:--- Com.thy 65% L6  
proof (prove): step 2  
goal:  
No subgoals!
```

The bottom status bar shows: "-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----"

Semantic Equivalence



The screenshot shows the Isabelle Proof General window titled "Isabelle Proof General: Com.thy". The interface includes a toolbar with icons for navigation and editing. The main text area contains the following Isabelle script:

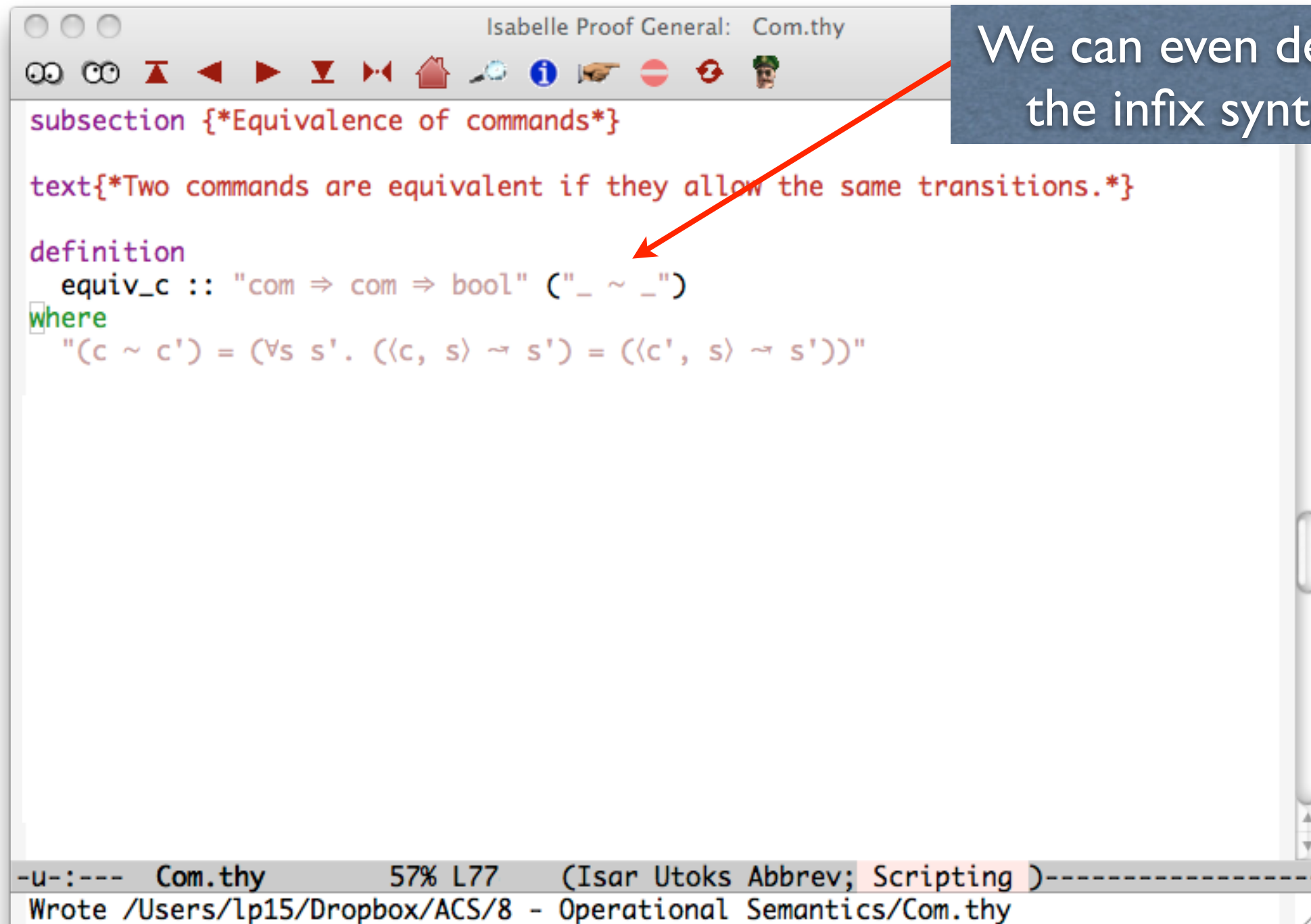
```
subsection {*Equivalence of commands*}

text{*Two commands are equivalent if they allow the same transitions.*}

definition
  equiv_c :: "com  $\Rightarrow$  com  $\Rightarrow$  bool" ("_  $\sim$  _")
where
  "(c  $\sim$  c') = ( $\forall s s'. (\langle c, s \rangle \rightsquigarrow s') = (\langle c', s \rangle \rightsquigarrow s')$ )"
```

The status bar at the bottom indicates the current file is "Com.thy", the cursor is at line 77 (57% down the page), and the current context is "(Isar Utoks Abbrev; Scripting)". It also shows the file path: "Wrote /Users/lp15/Dropbox/ACS/8 - Operational Semantics/Com.thy".

Semantic Equivalence



The screenshot shows the Isabelle Proof General interface with a file named 'Com.thy'. The editor contains the following text:

```
subsection {*Equivalence of commands*}

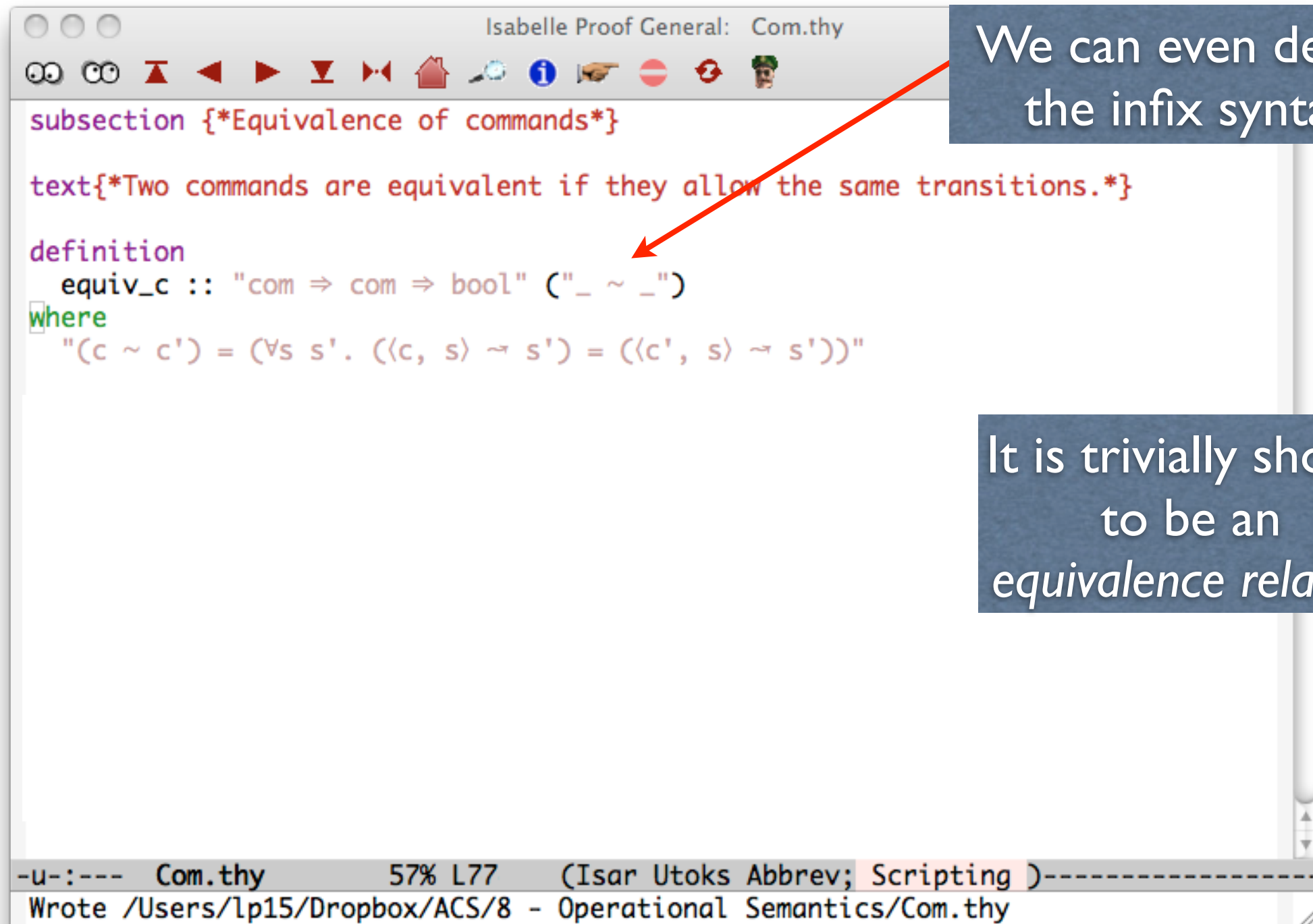
text{*Two commands are equivalent if they allow the same transitions.*}

definition
  equiv_c :: "com  $\Rightarrow$  com  $\Rightarrow$  bool" ("_  $\sim$  _")
where
  "(c  $\sim$  c') = ( $\forall$ s s'. ( $\langle$ c, s $\rangle \rightsquigarrow$  s') = ( $\langle$ c', s $\rangle \rightsquigarrow$  s'))"
```

The status bar at the bottom indicates the current position is 57% L77, and the file path is /Users/lp15/Dropbox/ACS/8 - Operational Semantics/Com.thy.

We can even define
the infix syntax

Semantic Equivalence



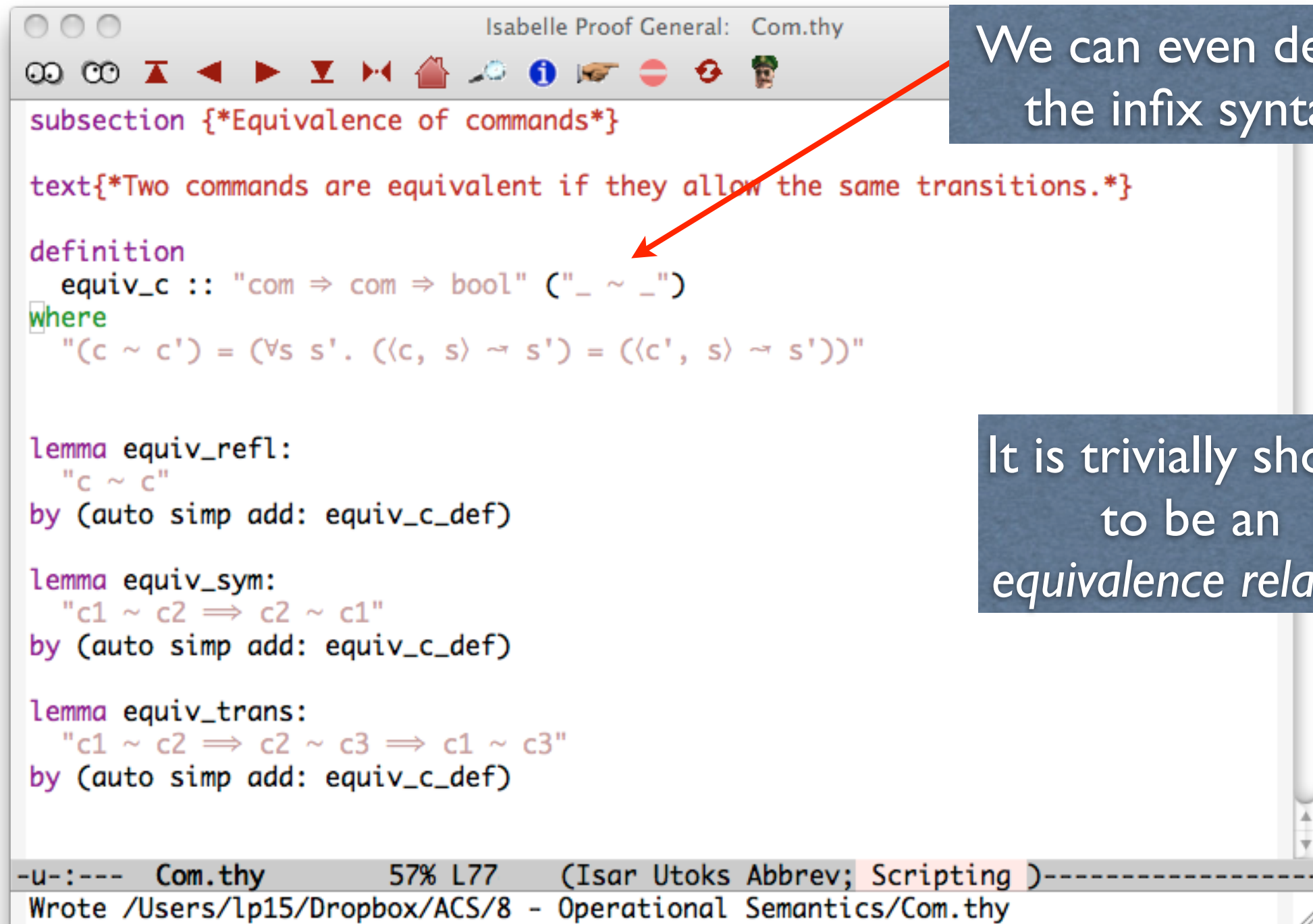
```
Isabelle Proof General: Com.thy
subsection {*Equivalence of commands*}
text{*Two commands are equivalent if they allow the same transitions.*}
definition
  equiv_c :: "com  $\Rightarrow$  com  $\Rightarrow$  bool" ("_  $\sim$  _")
where
  "(c  $\sim$  c') = ( $\forall$ s s'. ( $\langle$ c, s $\rangle \rightsquigarrow$  s') = ( $\langle$ c', s $\rangle \rightsquigarrow$  s'))"
```

The screenshot shows the Isabelle Proof General interface. The title bar reads "Isabelle Proof General: Com.thy". The editor contains a subsection titled "Equivalence of commands" with a text block explaining that two commands are equivalent if they allow the same transitions. A definition is provided for `equiv_c`, which takes two commands and returns a boolean, using the infix notation `_  $\sim$  _`. The definition states that `c  $\sim$  c'` holds if and only if for all states `s` and `s'`, the transition relation `$\langle$ c, s $\rangle \rightsquigarrow$  s'` is equivalent to `$\langle$ c', s $\rangle \rightsquigarrow$  s'`. The status bar at the bottom shows the file path `/Users/lp15/Dropbox/ACS/8 - Operational Semantics/Com.thy` and the current position at line 77.

We can even define the infix syntax

It is trivially shown to be an equivalence relation

Semantic Equivalence



```
Isabelle Proof General: Com.thy

subsection {*Equivalence of commands*}

text{*Two commands are equivalent if they allow the same transitions.*}

definition
  equiv_c :: "com  $\Rightarrow$  com  $\Rightarrow$  bool" ("_  $\sim$  _")
where
  "(c  $\sim$  c') = ( $\forall$ s s'. ( $\langle$ c, s $\rangle \rightsquigarrow$  s') = ( $\langle$ c', s $\rangle \rightsquigarrow$  s'))"

lemma equiv_refl:
  "c  $\sim$  c"
by (auto simp add: equiv_c_def)

lemma equiv_sym:
  "c1  $\sim$  c2  $\Rightarrow$  c2  $\sim$  c1"
by (auto simp add: equiv_c_def)

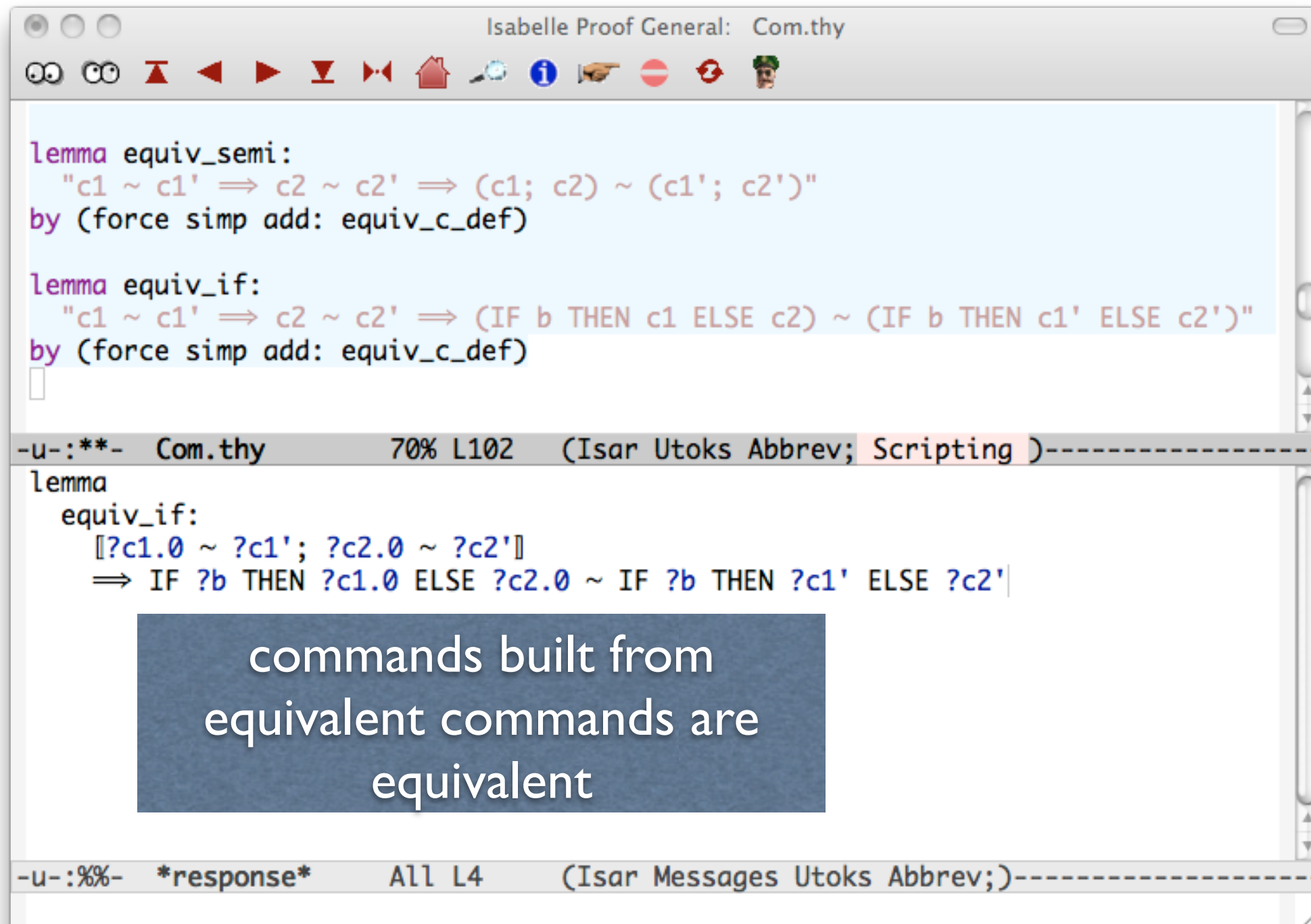
lemma equiv_trans:
  "c1  $\sim$  c2  $\Rightarrow$  c2  $\sim$  c3  $\Rightarrow$  c1  $\sim$  c3"
by (auto simp add: equiv_c_def)
```

-u-:--- Com.thy 57% L77 (Isar Utoks Abbrev; Scripting)-----
Wrote /Users/lp15/Dropbox/ACS/8 - Operational Semantics/Com.thy

We can even define
the infix syntax

It is trivially shown
to be an
equivalence relation

More Semantic Equivalence!



The screenshot shows the Isabelle Proof General interface with the file `Com.thy` open. The main editor contains two lemmas:

```
lemma equiv_semi:
  "c1 ~ c1'  $\Rightarrow$  c2 ~ c2'  $\Rightarrow$  (c1; c2) ~ (c1'; c2')"
by (force simp add: equiv_c_def)

lemma equiv_if:
  "c1 ~ c1'  $\Rightarrow$  c2 ~ c2'  $\Rightarrow$  (IF b THEN c1 ELSE c2) ~ (IF b THEN c1' ELSE c2')"
by (force simp add: equiv_c_def)
```

The lower pane shows the semantic representation of the `equiv_if` lemma:

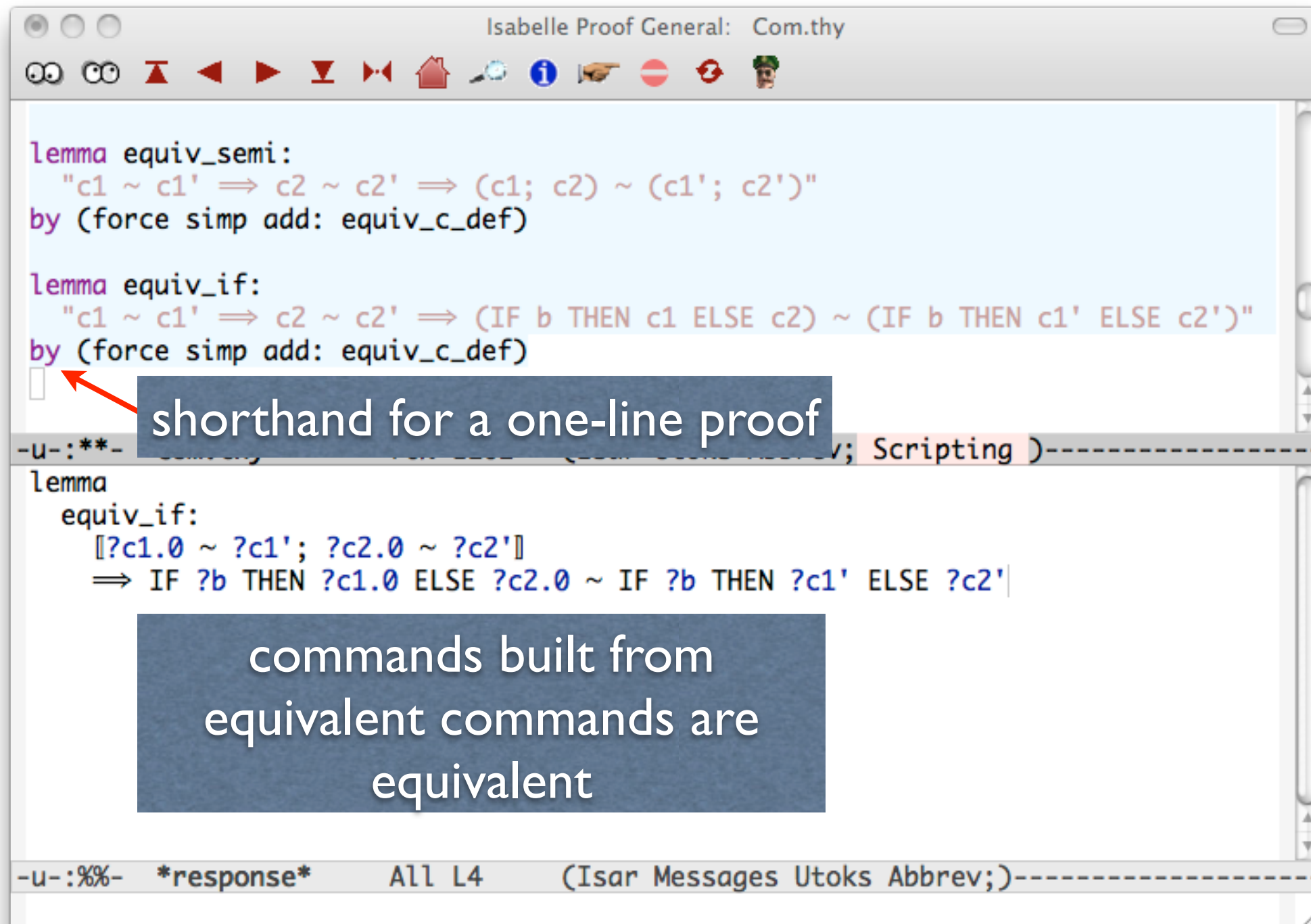
```
lemma
  equiv_if:
    [[?c1.0 ~ ?c1'; ?c2.0 ~ ?c2']]
     $\Rightarrow$  IF ?b THEN ?c1.0 ELSE ?c2.0 ~ IF ?b THEN ?c1' ELSE ?c2'
```

A blue box with white text is overlaid on the lower pane, stating:

commands built from
equivalent commands are
equivalent

The status bar at the bottom indicates the current context is `*response*` at line 4, and the message pane shows the message `(Isar Messages Utoks Abbrev;)`.

More Semantic Equivalence!



The screenshot shows the Isabelle Proof General interface with a window titled "Isabelle Proof General: Com.thy". The interface includes a toolbar with various icons for navigation and editing. The main text area contains two lemmas:

```
lemma equiv_semi:
  "c1 ~ c1'  $\Rightarrow$  c2 ~ c2'  $\Rightarrow$  (c1; c2) ~ (c1'; c2')"
by (force simp add: equiv_c_def)

lemma equiv_if:
  "c1 ~ c1'  $\Rightarrow$  c2 ~ c2'  $\Rightarrow$  (IF b THEN c1 ELSE c2) ~ (IF b THEN c1' ELSE c2')"
by (force simp add: equiv_c_def)
```

A red arrow points from the text "shorthand for a one-line proof" to the `by` keyword in the `equiv_if` lemma.

Below the main text area, there is a section titled "Scripting" which shows the internal representation of the `equiv_if` lemma:

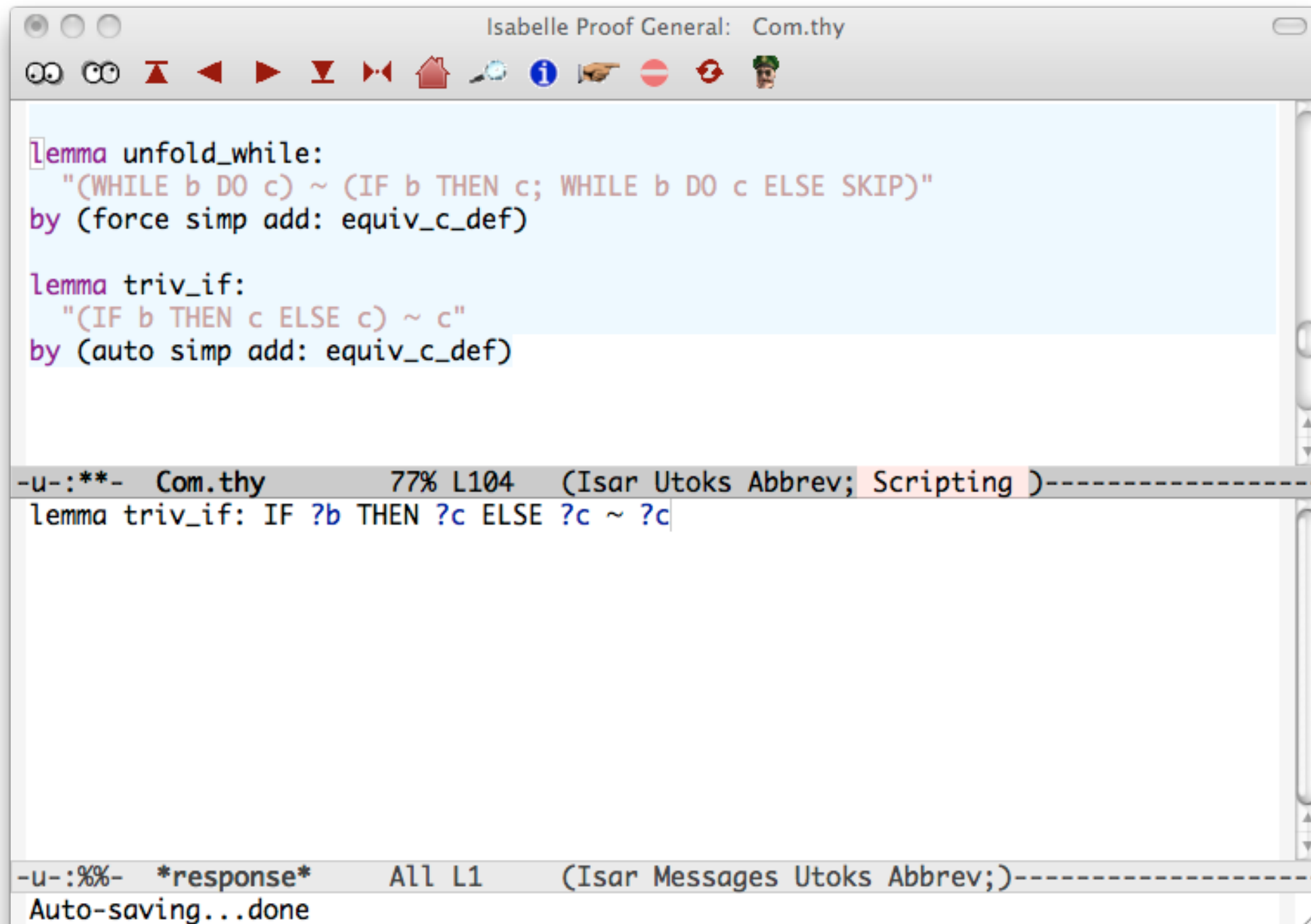
```
lemma
  equiv_if:
    [[?c1.0 ~ ?c1'; ?c2.0 ~ ?c2']]
     $\Rightarrow$  IF ?b THEN ?c1.0 ELSE ?c2.0 ~ IF ?b THEN ?c1' ELSE ?c2'
```

A blue box with white text is overlaid on the bottom part of the screenshot, containing the text:

commands built from
equivalent commands are
equivalent

The bottom status bar shows the following information: `-u-:%%- *response* All L4 (Isar Messages Utoks Abbrev;)`

And More!!



The screenshot shows the Isabelle Proof General interface. The main editor displays two lemmas:

```
lemma unfold_while:  
  "(WHILE b DO c) ~ (IF b THEN c; WHILE b DO c ELSE SKIP)"  
by (force simp add: equiv_c_def)  
  
lemma triv_if:  
  "(IF b THEN c ELSE c) ~ c"  
by (auto simp add: equiv_c_def)
```

The bottom-left panel shows the message for the second lemma:

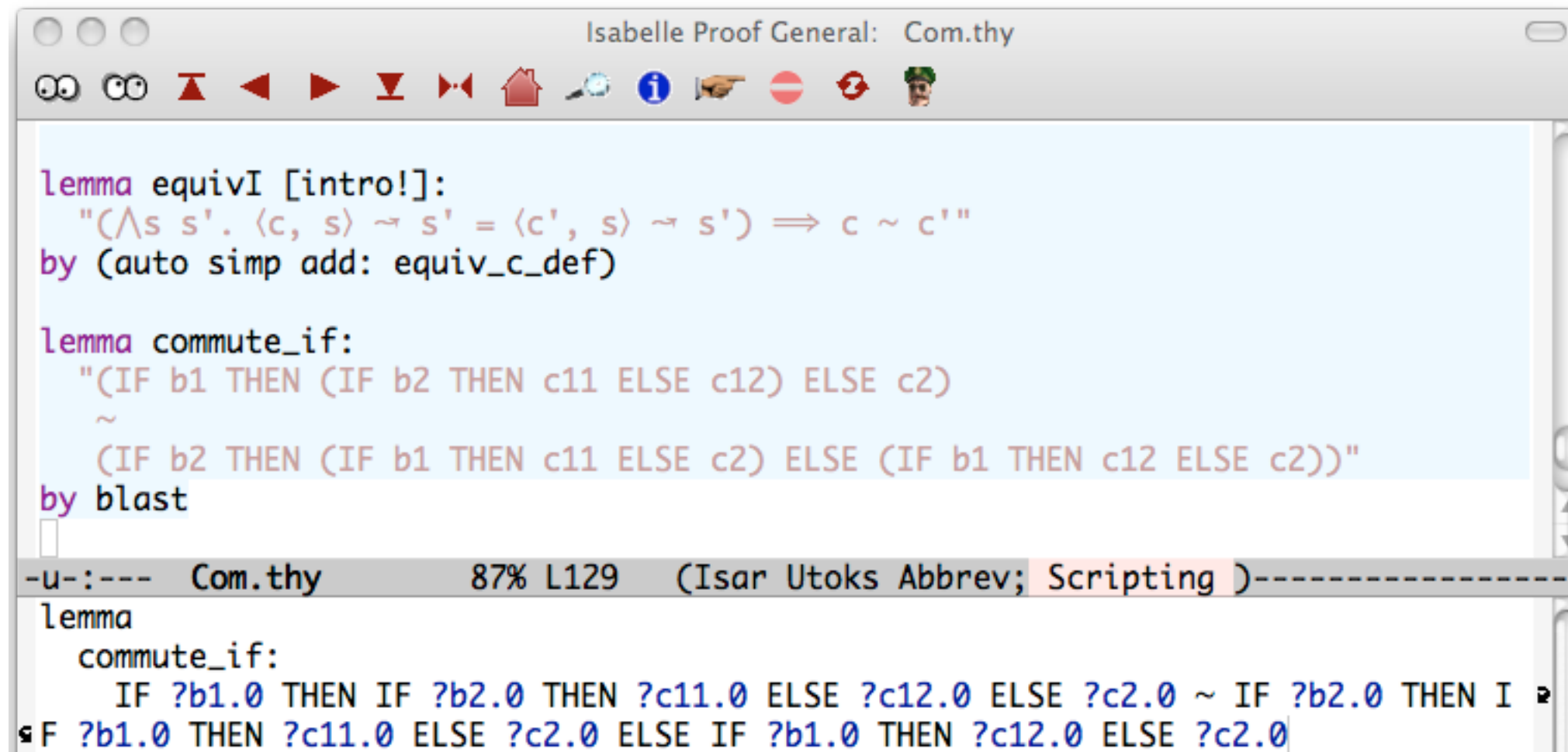
```
-u-:**- Com.thy          77% L104  (Isar Utoks Abbrev; Scripting )-----  
lemma triv_if: IF ?b THEN ?c ELSE ?c ~ ?c
```

The bottom-right panel shows the response message:

```
-u-:%%-  *response*      All L1    (Isar Messages Utoks Abbrev;)-----  
Auto-saving...done
```

A New Introduction Rule

$$\frac{\langle c, s \rangle \rightarrow s' \iff \langle c', s \rangle \rightarrow s'}{c \sim c'} \quad s \text{ and } s' \text{ not free...}$$



The screenshot shows the Isabelle Proof General interface with the file 'Com.thy' open. The main editor displays the following code:

```

lemma equivI [intro!]:
  "(\s s'. \langle c, s \rangle \rightsquigarrow s' = \langle c', s \rangle \rightsquigarrow s') \implies c \sim c'"
by (auto simp add: equiv_c_def)

lemma commute_if:
  "(IF b1 THEN (IF b2 THEN c11 ELSE c12) ELSE c2)
  ~
  (IF b2 THEN (IF b1 THEN c11 ELSE c2) ELSE (IF b1 THEN c12 ELSE c2))"
by blast
  
```

The status bar at the bottom indicates the current position is 87% L129 in 'Com.thy', with the text '(Isar Utoks Abbrev; Scripting)'.

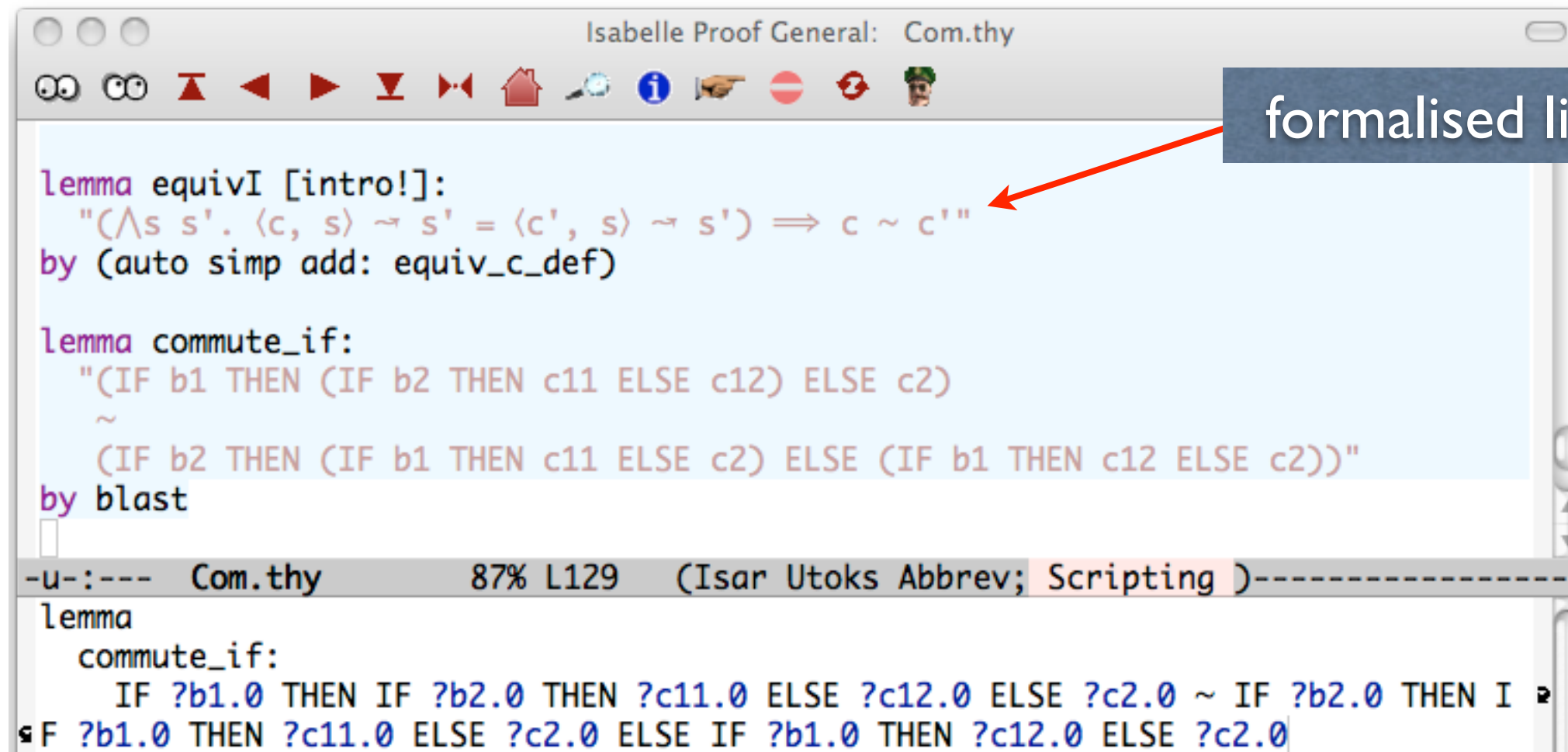
Below the status bar, a preview of the lemma 'commute_if' is shown in a different font style, using Isabelle's internal notation for variables and logical connectives:

```

lemma
  commute_if:
    IF ?b1.0 THEN IF ?b2.0 THEN ?c11.0 ELSE ?c12.0 ELSE ?c2.0 ~ IF ?b2.0 THEN I
  IF ?b1.0 THEN ?c11.0 ELSE ?c2.0 ELSE IF ?b1.0 THEN ?c12.0 ELSE ?c2.0
  
```

A New Introduction Rule

$$\frac{\langle c, s \rangle \rightarrow s' \iff \langle c', s \rangle \rightarrow s'}{c \sim c'} \quad s \text{ and } s' \text{ not free...}$$



Isabelle Proof General: Com.thy

```
lemma equivI [intro!]:  
  "(\s s'. \langle c, s \rangle \rightsquigarrow s' = \langle c', s \rangle \rightsquigarrow s') \implies c \sim c'"  
by (auto simp add: equiv_c_def)  
  
lemma commute_if:  
  "(IF b1 THEN (IF b2 THEN c11 ELSE c12) ELSE c2)  
  ~  
  (IF b2 THEN (IF b1 THEN c11 ELSE c2) ELSE (IF b1 THEN c12 ELSE c2))"  
by blast
```

-u-:--- Com.thy 87% L129 (Isar Utoks Abbrev; Scripting)-----

```
lemma  
  commute_if:  
    IF ?b1.0 THEN IF ?b2.0 THEN ?c11.0 ELSE ?c12.0 ELSE ?c2.0 ~ IF ?b2.0 THEN I  
IF ?b1.0 THEN ?c11.0 ELSE ?c2.0 ELSE IF ?b1.0 THEN ?c12.0 ELSE ?c2.0
```

formalised like this

A New Introduction Rule

$$\frac{\langle c, s \rangle \rightarrow s' \iff \langle c', s \rangle \rightarrow s'}{c \sim c'} \quad s \text{ and } s' \text{ not free...}$$

declared like this

```
lemma equivI [intro!]:  
  "(\s s'. \langle c, s \rangle \rightsquigarrow s' = \langle c', s \rangle \rightsquigarrow s') \implies c \sim c'"  
by (auto simp add: equiv_c_def)
```

```
lemma commute_if:  
  "(IF b1 THEN (IF b2 THEN c11 ELSE c12) ELSE c2)  
  ~  
  (IF b2 THEN (IF b1 THEN c11 ELSE c2) ELSE (IF b1 THEN c12 ELSE c2))"  
by blast
```

formalised like this

```
-u-:--- Com.thy      87% L129  (Isar Utoks Abbrev; Scripting )-----  
lemma  
  commute_if:  
    IF ?b1.0 THEN IF ?b2.0 THEN ?c11.0 ELSE ?c12.0 ELSE ?c2.0 ~ IF ?b2.0 THEN I  
IF ?b1.0 THEN ?c11.0 ELSE ?c2.0 ELSE IF ?b1.0 THEN ?c12.0 ELSE ?c2.0
```

A New Introduction Rule

$$\frac{\langle c, s \rangle \rightarrow s' \iff \langle c', s \rangle \rightarrow s'}{c \sim c'} \quad s \text{ and } s' \text{ not free...}$$

declared like this

```
lemma equivI [intro!]:  
  "(\s s'. \langle c, s \rangle \rightsquigarrow s' = \langle c', s \rangle \rightsquigarrow s') \implies c \sim c'"  
by (auto simp add: equiv_c_def)  
  
lemma commute_if:  
  "(IF b1 THEN (IF b2 THEN c11 ELSE c12) ELSE c2)  
  ~  
  (IF b2 THEN (IF b1 THEN c11 ELSE c2) ELSE (IF b1 THEN c12 ELSE c2))"  
by blast
```

formalised like this

used *implicitly* like this

```
-u-:--- Com.thy Abbrev; Scripting )-----  
lemma  
  commute_if:  
    IF ?b1.0 THEN IF ?b2.0 THEN ?c11.0 ELSE ?c12.0 ELSE ?c2.0 ~ IF ?b2.0 THEN I  
IF ?b1.0 THEN ?c11.0 ELSE ?c2.0 ELSE IF ?b1.0 THEN ?c12.0 ELSE ?c2.0
```

Final Remarks on Semantics

Final Remarks on Semantics

- Small-step semantics is treated similarly.

Final Remarks on Semantics

- Small-step semantics is treated similarly.
- Variable binding is crucial in larger examples, and should be formalised using the *nominal package*.
 - choosing a fresh variable
 - renaming bound variables consistently

Final Remarks on Semantics

- Small-step semantics is treated similarly.
- Variable binding is crucial in larger examples, and should be formalised using the *nominal package*.
 - choosing a fresh variable
 - renaming bound variables consistently
- Serious proofs will be complex and difficult!