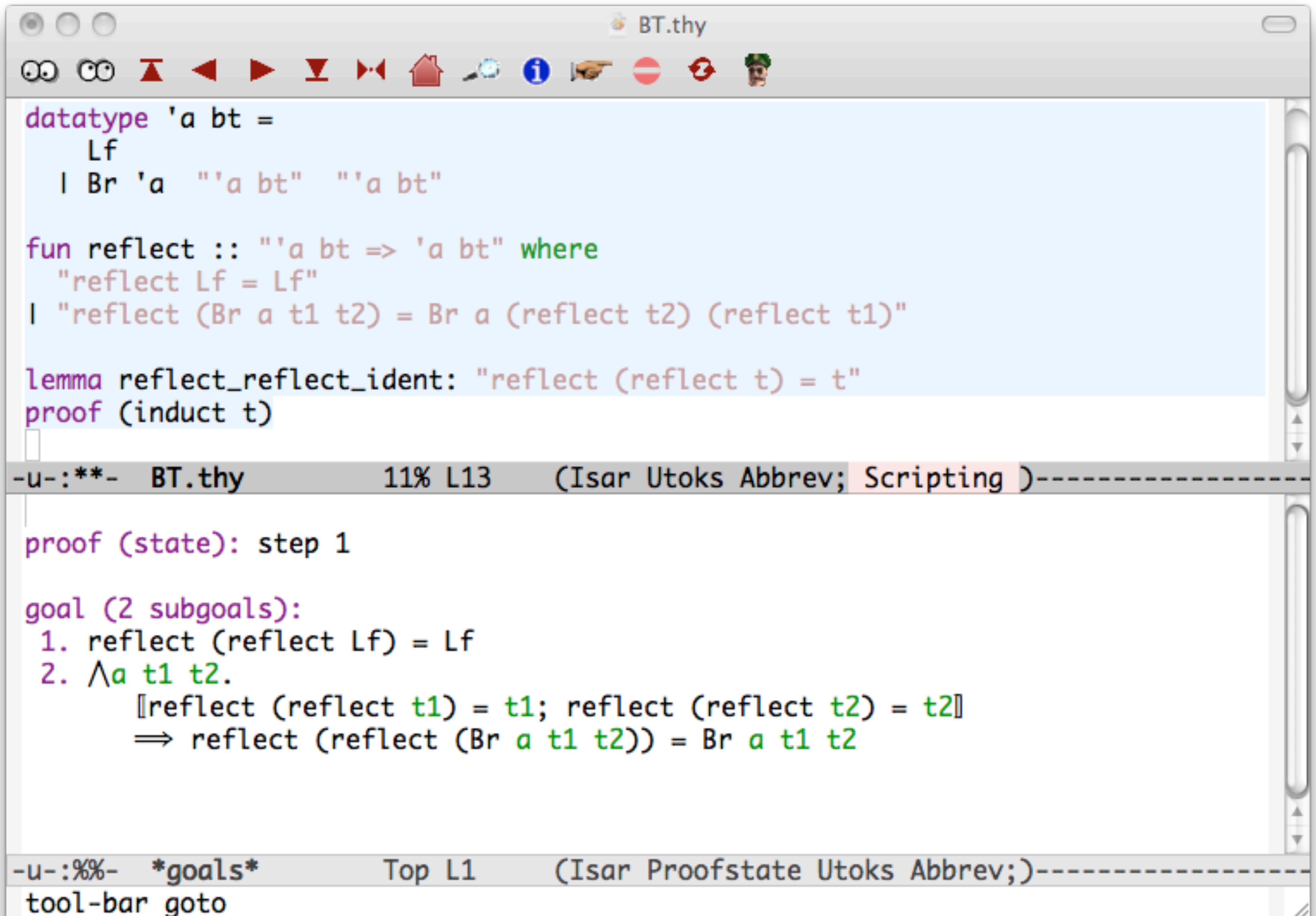
Interactive Formal Verification
11: Structured Induction Proofs

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

A Proof about Binary Trees



A Proof about Binary Trees

BT.thy

```
datatype 'a bt =
    Lf
  | Br 'a  "'a bt"  "'a bt"

fun reflect :: "'a bt => 'a bt" where
  "reflect Lf = Lf"
| "reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)"

lemma reflect_reflect_ident: "reflect (reflect t) = t"
proof (induct t)
```

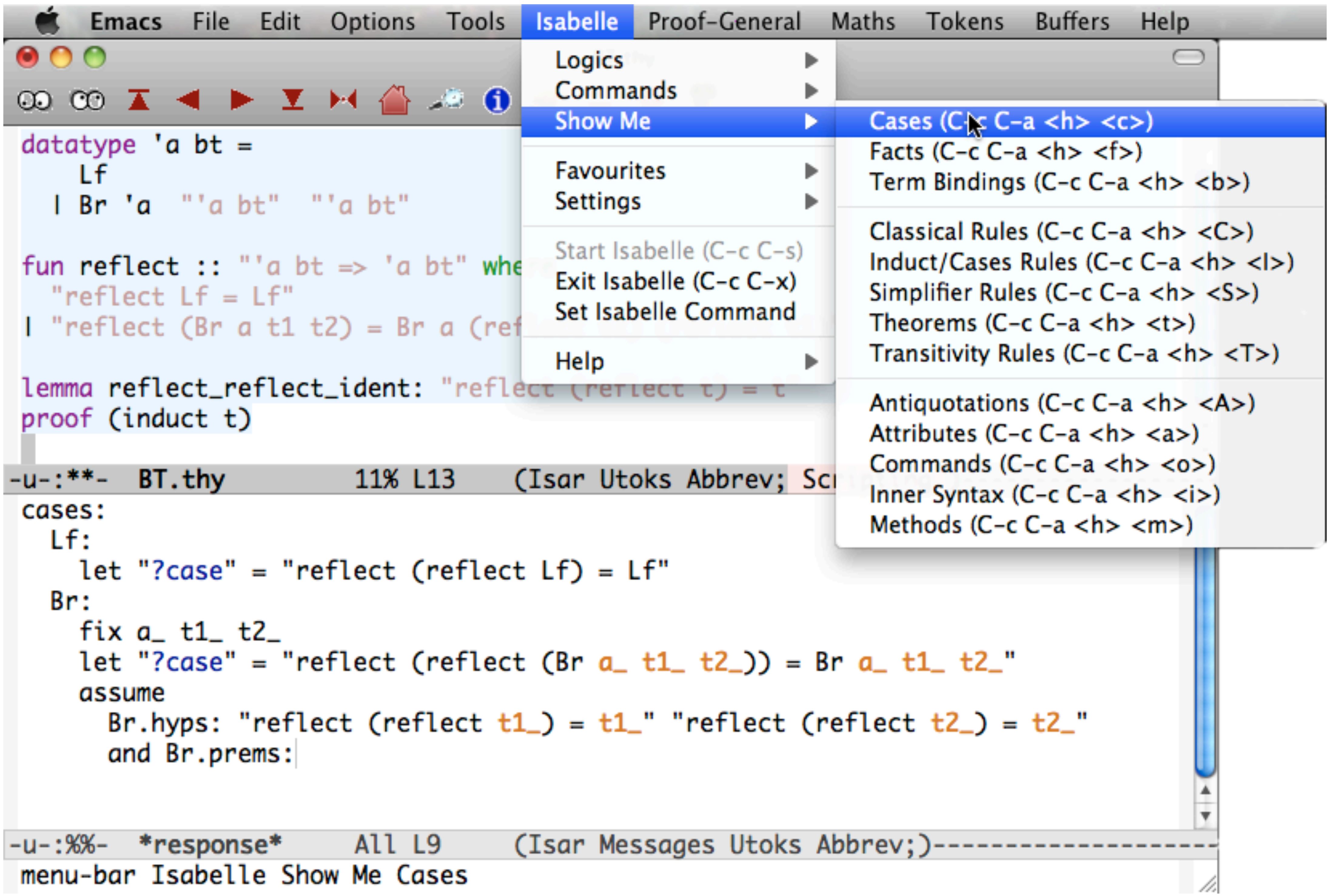
```
-u-:**-  BT.thy       11% L13     (Isar Utoks Abbrev; Scripting )----------------
```

```
proof (state): step 1

goal (2 subgoals):
 1. reflect (reflect Lf) = Lf
 2. ⋀a t1 t2.
       ⟦reflect (reflect t1) = t1; reflect (reflect t2) = t2⟧
       ⟹ reflect (reflect (Br a t1 t2)) = Br a t1 t2
```

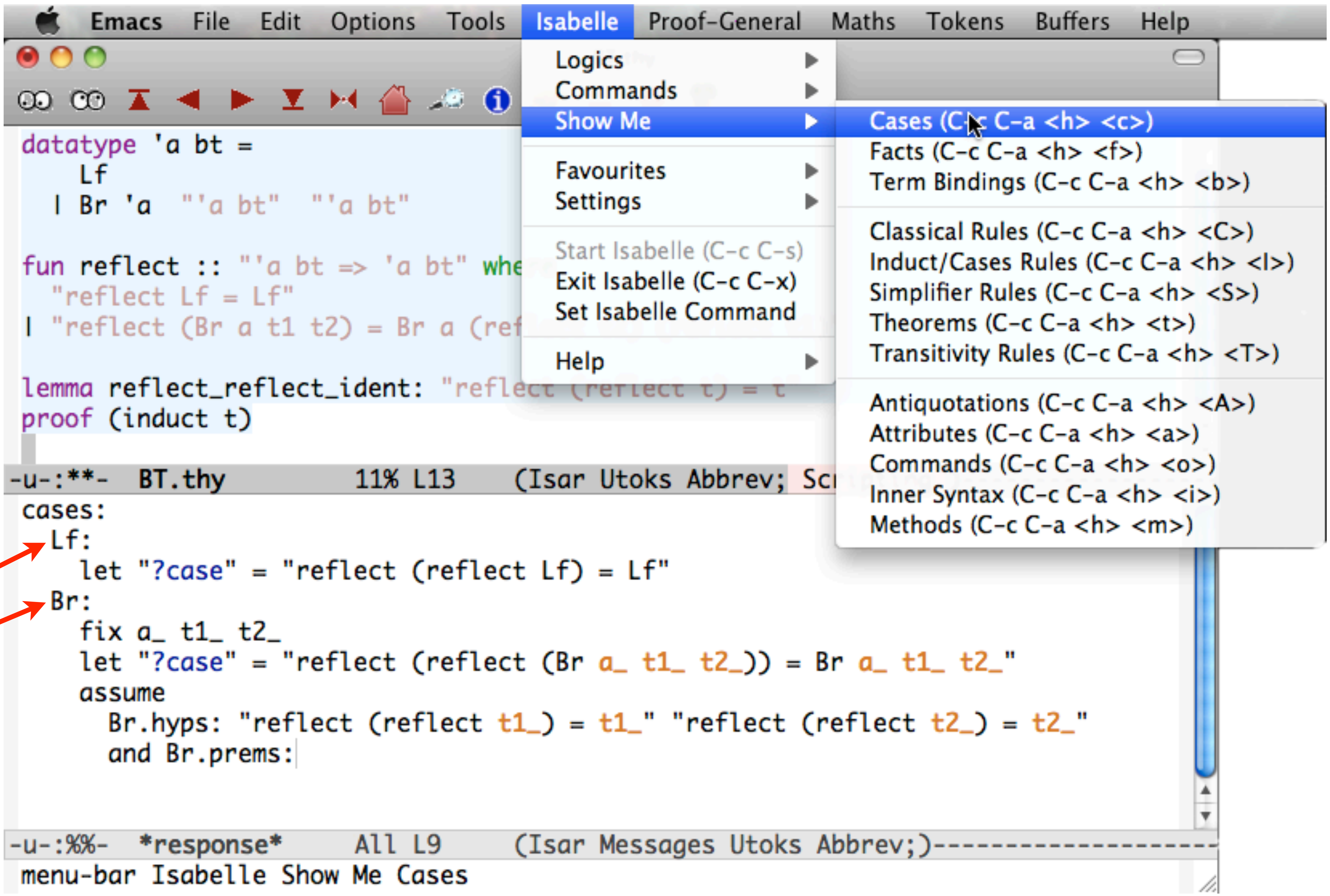
```
-u-:%%-  *goals*      Top L1      (Isar Proofstate Utoks Abbrev;)----------------
tool-bar goto
```

Must we copy each case and such big contexts?

Finding Predefined Cases

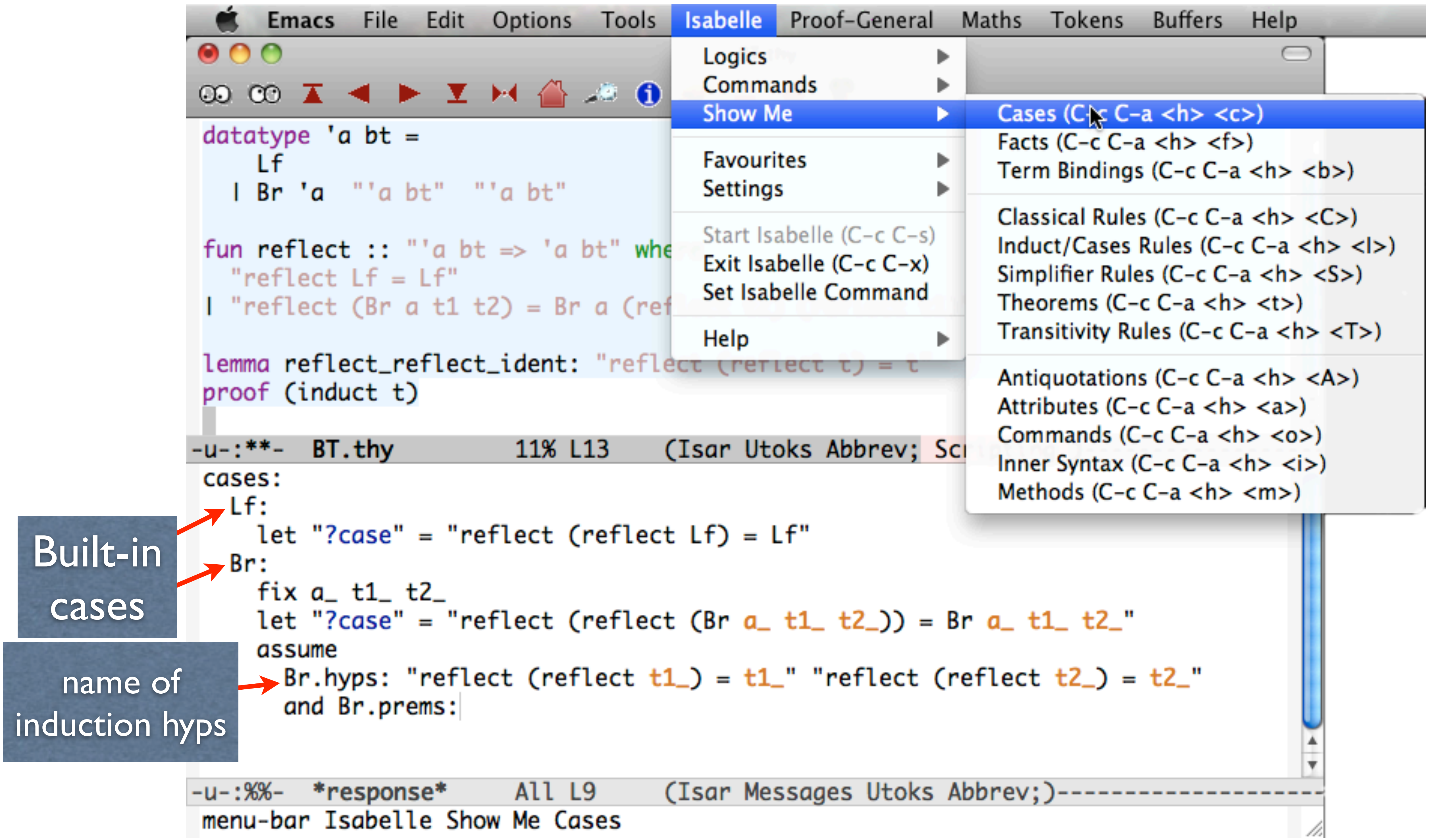


Finding Predefined Cases

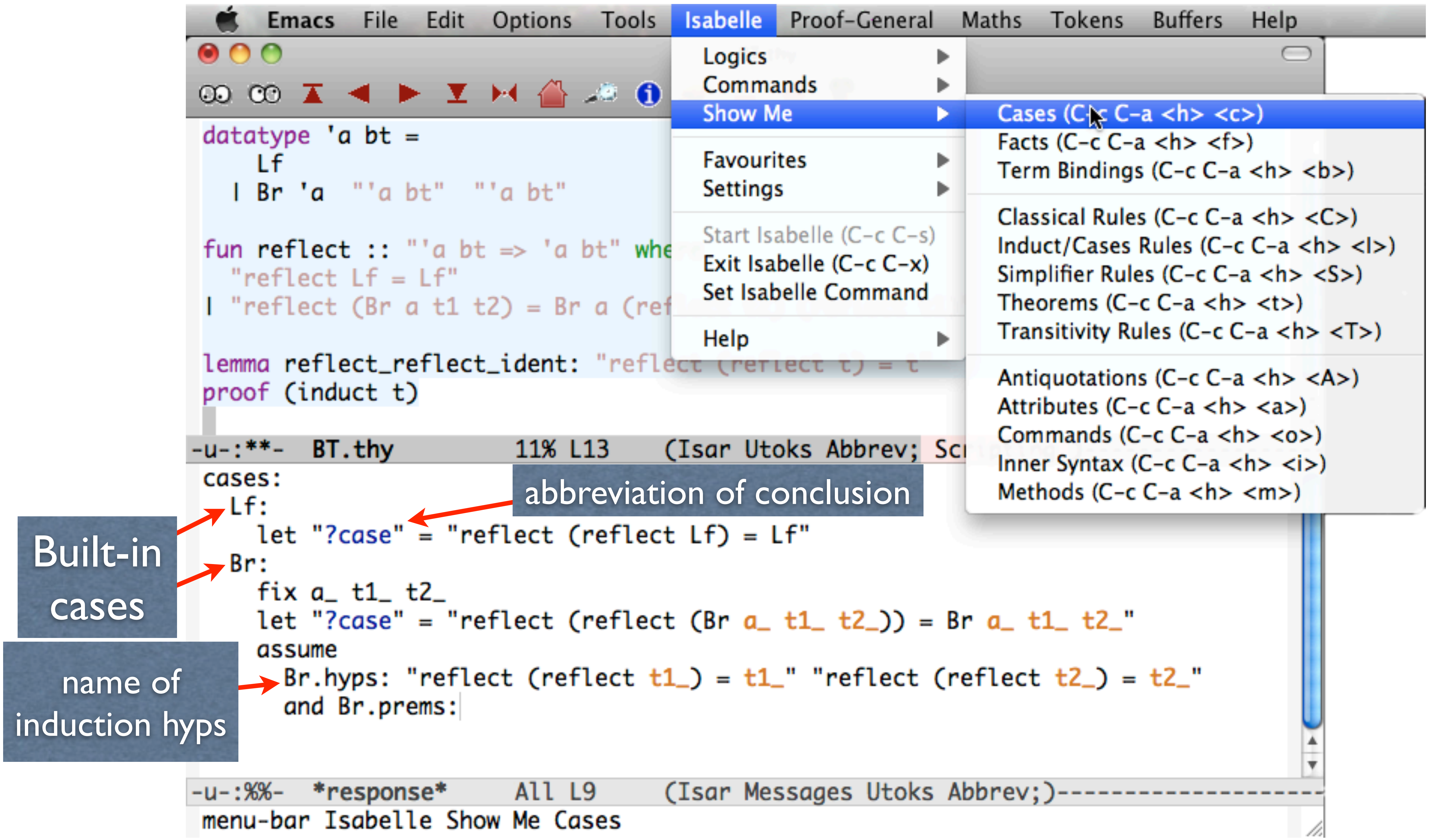


Built-in cases

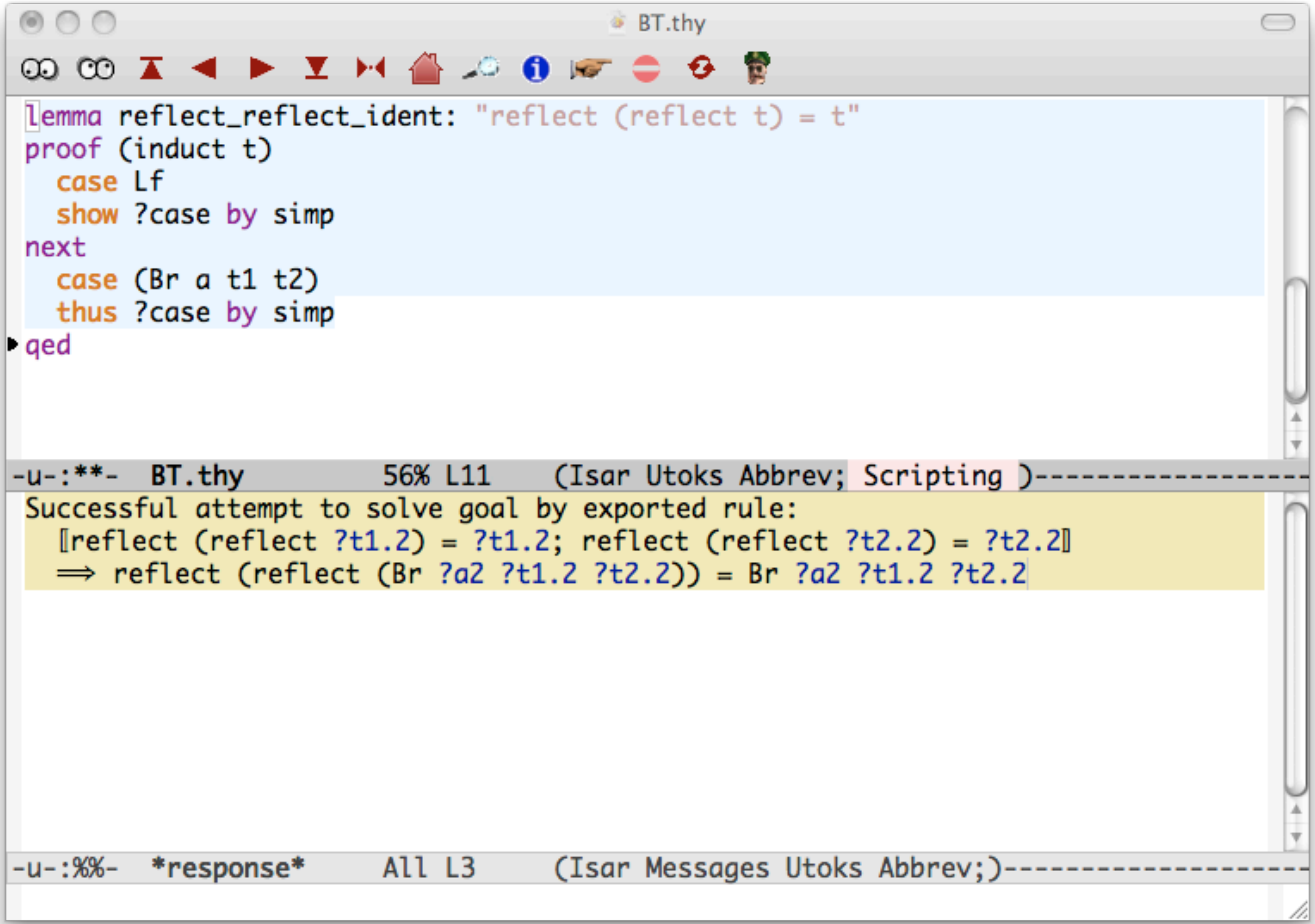
Finding Predefined Cases



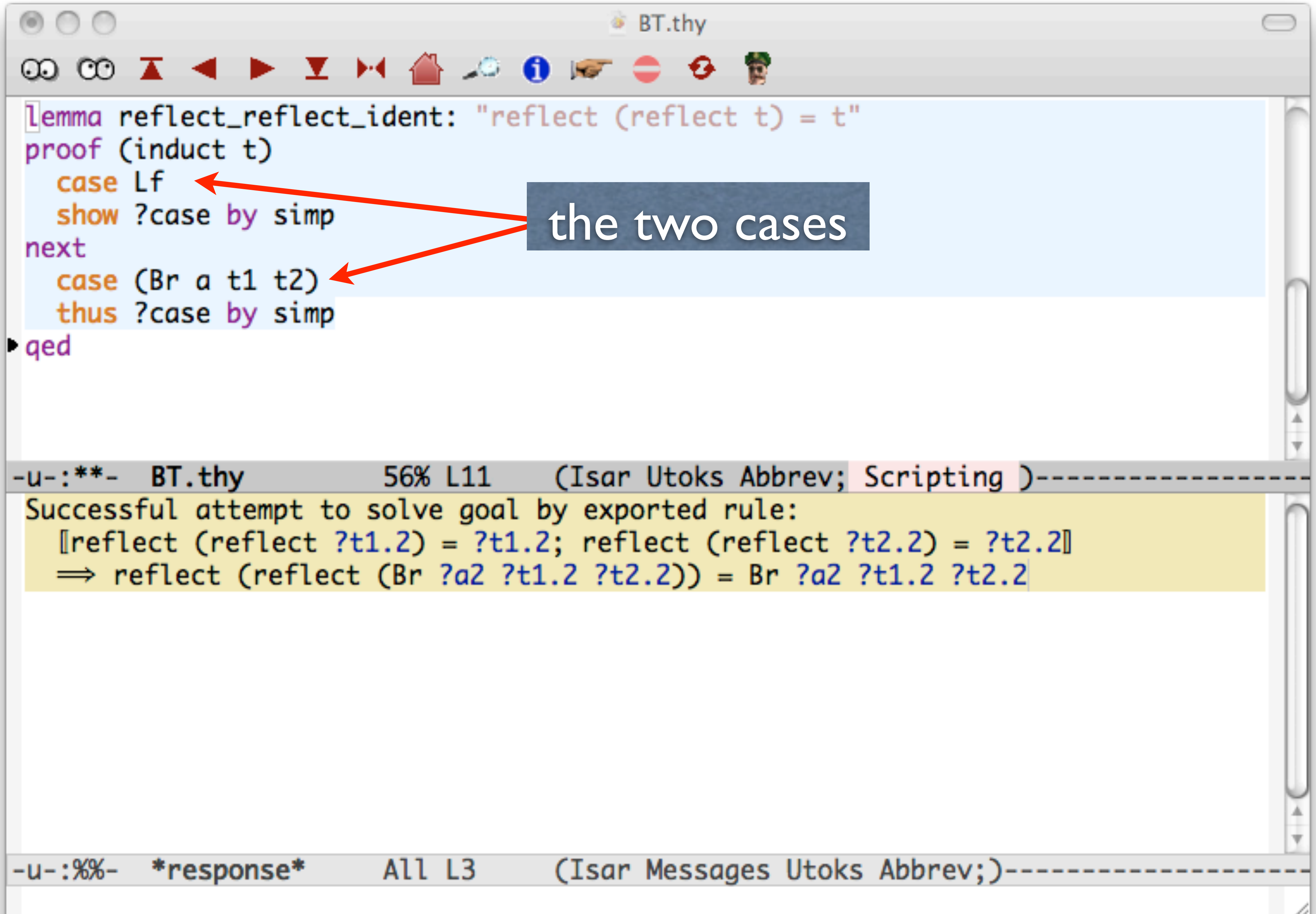
Finding Predefined Cases



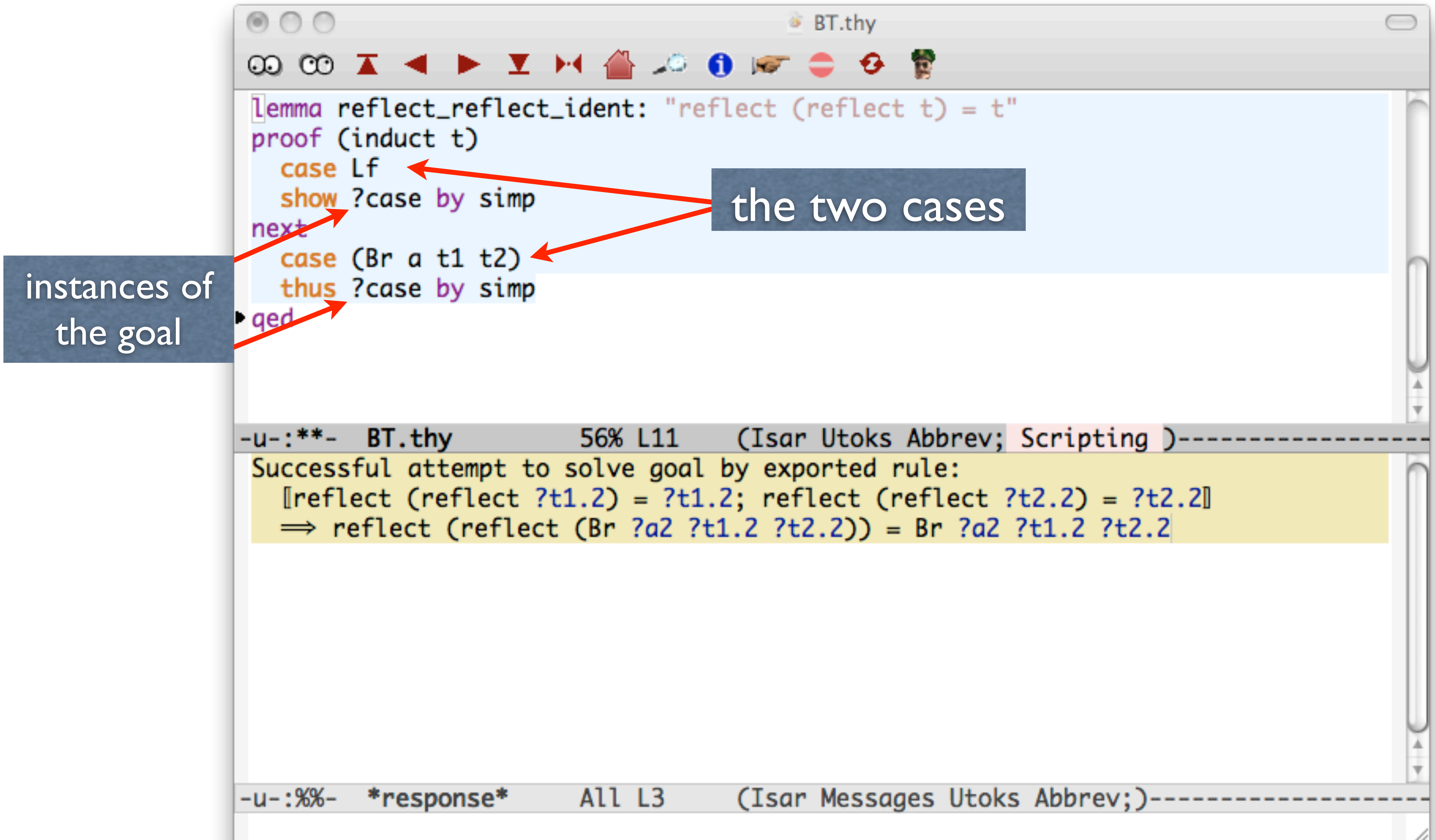
The Finished Proof



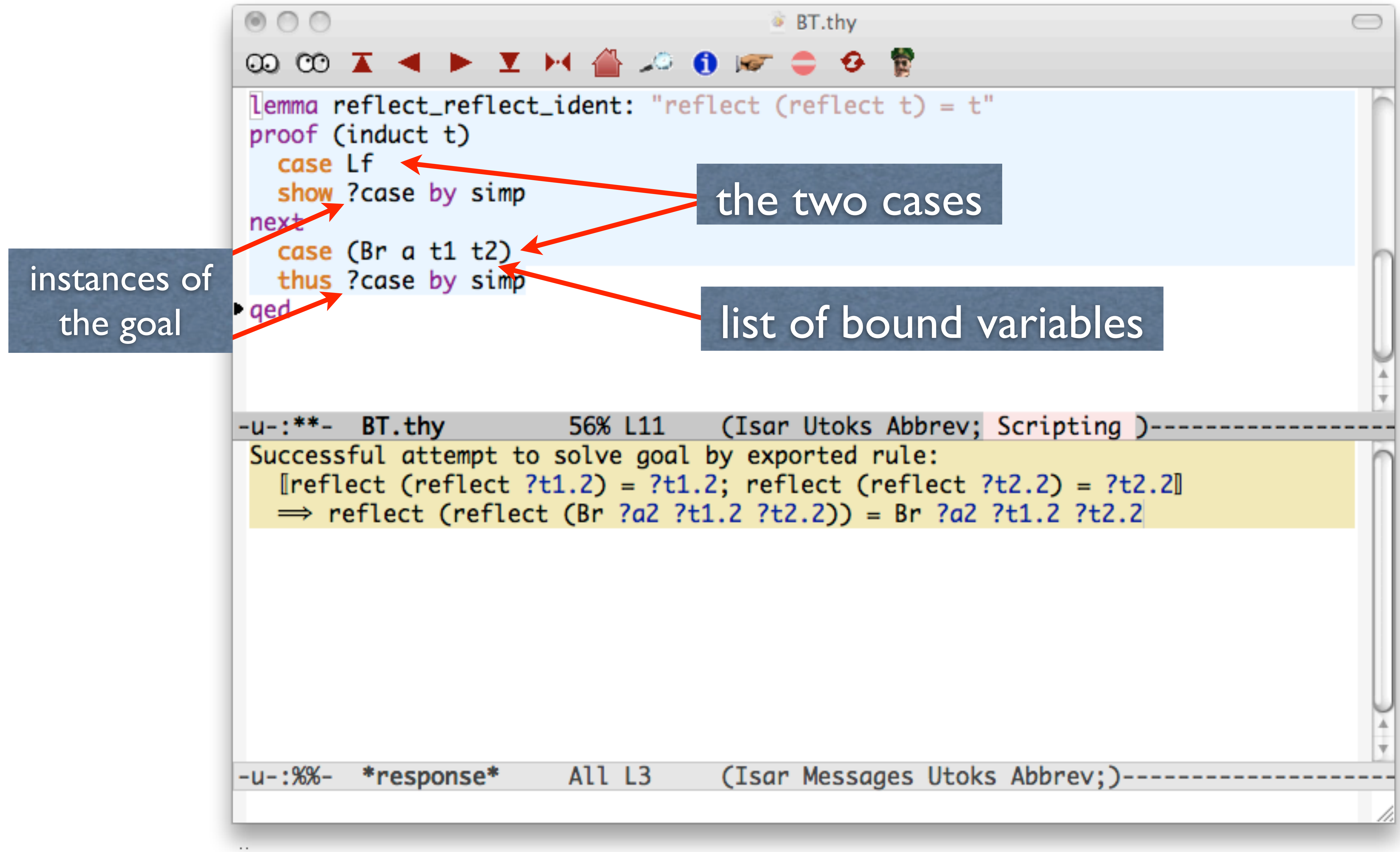
The Finished Proof



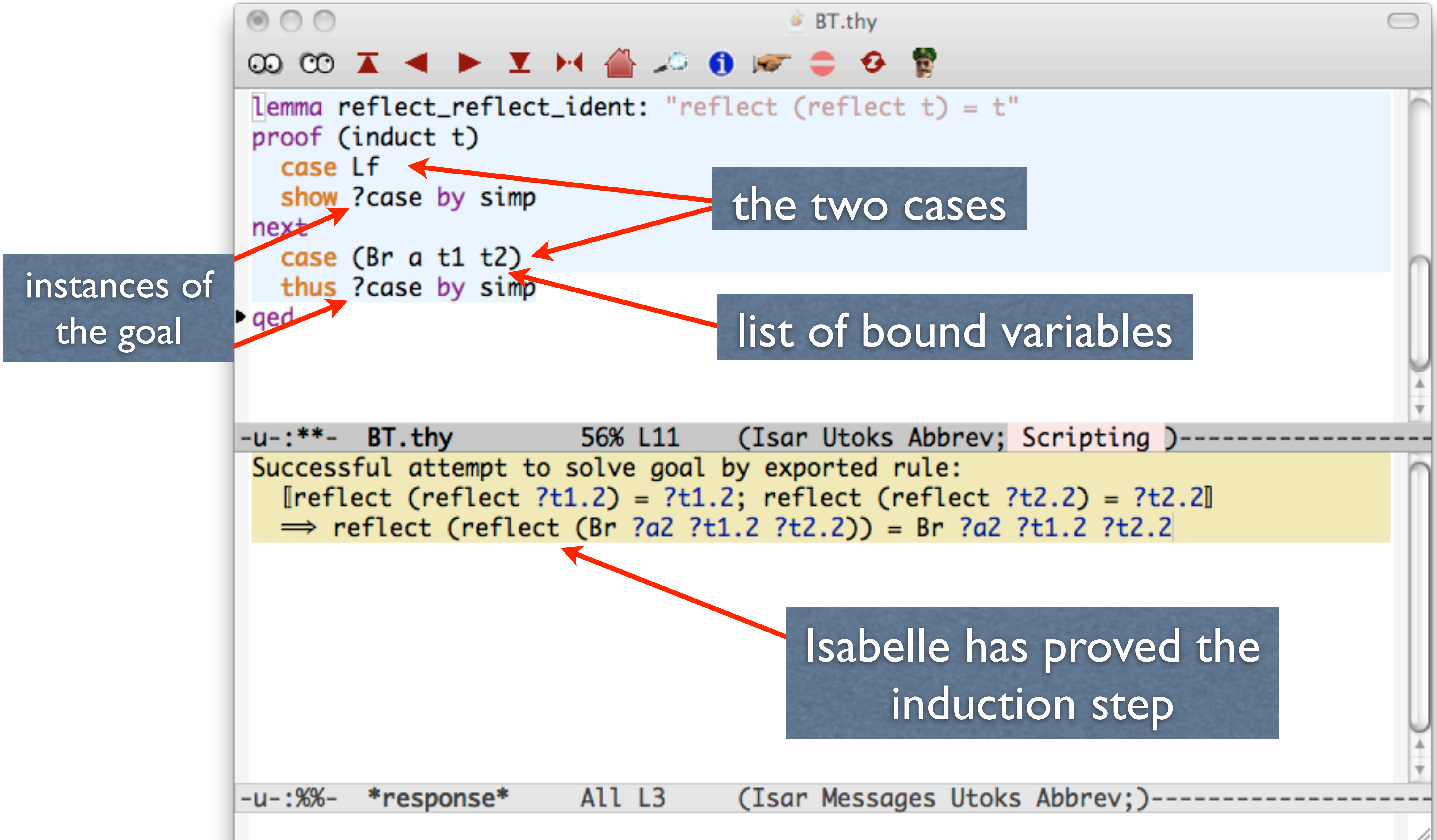
The Finished Proof



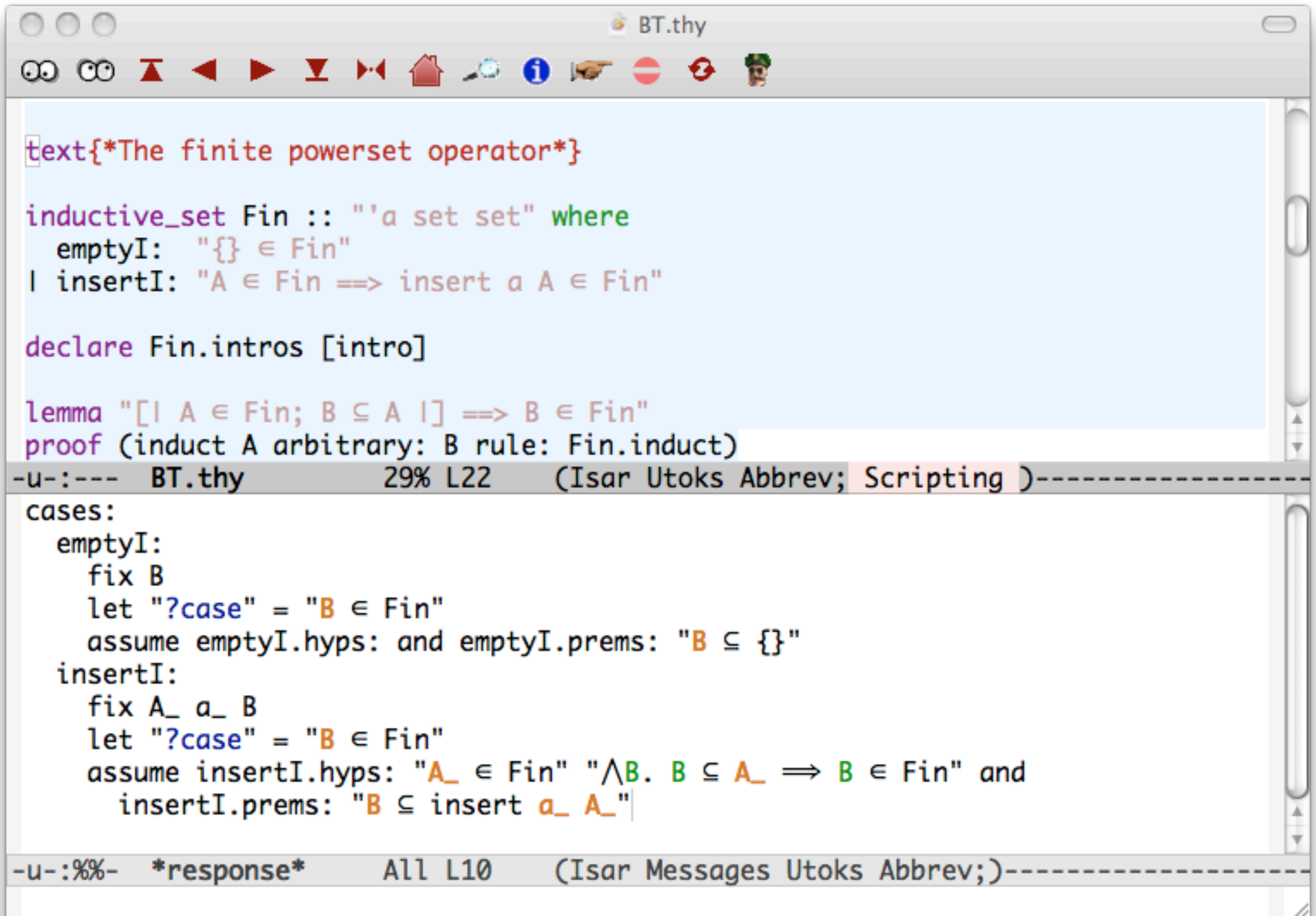
The Finished Proof



The Finished Proof



A More Sophisticated Proof



A More Sophisticated Proof

BT.thy

```
text{*The finite powerset operator*}

inductive_set Fin :: "'a set set" where
  emptyI:  "{} ∈ Fin"
| insertI: "A ∈ Fin ==> insert a A ∈ Fin"

declare Fin.intros [intro]

lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
-u-:---  BT.thy        29% L22     (Isar Utoks Abbrev; Scripting )------------------
cases:
  emptyI:
    fix B
    let "?case" = "B ∈ Fin"
    assume emptyI.hyps: and emptyI.prems: "B ⊆ {}"
  insertI:
    fix A_ a_ B
    let "?case" = "B ∈ Fin"
    assume insertI.hyps: "A_ ∈ Fin" "⋀B. B ⊆ A_ ⟹ B ∈ Fin" and
      insertI.prems: "B ⊆ insert a_ A_"
-u-:%%-  *response*    All L10     (Isar Messages Utoks Abbrev;)------------------
```

a *named* induction rule

A More Sophisticated Proof

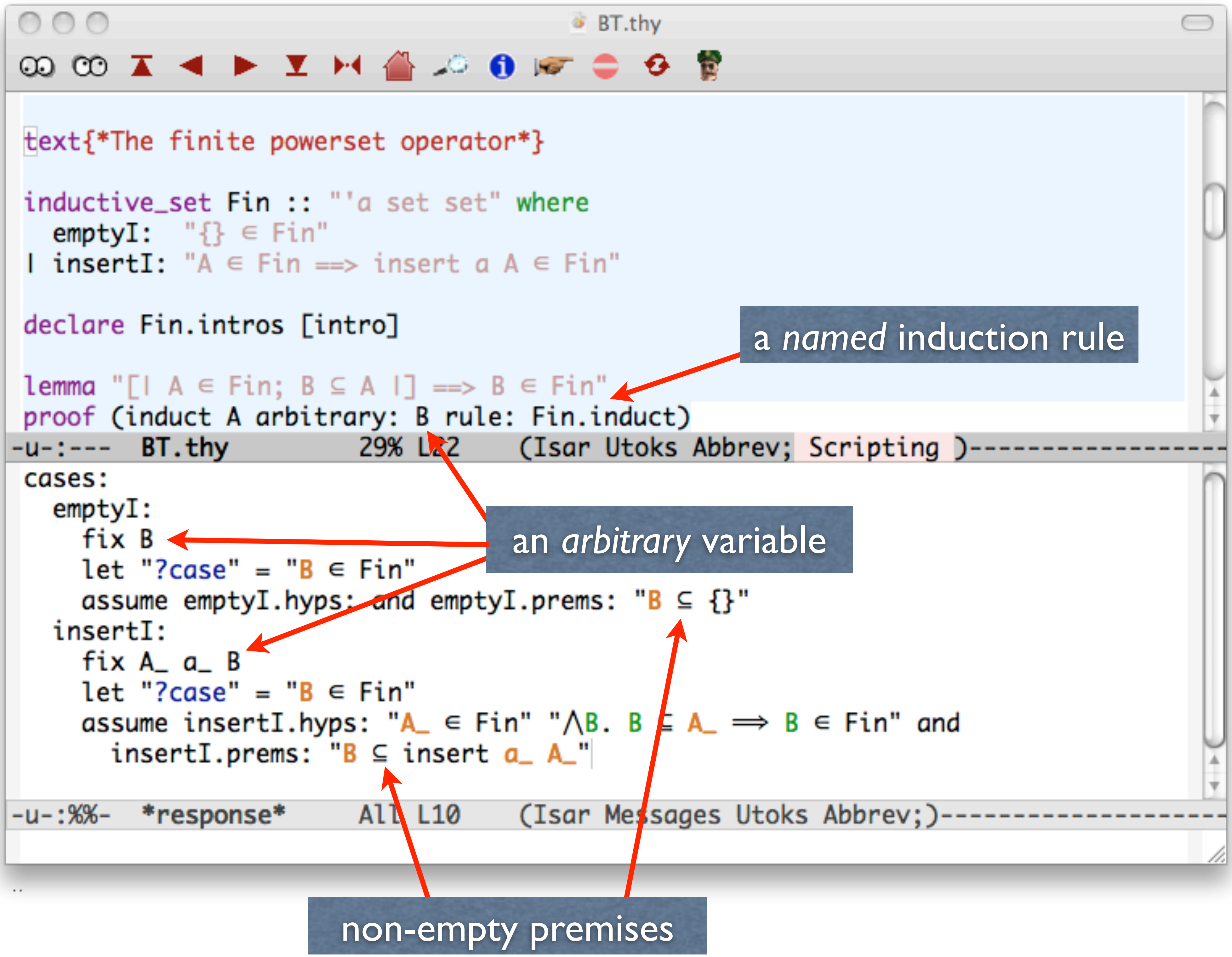
BT.thy

```
text{*The finite powerset operator*}

inductive_set Fin :: "'a set set" where
  emptyI:  "{} ∈ Fin"
| insertI: "A ∈ Fin ==> insert a A ∈ Fin"

declare Fin.intros [intro]

lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
-u-:---  BT.thy        29% L22     (Isar Utoks Abbrev; Scripting )----------------
cases:
  emptyI:
    fix B
    let "?case" = "B ∈ Fin"
    assume emptyI.hyps: and emptyI.prems: "B ⊆ {}"
  insertI:
    fix A_ a_ B
    let "?case" = "B ∈ Fin"
    assume insertI.hyps: "A_ ∈ Fin" "⋀B. B ⊆ A_ ⟹ B ∈ Fin" and
      insertI.prems: "B ⊆ insert a_ A_"
-u-:%%-  *response*    All L10     (Isar Messages Utoks Abbrev;)------------------
```

a *named* induction rule

an *arbitrary* variable

A More Sophisticated Proof



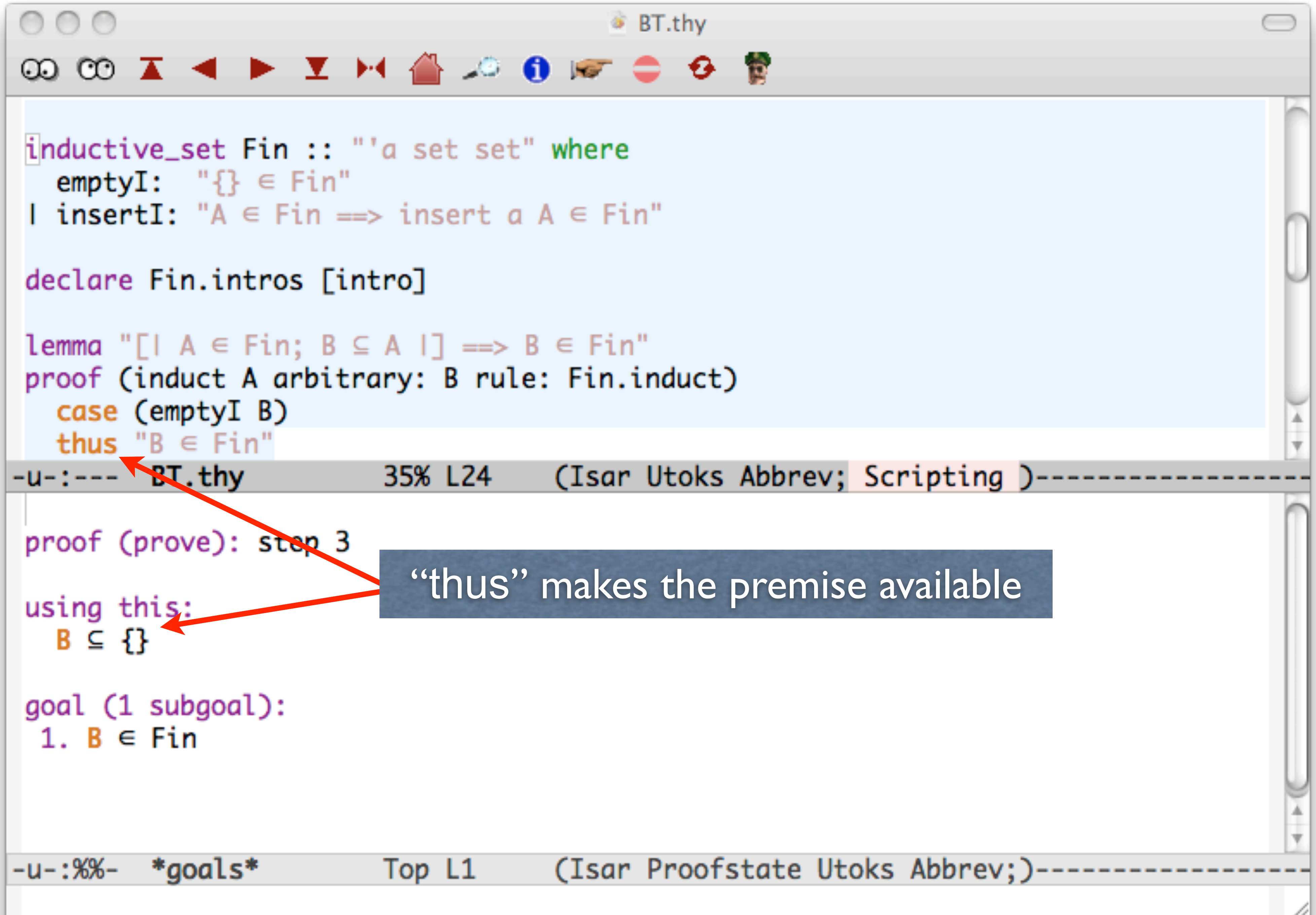
Proving the Base Case

BT.thy

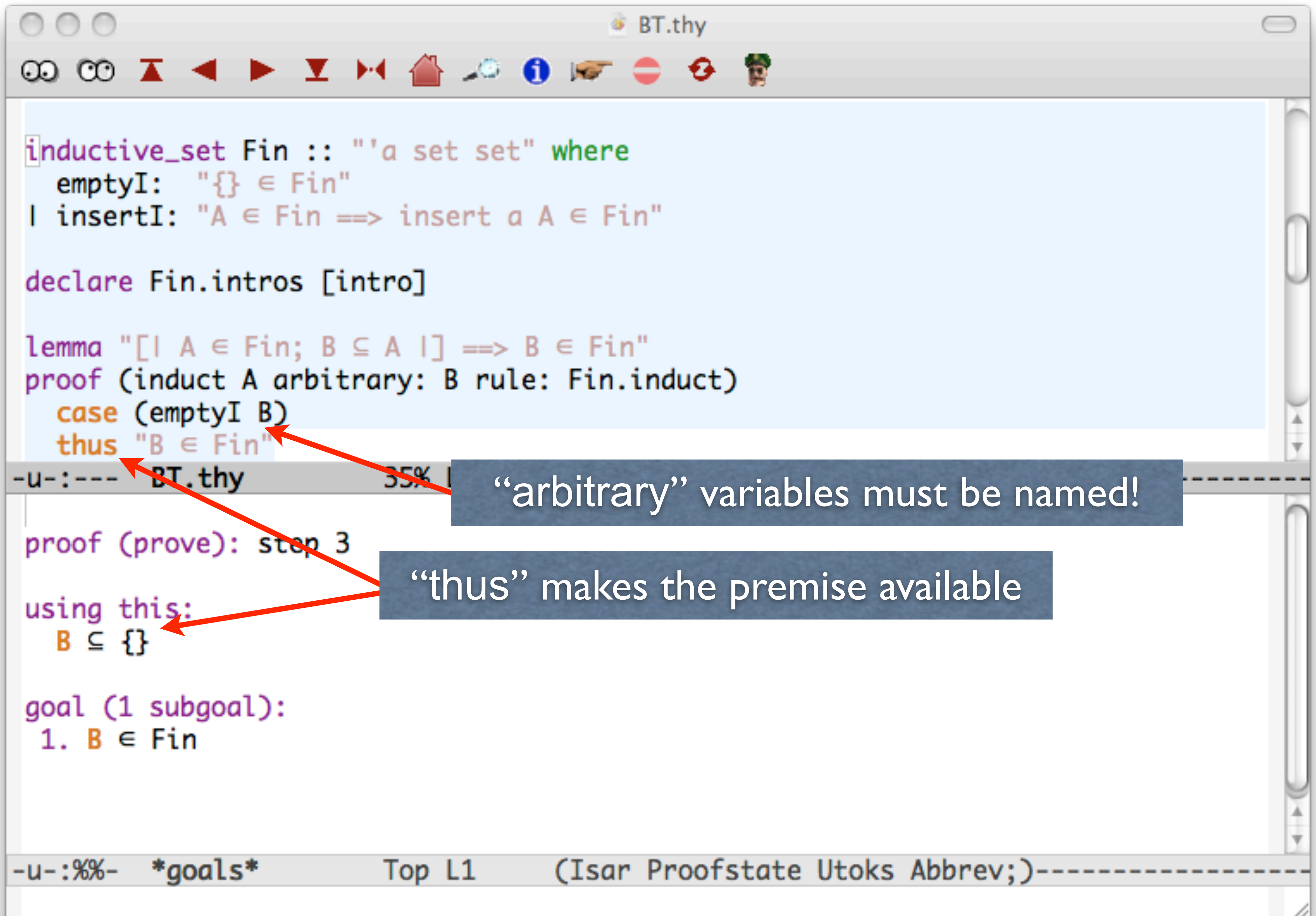
```
inductive_set Fin :: "'a set set" where
  emptyI:  "{} ∈ Fin"
| insertI: "A ∈ Fin ==> insert a A ∈ Fin"

declare Fin.intros [intro]

lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
-u-:---  BT.thy        35% L24     (Isar Utoks Abbrev; Scripting )----------------
```

```
proof (prove): step 3

using this:
  B ⊆ {}

goal (1 subgoal):
 1. B ∈ Fin

-u-:%%-  *goals*       Top L1      (Isar Proofstate Utoks Abbrev;)----------------
```

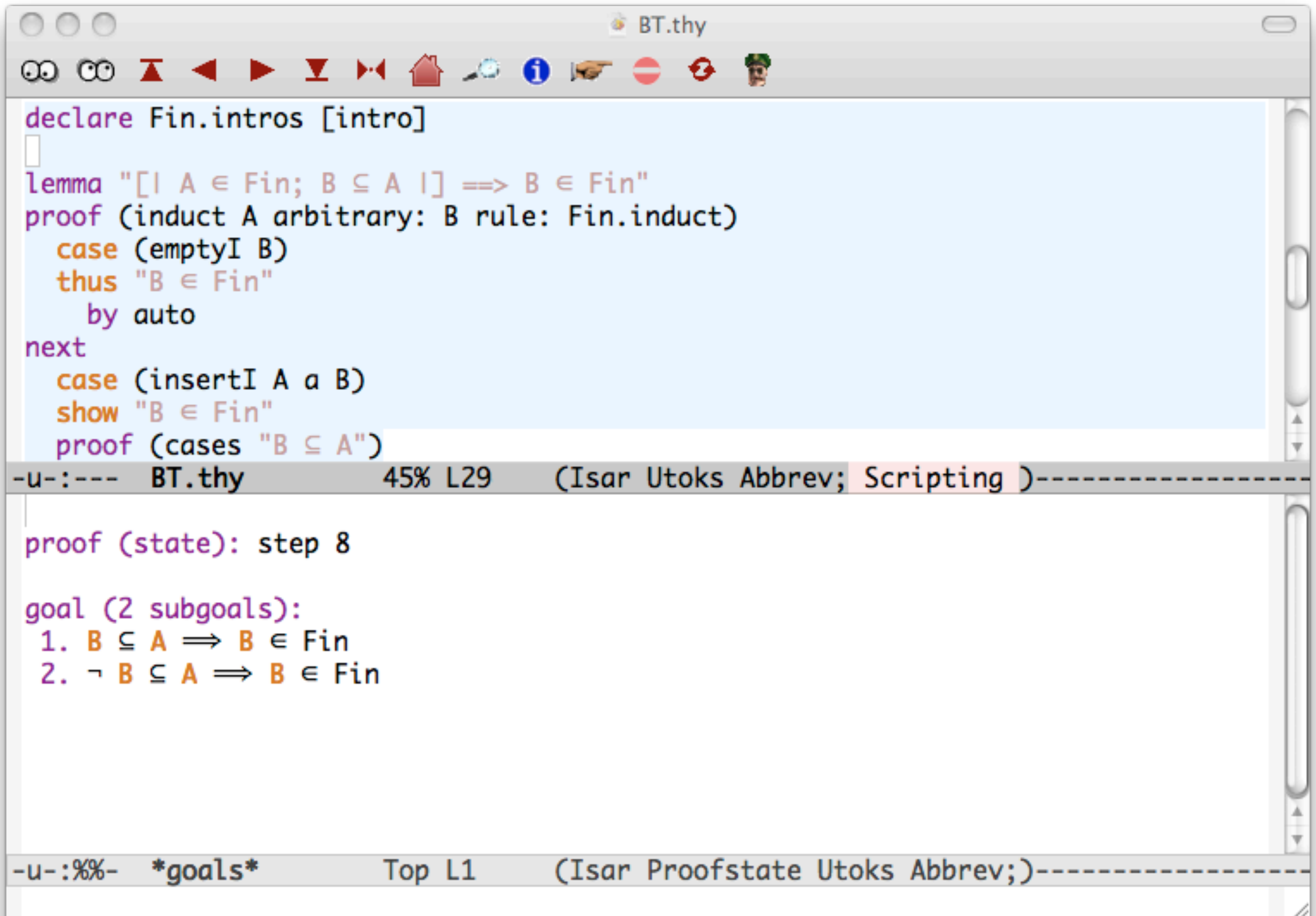
Proving the Base Case



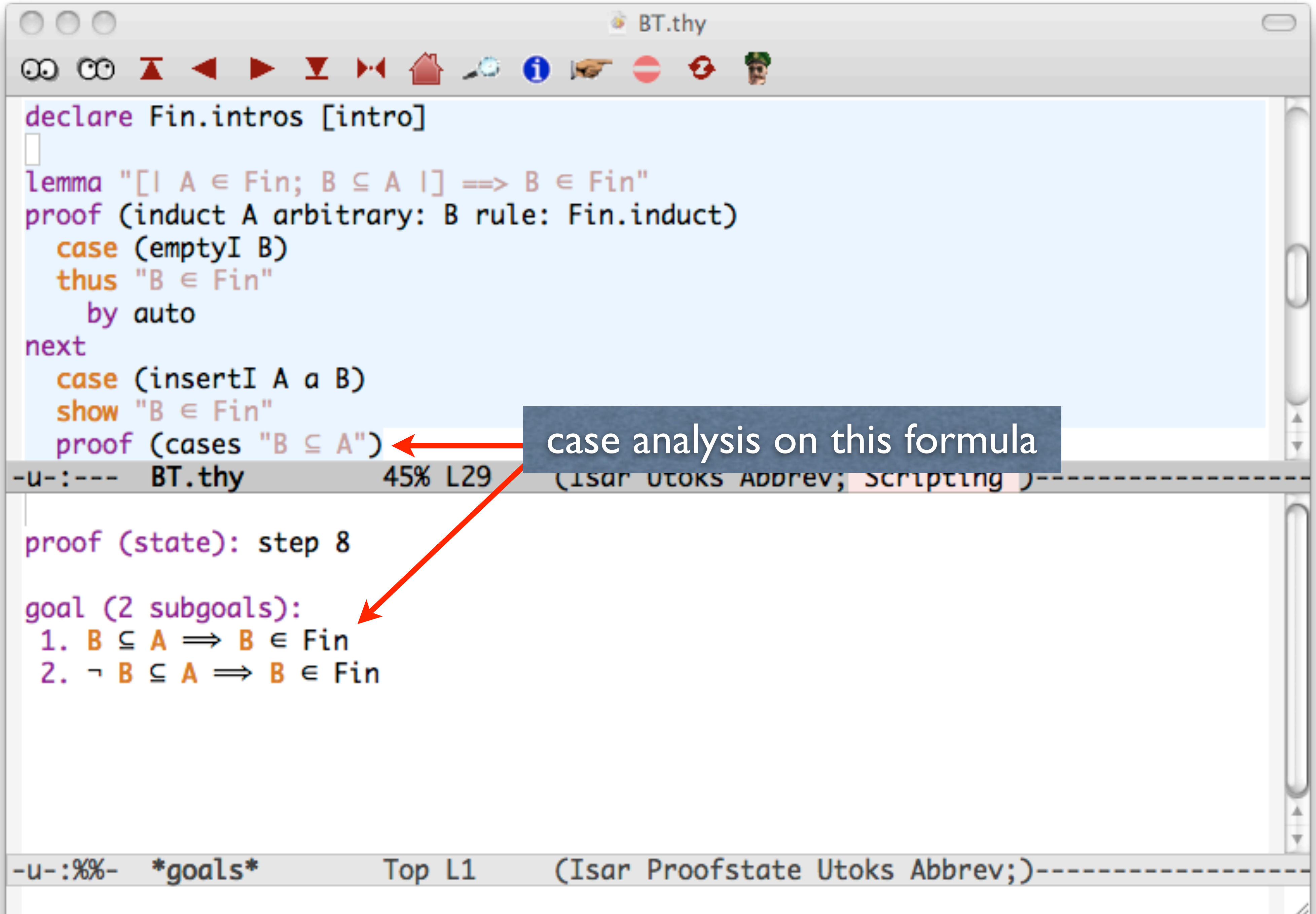
Proving the Base Case



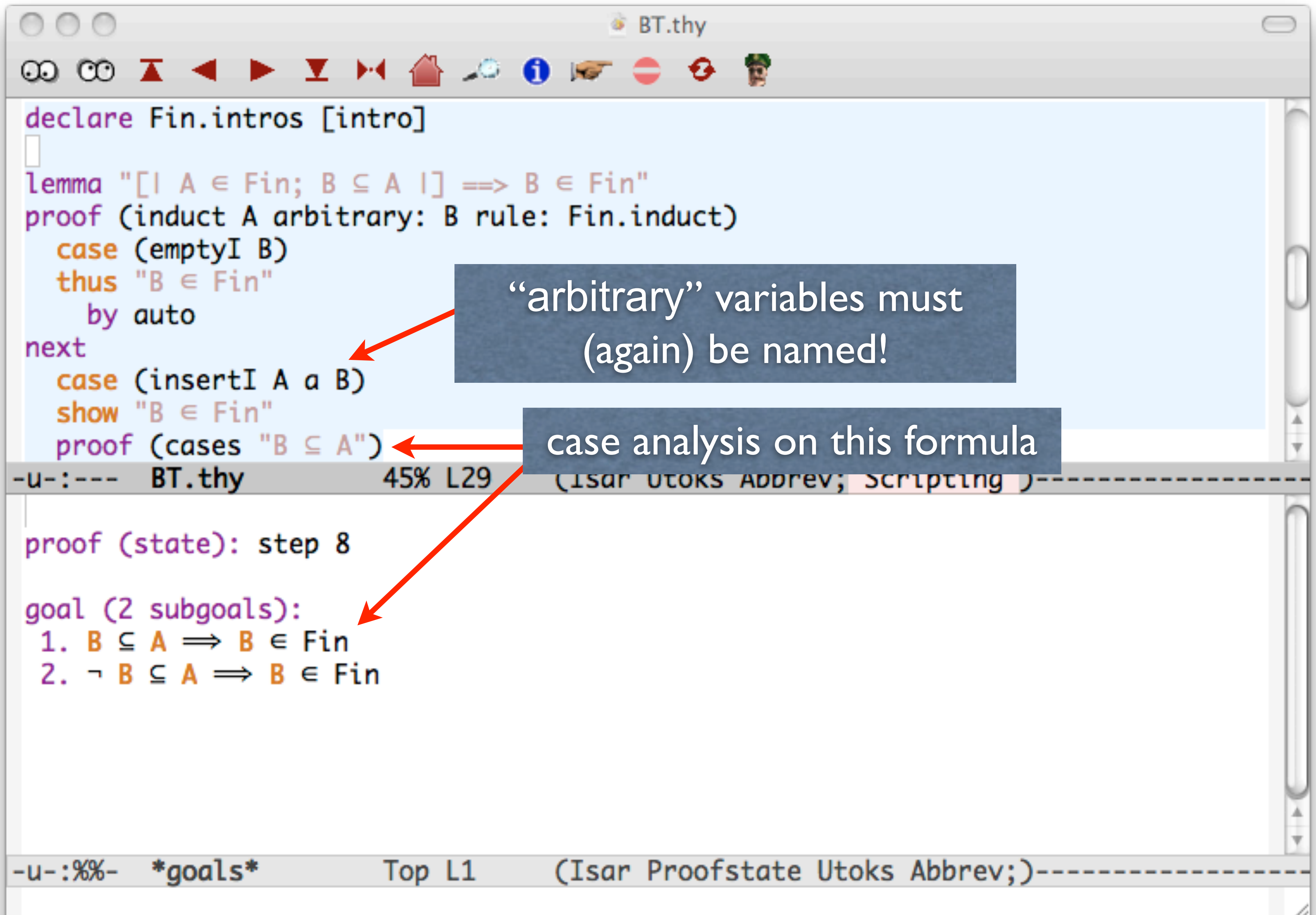
A Nested Case Analysis



A Nested Case Analysis



A Nested Case Analysis



The Complete Proof

BTplus.thy

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
    by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case True
    show "B ∈ Fin" using insertI True
      by auto
  next
    case False
    have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
      by auto
    hence "B = insert a (B - {a})" using False
      by auto
    also have "... ∈ Fin" using insertI Ba
      by blast
    finally show "B ∈ Fin" .
  qed
qed
```

-u-:--- BTplus.thy 20% L50 (Isar Utoks Abbrev; Scripting)----------------

The Complete Proof

BTplus.thy

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
    by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case True
    show "B ∈ Fin" using insertI True
      by auto
  next
    case False
    have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
      by auto
    hence "B = insert a (B - {a})" using False
      by auto
    also have "... ∈ Fin" using insertI Ba
      by blast
    finally show "B ∈ Fin" .
  qed
qed
-u-:---  BTplus.thy      20% L50     (Isar Utoks Abbrev; Scripting )----------------
```

true and false cases

The Complete Proof

BTplus.thy

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
    by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case True
    show "B ∈ Fin" using insertI True
      by auto
  next
    case False
    have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
      by auto
    hence "B = insert a (B - {a})" using False
      by auto
    also have "... ∈ Fin" using insertI Ba
      by blast
    finally show "B ∈ Fin" .
  qed
qed
-u-:---  BTplus.thy    20% L50    (Isar Utoks Abbrev; Scripting )----------------
```

induction hypothesis and premise

true and false cases

The Complete Proof

BTplus.thy

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
    by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case True
    show "B ∈ Fin" using insertI True
      by auto
  next
    case False
    have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
      by auto
    hence "B = insert a (B - {a})" using False
      by auto
    also have "... ∈ Fin" using insertI Ba
      by blast
    finally show "B ∈ Fin" .
  qed
qed
```

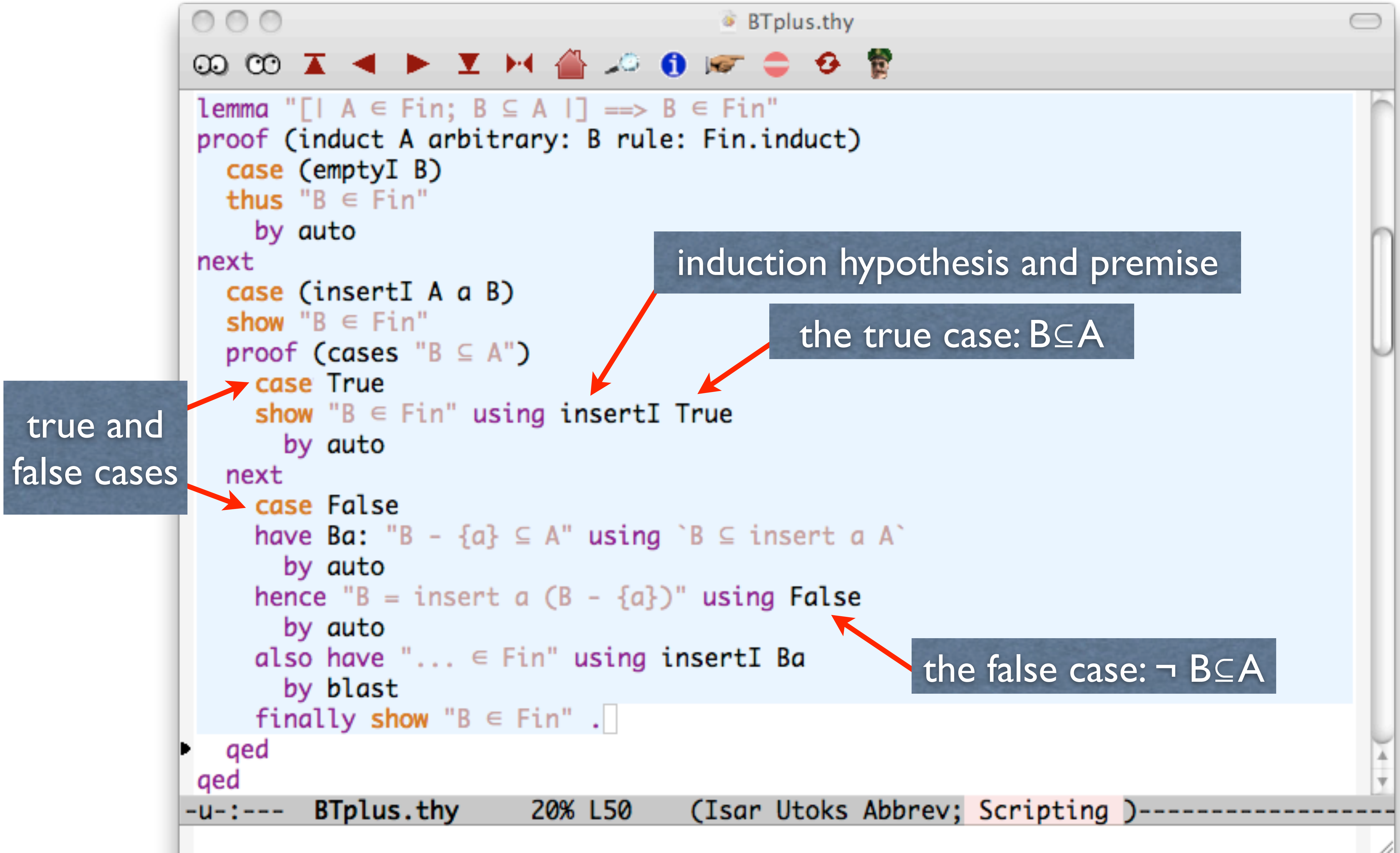
-u-:--- BTplus.thy 20% L50 (Isar Utoks Abbrev; Scripting)----------------

induction hypothesis and premise

the true case: B⊆A

true and false cases

The Complete Proof



The Complete Proof

BTplus.thy

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
    by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case True
    show "B ∈ Fin" using insertI True
      by auto
  next
    case False
    have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
      by auto
    hence "B = insert a (B - {a})" using False
      by auto
    also have "... ∈ Fin" using insertI Ba
      by blast
    finally show "B ∈ Fin" .
  qed
qed
```

-u-:--- BTplus.thy 20% L50 (Isar Utoks Abbrev; Scripting)------------------

induction hypothesis and premise

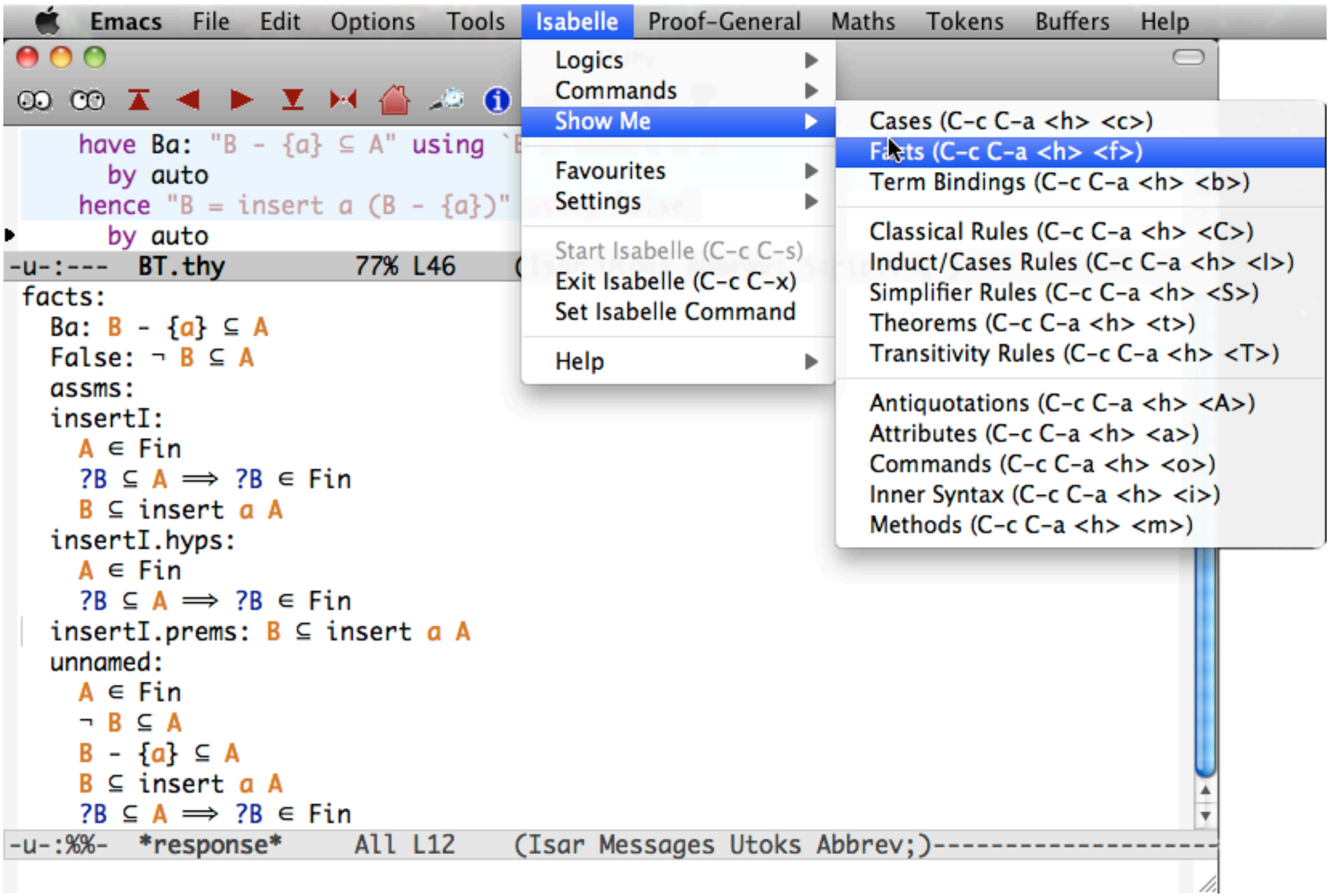
the true case: B⊆A

direct quotation of a fact

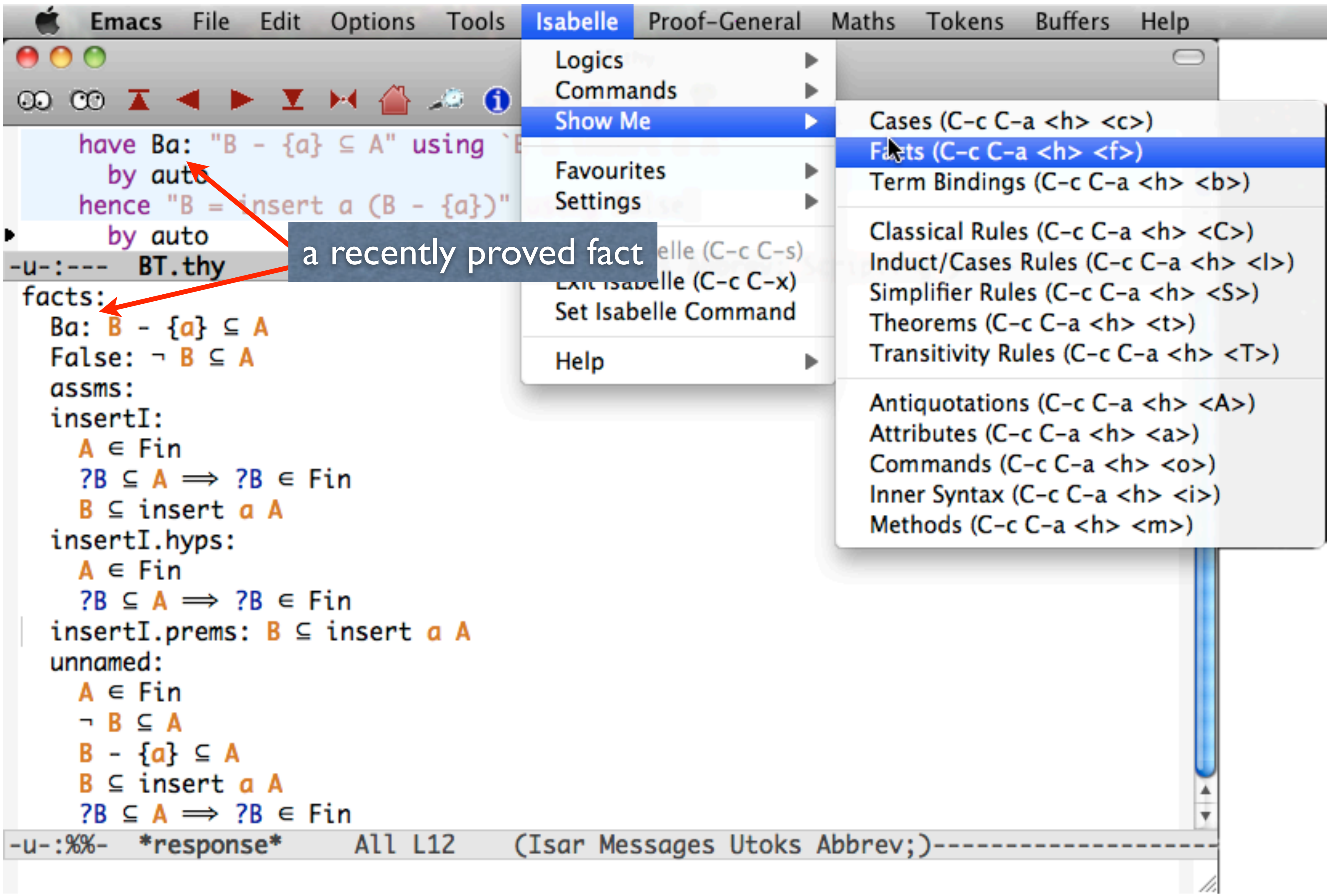
the false case: ¬ B⊆A

true and false cases

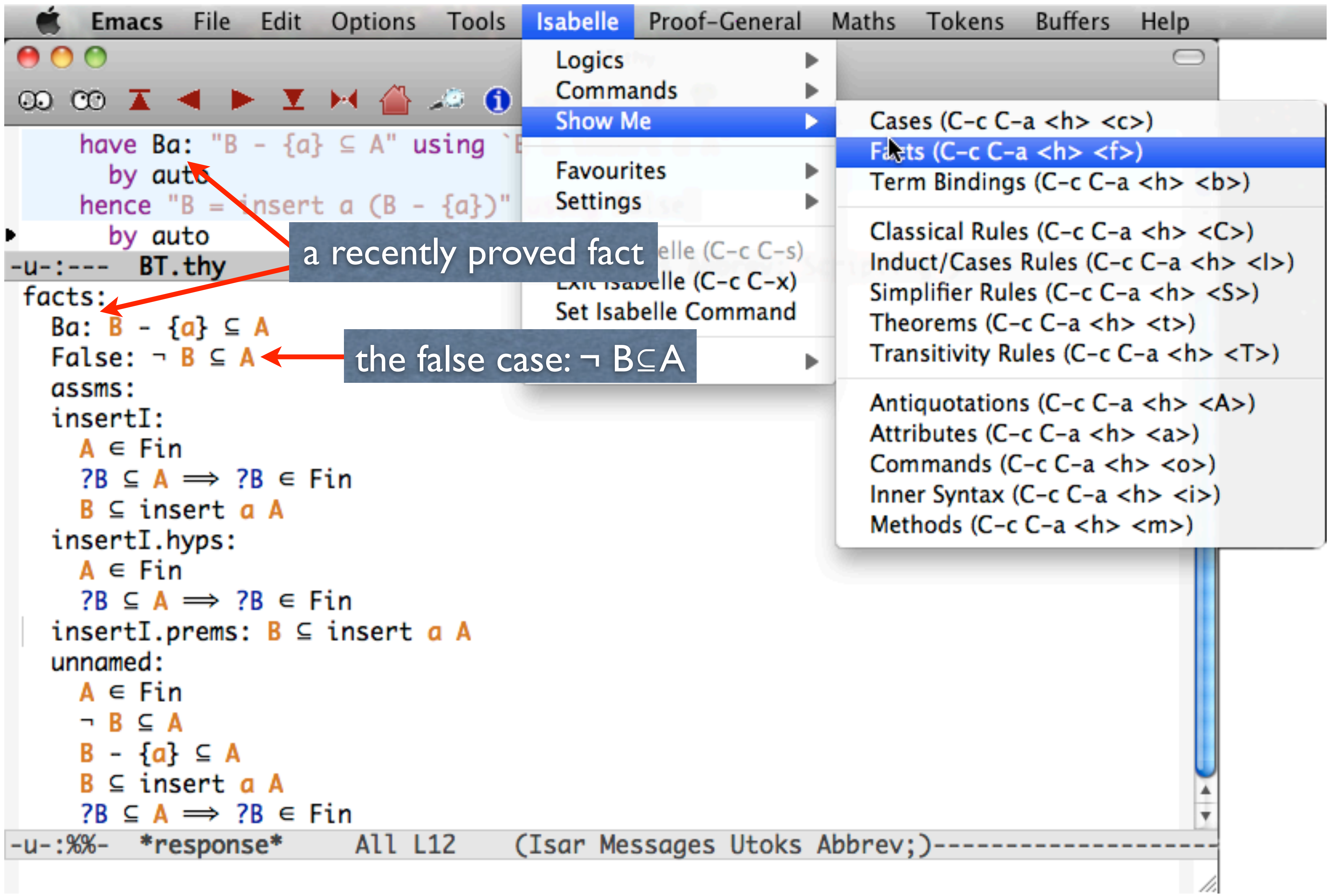
Which Theorems are Available?



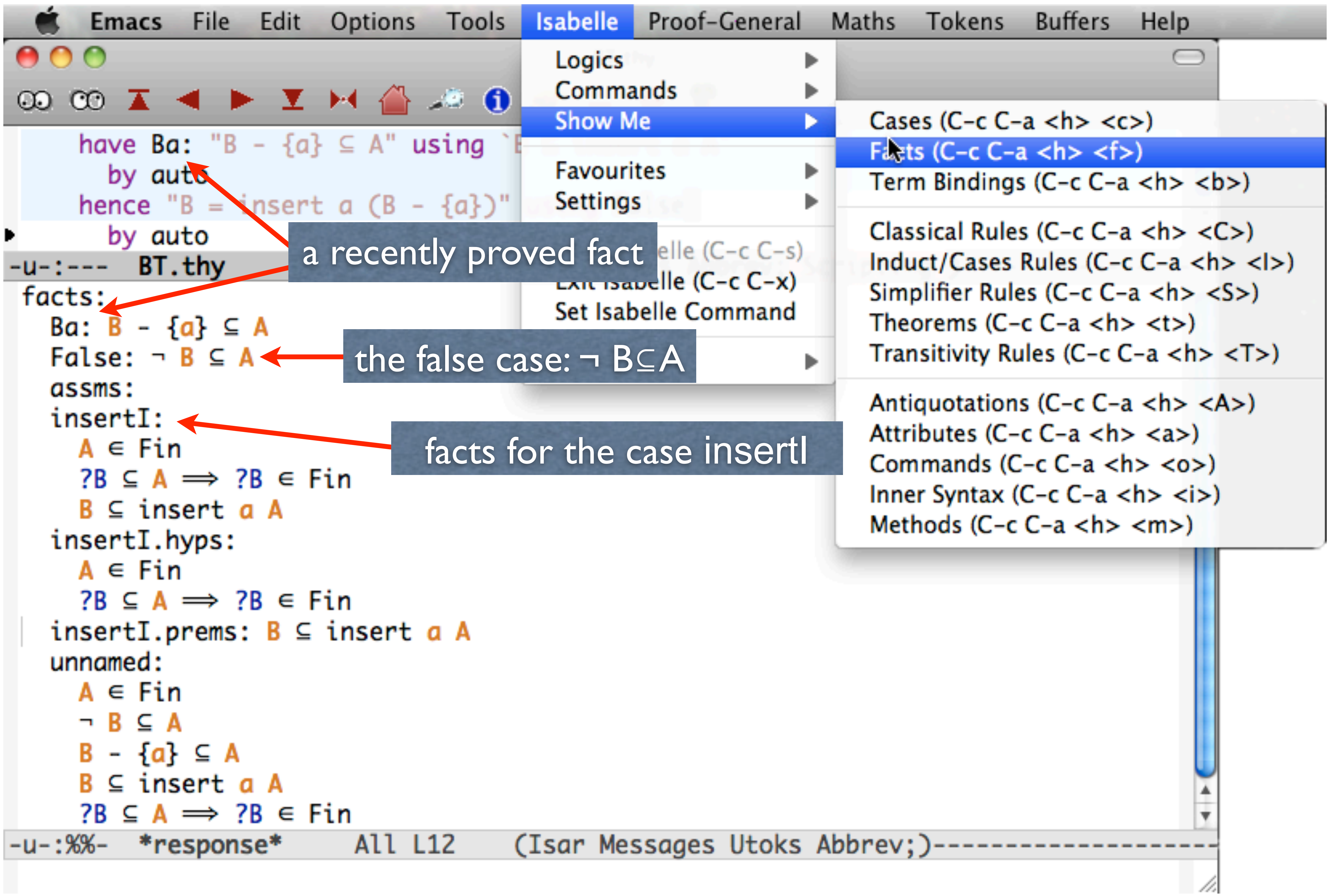
Which Theorems are Available?



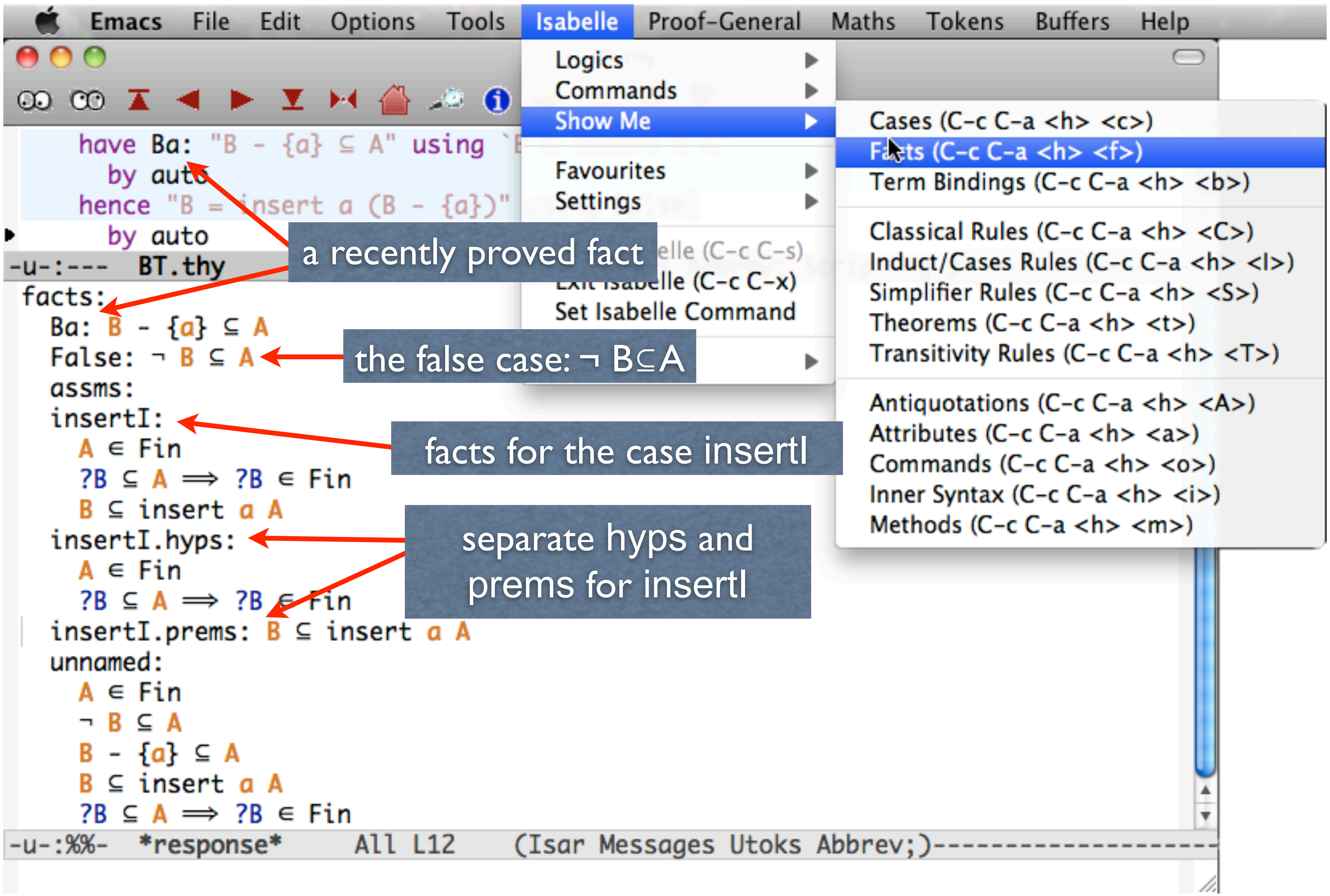
Which Theorems are Available?



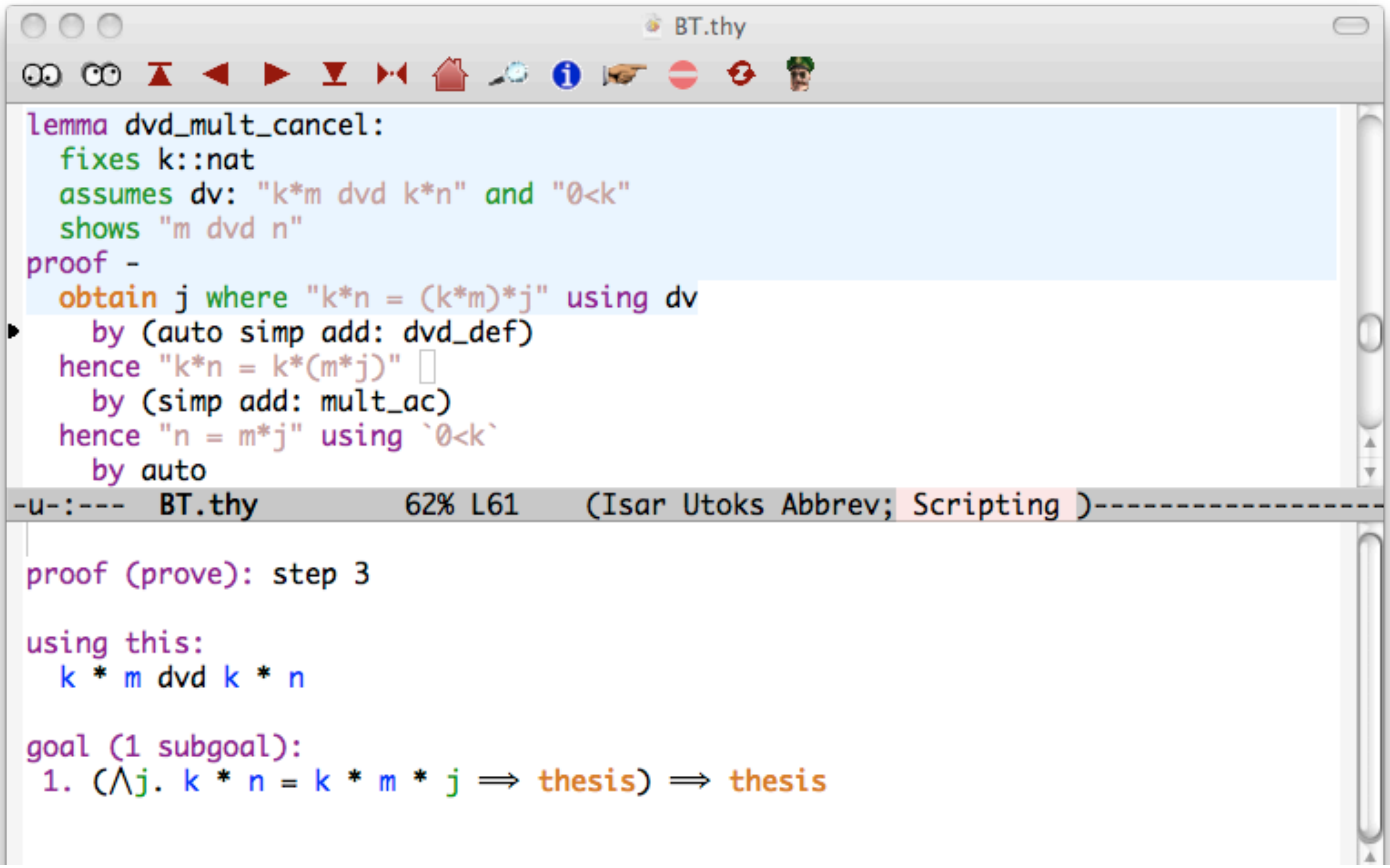
Which Theorems are Available?



Which Theorems are Available?

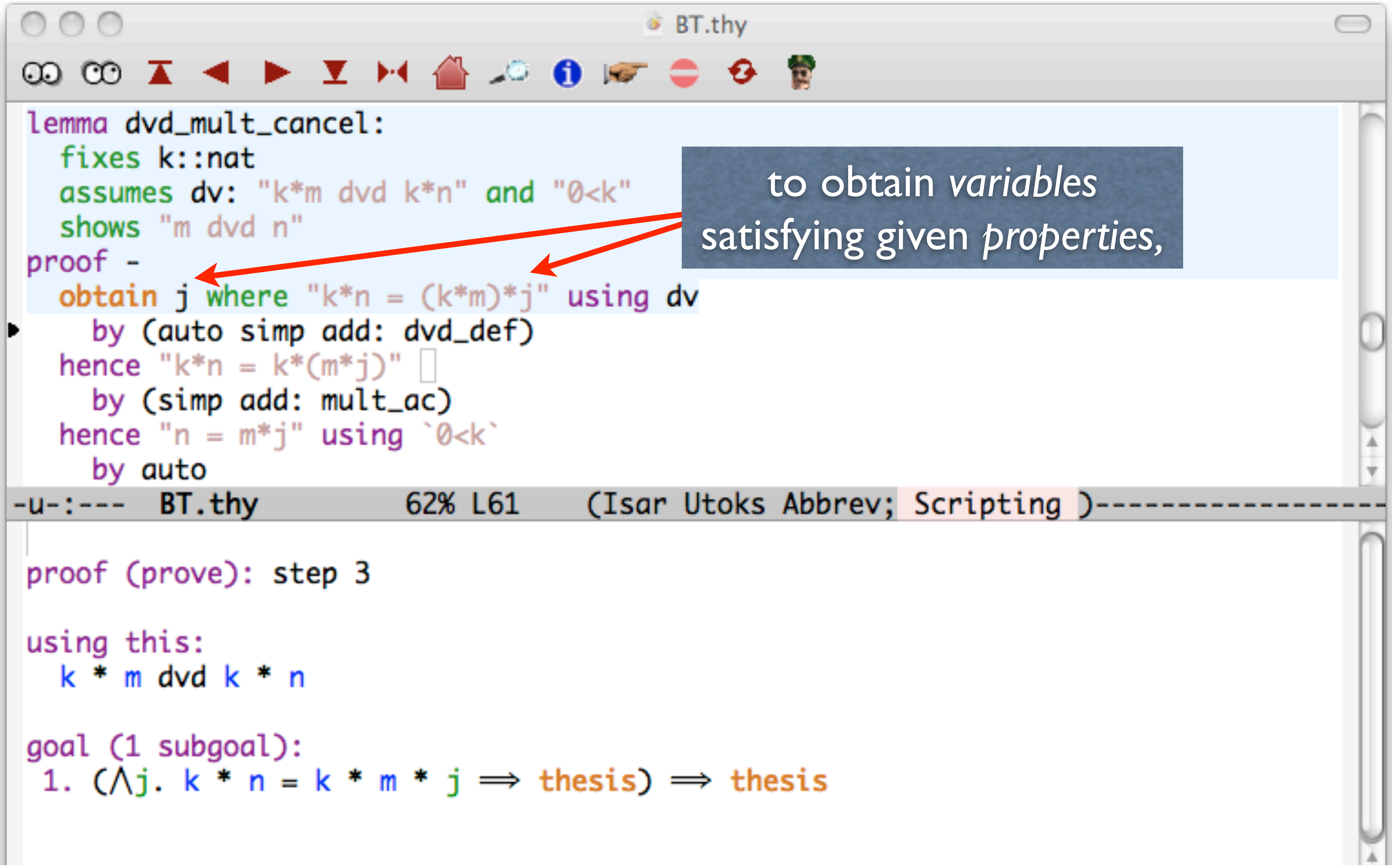


Existential Claims: "obtain"



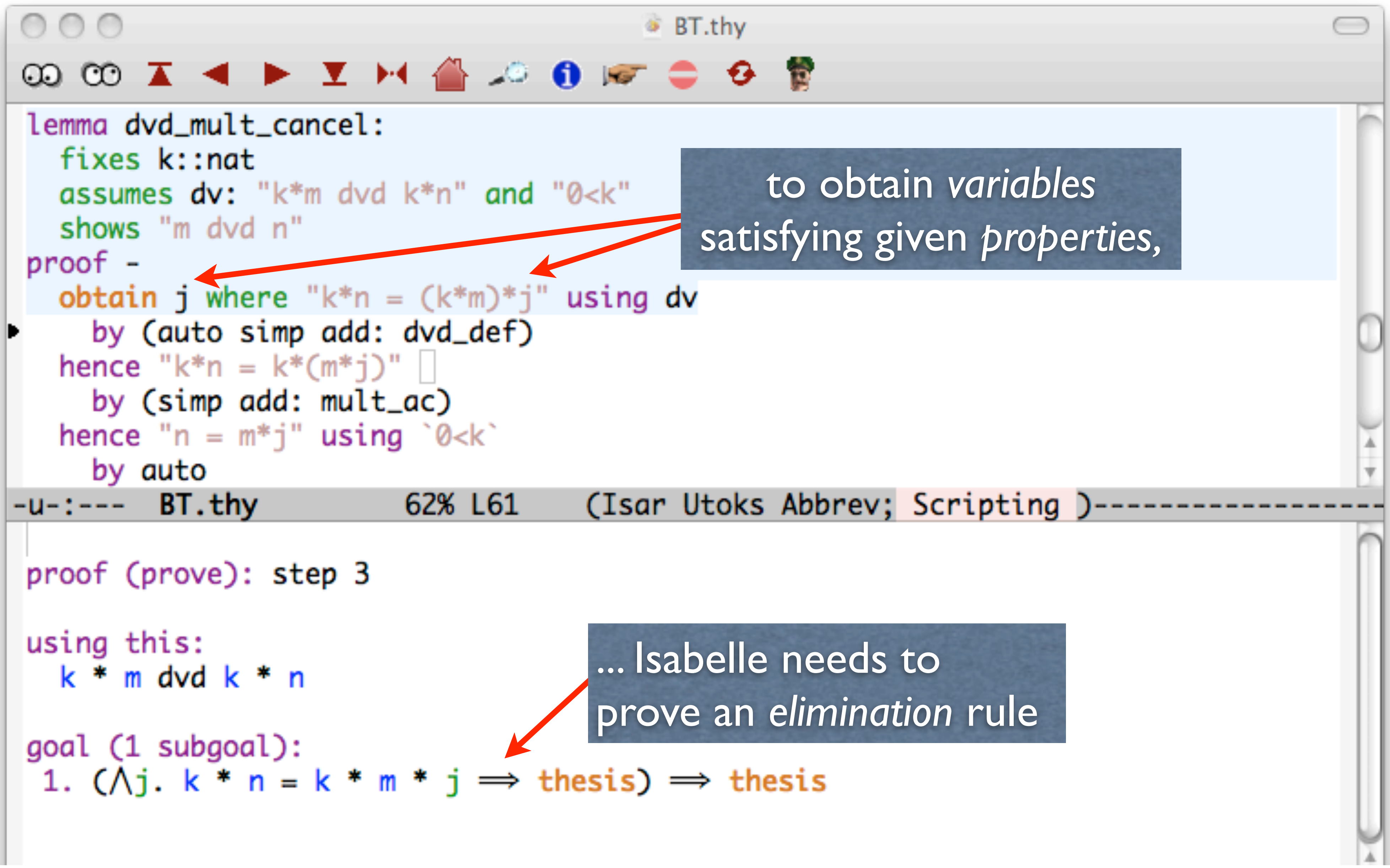
$$b \text{ dvd } a \;\leftrightarrow\; (\exists k.\, a = b \times k)$$

Existential Claims: "obtain"



$$b \text{ dvd } a \leftrightarrow (\exists k.\ a = b \times k)$$

Existential Claims: "obtain"



$$b \text{ dvd } a \leftrightarrow (\exists k.\ a = b \times k)$$

Continuing the Proof

BT.thy

```
lemma dvd_mult_cancel:
  fixes k::nat
  assumes dv: "k*m dvd k*n" and "0<k"
  shows "m dvd n"
proof -
  obtain j where "k*n = (k*m)*j" using dv
    by (auto simp add: dvd_def)
  hence "k*n = k*(m*j)"
    by (simp add: mult_ac)
  hence "n = m*j" using `0<k`
    by auto
-u-:---  BT.thy        62% L62     (Isar Utoks Abbrev; Scripting )------------------
```

```
proof (prove): step 5

using this:
  k * n = k * m * j

goal (1 subgoal):
 1. k * n = k * (m * j)

-u-:%%-  *goals*       Top L1      (Isar Proofstate Utoks Abbrev;)-----------------
tool-bar next
```

Continuing the Proof

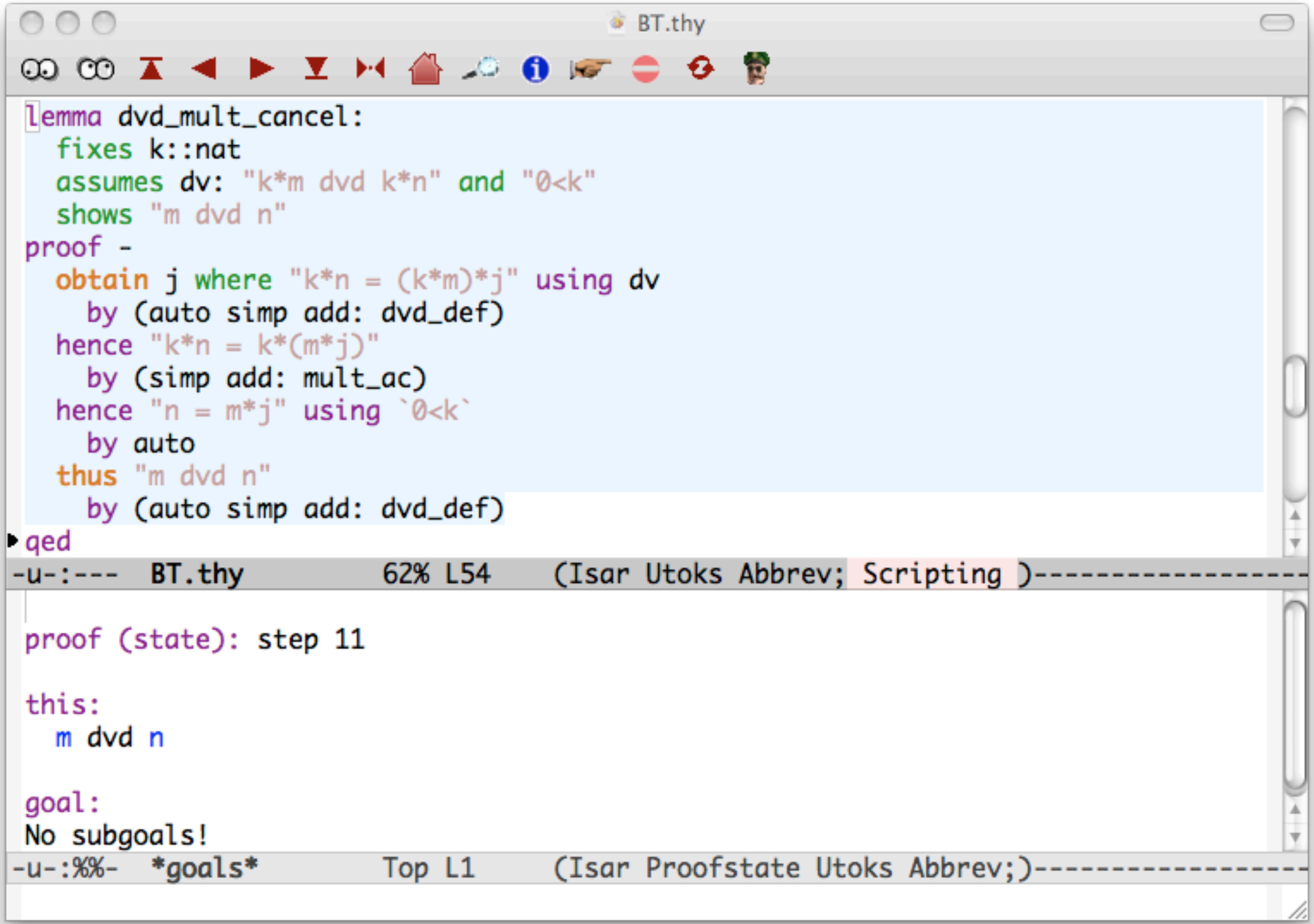
BT.thy

```
lemma dvd_mult_cancel:
  fixes k::nat
  assumes dv: "k*m dvd k*n" and "0<k"
  shows "m dvd n"
proof -
  obtain j where "k*n = (k*m)*j" using dv
    by (auto simp add: dvd_def)
  hence "k*n = k*(m*j)"
    by (simp add: mult_ac)
  hence "n = m*j" using `0<k`
    by auto
-u-:---  BT.thy        62% L62     (Isar Utoks Abbrev; Scripting )----------------
```

```
proof (prove): step 5

using this:
  k * n = k * m * j

goal (1 subgoal):
 1. k * n = k * (m * j)
-u-:%%-  *goals*       Top L1      (Isar Proofstate Utoks Abbrev;)----------------
tool-bar next
```

we now have the key property of j

The Finished Proof



The Finished Proof

BT.thy

```
lemma dvd_mult_cancel:
  fixes k::nat
  assumes dv: "k*m dvd k*n" and "0<k"
  shows "m dvd n"
proof -
  obtain j where "k*n = (k*m)*j" using dv
    by (auto simp add: dvd_def)
  hence "k*n = k*(m*j)"
    by (simp add: mult_ac)
  hence "n = m*j" using `0<k`
    by auto
  thus "m dvd n"
    by (auto simp add: dvd_def)
qed
-u-:---  BT.thy       62% L54    (Isar Utoks Abbrev; Scripting )----------------

proof (state): step 11

this:
  m dvd n

goal:
No subgoals!
-u-:%%-  *goals*      Top L1     (Isar Proofstate Utoks Abbrev;)----------------
```

removing k from the equality

Introducing “then”

BTplus.thy

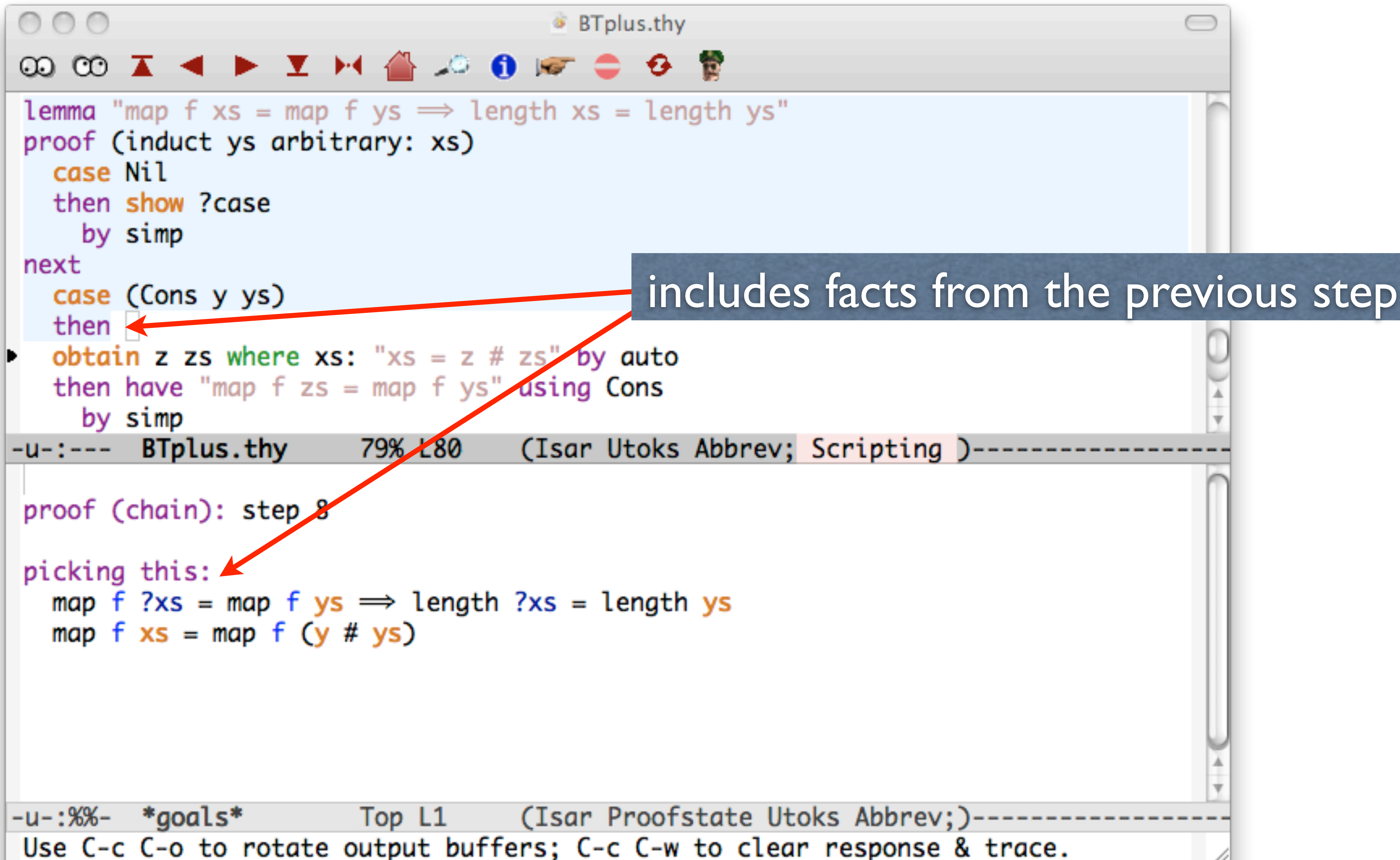
```
lemma "map f xs = map f ys ⟹ length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
```

```
-u-:---  BTplus.thy      79% L80     (Isar Utoks Abbrev; Scripting )----------------
```

```
proof (chain): step 8

picking this:
  map f ?xs = map f ys ⟹ length ?xs = length ys
  map f xs = map f (y # ys)
```

```
-u-:%%-  *goals*        Top L1      (Isar Proofstate Utoks Abbrev;)----------------
Use C-c C-o to rotate output buffers; C-c C-w to clear response & trace.
```

Introducing "then"



Introducing "then"

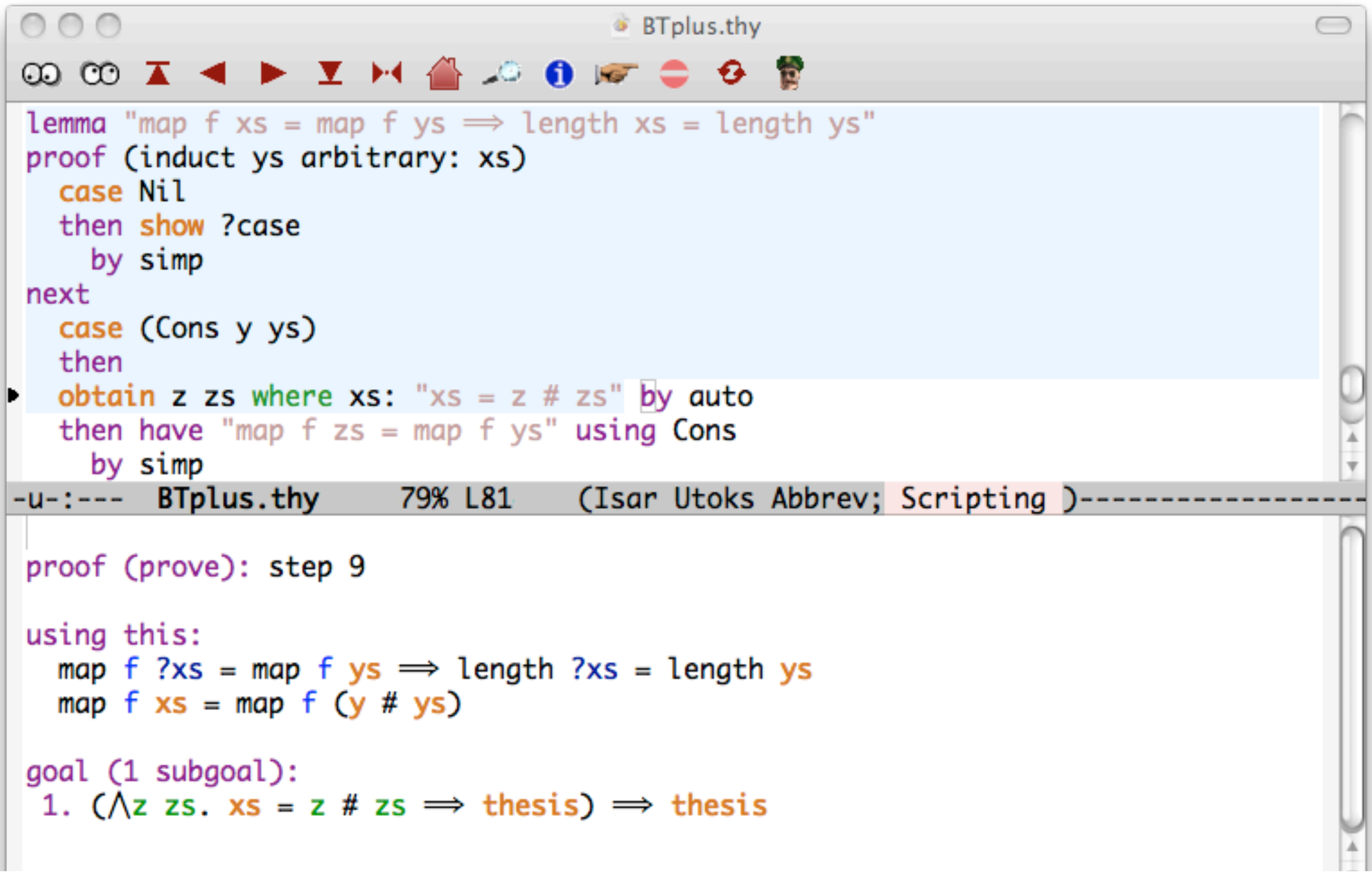
BTplus.thy

```
lemma "map f xs = map f ys ⟹ length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
-u-:---  BTplus.thy      79% L80     (Isar Utoks Abbrev; Scripting )-----------------

proof (chain): step 8

picking this:
  map f ?xs = map f ys ⟹ length ?xs = length ys
  map f xs = map f (y # ys)

-u-:%%-  *goals*         Top L1      (Isar Proofstate Utoks Abbrev;)-----------------
Use C-c C-o to rotate output buffers; C-c C-w to clear response & trace.
```

includes facts from the previous step

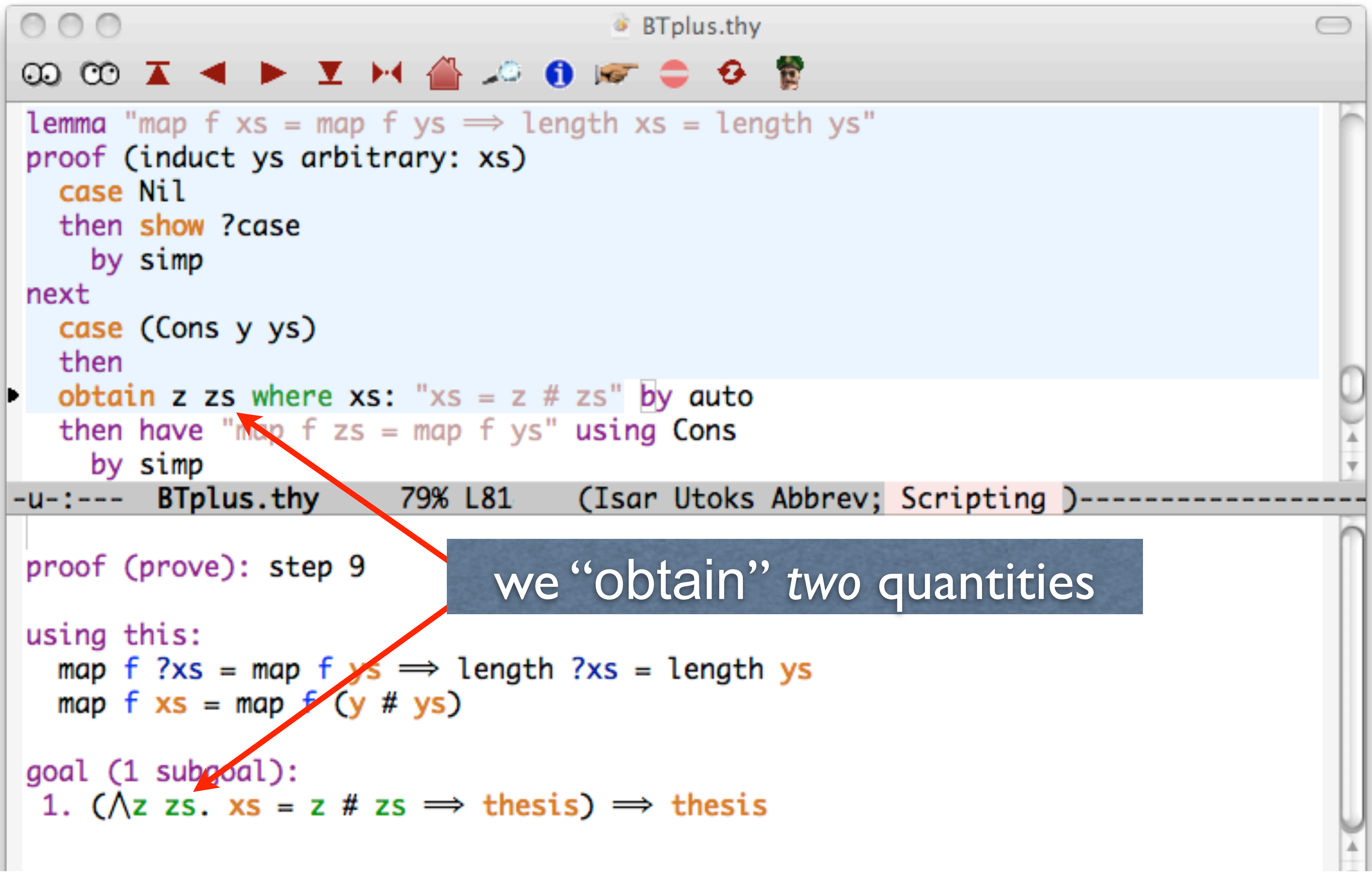
here, the induction context

Another Example of “obtain”



$(\text{map } f\ xs = y\#ys) \leftrightarrow (\exists z\ zs.\ xs = z\#zs\ \&\ f\ z = y\ \&\ \text{map } f\ zs = ys)$

Another Example of “obtain”

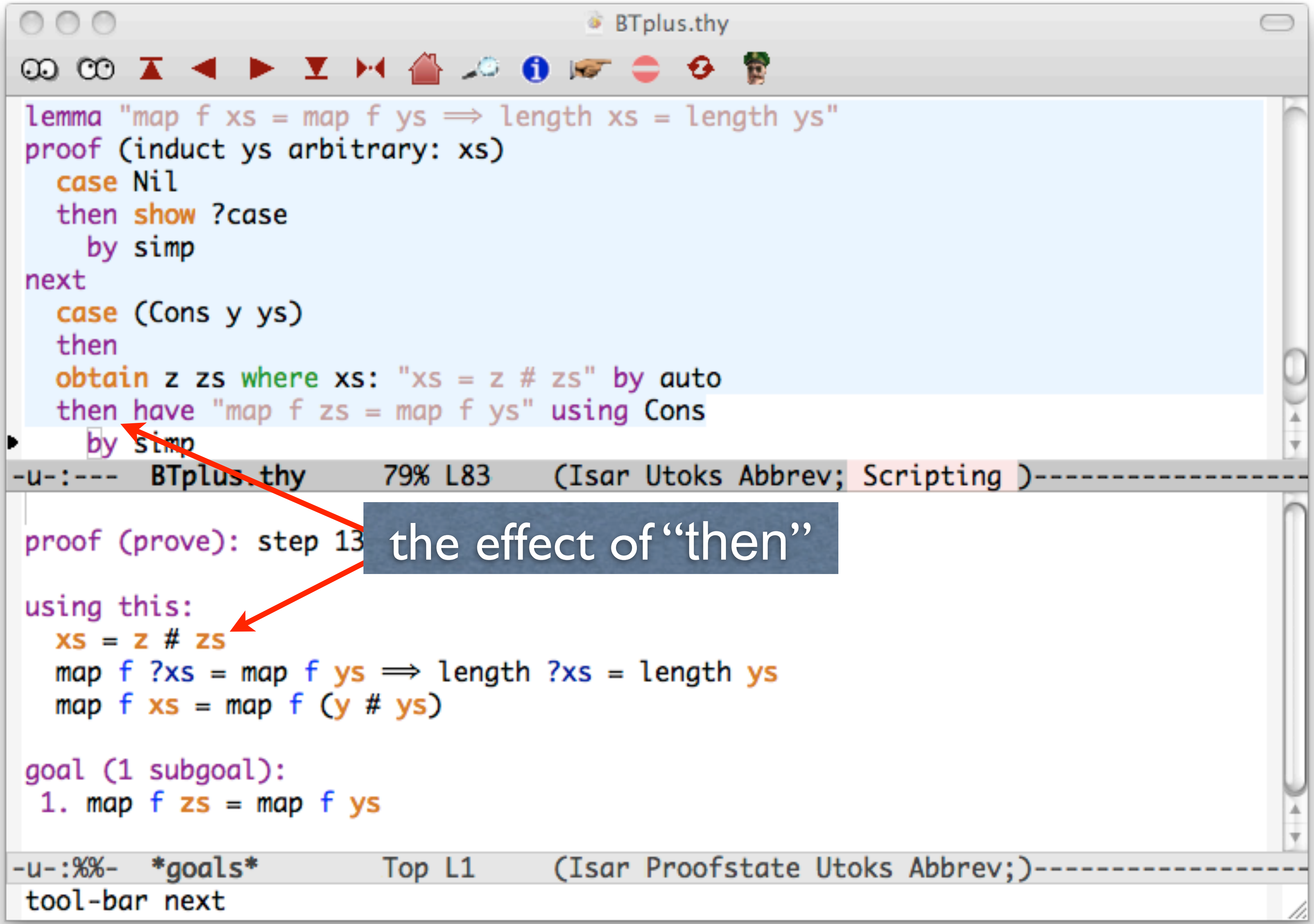


(map f xs = y#ys) $\leftrightarrow$ ($\exists$z zs. xs = z#zs & f z = y & map f zs = ys)

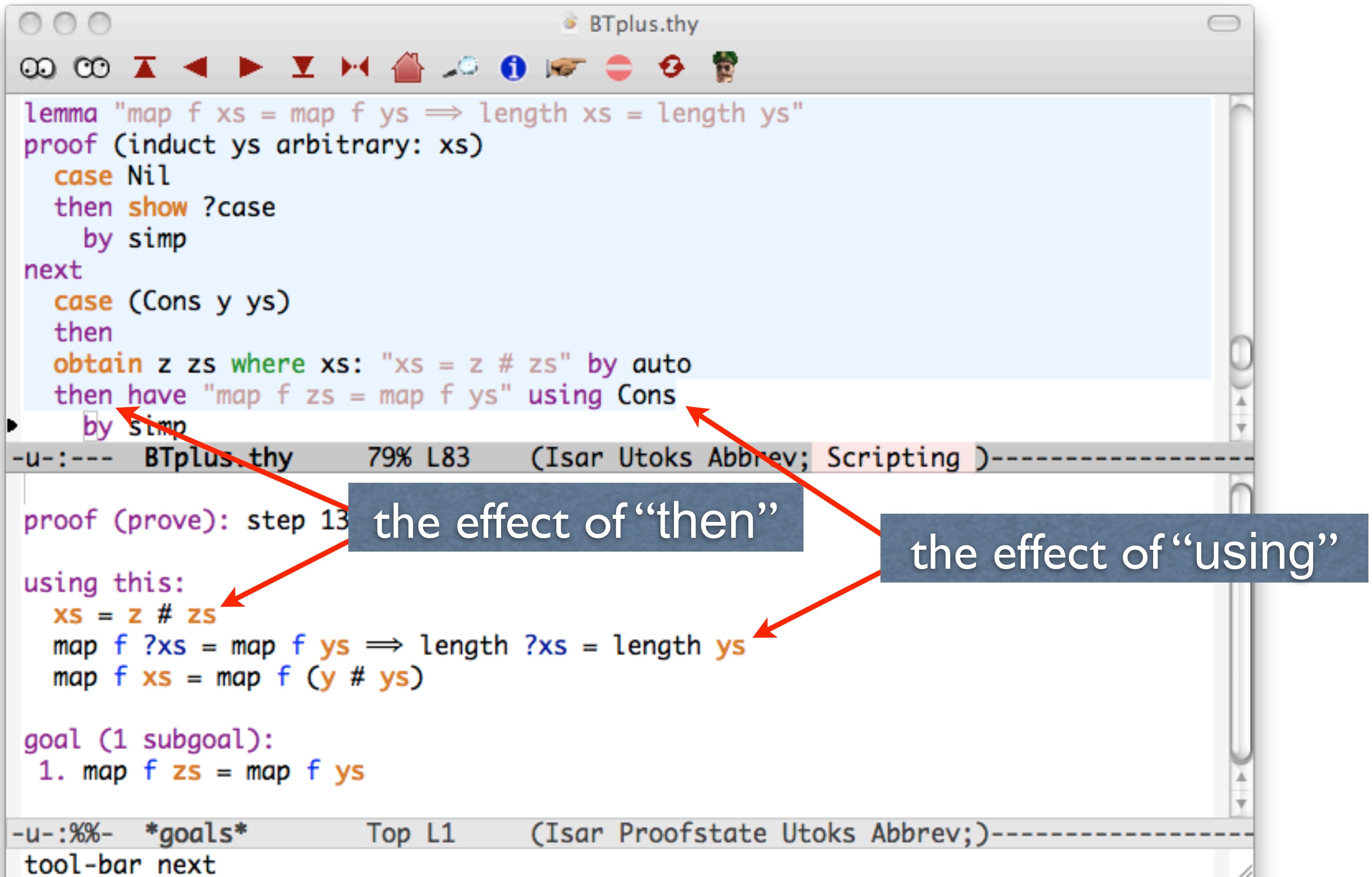
Facts from Two Sources

```
BTplus.thy

lemma "map f xs = map f ys ⟹ length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
-u-:---  BTplus.thy      79% L83     (Isar Utoks Abbrev; Scripting )-----------------

proof (prove): step 13

using this:
  xs = z # zs
  map f ?xs = map f ys ⟹ length ?xs = length ys
  map f xs = map f (y # ys)

goal (1 subgoal):
 1. map f zs = map f ys
-u-:%%-  *goals*         Top L1      (Isar Proofstate Utoks Abbrev;)-----------------
tool-bar next
```

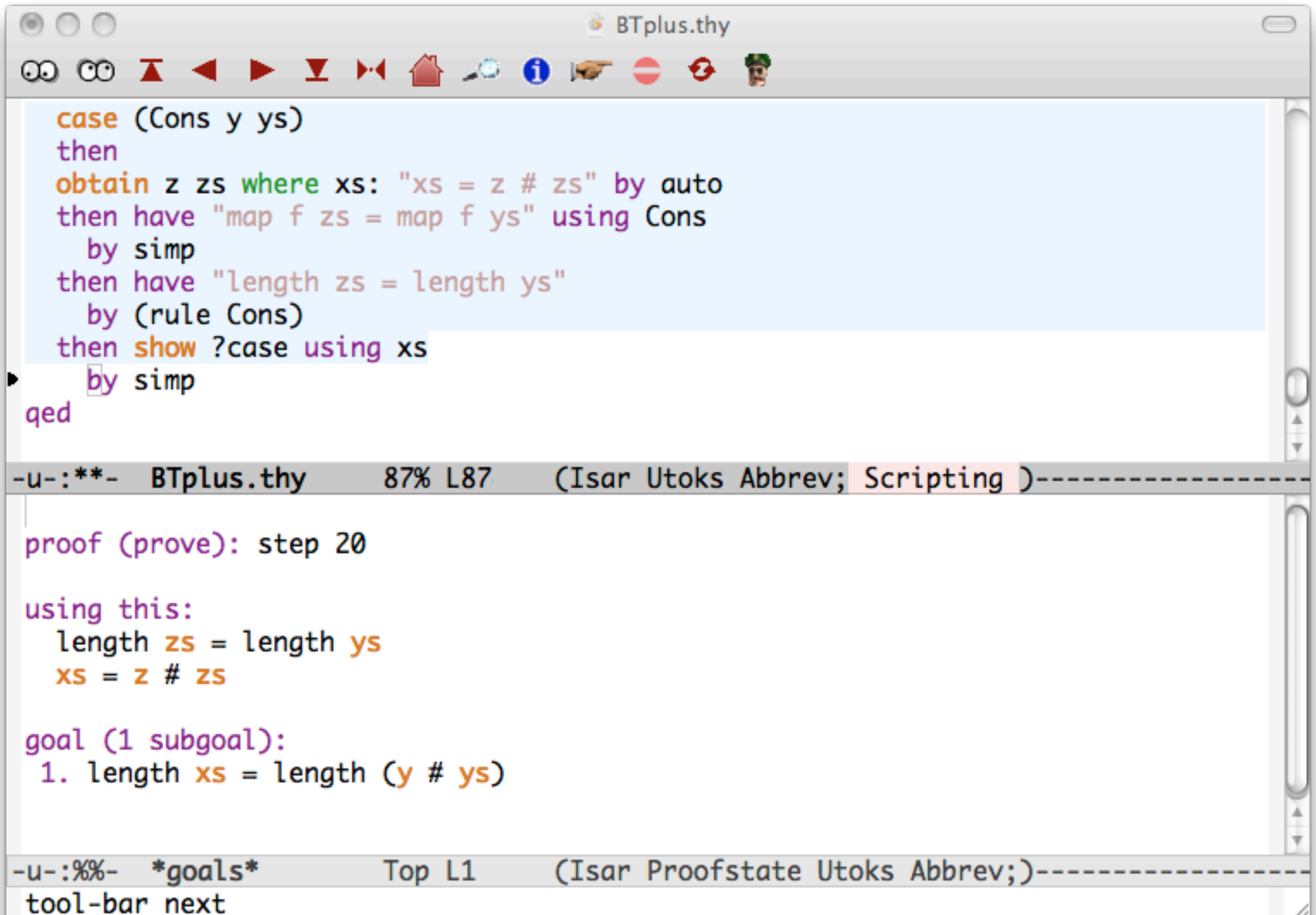
Facts from Two Sources



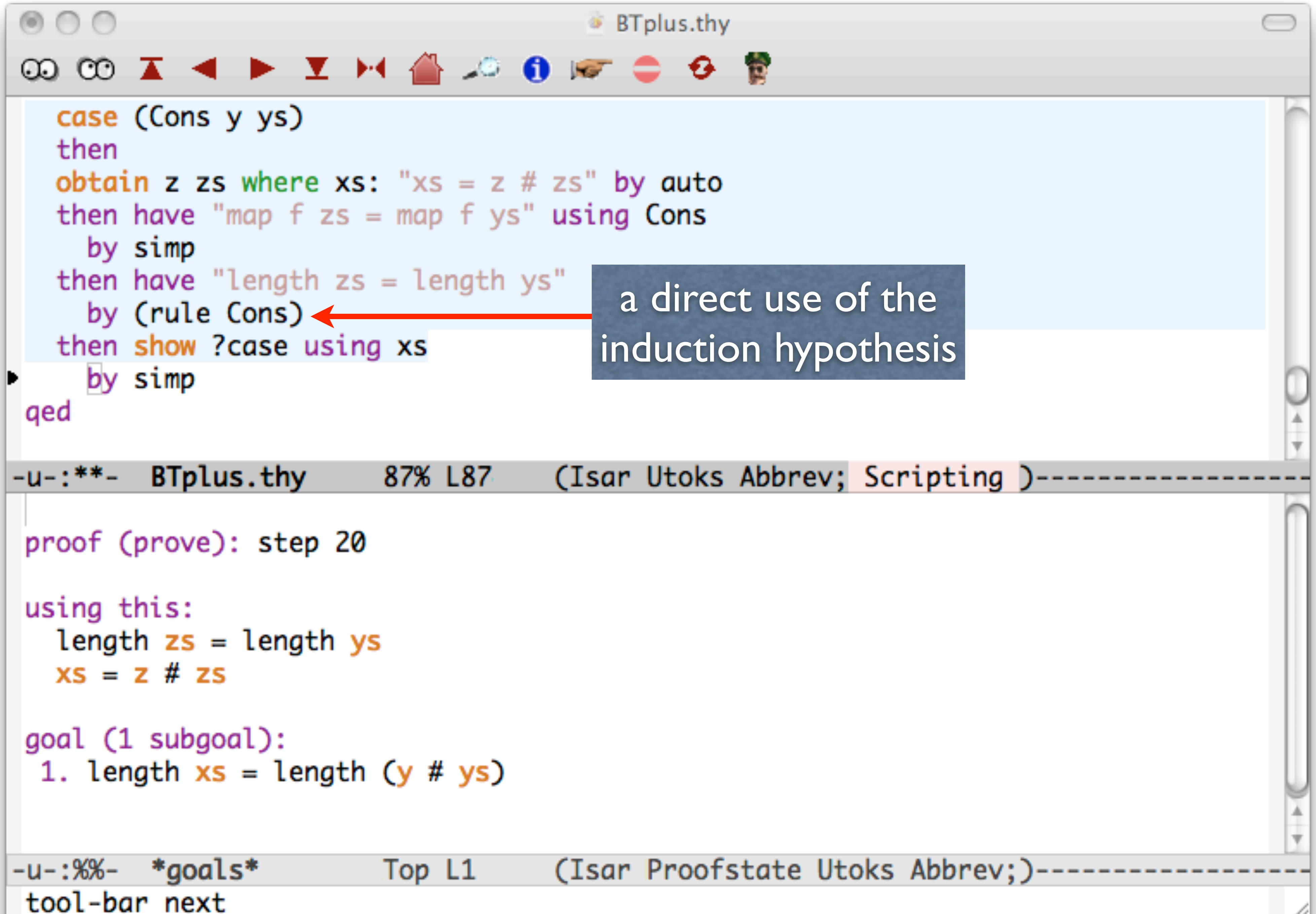
Facts from Two Sources



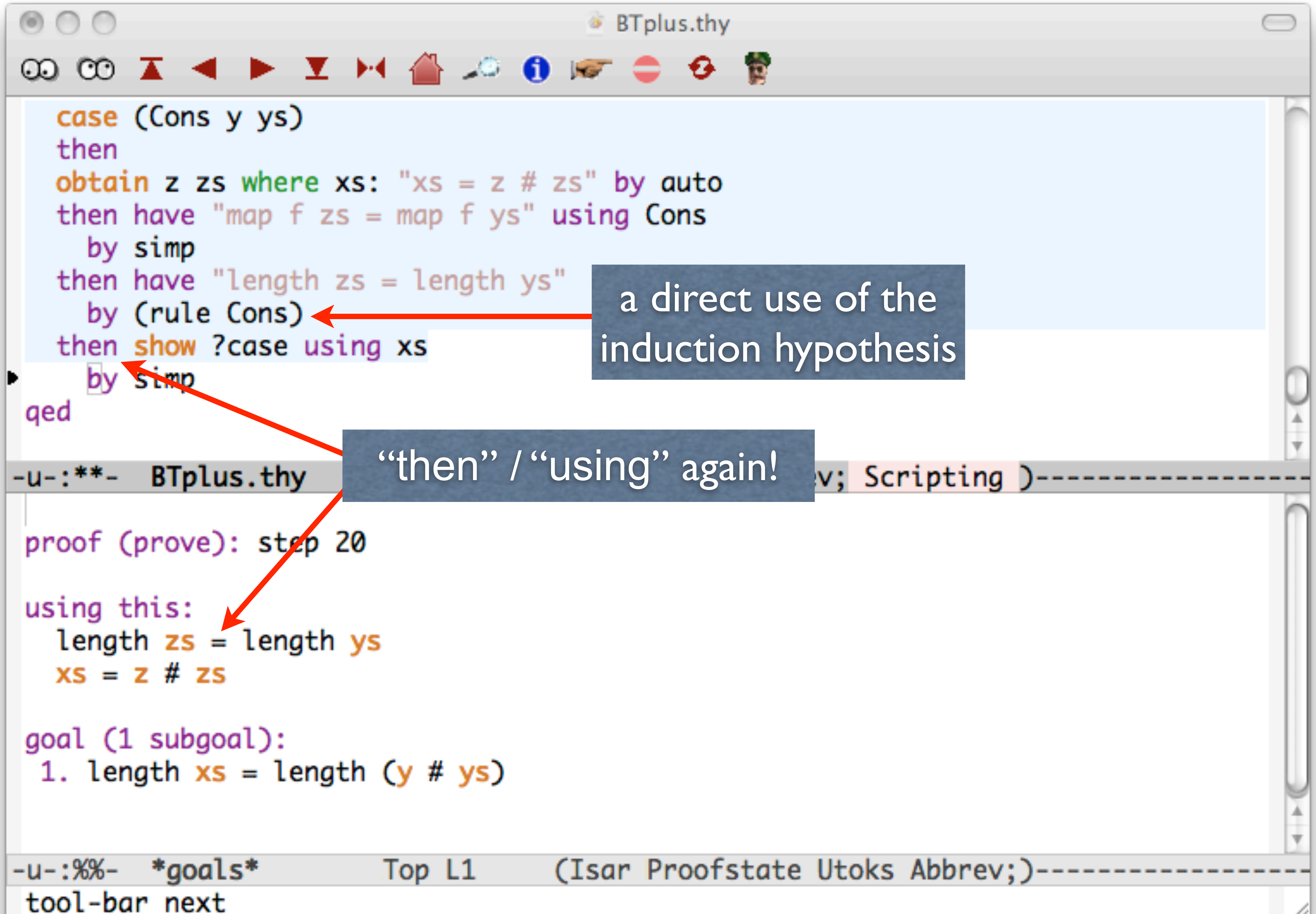
Finishing Up



Finishing Up



Finishing Up



The Complete Proof

```
BTplus.thy

lemma "map f xs = map f ys ⟹ length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
  then have "length zs = length ys"
    by (rule Cons)
  then show ?case using xs
    by simp
qed
-u-:**-  BTplus.thy      79% L73     (Isar Utoks Abbrev; Scripting )------------------
Successful attempt to solve goal by exported rule:
  ⟦⋀xs. map f xs = map f ?ysa2 ⟹ length xs = length ?ysa2;
   map f ?xsa2 = map f (?y2 # ?ysa2)⟧
  ⟹ length ?xsa2 = length (?y2 # ?ysa2)

-u-:%%-  *response*      All L4      (Isar Messages Utoks Abbrev;)-------------------
```

The Complete Proof

BTplus.thy

```
lemma "map f xs = map f ys ⟹ length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
  then have "length zs = length ys"
    by (rule Cons)
  then show ?case using xs
    by simp
qed
```

"then have" = "hence"

```
-u-:**-  BTplus.thy      79% L73     (Isar Utoks Abbrev; Scripting )-----------------
Successful attempt to solve goal by exported rule:
  ⟦⋀xs. map f xs = map f ?ysa2 ⟹ length xs = length ?ysa2;
   map f ?xsa2 = map f (?y2 # ?ysa2)⟧
  ⟹ length ?xsa2 = length (?y2 # ?ysa2)
-u-:%%-  *response*      All L4      (Isar Messages Utoks Abbrev;)-------------------
```

The Complete Proof

BTplus.thy

```
lemma "map f xs = map f ys ⟹ length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
  then have "length zs = length ys"
    by (rule Cons)
  then show ?case using xs
    by simp
qed
-u-:**-  BTplus.thy     79%
Successful attempt to solve goal by exported rule:
  ⟦⋀xs. map f xs = map f ?ysa2 ⟹ length xs = length ?ysa2;
   map f ?xsa2 = map f (?y2 # ?ysa2)⟧
  ⟹ length ?xsa2 = length (?y2 # ?ysa2)
-u-:%%-  *response*     All L4     (Isar Messages Utoks Abbrev;)
```

“then have” = “hence”

“then show” = “thus”

Additional Proof Structures

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  show "B ∈ Fin" using insertI True
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  hence "B = insert a (B - {a})" using False
    by auto
  also have "... ∈ Fin" using insertI Ba
    by blast
  finally show "B ∈ Fin" .
qed
```

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  with insertI show "B ∈ Fin"
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  with False have "B = insert a (B - {a})"
    by auto
  also from insertI Ba have "... ∈ Fin"
    by blast
  finally show "B ∈ Fin" .
qed
```

Additional Proof Structures

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  show "B ∈ Fin" using insertI True
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  hence "B = insert a (B - {a})" using False
    by auto
  also have "... ∈ Fin" using insertI Ba
    by blast
  finally show "B ∈ Fin" .
qed
```

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  with insertI show "B ∈ Fin"
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  with False have "B = insert a (B - {a})"
    by auto
  also from insertI Ba have "... ∈ Fin"
    by blast
  finally show "B ∈ Fin" .
qed
```

from ⟨*facts*⟩ ... = ... using ⟨*facts*⟩

Additional Proof Structures

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  show "B ∈ Fin" using insertI True
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  hence "B = insert a (B - {a})" using False
    by auto
  also have "... ∈ Fin" using insertI Ba
    by blast
  finally show "B ∈ Fin" .
qed
```

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  with insertI show "B ∈ Fin"
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  with False have "B = insert a (B - {a})"
    by auto
  also from insertI Ba have "... ∈ Fin"
    by blast
  finally show "B ∈ Fin" .
qed
```

from ⟨*facts*⟩ ... = ... using ⟨*facts*⟩

with ⟨*facts*⟩ ... = then from ⟨*facts*⟩ ...

Additional Proof Structures

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  show "B ∈ Fin" using insertI True
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  hence "B = insert a (B - {a})" using False
    by auto
  also have "... ∈ Fin" using insertI Ba
    by blast
  finally show "B ∈ Fin" .
qed
```

```
case (insertI A a B)
show "B ∈ Fin"
proof (cases "B ⊆ A")
  case True
  with insertI show "B ∈ Fin"
    by auto
next
  case False
  have Ba: "B - {a} ⊆ A" using `B ⊆ insert a A`
    by auto
  with False have "B = insert a (B - {a})"
    by auto
  also from insertI Ba have "... ∈ Fin"
    by blast
  finally show "B ∈ Fin" .
qed
```

from $\langle$*facts*$\rangle$ ... = ... using $\langle$*facts*$\rangle$

with $\langle$*facts*$\rangle$ ... = then from $\langle$*facts*$\rangle$ ...

(where ... is have / show / obtain)