

Interactive Formal Verification / / : Structured Induction Proofs

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

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)
  proof (state): step 1
    goal (2 subgoals):
      1. reflect (reflect Lf) = Lf
      2.  $\wedge a\ t1\ t2.$ 
          [[reflect (reflect t1) = t1; reflect (reflect t2) = t2]
            $\Rightarrow$  reflect (reflect (Br a t1 t2)) = Br a t1 t2
  
```

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

tool-bar goto

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)

proof (state): step 1
goal (2 subgoals):
1. reflect (reflect Lf) = Lf
2.  $\wedge a\ t1\ t2.$ 
   [[reflect (reflect t1) = t1; reflect (reflect t2) = t2]
   $\Rightarrow$  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

The screenshot shows the Emacs Isabelle interface. The menu bar includes Emacs, File, Edit, Options, Tools, Isabelle, Proof-General, Maths, Tokens, Buffers, and Help. The 'Isabelle' menu is open, and the 'Show Me' option is selected, which has opened a sub-menu. In this sub-menu, the 'Cases' option (with keyboard shortcut C-c C-a <h> <c>) is highlighted. Other options in the sub-menu include Facts, Term Bindings, Classical Rules, Induct/Cases Rules, Simplifier Rules, Theorems, Transitivity Rules, Antiquotations, Attributes, Commands, Inner Syntax, and Methods. The main editor window displays Isabelle code for a datatype 'a bt' and a function 'reflect'. The status bar at the bottom shows the current file 'BT.thy' at line 13, column 11, and the current buffer '*response*' at line 9, column 1.

```
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 (ref  
  
lemma reflect_reflect_ident: "reflect (reflect t) = t"  
proof (induct t)  
  
cases:  
  Lf:  
    let "?case" = "reflect (reflect Lf) = Lf"  
  Br:  
    fix a_ t1_ t2_  
    let "?case" = "reflect (reflect (Br a_ t1_ t2_)) = Br a_ t1_ t2_"  
    assume  
      Br.hyps: "reflect (reflect t1_) = t1_" "reflect (reflect t2_) = t2_"  
    and Br.premis:
```

Finding Predefined Cases

The screenshot shows the Emacs Isabelle interface. The menu bar includes Emacs, File, Edit, Options, Tools, Isabelle, Proof-General, Maths, Tokens, Buffers, and Help. The 'Isabelle' menu is open, and the 'Show Me' submenu is also open, with 'Cases (C-c C-a <h> <c>)' selected. The code editor displays the following Isabelle code:

```
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 (ref

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

cases:
Lf:
  let "?case" = "reflect (reflect Lf) = Lf"
Br:
  fix a_ t1_ t2_
  let "?case" = "reflect (reflect (Br a_ t1_ t2_)) = Br a_ t1_ t2_"
  assume
    Br.hyps: "reflect (reflect t1_) = t1_" "reflect (reflect t2_) = t2_"
  and Br.premis:
```

A callout box on the left contains the text "Built-in cases" with two red arrows pointing to the "cases:" and "Br:" sections of the code.

The status bar at the bottom shows: -u-:%%- *response* All L9 (Isar Messages Utoks Abbrev;)- menu-bar Isabelle Show Me Cases

Finding Predefined Cases

The screenshot shows the Emacs Isabelle interface. The menu bar includes 'Emacs', 'File', 'Edit', 'Options', 'Tools', 'Isabelle', 'Proof-General', 'Maths', 'Tokens', 'Buffers', and 'Help'. The 'Isabelle' menu is open, showing options like 'Logics', 'Commands', 'Show Me', 'Favourites', 'Settings', 'Start Isabelle', 'Exit Isabelle', 'Set Isabelle Command', and 'Help'. The 'Show Me' submenu is also open, listing various Isabelle constructs such as 'Cases', 'Facts', 'Term Bindings', 'Classical Rules', 'Induct/Cases Rules', 'Simplifier Rules', 'Theorems', 'Transitivity Rules', 'Antiquotations', 'Attributes', 'Commands', 'Inner Syntax', and 'Methods'. The 'Cases' option is highlighted. The code editor shows Isabelle code for a datatype 'a bt' and a function 'reflect'. A 'cases' block is shown with two cases: 'Lf' and 'Br'. Red arrows point from text boxes to the 'cases' block and the 'Br.hyps' line.

```
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 (ref  
  
lemma reflect_reflect_ident: "reflect (reflect t) = t"  
proof (induct t)  
  
-u-:**- BT.thy 11% L13 (Isar Utoks Abbrev; Sc  
cases:  
Lf:  
  let "?case" = "reflect (reflect Lf) = Lf"  
Br:  
  fix a_ t1_ t2_  
  let "?case" = "reflect (reflect (Br a_ t1_ t2_)) = Br a_ t1_ t2_"  
  assume  
  Br.hyps: "reflect (reflect t1_) = t1_" "reflect (reflect t2_) = t2_"  
  and Br.prem:  
  
-u-:%%- *response* All L9 (Isar Messages Utoks Abbrev;)-  
menu-bar Isabelle Show Me Cases
```

Built-in cases

name of induction hyps

Finding Predefined Cases

The screenshot shows the Emacs Isabelle interface. The menu bar includes Emacs, File, Edit, Options, Tools, Isabelle, Proof-General, Maths, Tokens, Buffers, and Help. The 'Isabelle' menu is open, showing options like Logics, Commands, Show Me, Favourites, Settings, Start Isabelle, Exit Isabelle, Set Isabelle Command, and Help. The 'Show Me' submenu is also open, listing various Isabelle constructs such as Cases, Facts, Term Bindings, Classical Rules, Induct/Cases Rules, Simplifier Rules, Theorems, Transitivity Rules, Antiquotations, Attributes, Commands, Inner Syntax, and Methods.

The code editor displays the following Isabelle code:

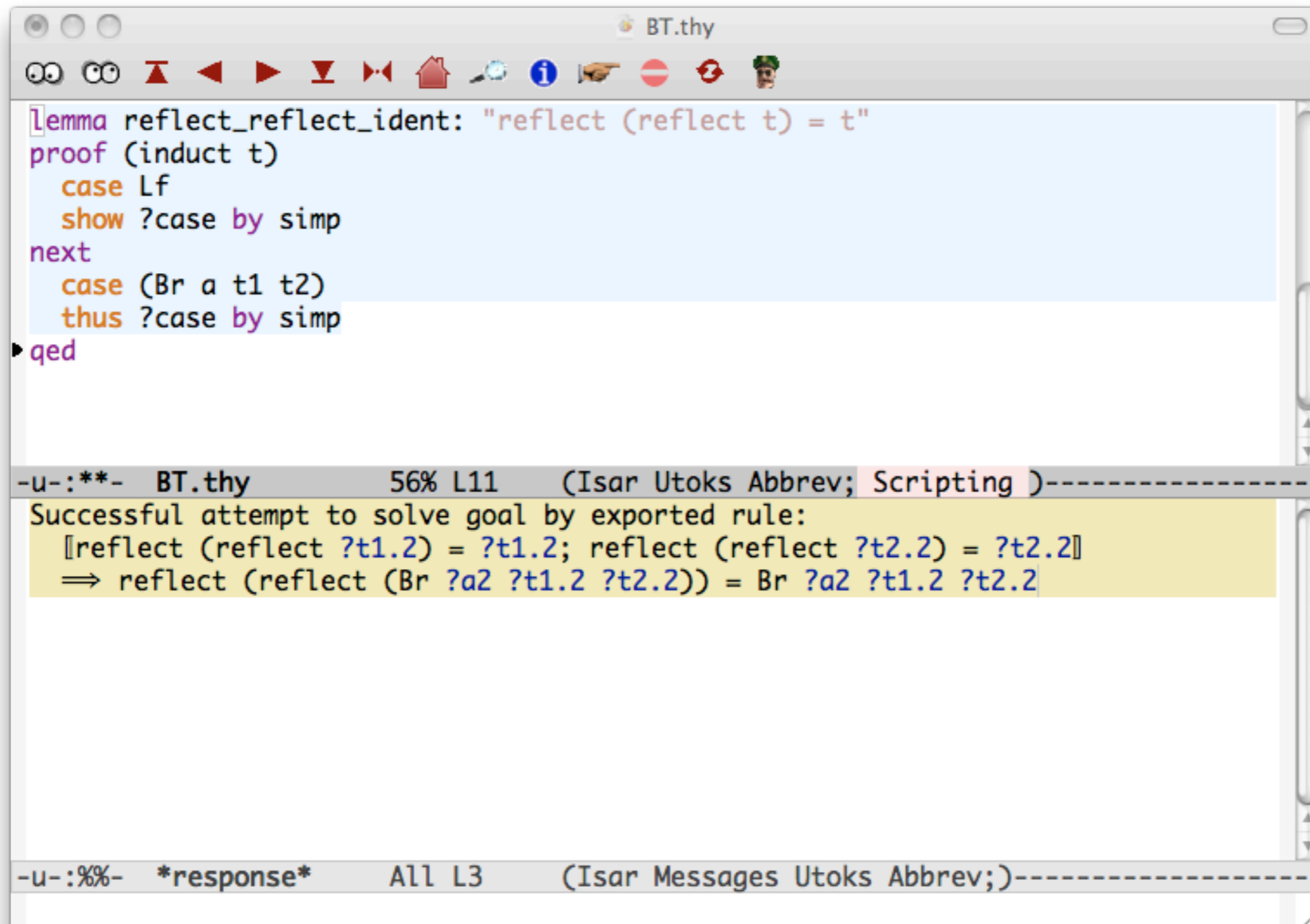
```
datatype 'a bt =  
  Lf  
  | Br 'a "'a bt" "'a bt"  
  
fun reflect :: "'a bt => 'a bt" where  
  "reflect Lf = Lf"  
| "reflect (Br a t1 t2) = Br a (ref  
  
lemma reflect_reflect_ident: "reflect (reflect t) = t"  
proof (induct t)  
  
cases:  
  Lf:  
    let "?case" = "reflect (reflect Lf) = Lf"  
  Br:  
    fix a_ t1_ t2_  
    let "?case" = "reflect (reflect (Br a_ t1_ t2_)) = Br a_ t1_ t2_"  
    assume  
      Br.hyps: "reflect (reflect t1_) = t1_" "reflect (reflect t2_) = t2_"  
    and Br.premis:
```

Annotations in the image include:

- A blue box labeled "Built-in cases" with red arrows pointing to the `Lf:` and `Br:` sections of the `cases:` block.
- A blue box labeled "name of induction hyps" with a red arrow pointing to the `Br.hyps:` line.
- A blue box labeled "abbreviation of conclusion" with a red arrow pointing to the `?case` variable in the `Lf:` case.

The status bar at the bottom shows: `-u-:%%- *response* All L9 (Isar Messages Utoks Abbrev;)-` and the menu bar indicates `menu-bar Isabelle Show Me Cases`.

The Finished Proof



The screenshot shows a window titled "BT.thy" with a toolbar at the top. The main text area contains the following proof script:

```
[lemma reflect_reflect_ident: "reflect (reflect t) = t"  
proof (induct t)  
  case Lf  
  show ?case by simp  
next  
  case (Br a t1 t2)  
  thus ?case by simp  
qed
```

Below the code is a message window with a grey header: "-u-:**- BT.thy 56% L11 (Isar Utoks Abbrev; Scripting)-----". The message content is highlighted in yellow and reads:

```
Successful attempt to solve goal by exported rule:  
[[reflect (reflect ?t1.2) = ?t1.2; reflect (reflect ?t2.2) = ?t2.2]]  
⇒ reflect (reflect (Br ?a2 ?t1.2 ?t2.2)) = Br ?a2 ?t1.2 ?t2.2
```

At the bottom, another message window header is visible: "-u-:%%- *response* All L3 (Isar Messages Utoks Abbrev;)------".

The Finished Proof

The screenshot shows a theorem prover interface with a code editor and a message window. The code editor contains the following text:

```
[lemma reflect_reflect_ident: "reflect (reflect t) = t"  
proof (induct t)  
  case Lf  
  show ?case by simp  
next  
  case (Br a t1 t2)  
  thus ?case by simp  
qed
```

Two red arrows point from a dark blue box containing the text "the two cases" to the lines `case Lf` and `case (Br a t1 t2)` in the code. Below the code editor, a message window displays the following text:

```
-u-:***- BT.thy 56% L11 (Isar Utoks Abbrev; Scripting )-----  
Successful attempt to solve goal by exported rule:  
[[reflect (reflect ?t1.2) = ?t1.2; reflect (reflect ?t2.2) = ?t2.2]  
=> reflect (reflect (Br ?a2 ?t1.2 ?t2.2)) = Br ?a2 ?t1.2 ?t2.2
```

At the bottom of the interface, another message window shows:

```
-u-:%%- *response* All L3 (Isar Messages Utoks Abbrev;)
```

The Finished Proof

```
BT.thy
[lemma reflect_reflect_ident: "reflect (reflect t) = t"
proof (induct t)
  case Lf
  show ?case by simp
next
  case (Br a t1 t2)
  thus ?case by simp
qed
```

the two cases

instances of the goal

-u-:***- BT.thy 56% L11 (Isar Utoks Abbrev; Scripting)-----

Successful attempt to solve goal by exported rule:
[[reflect (reflect ?t1.2) = ?t1.2; reflect (reflect ?t2.2) = ?t2.2]]
⇒ reflect (reflect (Br ?a2 ?t1.2 ?t2.2)) = Br ?a2 ?t1.2 ?t2.2

-u-:%%- *response* All L3 (Isar Messages Utoks Abbrev;)

The Finished Proof

```
BT.thy
[lemma reflect_reflect_ident: "reflect (reflect t) = t"
proof (induct t)
  case Lf
  show ?case by simp
next
  case (Br a t1 t2)
  thus ?case by simp
qed
```

Successful attempt to solve goal by exported rule:
[[reflect (reflect ?t1.2) = ?t1.2; reflect (reflect ?t2.2) = ?t2.2]]
⇒ reflect (reflect (Br ?a2 ?t1.2 ?t2.2)) = Br ?a2 ?t1.2 ?t2.2

instances of the goal

the two cases

list of bound variables

The Finished Proof

```
BT.thy
[lemma reflect_reflect_ident: "reflect (reflect t) = t"
proof (induct t)
  case Lf
  show ?case by simp
next
  case (Br a t1 t2)
  thus ?case by simp
qed
```

Successful attempt to solve goal by exported rule:
[[reflect (reflect ?t1.2) = ?t1.2; reflect (reflect ?t2.2) = ?t2.2]]
⇒ reflect (reflect (Br ?a2 ?t1.2 ?t2.2)) = Br ?a2 ?t1.2 ?t2.2

response All L3 (Isar Messages Utoks Abbrev;)

instances of the goal

the two cases

list of bound variables

Isabelle has proved the induction step

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.prem: "B ⊆ {}"
  insertI:
    fix A_ a_ B
    let "?case" = "B ∈ Fin"
    assume insertI.hyps: "A_ ∈ Fin" "\B. B ⊆ A_ ==> B ∈ Fin" and
      insertI.prem: "B ⊆ insert a_ A_"
-u-:%%- *response*     All L10    (Isar Messages Utoks Abbrev;)
```

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.prem: "B ⊆ {}"
  insertI:
    fix A_ a_ B
    let "?case" = "B ∈ Fin"
    assume insertI.hyps: "A_ ∈ Fin" "∧B. B ⊆ A_ ==> B ∈ Fin" and
      insertI.prem: "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% L2 (Isar Utoks Abbrev; Scripting )-----
cases:
  emptyI:
    fix B
    let "?case" = "B ∈ Fin"
    assume emptyI.hyps: and emptyI.prem: "B ⊆ {}"
  insertI:
    fix A_ a_ B
    let "?case" = "B ∈ Fin"
    assume insertI.hyps: "A_ ∈ Fin" "∧B. B ⊆ A_ ==> B ∈ Fin" and
      insertI.prem: "B ⊆ insert a_ A_"
-u-:%%- *response* All L10 (Isar Messages Utoks Abbrev;)
```

a named induction rule

an arbitrary variable

A More Sophisticated Proof

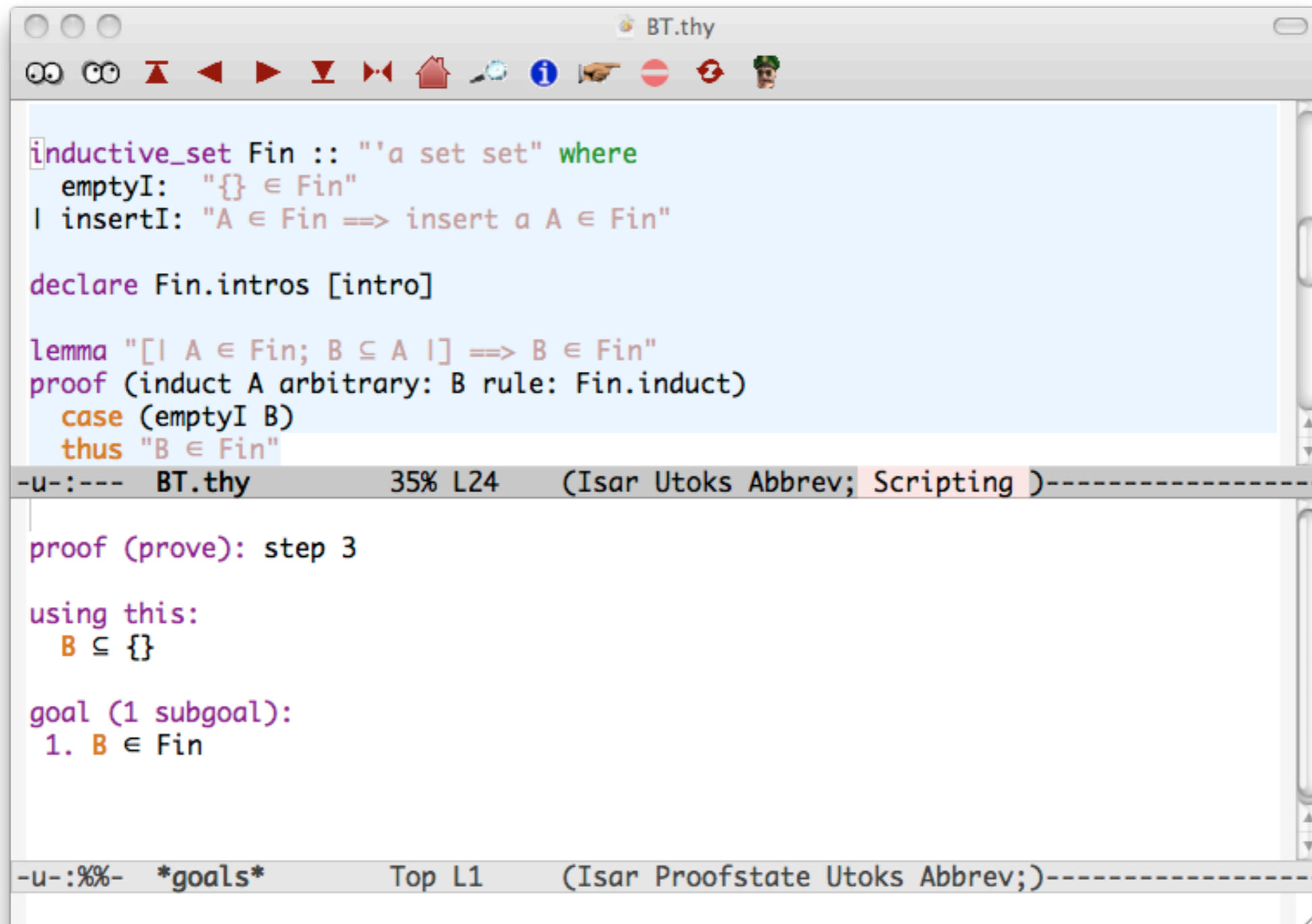
```
BT.thy
text{*The finite powerset operator*}
inductive_set Fin :: "'a set set" where
  emptyI: "{} ∈ Fin"
  | insertI: "A ∈ Fin ==> insert a A ∈ Fin"
declare Fin.intros [intro]
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
-u-:--- BT.thy 29% L2 (Isar Utoks Abbrev; Scripting )-----
cases:
  emptyI:
    fix B
    let "?case" = "B ∈ Fin"
    assume emptyI.hyps: and emptyI.premis: "B ⊆ {}"
  insertI:
    fix A_ a_ B
    let "?case" = "B ∈ Fin"
    assume insertI.hyps: "A_ ∈ Fin" "∧B. B ⊆ A_ ==> B ∈ Fin" and
      insertI.premis: "B ⊆ insert a_ A_"
-u-:%%- *response* All L10 (Isar Messages Utoks Abbrev;)
```

a *named* induction rule

an *arbitrary* variable

non-empty premises

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

```
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;)-----
```

“thus” makes the premise available

Proving the Base Case

```
BT.thy

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

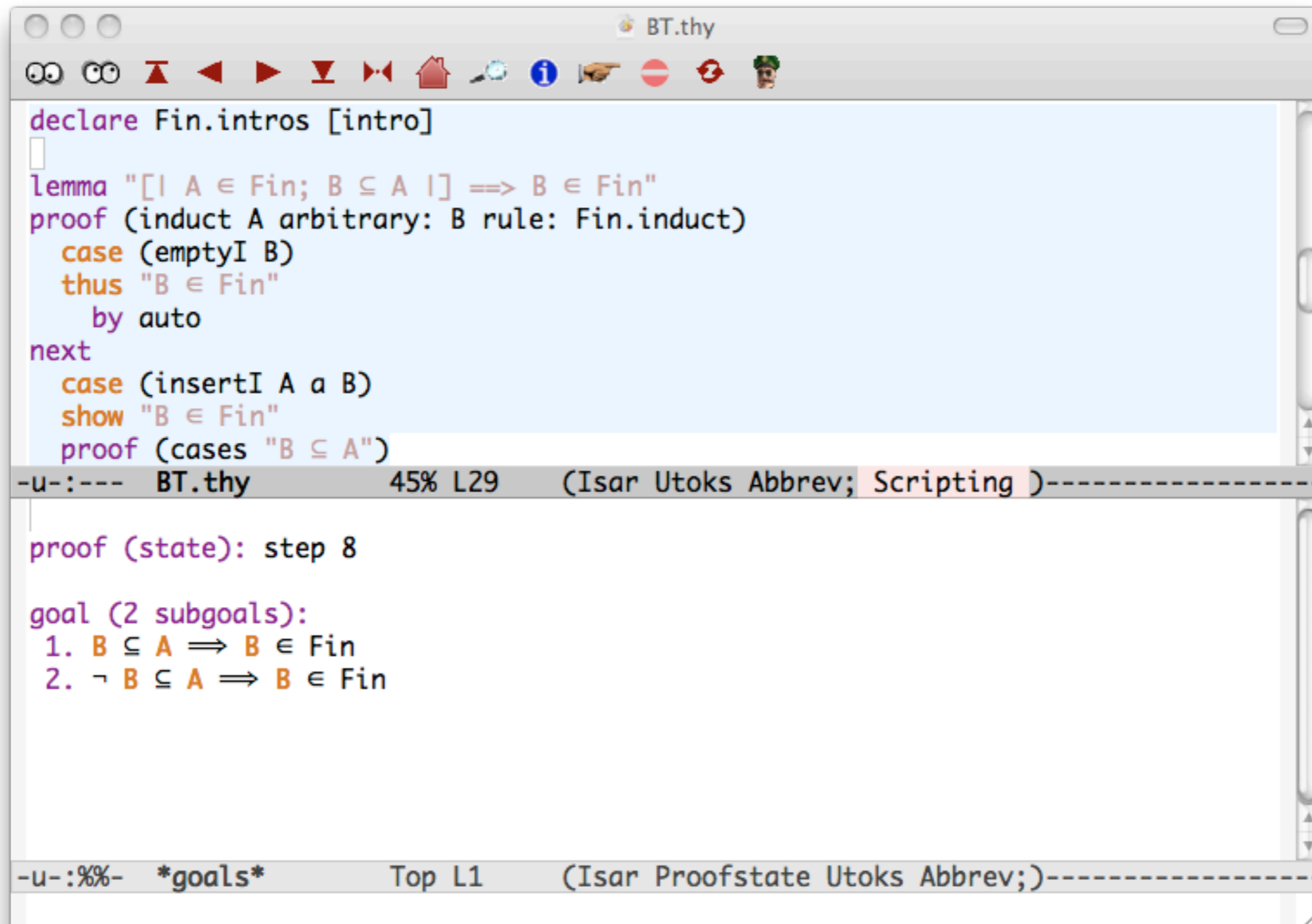
declare Fin.intros [intro]

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% |
proof (prove): step 3
using this:
  B ⊆ {}
goal (1 subgoal):
  1. B ∈ Fin
- u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
```

“arbitrary” variables must be named!

“thus” makes the premise available

A Nested Case Analysis



```
BT.thy
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"
  by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case true
    thus "B ∈ Fin"
    by auto
  case false
  thus "B ∈ Fin"
  by auto
qed

-u-:--- BT.thy 45% L29 (Isar Utoks Abbrev; Scripting )-----

proof (state): step 8

goal (2 subgoals):
1. B ⊆ A ==> B ∈ Fin
2. ¬ B ⊆ A ==> B ∈ Fin

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

A Nested Case Analysis

```
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"
  by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case (state)
    proof (state): step 8
      goal (2 subgoals):
      1. B ⊆ A ⇒ B ∈ Fin
      2. ¬ B ⊆ A ⇒ B ∈ Fin
```

BT.thy

45% L29 (Isar Utoks Abbrev; Scripting)

goals Top L1 (Isar Proofstate Utoks Abbrev;)

A Nested Case Analysis

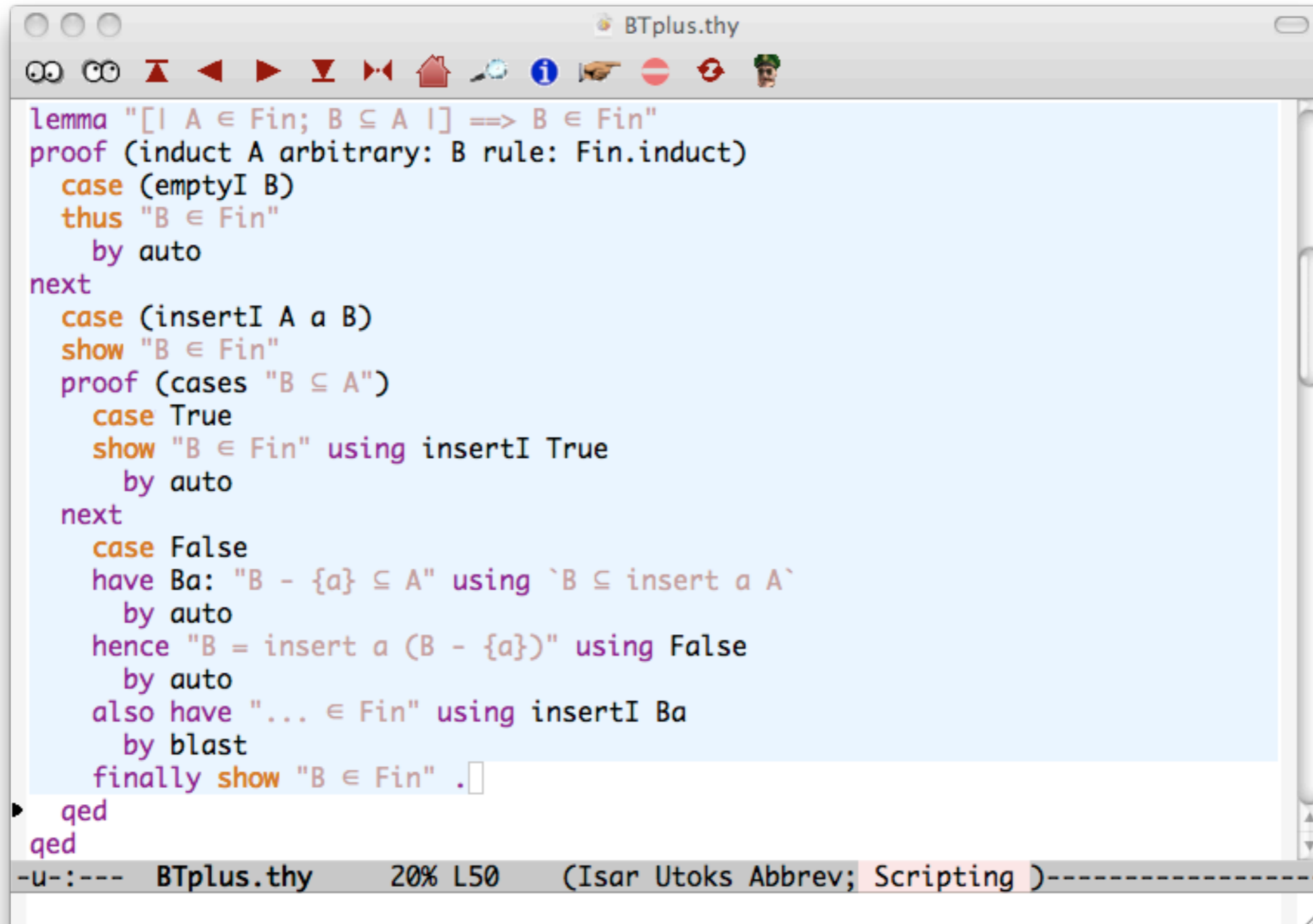
```
declare Fin.intros [intro]
[]
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
proof (induct A arbitrary: B rule: Fin.induct)
  case (emptyI B)
  thus "B ∈ Fin"
  by auto
next
  case (insertI A a B)
  show "B ∈ Fin"
  proof (cases "B ⊆ A")
    case (emptyI B)
    thus "B ∈ Fin"
    by auto
  case (insertI A a B)
  thus "B ∈ Fin"
  by auto
qed
```

“arbitrary” variables must (again) be named!

case analysis on this formula

```
proof (state): step 8
goal (2 subgoals):
1. B ⊆ A ==> B ∈ Fin
2. ¬ B ⊆ A ==> B ∈ Fin
```

The Complete Proof



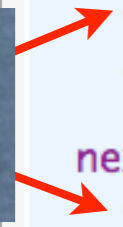
```
BTplus.thy
[Navigation icons: back, forward, search, etc.]

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
```

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
```

induction hypothesis and premise

true and false cases

-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
```

induction hypothesis and premise

the true case: $B \subseteq A$

true and false cases

the false case: $\neg B \subseteq A$

The Complete Proof

```
BTplus.thy
[... navigation icons ...]
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
```

induction hypothesis and premise

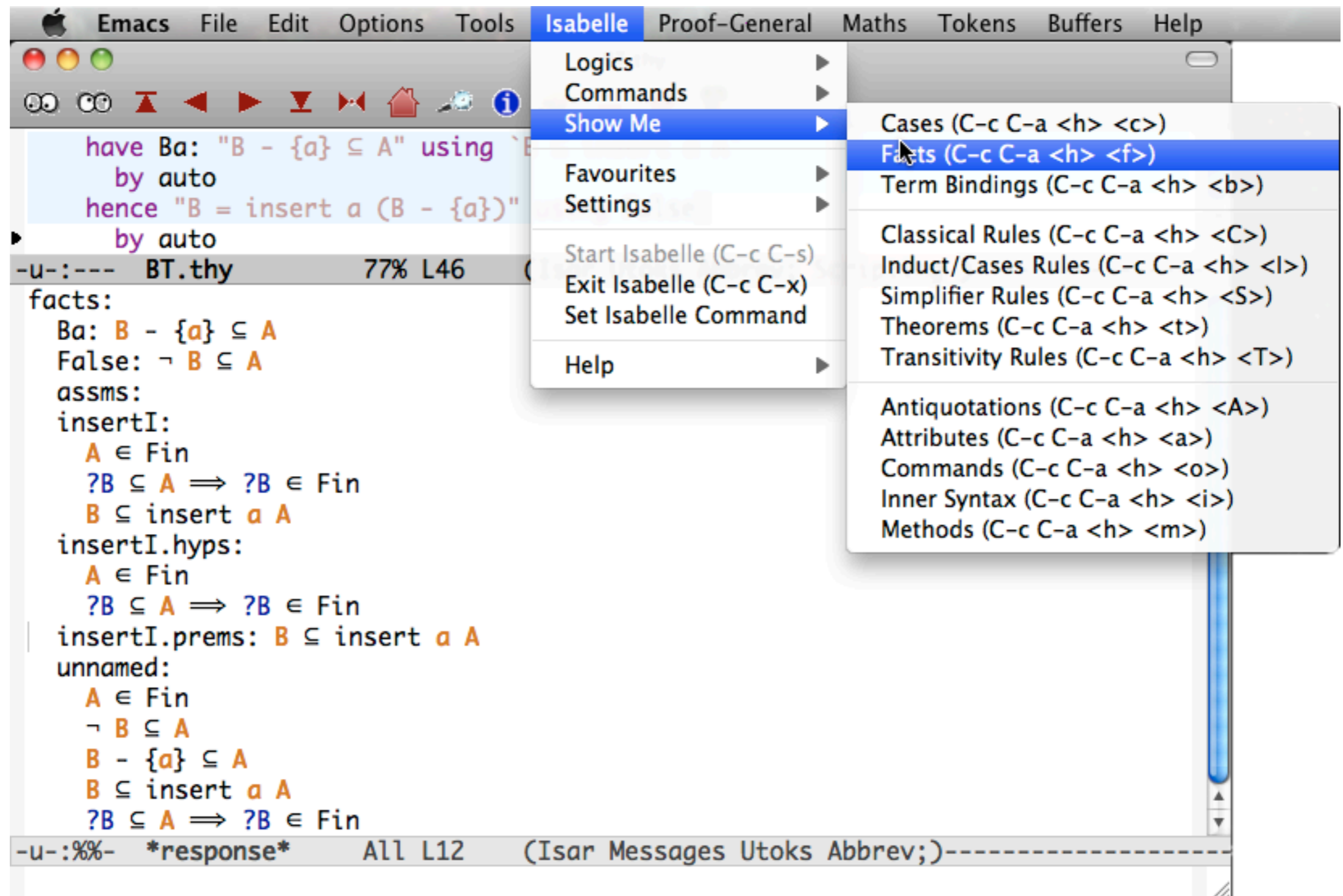
the true case: $B \subseteq A$

direct quotation of a fact

the false case: $\neg B \subseteq A$

true and false cases

Which Theorems are Available?



The screenshot shows the Emacs Isabelle interface. The menu is open, displaying the following options:

- Logics
- Commands
- Show Me
 - Cases (C-c C-a <h> <c>)
 - Facts (C-c C-a <h> <f>)
 - Term Bindings (C-c C-a <h>)
- Favourites
- Settings
- Start Isabelle (C-c C-s)
- Exit Isabelle (C-c C-x)
- Set Isabelle Command
- Help

The main window displays the following code:

```
have Ba: "B - {a} ⊆ A" using `E
  by auto
hence "B = insert a (B - {a})"
  by auto
```

The status bar shows: -u-:--- BT.thy 77% L46

The bottom status bar shows: -u-:%%- *response* All L12 (Isar Messages Utoks Abbrev;)-

Which Theorems are Available?

The screenshot shows the Emacs Isabelle interface. The menu is open, displaying various theorem categories. A callout box points to the 'Facts' entry in the menu and the 'facts:' section in the script, indicating that facts are recently proved.

Menu Items:

- Logics
- Commands
- Show Me
 - Cases (C-c C-a <h> <c>)
 - Facts (C-c C-a <h> <f>)
 - Term Bindings (C-c C-a <h>)
- Favourites
- Settings
 - Classical Rules (C-c C-a <h> <C>)
 - Induct/Cases Rules (C-c C-a <h> <I>)
 - Simplifier Rules (C-c C-a <h> <S>)
 - Theorems (C-c C-a <h> <t>)
 - Transitivity Rules (C-c C-a <h> <T>)
- Help
 - Antiquotations (C-c C-a <h> <A>)
 - Attributes (C-c C-a <h> <a>)
 - Commands (C-c C-a <h> <o>)
 - Inner Syntax (C-c C-a <h> <i>)
 - Methods (C-c C-a <h> <m>)

Script Content:

```
have Ba: "B - {a} ⊆ A" using `E
  by auto
hence "B = insert a (B - {a})"
  by auto
-u:--- BT.thy
facts:
  Ba: B - {a} ⊆ A
  False: ¬ B ⊆ A
assms:
insertI:
  A ∈ Fin
  ?B ⊆ A ⇒ ?B ∈ Fin
  B ⊆ insert a A
insertI.hyps:
  A ∈ Fin
  ?B ⊆ A ⇒ ?B ∈ Fin
insertI.prem: B ⊆ insert a A
unnamed:
  A ∈ Fin
  ¬ B ⊆ A
  B - {a} ⊆ A
  B ⊆ insert a A
  ?B ⊆ A ⇒ ?B ∈ Fin
-u:%%- *response* All L12 (Isar Messages Utoks Abbrev;)
```

Which Theorems are Available?

The screenshot shows the Emacs Isabelle interface. The menu bar includes: Emacs, File, Edit, Options, Tools, Isabelle, Proof-General, Maths, Tokens, Buffers, Help. The 'Isabelle' menu is open, showing options: Logics, Commands, Show Me, Favourites, Settings, Isabelle (C-c C-s), Exit Isabelle (C-c C-x), and Set Isabelle Command. The 'Show Me' submenu is also open, listing various theorem categories with their shortcuts: Cases (C-c C-a <h> <c>), Facts (C-c C-a <h> <f>), Term Bindings (C-c C-a <h>), Classical Rules (C-c C-a <h> <C>), Induct/Cases Rules (C-c C-a <h> <I>), Simplifier Rules (C-c C-a <h> <S>), Theorems (C-c C-a <h> <t>), Transitivity Rules (C-c C-a <h> <T>), Antiquotations (C-c C-a <h> <A>), Attributes (C-c C-a <h> <a>), Commands (C-c C-a <h> <o>), Inner Syntax (C-c C-a <h> <i>), and Methods (C-c C-a <h> <m>).

The main window displays a proof script for BT.thy. The script includes the following sections:

```
have Ba: "B - {a} ⊆ A" using `E
  by auto
hence "B = insert a (B - {a})"
  by auto
```

Below this, the 'facts' section lists:

```
facts:
  Ba: B - {a} ⊆ A
  False: ¬ B ⊆ A
```

The 'assms' section lists:

```
insertI:
  A ∈ Fin
  ?B ⊆ A ⇒ ?B ∈ Fin
  B ⊆ insert a A
```

The 'insertI.hyps' section lists:

```
insertI.hyps:
  A ∈ Fin
  ?B ⊆ A ⇒ ?B ∈ Fin
```

The 'insertI.premis' section lists:

```
insertI.premis: B ⊆ insert a A
```

The 'unnamed' section lists:

```
unnamed:
  A ∈ Fin
  ¬ B ⊆ A
  B - {a} ⊆ A
  B ⊆ insert a A
  ?B ⊆ A ⇒ ?B ∈ Fin
```

Two red arrows point from text boxes to the script. One points to the 'by auto' line of the 'have' statement, labeled "a recently proved fact". The other points to the 'False' line in the 'facts' section, labeled "the false case: ¬ B ⊆ A".

The status bar at the bottom shows: -u-:%%- *response* All L12 (Isar Messages Utoks Abbrev;)-

Which Theorems are Available?

The screenshot shows the Emacs Isabelle interface. The menu is open, displaying various theorem categories. The 'Facts' option is highlighted. The proof script in the background contains several sections: 'facts:', 'assms:', 'insertI:', 'insertI.hyps:', 'insertI.prem:', and 'unnamed:'. Red arrows point from text boxes to specific elements in the script.

Menu Items:

- Logics
- Commands
- Show Me
 - Cases (C-c C-a <h> <c>)
 - Facts (C-c C-a <h> <f>)
 - Term Bindings (C-c C-a <h>)
- Favourites
- Settings
 - Classical Rules (C-c C-a <h> <C>)
 - Induct/Cases Rules (C-c C-a <h> <I>)
 - Simplifier Rules (C-c C-a <h> <S>)
 - Theorems (C-c C-a <h> <t>)
 - Transitivity Rules (C-c C-a <h> <T>)
- Antiquotations (C-c C-a <h> <A>)
- Attributes (C-c C-a <h> <a>)
- Commands (C-c C-a <h> <o>)
- Inner Syntax (C-c C-a <h> <i>)
- Methods (C-c C-a <h> <m>)

Proof Script Content:

```
have Ba: "B - {a} ⊆ A" using `E
  by auto
hence "B = insert a (B - {a})"
  by auto
-u:--- BT.thy
facts:
  Ba: B - {a} ⊆ A
  False: ¬ B ⊆ A
assms:
insertI:
  A ∈ Fin
  ?B ⊆ A ⇒ ?B ∈ Fin
  B ⊆ insert a A
insertI.hyps:
  A ∈ Fin
  ?B ⊆ A ⇒ ?B ∈ Fin
insertI.prem: B ⊆ insert a A
unnamed:
  A ∈ Fin
  ¬ B ⊆ A
  B - {a} ⊆ A
  B ⊆ insert a A
  ?B ⊆ A ⇒ ?B ∈ Fin
-u:%%- *response* All L12 (Isar Messages Utoks Abbrev;)-
```

Annotations:

- a recently proved fact (points to the highlighted 'have' block)
- the false case: $\neg B \subseteq A$ (points to 'False: $\neg B \subseteq A$ ')
- facts for the case insertI (points to the 'insertI:' section)

Which Theorems are Available?

The screenshot shows the Emacs Isabelle interface with the 'Isabelle' menu open. The menu items are:

- Logics
- Commands
- Show Me
- Favourites
- Settings
- elle (C-c C-s)
- Isabelle (C-c C-x)
- Set Isabelle Command

The 'Show Me' submenu is open, listing various theorem and fact categories:

- Cases (C-c C-a <h> <c>)
- Facts (C-c C-a <h> <f>)
- Term Bindings (C-c C-a <h>)
- Classical Rules (C-c C-a <h> <C>)
- Induct/Cases Rules (C-c C-a <h> <I>)
- Simplifier Rules (C-c C-a <h> <S>)
- Theorems (C-c C-a <h> <t>)
- Transitivity Rules (C-c C-a <h> <T>)
- Antiquotations (C-c C-a <h> <A>)
- Attributes (C-c C-a <h> <a>)
- Commands (C-c C-a <h> <o>)
- Inner Syntax (C-c C-a <h> <i>)
- Methods (C-c C-a <h> <m>)

The main editor window shows the following code:

```
have Ba: "B - {a} ⊆ A" using `E
  by auto
hence "B = insert a (B - {a})"
  by auto
```

Below this, the 'facts' section is expanded:

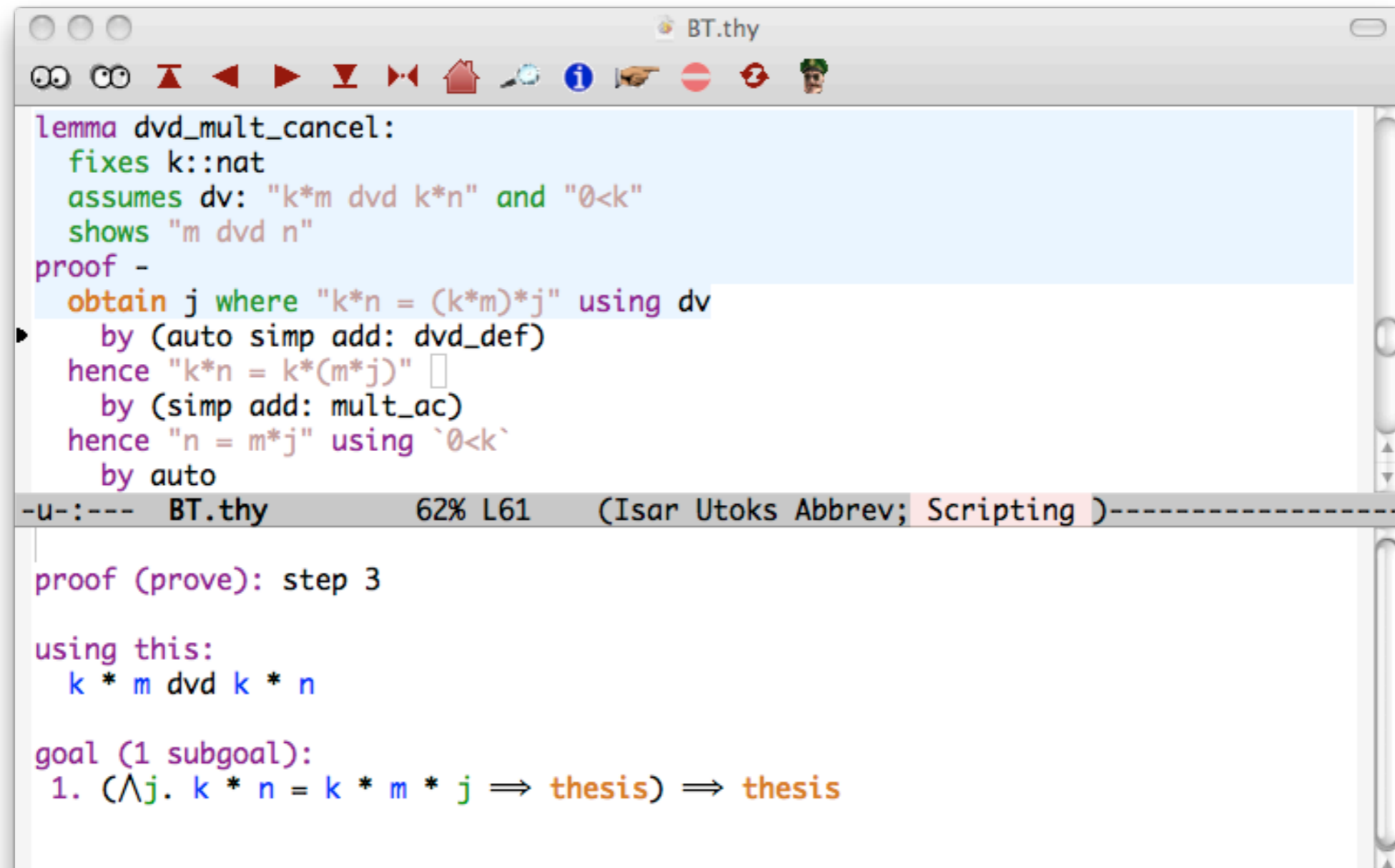
```
facts:
  Ba: B - {a} ⊆ A
  False: ¬ B ⊆ A
  assms:
  insertI:
    A ∈ Fin
    ?B ⊆ A ⇒ ?B ∈ Fin
    B ⊆ insert a A
  insertI.hyps:
    A ∈ Fin
    ?B ⊆ A ⇒ ?B ∈ Fin
  insertI.prem: B ⊆ insert a A
  unnamed:
    A ∈ Fin
    ¬ B ⊆ A
    B - {a} ⊆ A
    B ⊆ insert a A
    ?B ⊆ A ⇒ ?B ∈ Fin
```

Annotations with red arrows point to specific parts of the code:

- "a recently proved fact" points to the `have Ba` line.
- "the false case: $\neg B \subseteq A$ " points to the `False` line.
- "facts for the case insertI" points to the `insertI` section.
- "separate hyps and prems for insertI" points to the `insertI.hyps` and `insertI.prem` lines.

The status bar at the bottom shows: `-u-:%%- *response* All L12 (Isar Messages Utoks Abbrev;)-`

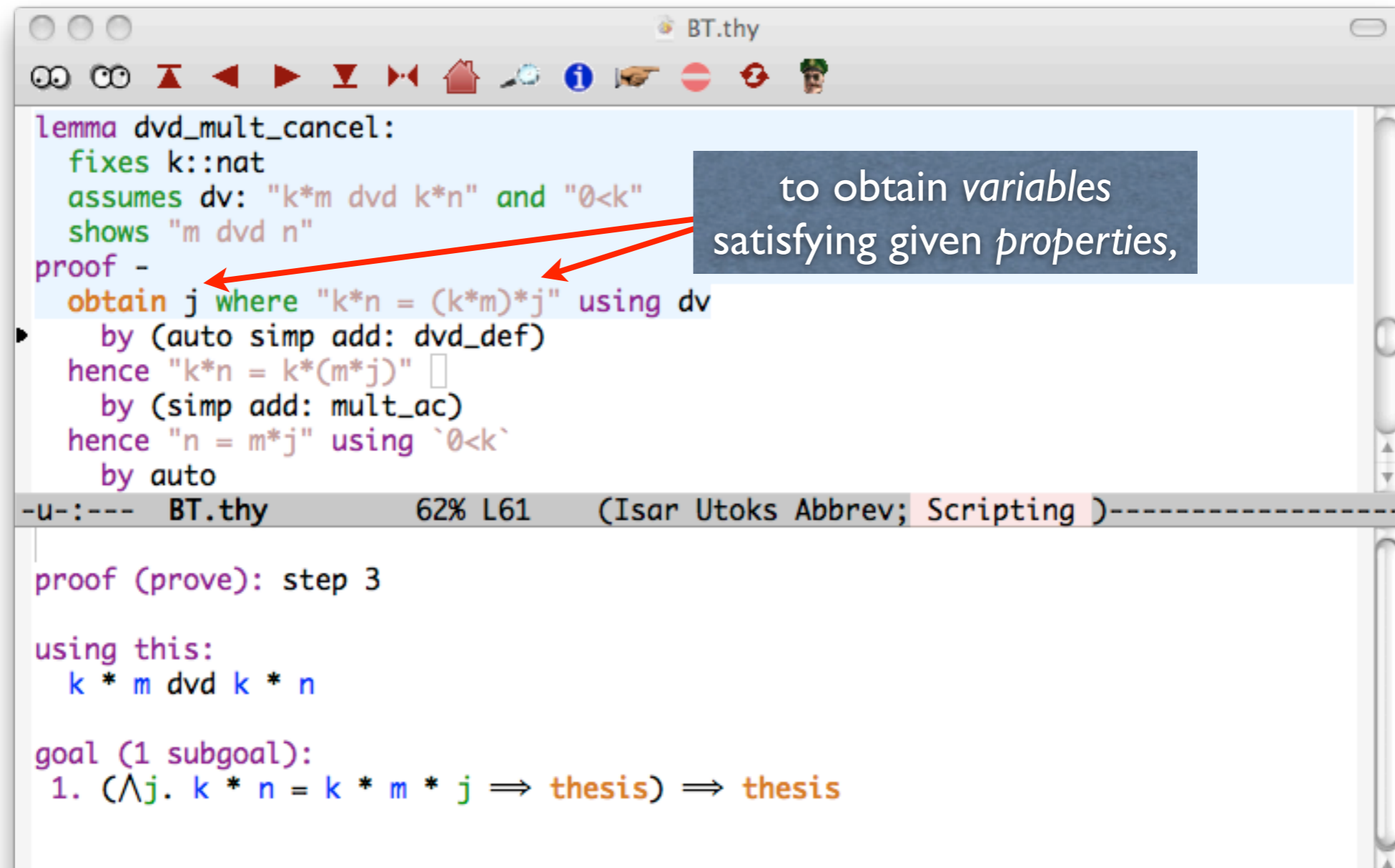
Existential Claims: “obtain”



```
BT.thy
--u-:--- BT.thy 62% L61 (Isar Utoks Abbrev; Scripting )-----
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
proof (prove): step 3
using this:
  k * m dvd k * n
goal (1 subgoal):
  1. (∧j. k * n = k * m * j ⇒ thesis) ⇒ thesis
```

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

Existential Claims: “obtain”



The screenshot shows a window titled "BT.thy" with a toolbar and a code editor. The code defines a lemma `dvd_mult_cancel` and its proof. A blue box with white text and two red arrows points to the `obtain` statement in the proof. The text in the box says "to obtain variables satisfying given properties,". The code in the editor is as follows:

```
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
```

Below the code editor, a status bar shows "BT.thy 62% L61 (Isar Utoks Abbrev; Scripting)". The main proof area shows:

```
proof (prove): step 3
  using this:
    k * m dvd k * n
  goal (1 subgoal):
    1. (∧j. k * n = k * m * j ⇒ thesis) ⇒ thesis
```

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

Existential Claims: “obtain”

```
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
```

to obtain *variables* satisfying given *properties*,

```
proof (prove): step 3
using this:
  k * m dvd k * n
goal (1 subgoal):
  1. (∧j. k * n = k * m * j ⇒ thesis) ⇒ thesis
```

... Isabelle needs to prove an *elimination rule*

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

Continuing the Proof

```
BT.thy
lemmas 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
```

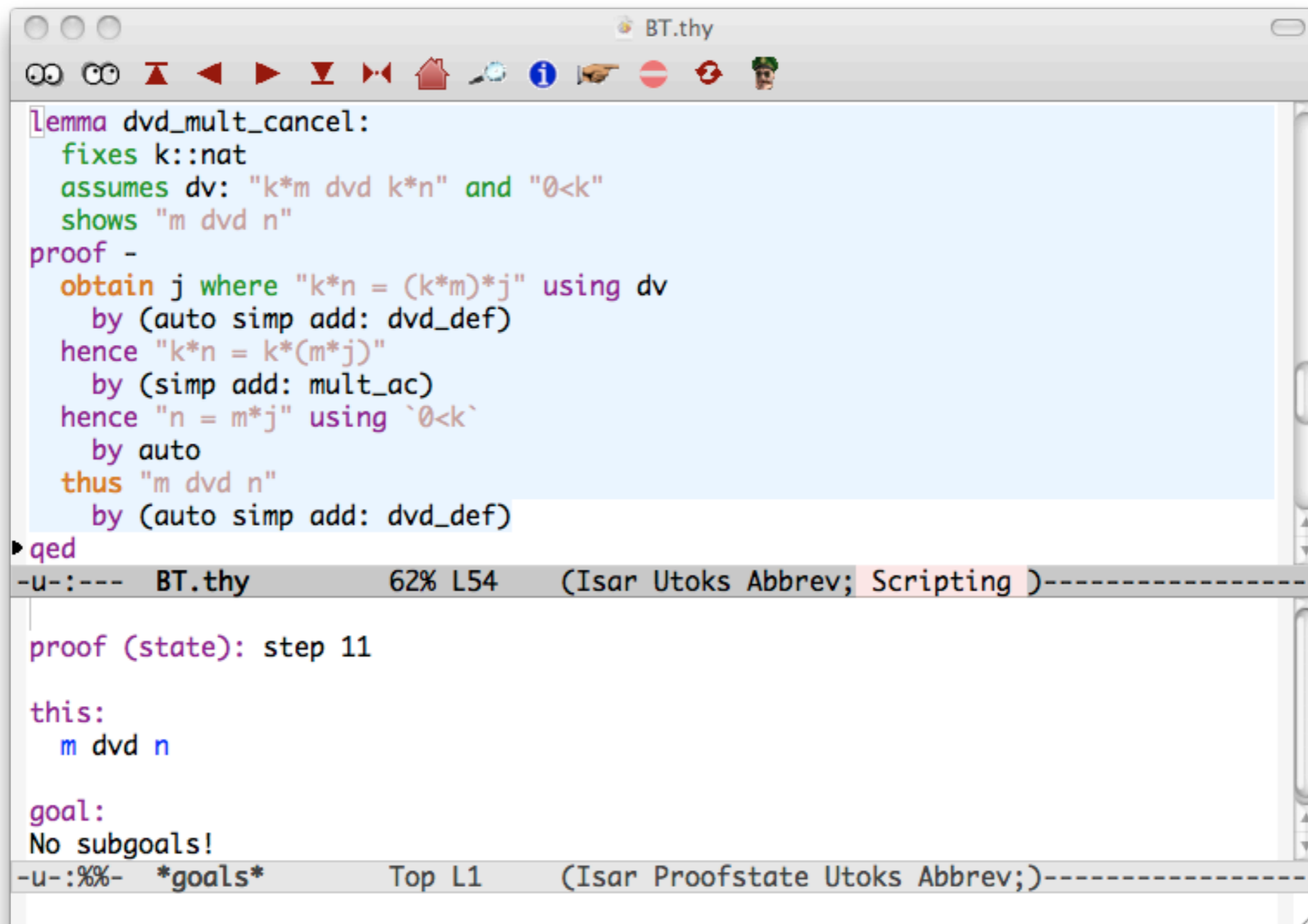
Continuing the Proof

```
BT.thy
lemmas 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

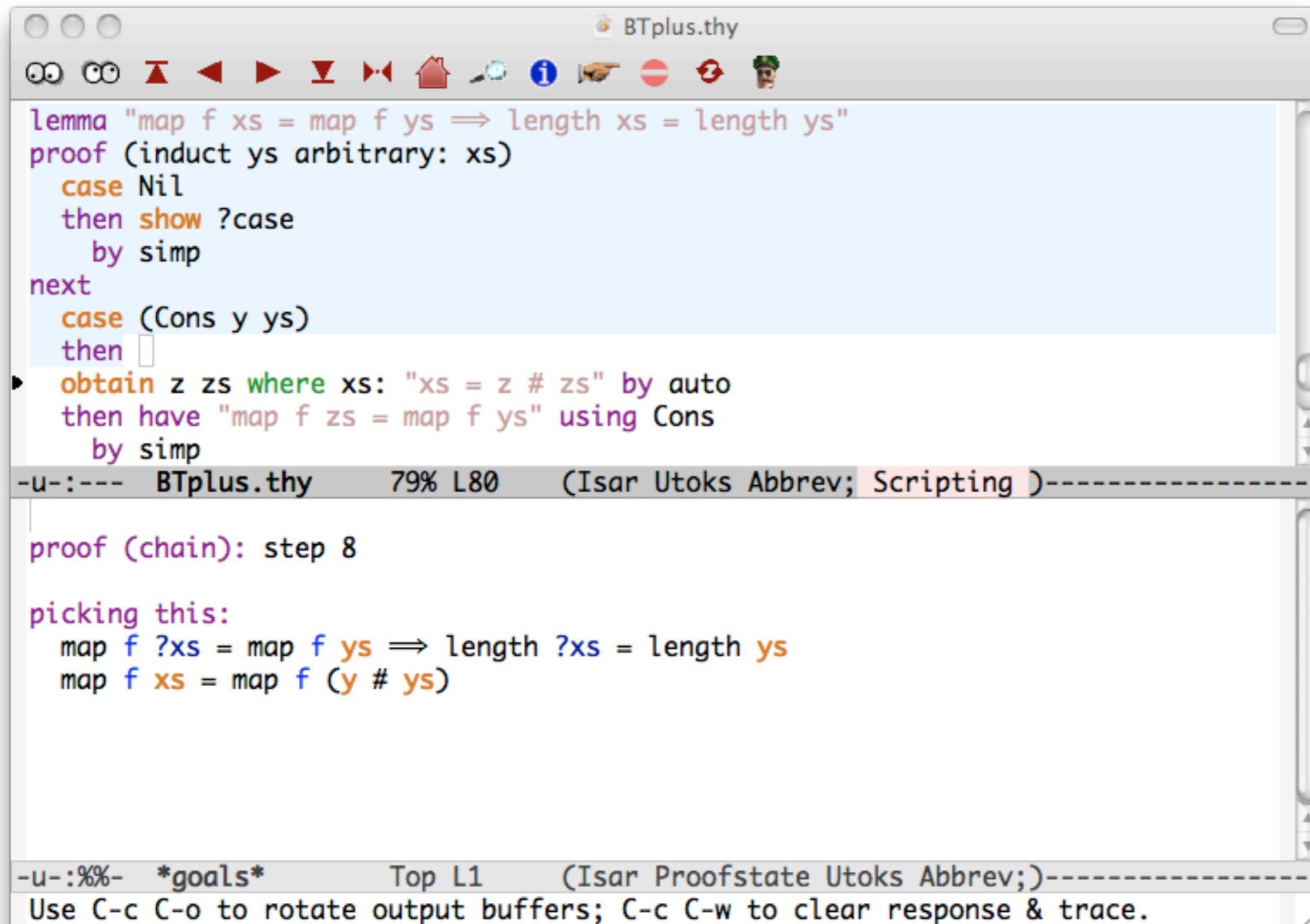


```
BT.thy
[lemma dvd_mult_cancel:
  fixes k::nat
  assumes dv: "k*m dvd k*n" and "0<k"
  shows "m dvd n"
proof -
  obtain j where "k*n = (k*m)*j" using dv
    by (auto simp add: dvd_def)
  hence "k*n = k*(m*j)"
    by (simp add: mult_ac)
  hence "n = m*j" using `0<k`
    by auto
  thus "m dvd n"
    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;)-----
```

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` ← removing k from
    by auto                                     the equality
  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;)-----
```


Introducing “then”

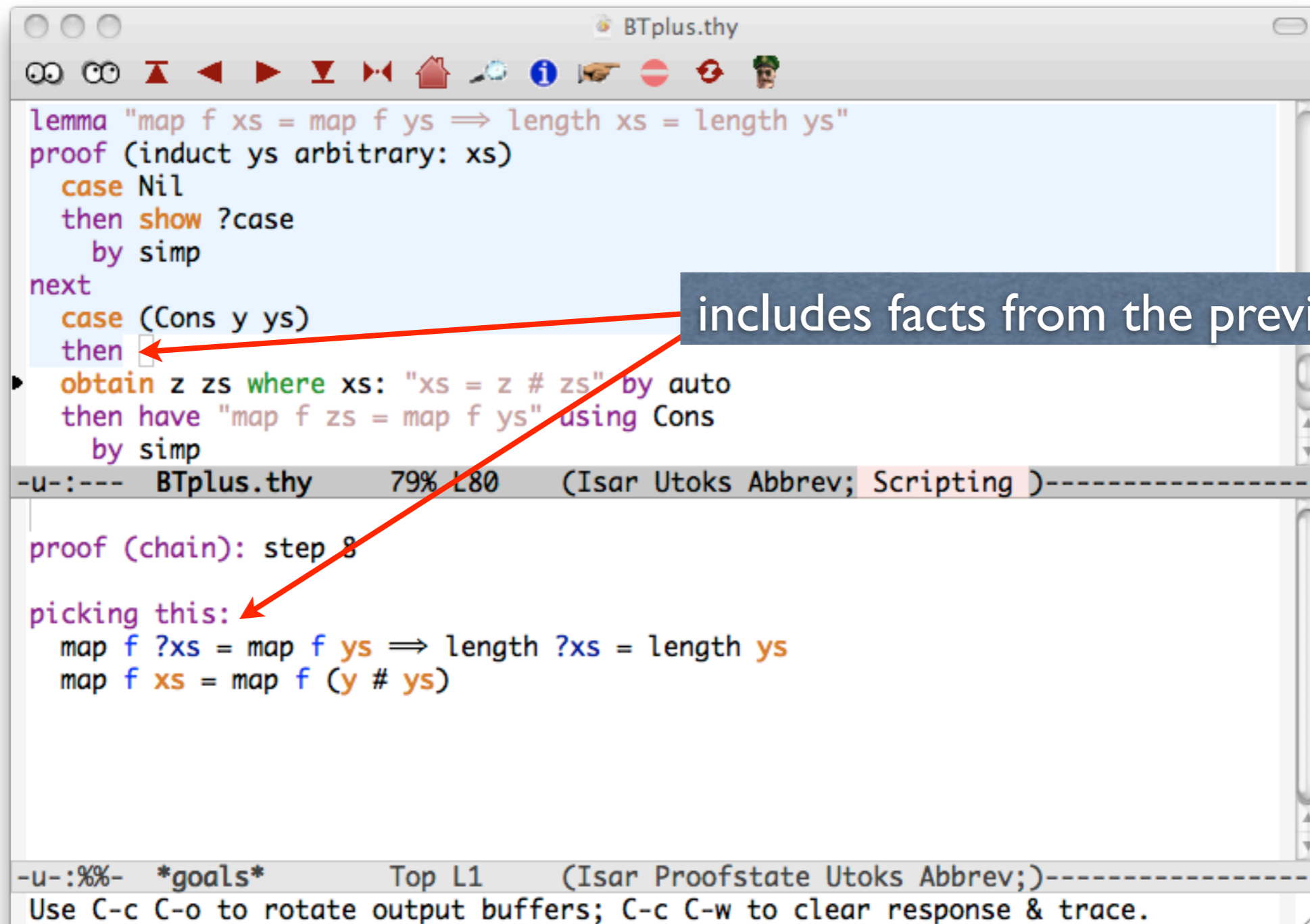


```
BTplus.thy
lemmas "map f xs = map f ys => length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
-u-:--- BTplus.thy 79% L80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



```
BTplus.thy
lemma "map f xs = map f ys  $\implies$  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  $\implies$  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

Introducing “then”

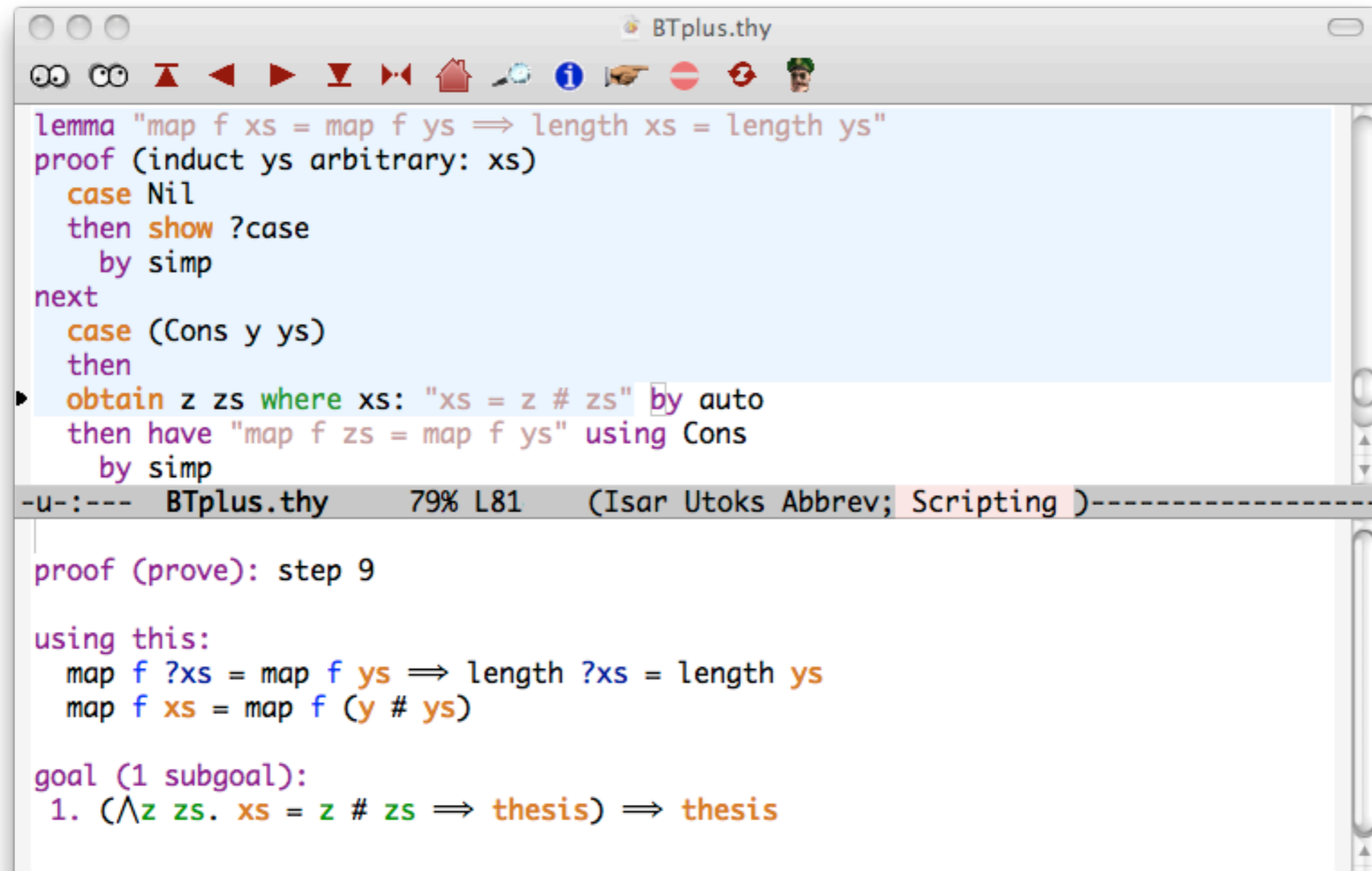
```
BTplus.thy
lemma "map f xs = map f ys  $\implies$  length xs = length ys"
proof (induct ys arbitrary: xs)
  case Nil
  then show ?case
    by simp
next
  case (Cons y ys)
  then
  obtain z zs where xs: "xs = z # zs" by auto
  then have "map f zs = map f ys" using Cons
    by simp
- u-:--- BTplus.thy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

proof (chain): step 8
picking this:
  map f ?xs = map f ys  $\implies$  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

picking this:

Another Example of “obtain”



```
BTplus.thy
lemma "map f xs = map f ys  $\implies$  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% L81 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 9

using this:
  map f ?xs = map f ys  $\implies$  length ?xs = length ys
  map f xs = map f (y # ys)

goal (1 subgoal):
1. ( $\wedge$  z zs. xs = z # zs  $\implies$  thesis)  $\implies$  thesis
```

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

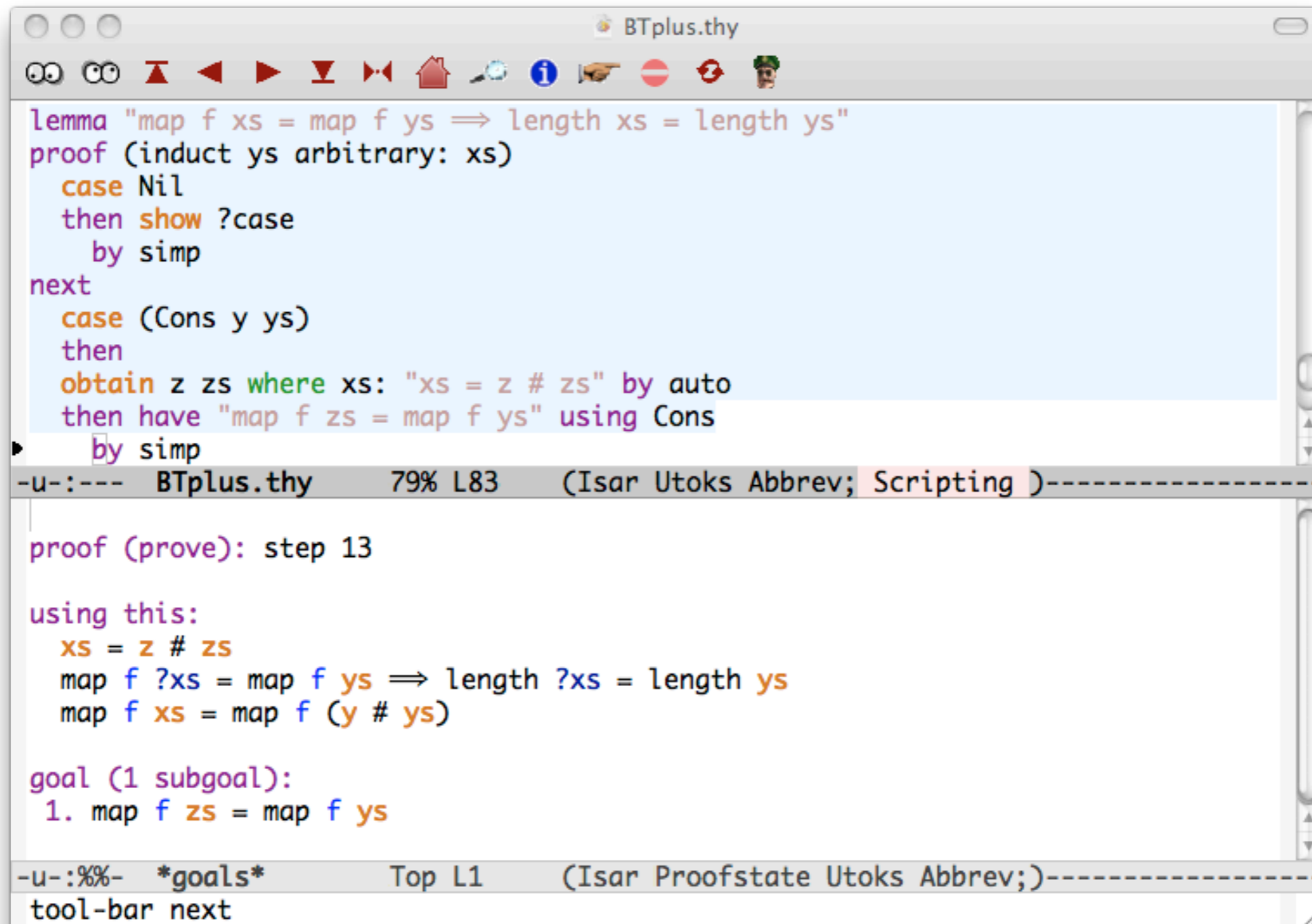
Another Example of “obtain”

```
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% L81 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 9
using this:
  map f ?xs = map f ys => length ?xs = length ys
  map f xs = map f (y # ys)
goal (1 subgoal):
1. (∧ z zs. xs = z # zs => thesis) => thesis
```

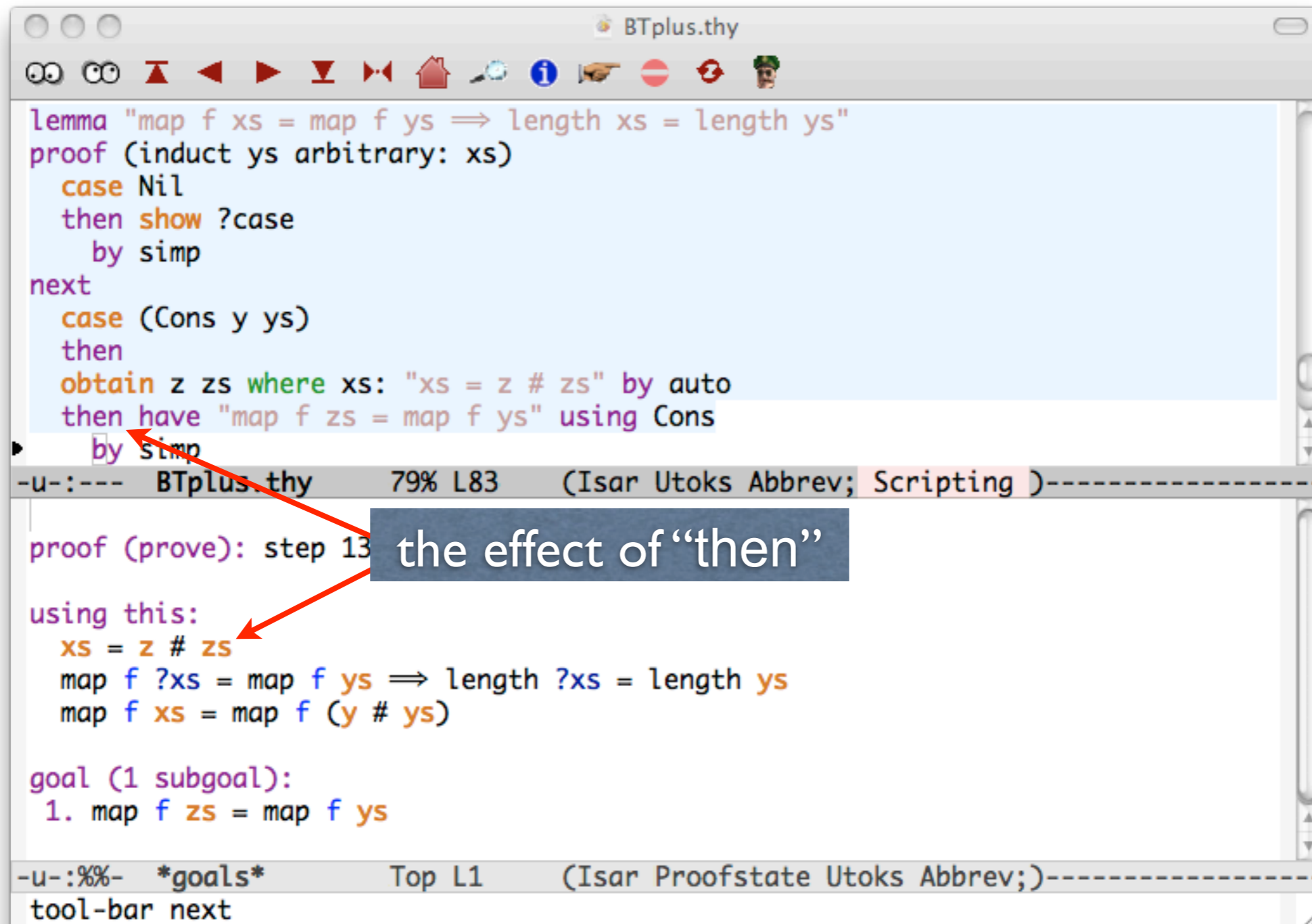
$$(\text{map } f \text{ } xs = y\#ys) \leftrightarrow (\exists z \text{ } zs. xs = z\#zs \ \& \ f \ z = y \ \& \ \text{map } f \ zs = ys)$$

Facts from Two Sources



```
BTplus.thy
lemmma "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



```
BTplus.thy
lemmma "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 the effect of "then"
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
```

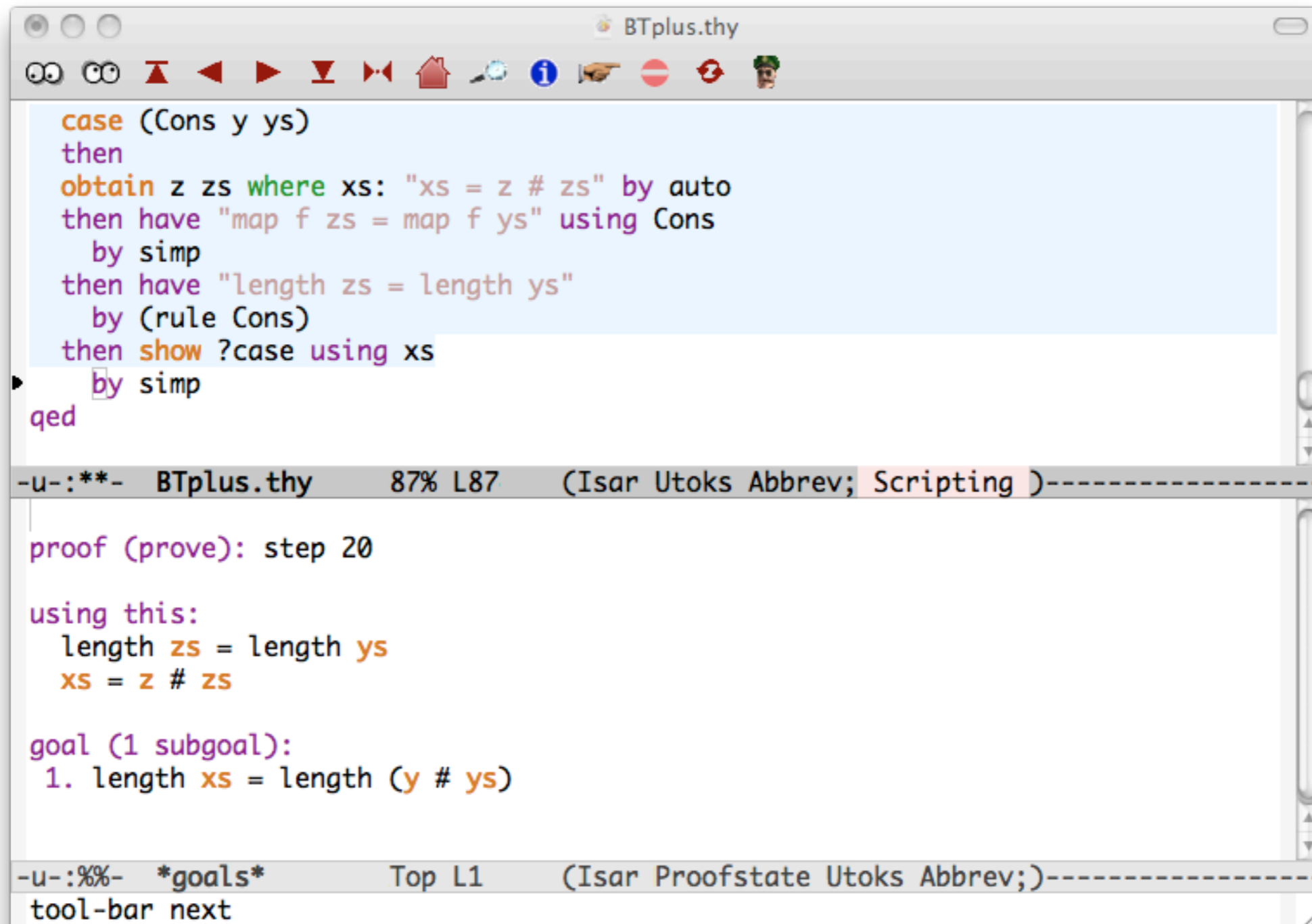
Facts from Two Sources

```
BTplus.thy
lemma "map f xs = map f ys  $\implies$  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  $\implies$  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
```

the effect of "then"

the effect of "using"

Finishing Up



```
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 87% L87 (Isar Utoks Abbrev; Scripting)-----

```
proof (prove): step 20

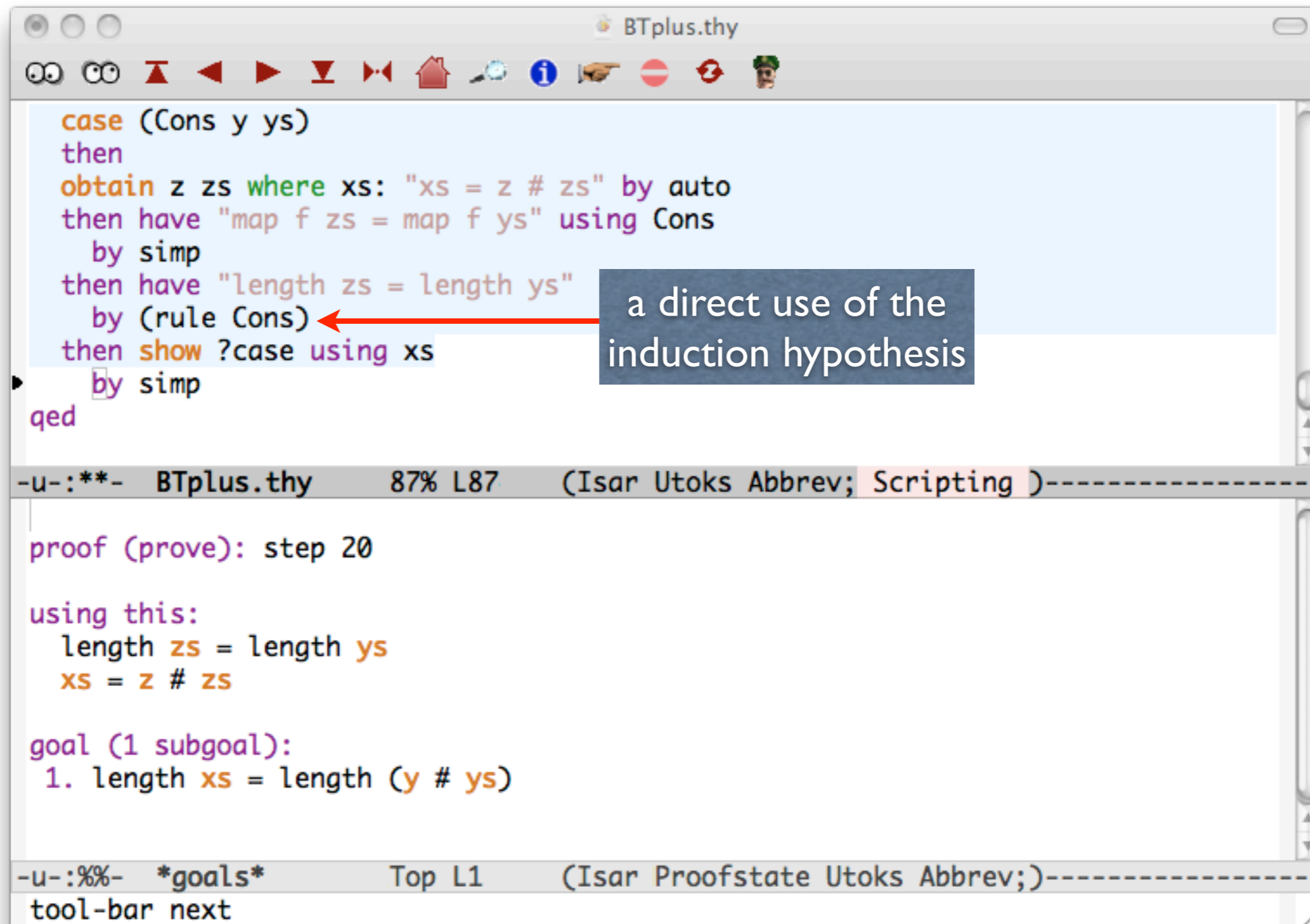
using this:
  length zs = length ys
  xs = z # zs

goal (1 subgoal):
  1. length xs = length (y # ys)
```

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

tool-bar next

Finishing Up



The screenshot shows a theorem prover interface with a proof script in the top pane and its current state in the bottom pane. A red arrow points from a text box to the `by (rule Cons)` line in the script.

```
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
```

a direct use of the induction hypothesis

```
-u-:**- BTplus.thy      87% L87      (Isar Utoks Abbrev; Scripting )-----
```

```
proof (prove): step 20

using this:
  length zs = length ys
  xs = z # zs

goal (1 subgoal):
  1. length xs = length (y # ys)
```

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

Finishing Up

```
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
```

a direct use of the induction hypothesis

"then" / "using" again!

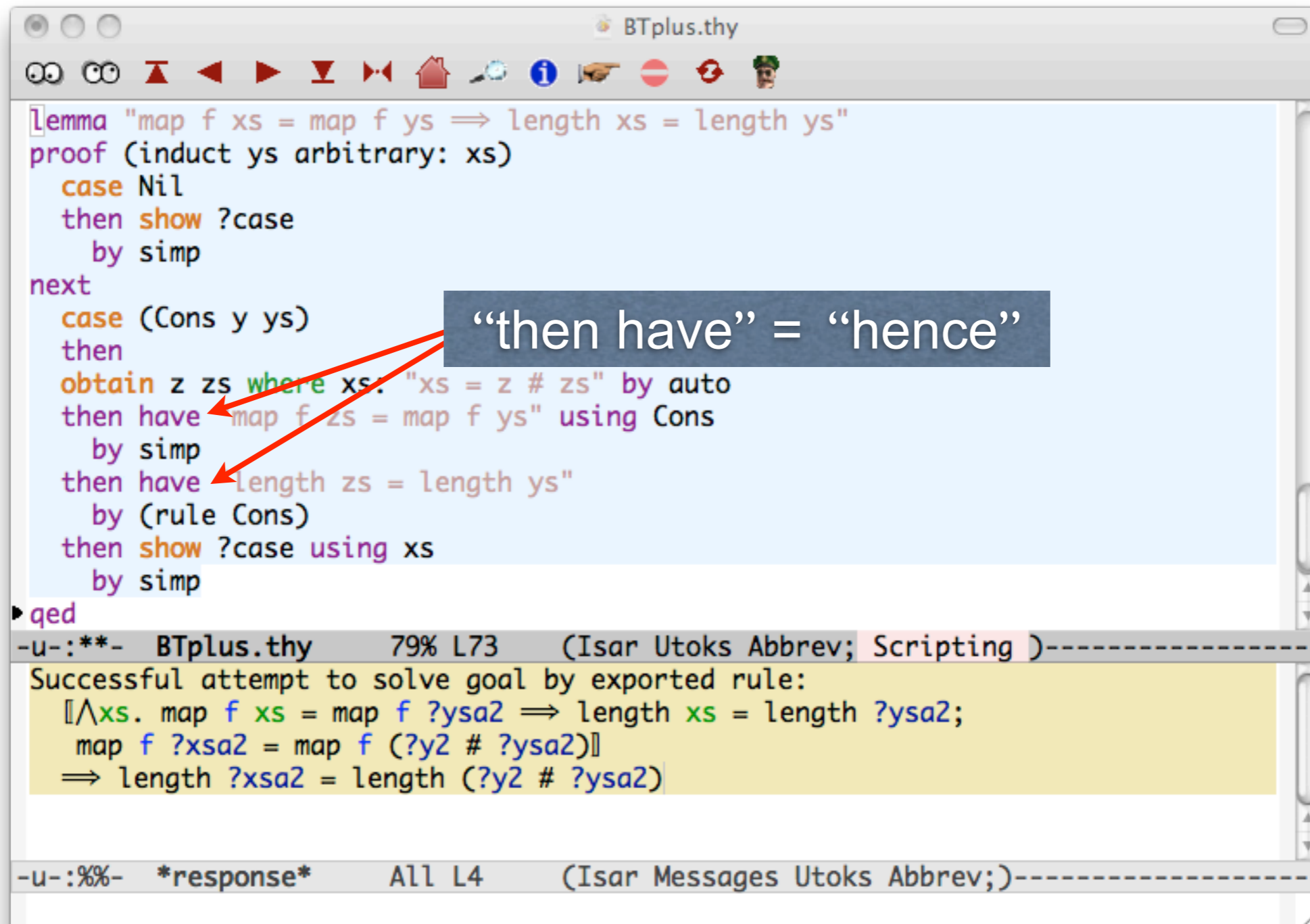
```
-u-:***- BTplus.thy
proof (prove): step 20
using this:
  length zs = length ys
  xs = z # zs
goal (1 subgoal):
1. length xs = length (y # ys)
```

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

The Complete Proof

```
BTplus.thy
[lemma "map f xs = map f ys  $\implies$  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:
[[ $\wedge$ xs. map f xs = map f ?ysa2  $\implies$  length xs = length ?ysa2;
  map f ?xsa2 = map f (?y2 # ?ysa2)]
 $\implies$  length ?xsa2 = length (?y2 # ?ysa2)
-u-:%%- *response* All L4 (Isar Messages Utoks Abbrev;)
```

The Complete Proof



The screenshot shows a window titled "BTplus.thy" with a proof script and its execution output. The proof script is as follows:

```
[lemma "map f xs = map f ys  $\implies$  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
```

Two red arrows point from a dark blue box containing the text "then have" = "hence" to the two occurrences of "then have" in the proof script.

The execution output is as follows:

```
-u-:**- BTplus.thy 79% L73 (Isar Utoks Abbrev; Scripting )-----  
Successful attempt to solve goal by exported rule:  
[[ $\wedge$ xs. map f xs = map f ?ysa2  $\implies$  length xs = length ?ysa2;  
  map f ?xsa2 = map f (?y2 # ?ysa2)]]  
 $\implies$  length ?xsa2 = length (?y2 # ?ysa2)
```

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

The Complete Proof

```
BTplus.thy
[lemma "map f xs = map f ys  $\implies$  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”

“then show” = “thus”

Successful attempt to solve goal by exported rule:
[[\wedge xs. map f xs = map f ?ysa2 \implies length xs = length ?ysa2;
map f ?xsa2 = map f (?y2 # ?ysa2)]]
 \implies length ?xsa2 = length (?y2 # ?ysa2)

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

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

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 $\langle facts \rangle$...

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)