

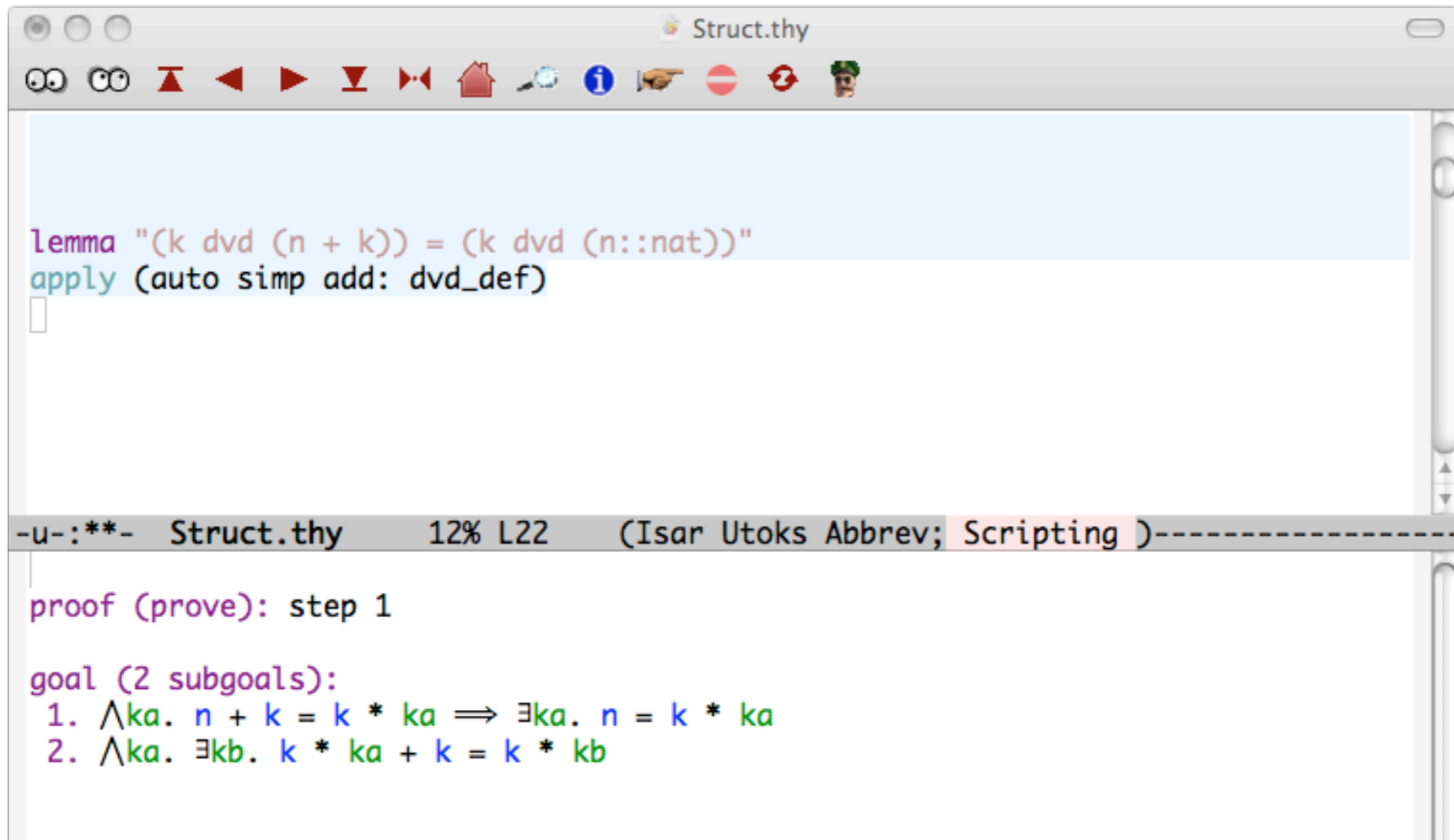
Interactive Formal Verification

/ 0: Structured Proofs

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

A Proof about “Divides”

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



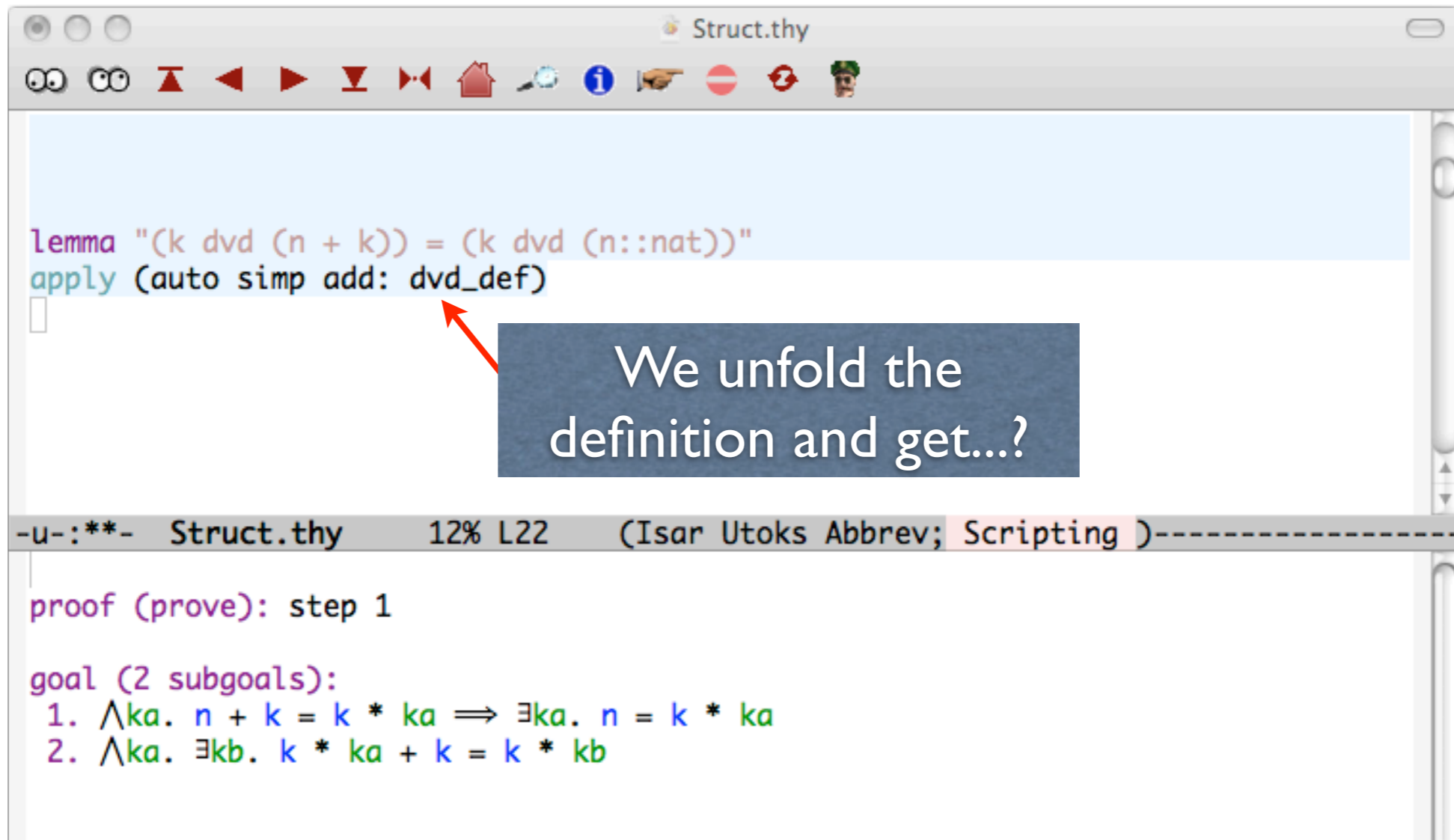
```
Struct.thy
- - - - -
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
apply (auto simp add: dvd_def)
□

- u-:***- Struct.thy 12% L22 (Isar Utoks Abbrev; Scripting )-----
proof (prove): step 1

goal (2 subgoals):
1.  $\forall ka. n + k = k * ka \implies \exists ka. n = k * ka$ 
2.  $\forall ka. \exists kb. k * ka + k = k * kb$ 
```

A Proof about “Divides”

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



The screenshot shows a proof assistant window titled "Struct.thy". The main editor contains the following code:

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"  
apply (auto simp add: dvd_def)  
□
```

A red arrow points from a text box to the `dvd_def` in the `apply` line. The text box contains the text: "We unfold the definition and get...?".

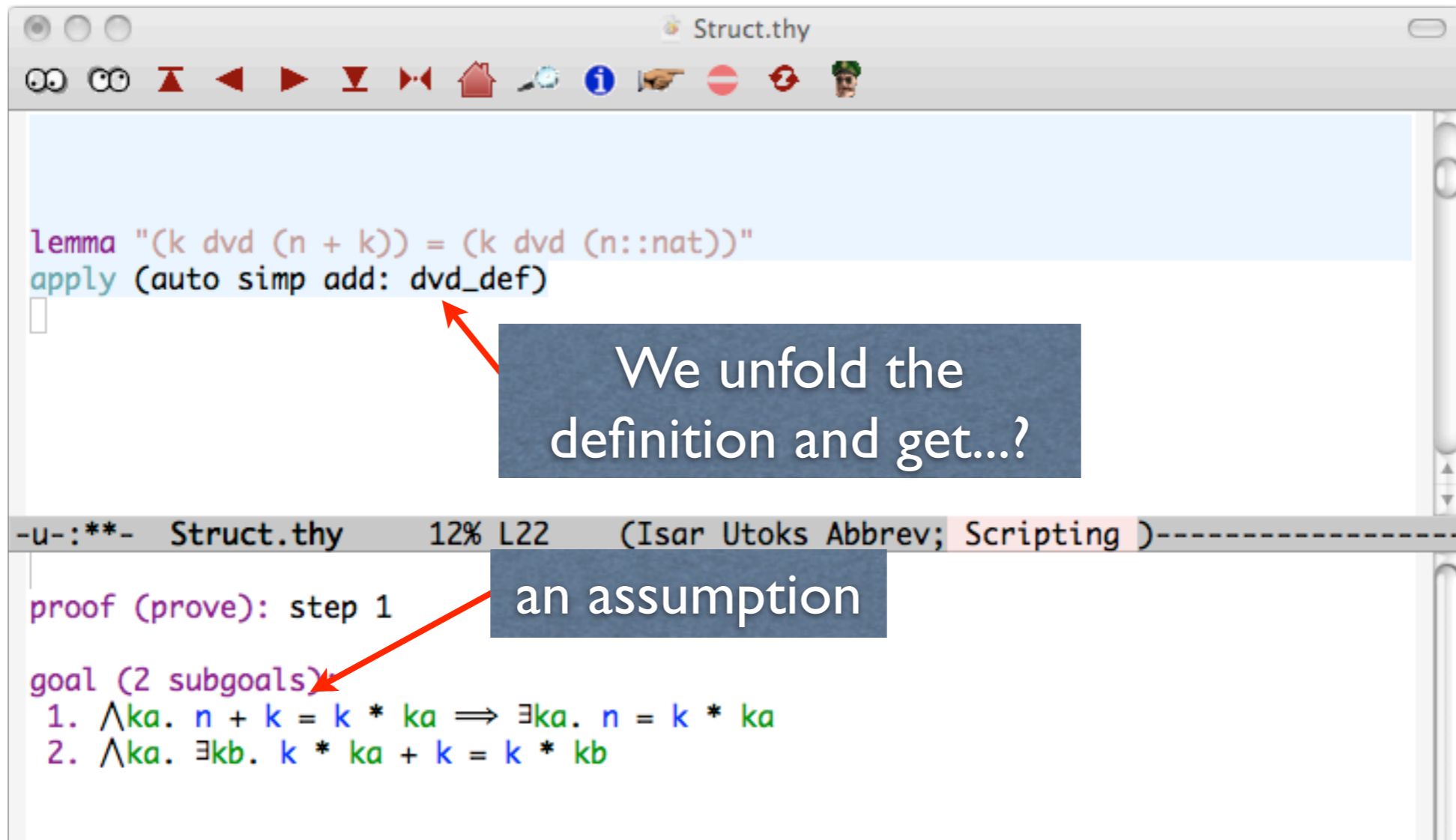
The status bar at the bottom shows: `-u-:***- Struct.thy 12% L22 (Isar Utoks Abbrev; Scripting)`

The proof steps are shown below the status bar:

```
proof (prove): step 1  
goal (2 subgoals):  
1.  $\forall ka. n + k = k * ka \implies \exists ka. n = k * ka$   
2.  $\forall ka. \exists kb. k * ka + k = k * kb$ 
```

A Proof about “Divides”

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



The screenshot shows a window titled "Struct.thy" with a toolbar and a text editor. The editor contains the following code:

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"  
apply (auto simp add: dvd_def)  
□
```

Below the editor is a status bar with the text: "-u-:***- Struct.thy 12% L22 (Isar Utoks Abbrev; Scripting)-----".

Below the status bar, the proof steps are shown:

```
proof (prove): step 1  
goal (2 subgoals)  
1.  $\wedge ka. n + k = k * ka \implies \exists ka. n = k * ka$   
2.  $\wedge ka. \exists kb. k * ka + k = k * kb$ 
```

Two callout boxes with arrows pointing to the code:

- A box containing "We unfold the definition and get...?" points to the `apply (auto simp add: dvd_def)` line.
- A box containing "an assumption" points to the `goal (2 subgoals)` line.

A Proof about “Divides”

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

```
Struct.thy  
lemma "(k dvd (n + k)) = (k dvd (n::nat))"  
apply (auto simp add: dvd_def)  
□  
-u-:***- Struct.thy 12% L22 (Isar Utoks Abbrev; Scripting )-----  
proof (prove): step 1  
goal (2 subgoals)  
1.  $\wedge ka. n + k = k * ka \implies \exists ka. n = k * ka$   
2.  $\wedge ka. \exists kb. k * ka + k = k * kb$ 
```

We unfold the definition and get...?

an assumption

locally bound variables

A Proof about “Divides”

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

```
Struct.thy
[Icons: zoom, undo, redo, search, etc.]

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
apply (auto simp add: dvd_def)
[ ]

-u-:***- Struct.thy 12% L22 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 1
goal (2 subgoals)
1.  $\wedge ka. n + k = k * ka \implies \exists ka. n = k * ka$ 
2.  $\wedge ka. \exists kb. k * ka + k = k * kb$ 

[Annotations:
- 'We unfold the definition and get...?' points to 'dvd_def' in the apply command.
- 'an assumption' points to 'step 1' in the proof command.
- 'locally bound variables' points to 'ka' and 'kb' in the subgoals.
- 'A messy proof with two subgoals...' points to the goal block.]
```

Complex Subgoals

Complex Subgoals

- Isabelle provides many tactics that refer to bound variables and assumptions.
- Assumptions are often found by matching.
- Bound variables can be referred to by name, but these names are fragile.

Complex Subgoals

- Isabelle provides many tactics that refer to bound variables and assumptions.
 - Assumptions are often found by matching.
 - Bound variables can be referred to by name, but these names are fragile.
- *Structured proofs* provide a robust means of referring to these elements by name.

Complex Subgoals

- Isabelle provides many tactics that refer to bound variables and assumptions.
 - Assumptions are often found by matching.
 - Bound variables can be referred to by name, but these names are fragile.
- *Structured proofs* provide a robust means of referring to these elements by name.
- Structured proofs are typically verbose but much more readable than linear apply-proofs.

A Structured Proof

```
Struct.thy
lemmas dvd_def: dvd_def
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  hence "n = k * (m - 1)"
    by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
  thus "∃m'. n = k * m'"
  by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
  by (metis mult_Suc_right nat_add_commute)
qed
-u-:--- Struct.thy      2% L11  (Isar Utoks Abbrev; Scripting )-----
|
| proof (prove): step 6
|
| using this:
|   n = k * (m - 1)
|
| goal (1 subgoal):
| 1. ∃m'. n = k * m'
-u-:%%- *goals*      Top L1  (Isar Proofstate Utoks Abbrev;)-----
|
| tool-bar goto
```

But how do you write them?

The Elements of Isar

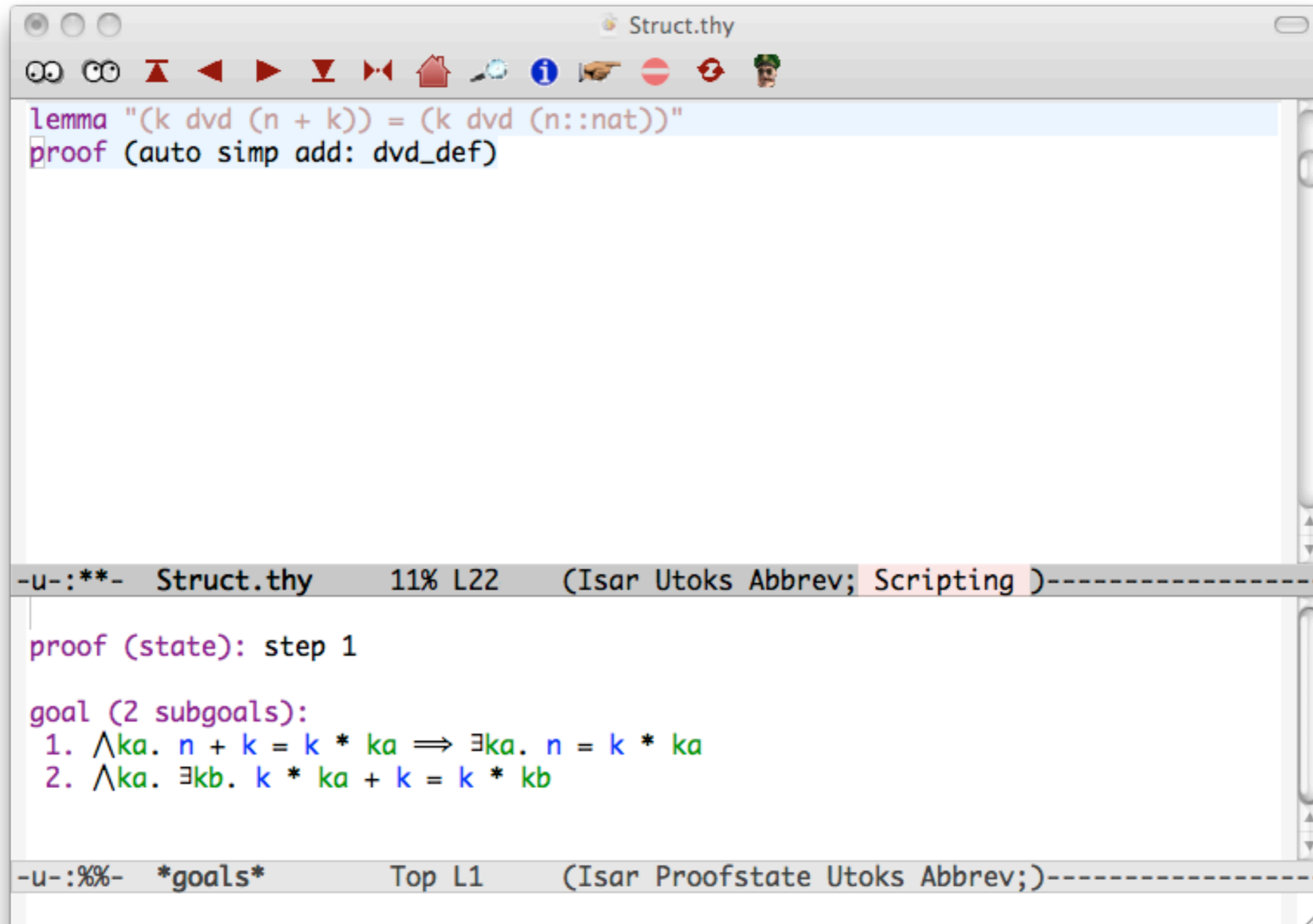
The Elements of Isar

- A *proof context* holds local variables and assumptions of a subgoal.
- In a context, the variables are free and the assumptions are simply theorems.
- Closing a context yields a theorem having the structure of a subgoal.

The Elements of Isar

- A *proof context* holds local variables and assumptions of a subgoal.
- In a context, the variables are free and the assumptions are simply theorems.
- Closing a context yields a theorem having the structure of a subgoal.
- The Isar language lets us state and prove intermediate results, express inductions, etc.

Getting Started



The screenshot shows a window titled "Struct.thy" with a toolbar containing various navigation icons. The main text area contains the following code:

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"  
proof (auto simp add: dvd_def)
```

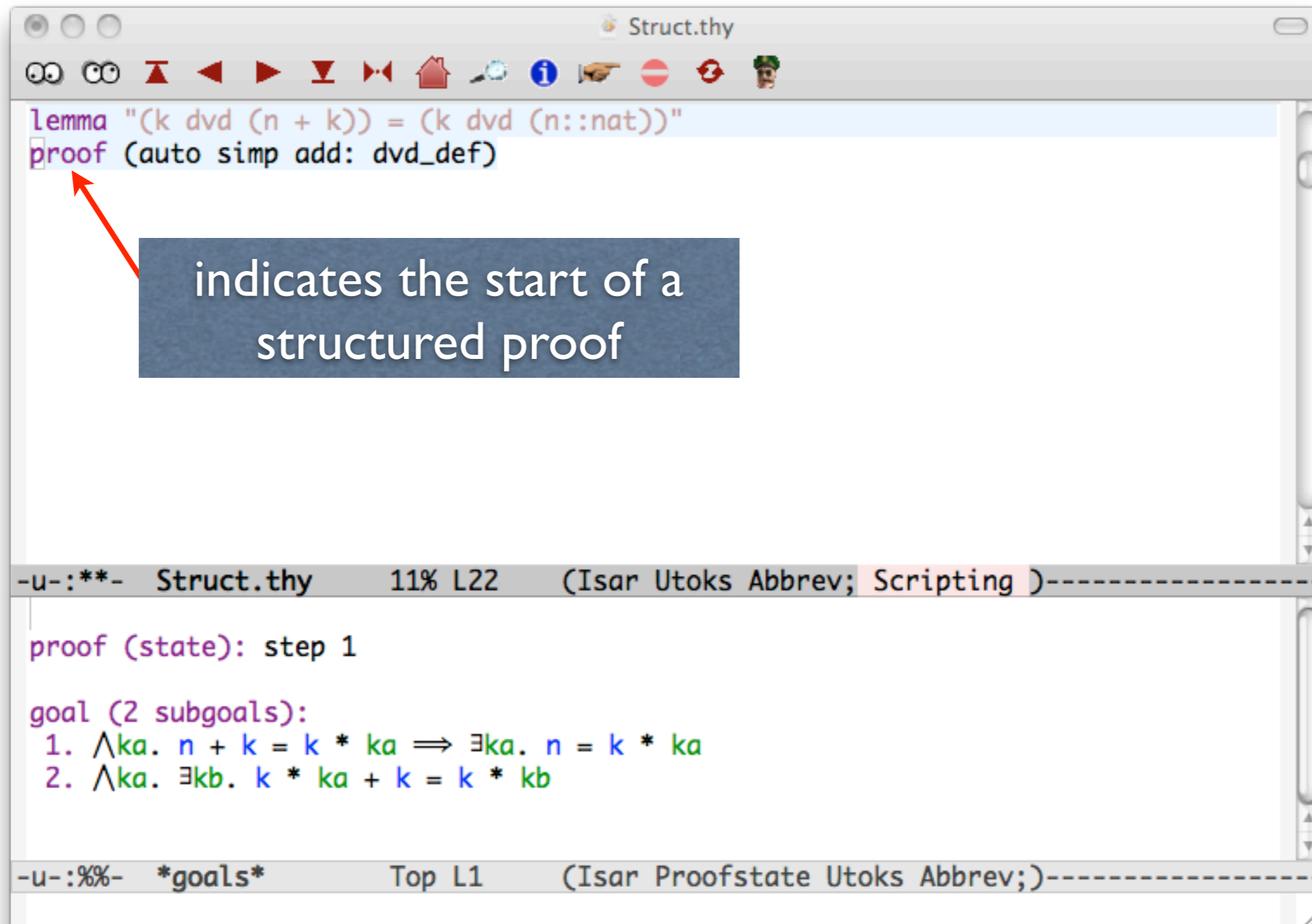
Below the code is a status bar with the text: "-u-:**- Struct.thy 11% L22 (Isar Utoks Abbrev; Scripting)-----".

The next section shows the proof state:

```
proof (state): step 1  
goal (2 subgoals):  
1.  $\forall ka. n + k = k * ka \implies \exists ka. n = k * ka$   
2.  $\forall ka. \exists kb. k * ka + k = k * kb$ 
```

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

Getting Started



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

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"  
proof (auto simp add: dvd_def)
```

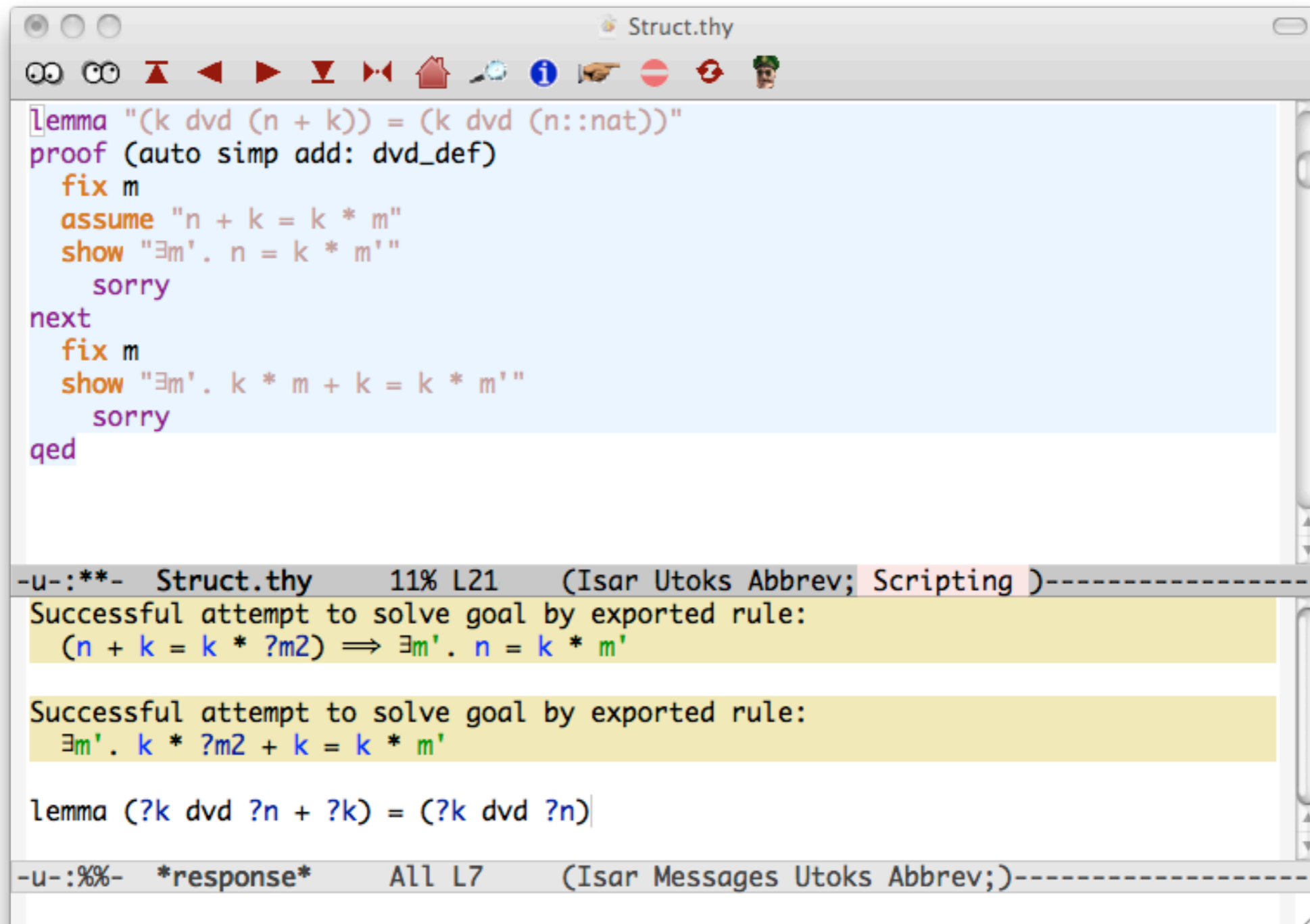
A red arrow points from a dark blue callout box to the `proof` keyword. The callout box contains the text: "indicates the start of a structured proof".

Below the main text area, there are two status bars. The first status bar shows: `-u-:**- Struct.thy 11% L22 (Isar Utoks Abbrev; Scripting)`. The second status bar shows: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)`.

The main text area also displays the structured proof state:

```
proof (state): step 1  
goal (2 subgoals):  
1.  $\wedge ka. n + k = k * ka \implies \exists ka. n = k * ka$   
2.  $\wedge ka. \exists kb. k * ka + k = k * kb$ 
```


The Proof Skeleton



```
Struct.thy
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed]

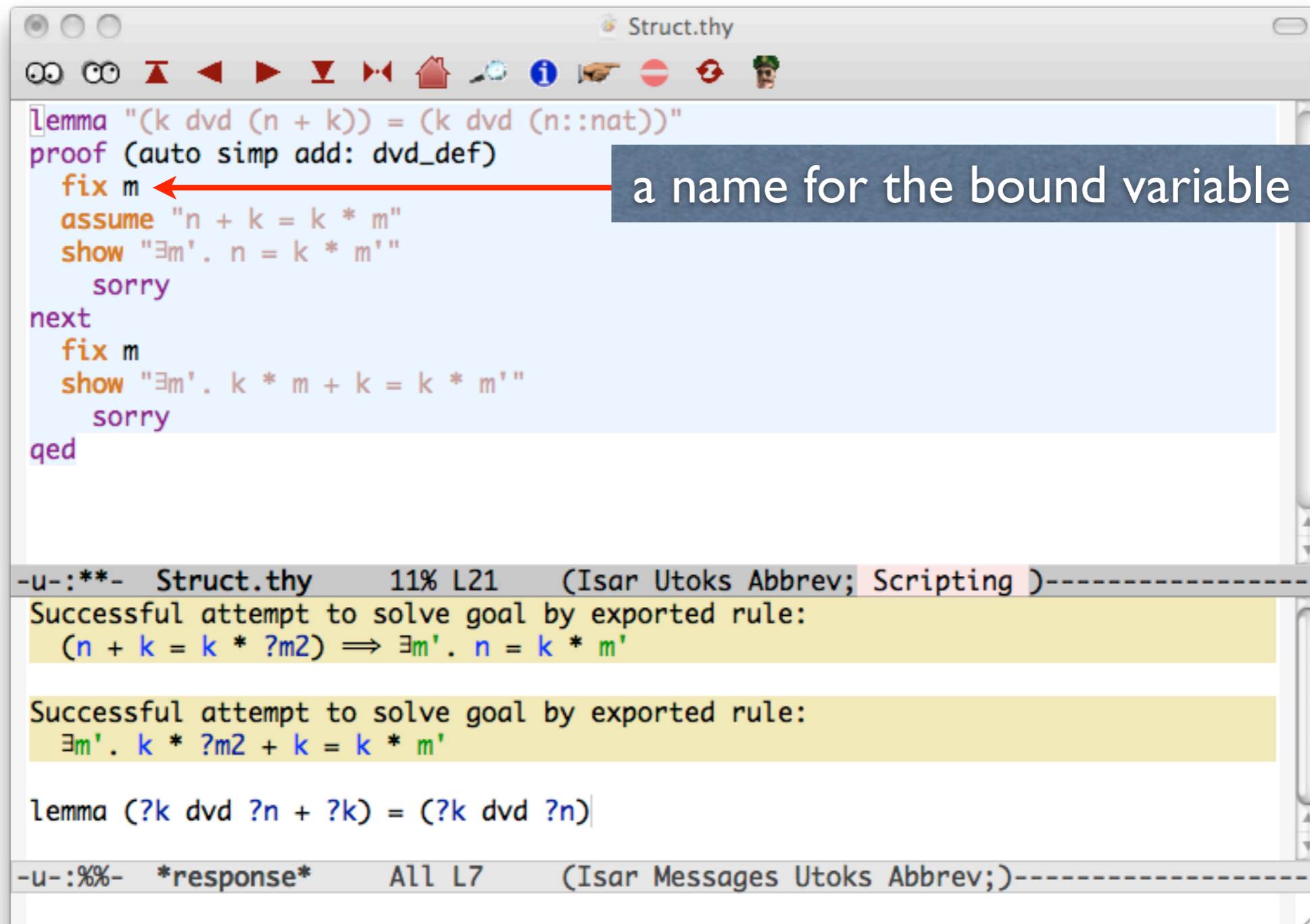
-u-:**- Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)------
```

The Proof Skeleton



The screenshot shows a proof editor window titled "Struct.thy". The main text area contains the following code:

```
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m ←
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed
```

A red arrow points from the text "a name for the bound variable" to the `fix m` line in the first block of the proof.

The bottom panel shows the execution output:

```
-u-:**- Struct.thy      11% L21  (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
-u-:%%-  *response*    All L7   (Isar Messages Utoks Abbrev;)------
```

The Proof Skeleton

assumption

a name for the bound variable

```
Struct.thy
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed]

-u-:**- Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)------
```

The Proof Skeleton

assumption

conclusion

```
Struct.thy
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed]

-u-:**- Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)------
```

a name for the bound variable

The Proof Skeleton

assumption

conclusion

dummy
proofs

```
Struct.thy
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m ← a name for the bound variable
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed]

-u-:**- Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)------
```

The Proof Skeleton

assumption

conclusion

dummy
proofs

```
Struct.thy
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed]

-u-:**- Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)------
```

a name for the bound variable

separates proofs of goals

The Proof Skeleton

```
Struct.thy
[lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume "n + k = k * m"
  show "∃m'. n = k * m'"
  sorry
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed
```

-u-:**- Struct.thy 11% L21 (Isar Utoks Abbrev; Scripting)-----

Successful attempt to solve goal by exported rule:
 $(n + k = k * ?m2) \Rightarrow \exists m'. n = k * m'$

Successful attempt to solve goal by exported rule:
 $\exists m'. k * ?m2 + k = k * m'$

lemma (?k dvd ?n + ?k) = (?k dvd ?n)

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)------

assumption

conclusion

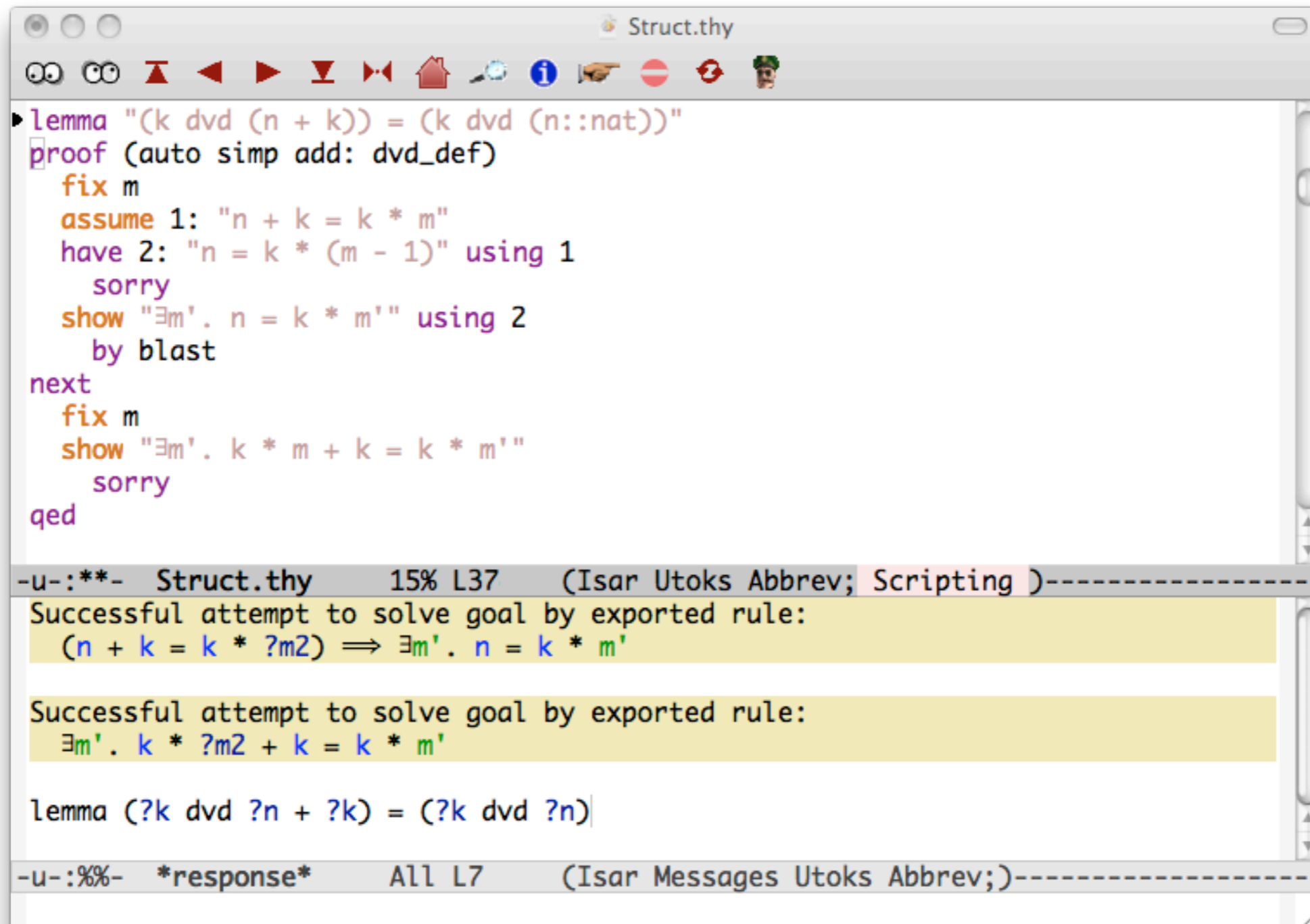
dummy
proofs

a name for the bound variable

separates proofs of goals

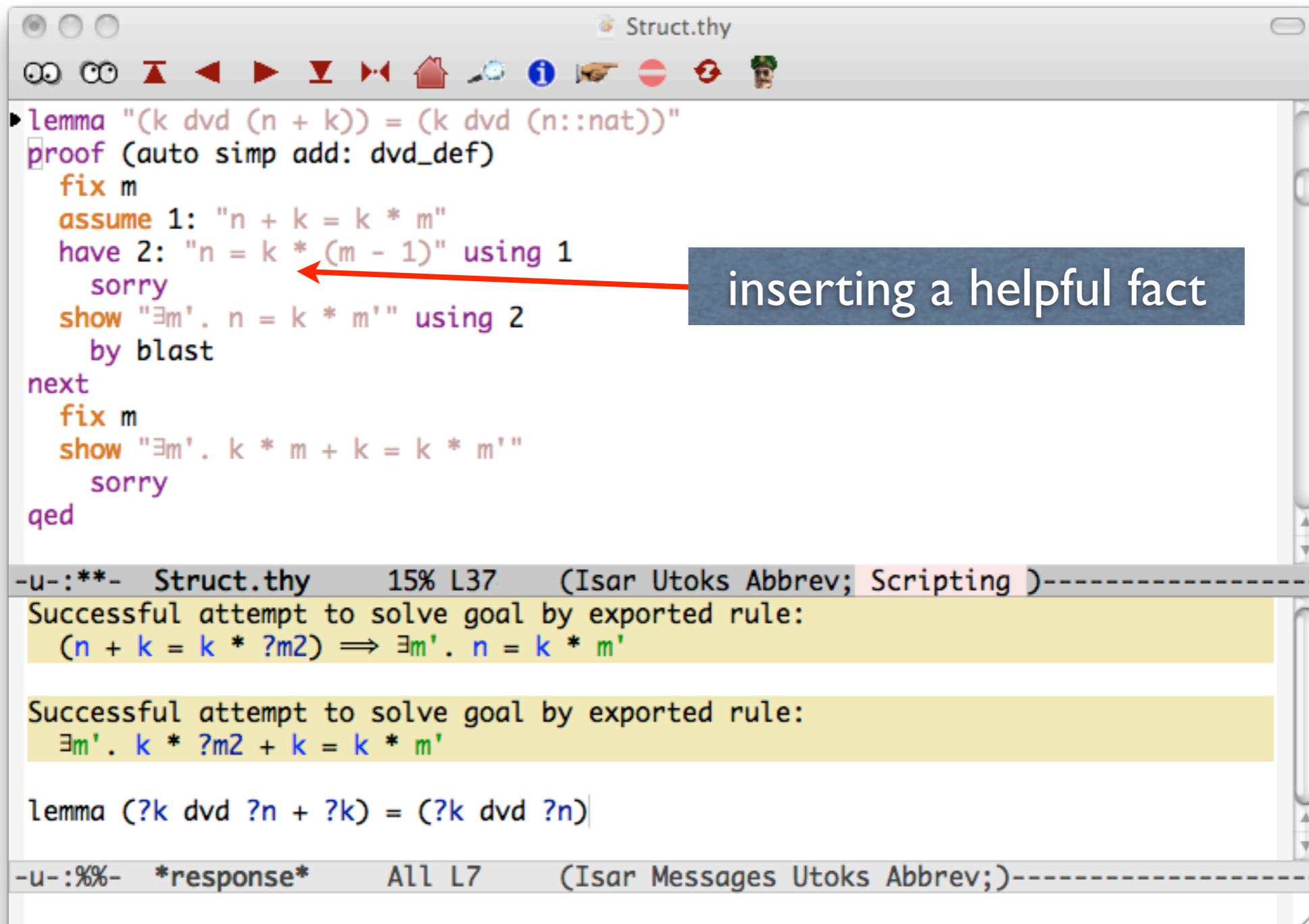
terminates the proof

Fleshing Out that Skeleton



```
Struct.thy
--u-:***- Struct.thy 15% L37 (Isar Utoks Abbrev; Scripting )-----
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    sorry
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
    sorry
qed
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'
Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'
lemma (?k dvd ?n + ?k) = (?k dvd ?n)
--u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)
```


Fleshing Out that Skeleton



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

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    sorry
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
    sorry
qed
```

An annotation "inserting a helpful fact" with a red arrow points to the line `have 2: "n = k * (m - 1)" using 1`.

The execution output shows the following messages:

```
-u-:**- Struct.thy 15% L37 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)
```

Fleshing Out that Skeleton

labels for facts

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    sorry
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
    sorry
qed
```

inserting a helpful fact

```
-u-:**- Struct.thy 15% L37 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)|

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)
```

Fleshing Out that Skeleton

The image shows a screenshot of a theorem prover interface with several annotations. Three blue boxes with white text are connected to the code by red arrows:

- labels for facts**: Points to the `assume 1` and `have 2` lines.
- more labels**: Points to the `show` line in the first block.
- inserting a helpful fact**: Points to the `using 2` in the `show` line.

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
  sorry
  show "∃m'. n = k * m'" using 2
  by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed
```

The bottom part of the screenshot shows the output of the prover:

```
-u-:***- Struct.thy 15% L37 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  (n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
  ∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
```

-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)

Fleshing Out that Skeleton

The image shows a screenshot of a theorem prover interface with several annotations. Red arrows point from text boxes to specific parts of the code. The code is as follows:

```
lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
  sorry
  show "∃m'. n = k * m'" using 2
  by blast
next
fix m
show "∃m'. k * m + k = k * m'"
sorry
qed
```

Annotations:

- labels for facts**: points to the `assume 1` line.
- more labels**: points to the `have 2` line.
- inserting a helpful fact**: points to the `using 2` in the `show` line.
- a real proof!**: points to the `by blast` line.

The bottom part of the screenshot shows the output of the prover:

```
-u-:***- Struct.thy 15% L37 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
(n + k = k * ?m2) ⇒ ∃m'. n = k * m'

Successful attempt to solve goal by exported rule:
∃m'. k * ?m2 + k = k * m'

lemma (?k dvd ?n + ?k) = (?k dvd ?n)
```

The bottom status bar shows: `-u-:%%- *response* All L7 (Isar Messages Utoks Abbrev;)`

Completing the Proof

```
Struct.thy
lemmas (k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed
-u-:***- Struct.thy 20% L65 (Isar Utoks Abbrev; Scripting )-----
Sledgehammer: external prover "spass" for subgoal 1:
∃m'. k * m + k = k * m'
Try this command: apply (metis mult_Suc_right nat_add_commute)
For minimizing the number of lemmas try this command:
atp_minimize [atp=spass] mult_Suc_right nat_add_commute

Sledgehammer: external prover "e" for subgoal 1:
∃m'. k * m + k = k * m'
-u-:%%- *response* Top L1 (Isar Messages Utoks Abbrev;)-----
menu-bar Isabelle Commands Sledgehammer
```

Completing the Proof

```
Struct.thy
lemmas dvd_def: "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
  fix m
  assume 1: "n + k = k * m"
  have 2: "n = k * (m - 1)" using 1
    by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
  show "∃m'. n = k * m'" using 2
    by blast
next
  fix m
  show "∃m'. k * m + k = k * m'"
  sorry
qed
```

← found using sledgehammer

```
-u-:***- Struct.thy      20% L65  (Isar Utoks Abbrev; Scripting )-----
Sledgehammer: external prover "spass" for subgoal 1:
∃m'. k * m + k = k * m'
Try this command: apply (metis mult_Suc_right nat_add_commute)
For minimizing the number of lemmas try this command:
atp_minimize [atp=spass] mult_Suc_right nat_add_commute

Sledgehammer: external prover "e" for subgoal 1:
∃m'. k * m + k = k * m'
-u-:%%-  *response*      Top L1   (Isar Messages Utoks Abbrev;)-----
menu-bar Isabelle Commands Sledgehammer
```

Completing the Proof

```
Struct.thy
Sledgehammer: external prover "spass" for subgoal 1:
∃m'. k * m + k = k * m'
Try this command: apply (metis mult_Suc_right nat_add_commute)
For minimizing the number of lemmas try this command:
atp_minimize [atp=spass] mult_Suc_right nat_add_commute

Sledgehammer: external prover "e" for subgoal 1:
∃m'. k * m + k = k * m'
-----
*response* Top L1 (Isar Messages Utoks Abbrev;)-----
menu-bar Isabelle Commands Sledgehammer
```

lemma "(k dvd (n + k)) = (k dvd (n::nat))"
proof (auto simp add: dvd_def)
 fix m
 assume 1: "n + k = k * m"
 have 2: "n = k * (m - 1)" using 1
 by (metis diff_add_inverse diff_mult_distrib2 nat_add_commute nat_mult_1_right)
 show "∃m'. n = k * m'" using 2
 by blast
next
 fix m
 show "∃m'. k * m + k = k * m'"
 sorry
qed

found using sledgehammer

sledgehammer does it again!

Streamlining the Proof

```
assume 1: "n + k = k * m"  
have 2: "n = k * (m - 1)" using 1  
  by (metis diff_add_inverse diff  
show "∃m'. n = k * m'" using 2
```


Streamlining the Proof

<code>assume 1: "n + k = k * m"</code>	→	<code>assume "n + k = k * m"</code>
<code>have 2: "n = k * (m - 1)" using 1</code>	→	<code>hence "n = k * (m - 1)"</code>
<code> by (metis diff_add_inverse diff)</code>		<code> by (metis diff_add_inverse diff)</code>
<code>show "∃m'. n = k * m'" using 2</code>	→	<code>thus "∃m'. n = k * m'"</code>

Streamlining the Proof

<code>assume 1: "n + k = k * m"</code>	→	<code>assume "n + k = k * m"</code>
<code>have 2: "n = k * (m - 1)" using 1</code>	→	<code>hence "n = k * (m - 1)"</code>
<code>by (metis diff_add_inverse diff)</code>		<code>by (metis diff_add_inverse diff)</code>
<code>show "∃m'. n = k * m'" using 2</code>	→	<code>thus "∃m'. n = k * m'"</code>

- hence means have — using the previous fact

Streamlining the Proof

<code>assume 1: "n + k = k * m"</code>	\longrightarrow	<code>assume "n + k = k * m"</code>
<code>have 2: "n = k * (m - 1)" using 1</code>	\longrightarrow	<code>hence "n = k * (m - 1)"</code>
<code> by (metis diff_add_inverse diff)</code>		<code> by (metis diff_add_inverse diff)</code>
<code>show "∃m'. n = k * m'" using 2</code>	\longrightarrow	<code>thus "∃m'. n = k * m'"</code>

- `hence` means `have` — using the previous fact
- `thus` means `show` — using the previous fact

Streamlining the Proof

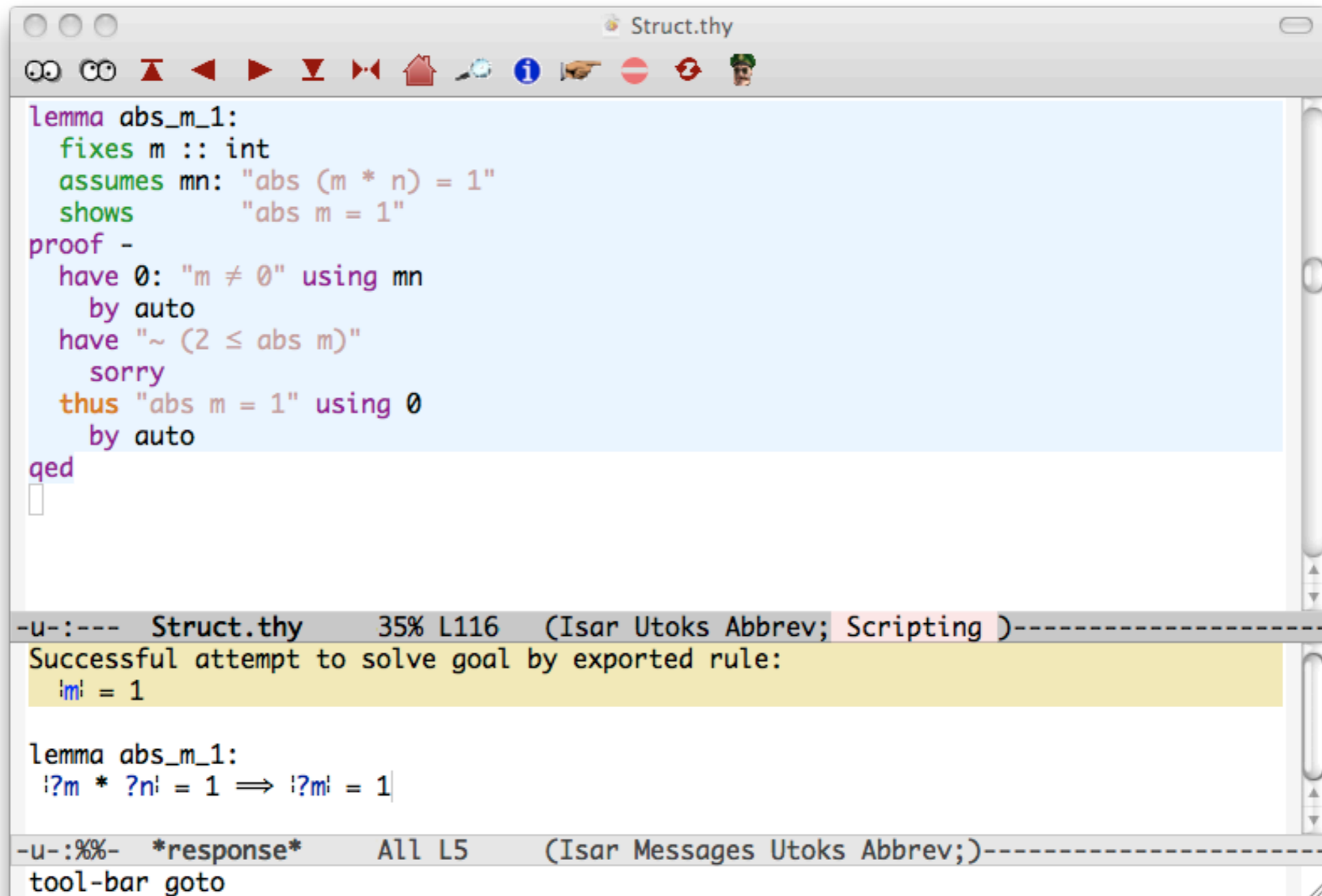
```
assume 1: "n + k = k * m"
have 2: "n = k * (m - 1)" using 1
  by (metis diff_add_inverse diff)
show "∃m'. n = k * m'" using 2
```

→

```
assume "n + k = k * m"
hence "n = k * (m - 1)"
  by (metis diff_add_inverse diff)
thus "∃m'. n = k * m'"
```

- hence means have — using the previous fact
- thus means show — using the previous fact
- There are numerous other tricks of this sort!

Another Proof Skeleton



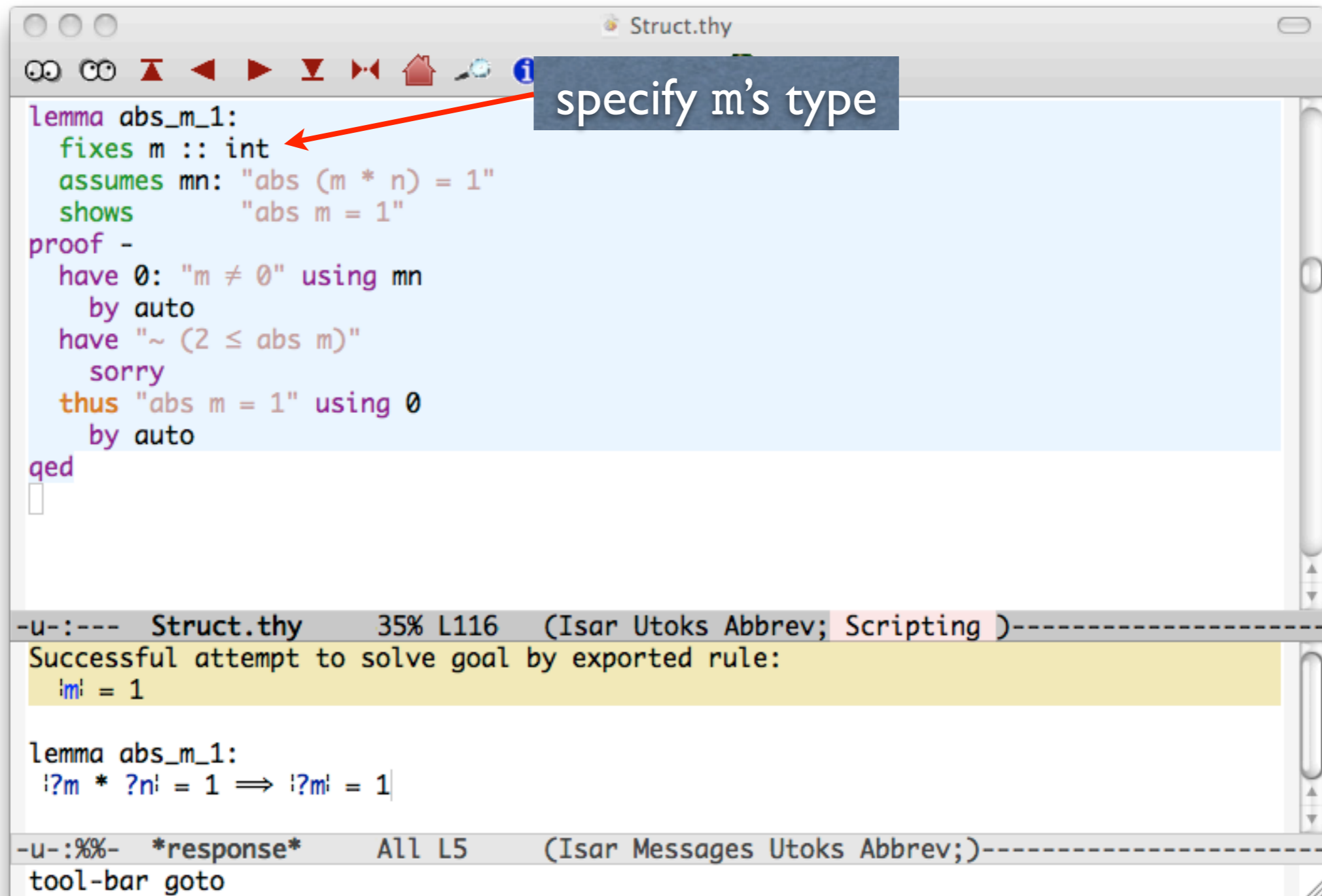
```
Struct.thy
lemmas abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
□

-u-:--- Struct.thy 35% L116 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  |m| = 1

lemma abs_m_1:
  !?m * ?n = 1 ⇒ !?m = 1

-u-:%%- *response* All L5 (Isar Messages Utoks Abbrev;)-----
tool-bar goto
```

Another Proof Skeleton



```
Struct.thy
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
□
```

specify m's type

```
-u-:--- Struct.thy 35% L116 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  |m| = 1

lemma abs_m_1:
  !?m * ?n = 1 ⇒ !?m = 1|

-u-:%%- *response* All L5 (Isar Messages Utoks Abbrev;)-----
tool-bar goto
```

Another Proof Skeleton

```
Struct.thy
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
```

specify m's type

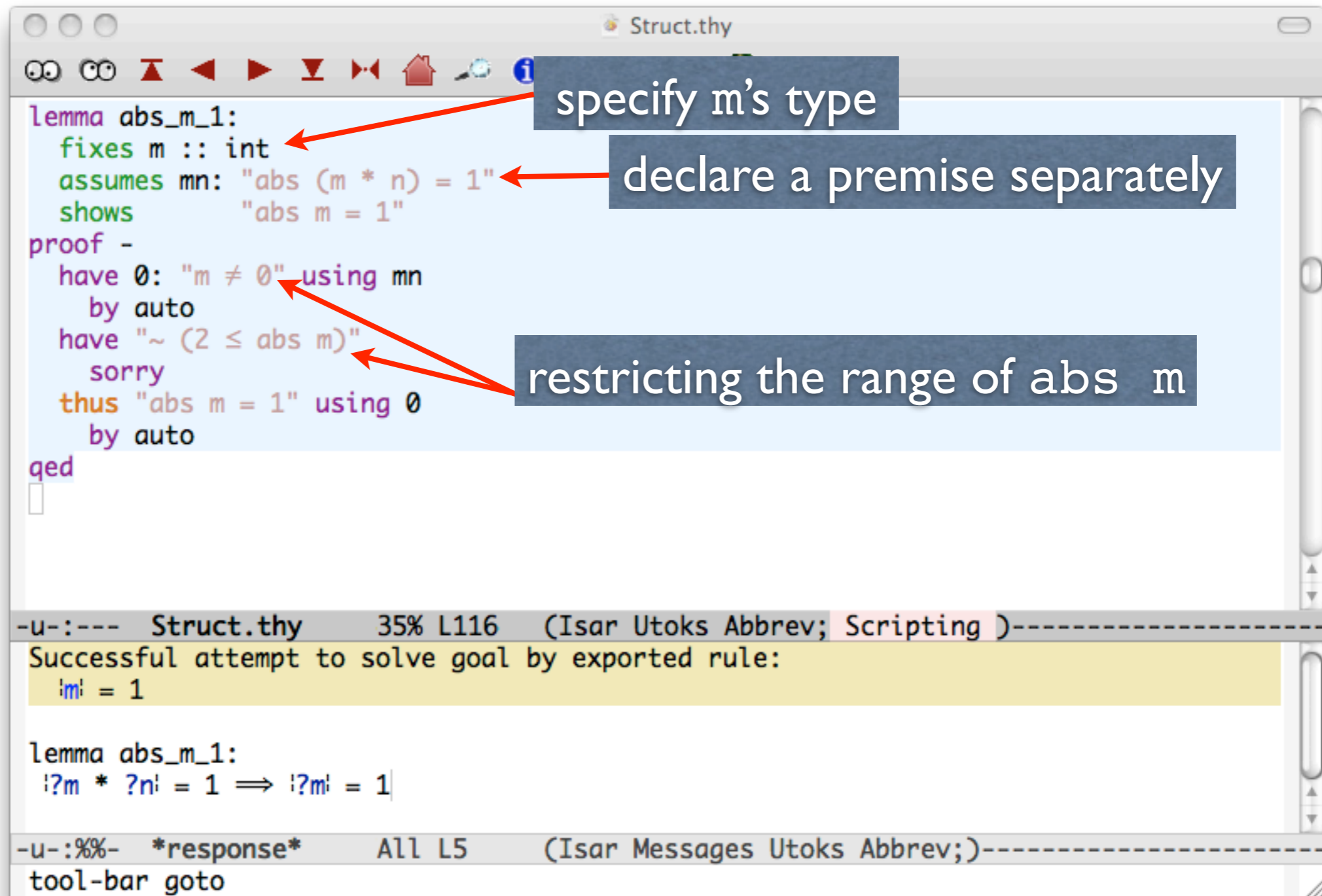
declare a premise separately

-u-:--- Struct.thy 35% L116 (Isar Utoks Abbrev; Scripting)-----
Successful attempt to solve goal by exported rule:
|m| = 1

```
lemma abs_m_1:
  !?m * ?n = 1 ==> !?m = 1
```

-u-:%%- *response* All L5 (Isar Messages Utoks Abbrev;)-----
tool-bar goto

Another Proof Skeleton



The image shows a screenshot of a theorem prover interface with a proof skeleton and its execution output. The proof skeleton is as follows:

```
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
```

Annotations with arrows point to specific parts of the code:

- "specify m's type" points to `fixes m :: int`.
- "declare a premise separately" points to `assumes mn: "abs (m * n) = 1"`.
- "restricting the range of abs m" points to `have 0: "m ≠ 0"` and `have "~ (2 ≤ abs m)"`.

The execution output at the bottom shows:

```
-u-:--- Struct.thy 35% L116 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  |m| = 1

lemma abs_m_1:
  !?m * ?n = 1 ⇒ !?m = 1|

-u-:%%- *response* All L5 (Isar Messages Utoks Abbrev;)-----
tool-bar goto
```


Another Proof Skeleton

```
Struct.thy
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
```

specify m's type

declare a premise separately

restricting the range of abs m

makes the conclusion trivial

-u-:--- Struct.thy 35% L116 (Isar Utoks Abbrev; Scripting)-----
Successful attempt to solve goal by exported rule:
|m| = 1

```
lemma abs_m_1:
  !?m * ?n = 1 ==> !?m = 1
```

-u-:%%- *response* All L5 (Isar Messages Utoks Abbrev;)-----
tool-bar goto

Another Proof Skeleton

The screenshot shows a proof editor window titled "Struct.thy". The main text area contains the following code:

```
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    sorry
  thus "abs m = 1" using 0
    by auto
qed
```

Annotations with arrows point to specific parts of the code:

- "specify m's type" points to `fixes m :: int`.
- "declare a premise separately" points to `assumes mn: "abs (m * n) = 1"`.
- "null proof step" points to the `proof -` line.
- "restricting the range of abs m" points to `have "~ (2 ≤ abs m)"`.
- "makes the conclusion trivial" points to `by auto` at the end of the proof.

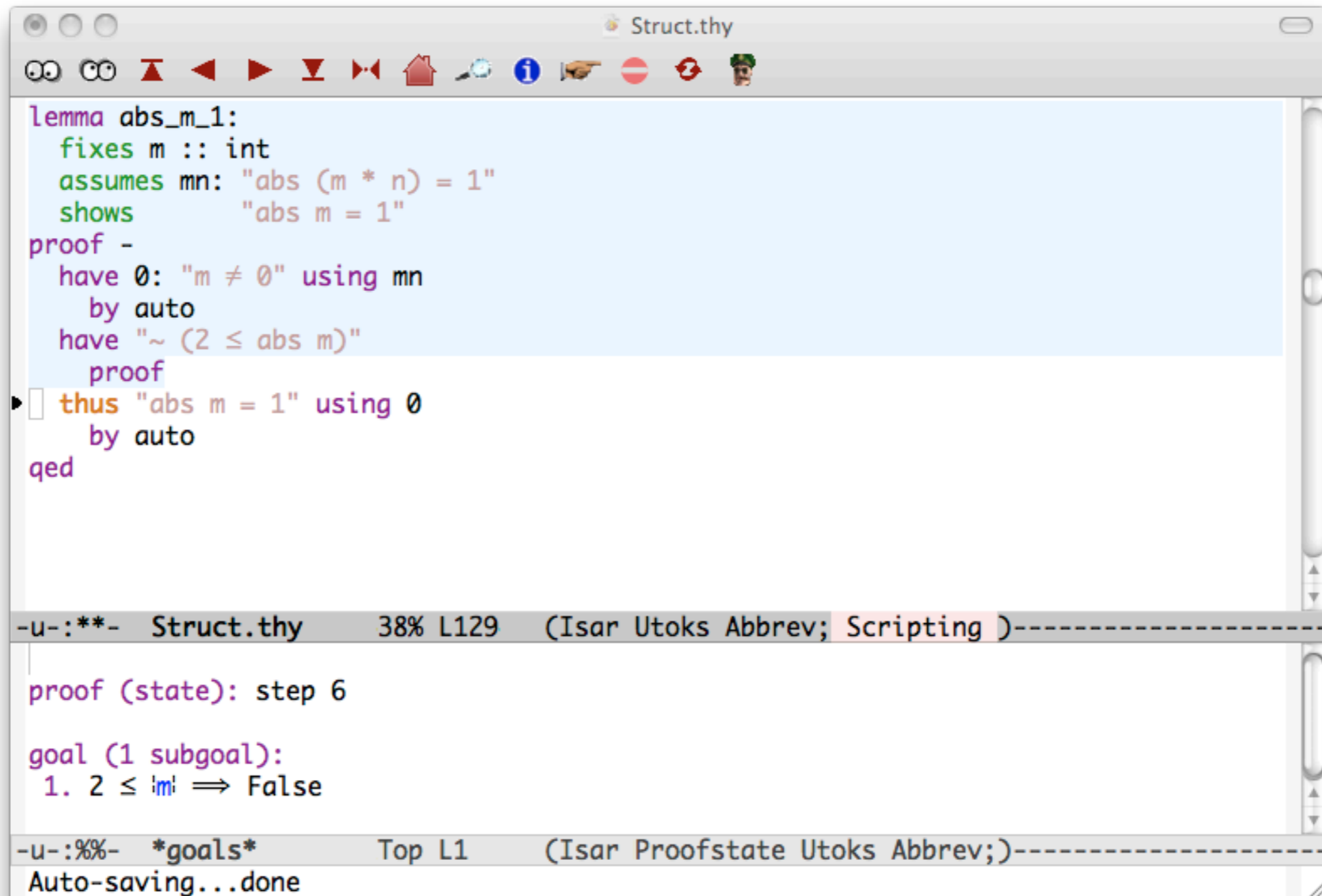
The bottom status bar shows the following information:

```
-u-:--- Struct.thy 35% L116 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
  |m| = 1

lemma abs_m_1:
  !?m * ?n = 1 ==> !?m = 1|

-u-:%%- *response* All L5 (Isar Messages Utoks Abbrev;)-----
tool-bar goto
```

Starting a Nested Proof



The screenshot shows a window titled "Struct.thy" with a toolbar and a text editor. The editor contains the following Isabelle/HOL code:

```
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
   thus "abs m = 1" using 0
    by auto
qed
```

The cursor is positioned at the start of the nested proof block. Below the editor, a status bar shows the current position: "-u-:**- Struct.thy 38% L129 (Isar Utoks Abbrev; Scripting)".

Below the status bar, the proof state is displayed:

```
proof (state): step 6
goal (1 subgoal):
  1. 2 ≤ |m| ⇒ False
```

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

At the very bottom, a message reads: "Auto-saving...done".

Starting a Nested Proof

The screenshot shows a theorem prover interface with a source editor and a command-line window. The source editor contains the following code:

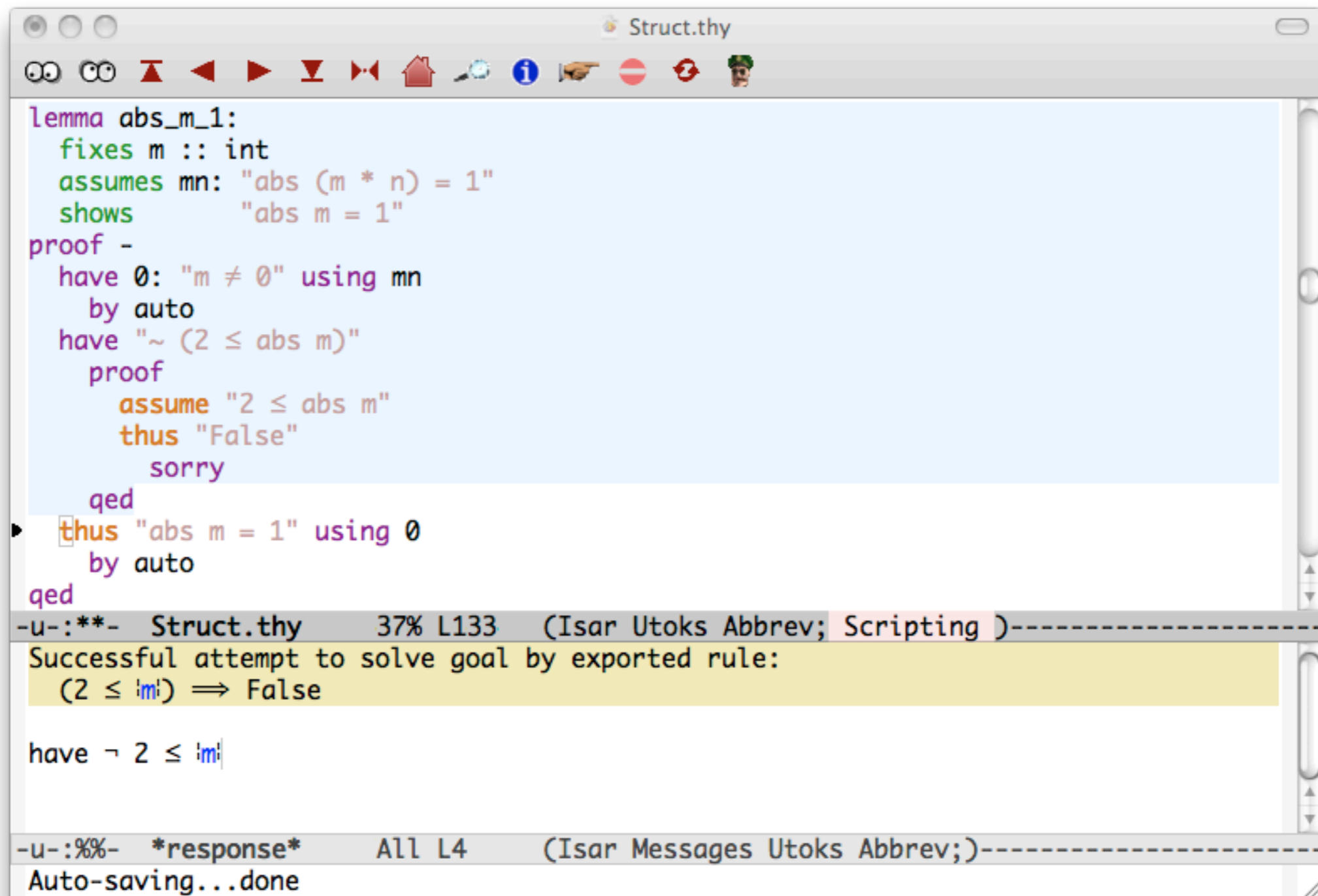
```
lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
  thus "abs m = 1" using 0
    by auto
qed
```

A red arrow points from a purple box labeled "default proof step" to the `proof` block in the source editor. Another red arrow points from the same box to the `proof (state): step 6` line in the command-line window. The command-line window also shows the goal state:

```
proof (state): step 6
goal (1 subgoal):
  1. 2 ≤ |m| ⇒ False
```

The status bar at the bottom of the command-line window indicates the current state: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-`. The source editor status bar shows: `-u-:**- Struct.thy 38% L129 (Isar Utoks Abbrev; Scripting)-`. The text "Auto-saving...done" is visible at the bottom left of the command-line window.

A Nested Proof Skeleton



```
Struct.thy
lemmma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
      assume "2 ≤ abs m"
      thus "False"
        sorry
    qed
  thus "abs m = 1" using 0
    by auto
qed
```

-u-:**- Struct.thy 37% L133 (Isar Utoks Abbrev; Scripting)-----

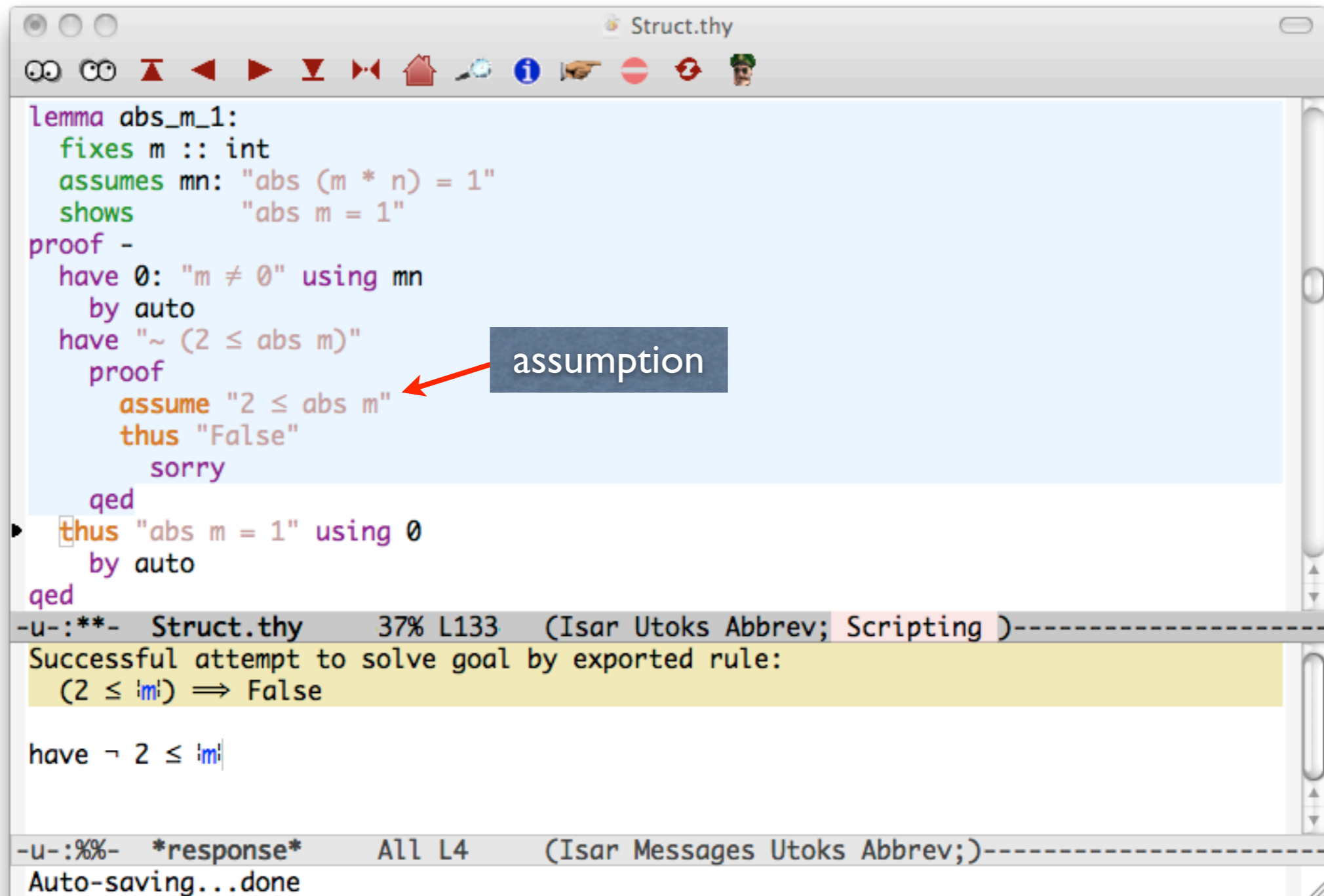
Successful attempt to solve goal by exported rule:
(2 ≤ |m|) ⇒ False

have ¬ 2 ≤ |m|

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

Auto-saving...done

A Nested Proof Skeleton



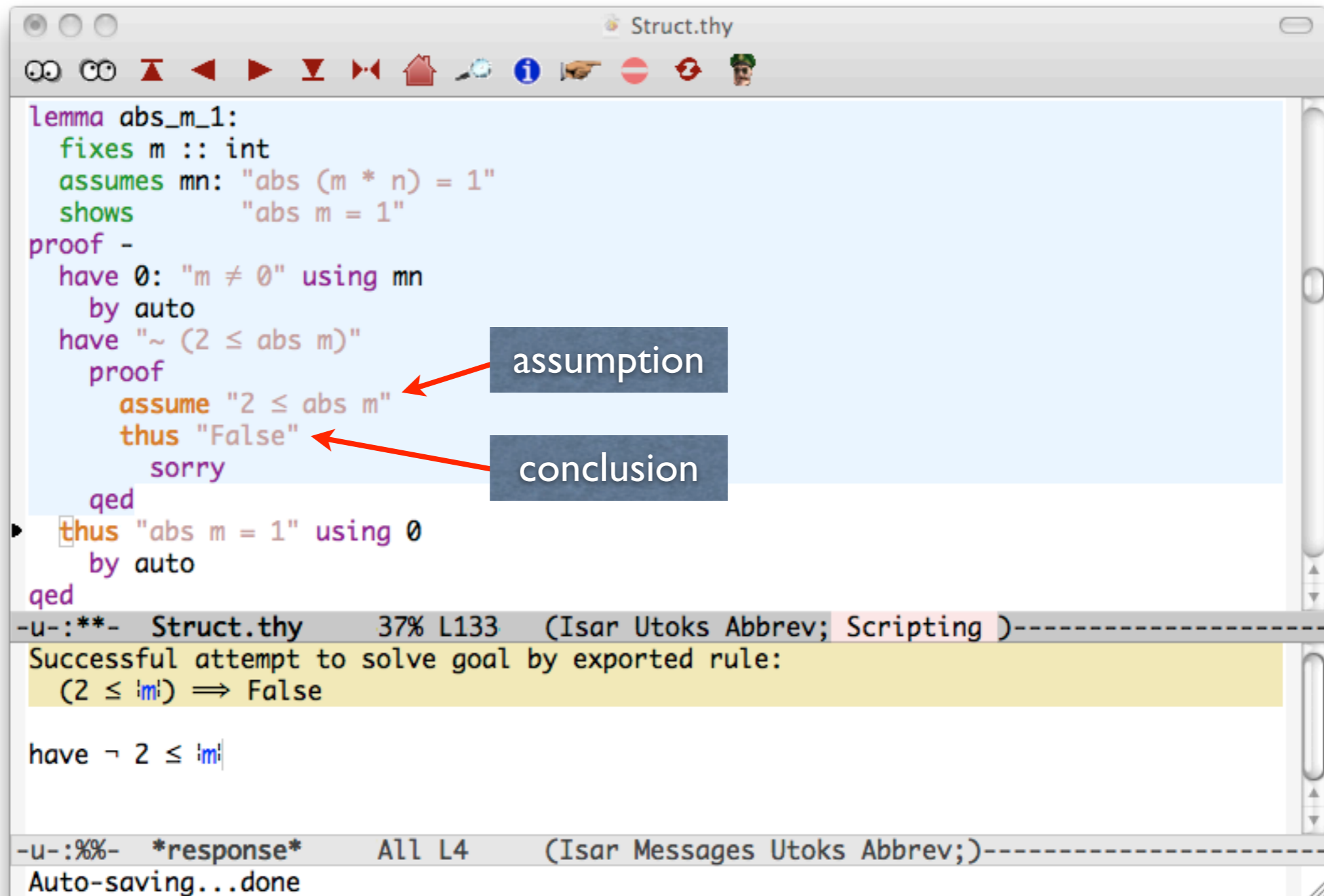
```
Struct.thy
lemmas abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
      assume "2 ≤ abs m"
      thus "False"
        sorry
    qed
  thus "abs m = 1" using 0
    by auto
qed

-u-:***- Struct.thy 37% L133 (Isar Utoks Abbrev; Scripting )-----
Successful attempt to solve goal by exported rule:
(2 ≤ |m|) ⇒ False

have ¬ 2 ≤ |m|

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

A Nested Proof Skeleton



```
Struct.thy
lemmma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
    proof
      assume "2 ≤ abs m"
      thus "False"
        sorry
    qed
  thus "abs m = 1" using 0
    by auto
qed
```

assumption

conclusion

-u-:**- Struct.thy 37% L133 (Isar Utoks Abbrev; Scripting)-----

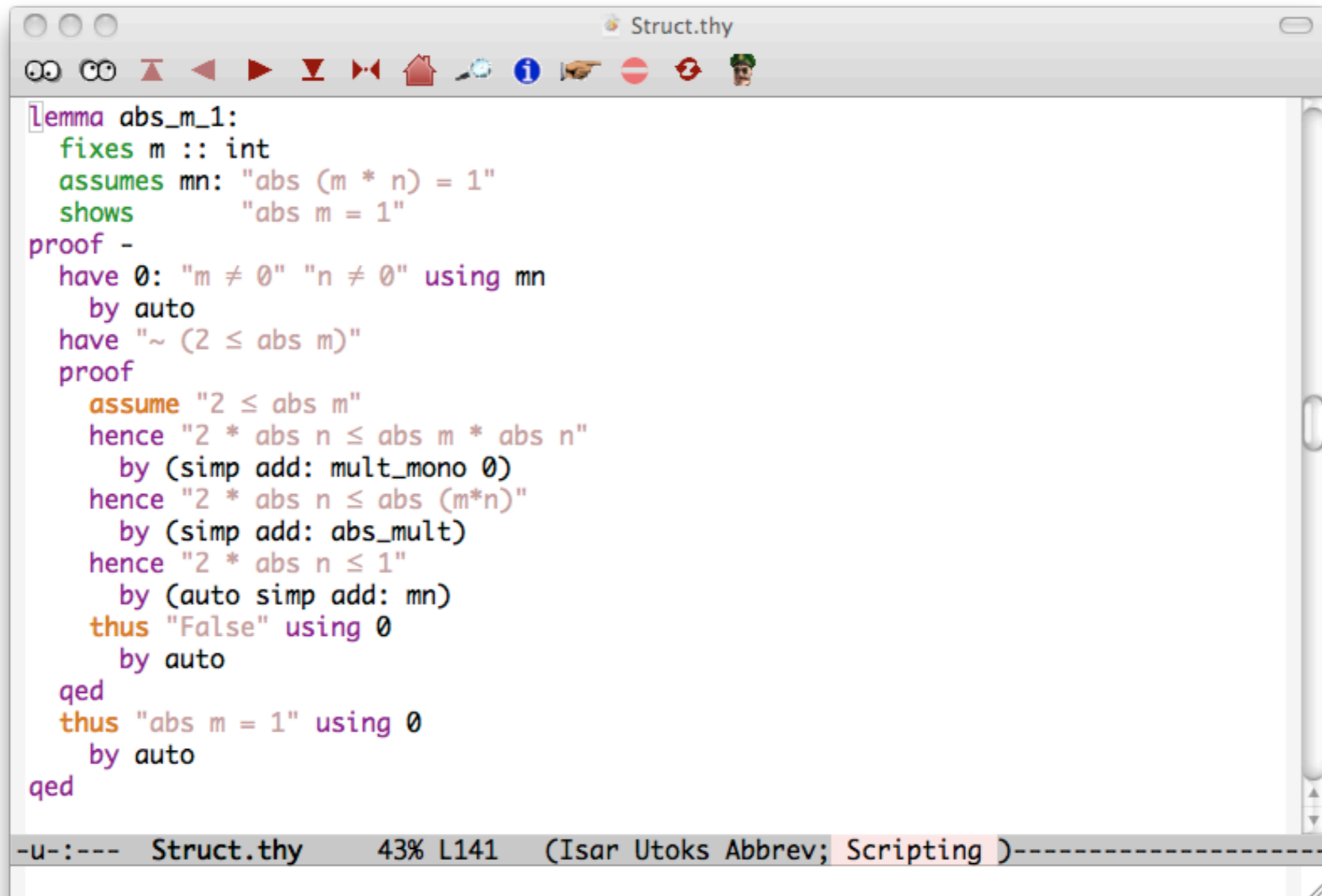
Successful attempt to solve goal by exported rule:
(2 ≤ |m|) ⇒ False

have ¬ 2 ≤ |m|

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

Auto-saving...done

A Complete Proof



```
Struct.thy
[lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" "n ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
  proof
    assume "2 ≤ abs m"
    hence "2 * abs n ≤ abs m * abs n"
      by (simp add: mult_mono 0)
    hence "2 * abs n ≤ abs (m*n)"
      by (simp add: abs_mult)
    hence "2 * abs n ≤ 1"
      by (auto simp add: mn)
    thus "False" using 0
      by auto
  qed
  thus "abs m = 1" using 0
    by auto
qed
```

-u-:--- Struct.thy 43% L141 (Isar Utoks Abbrev; Scripting)-----

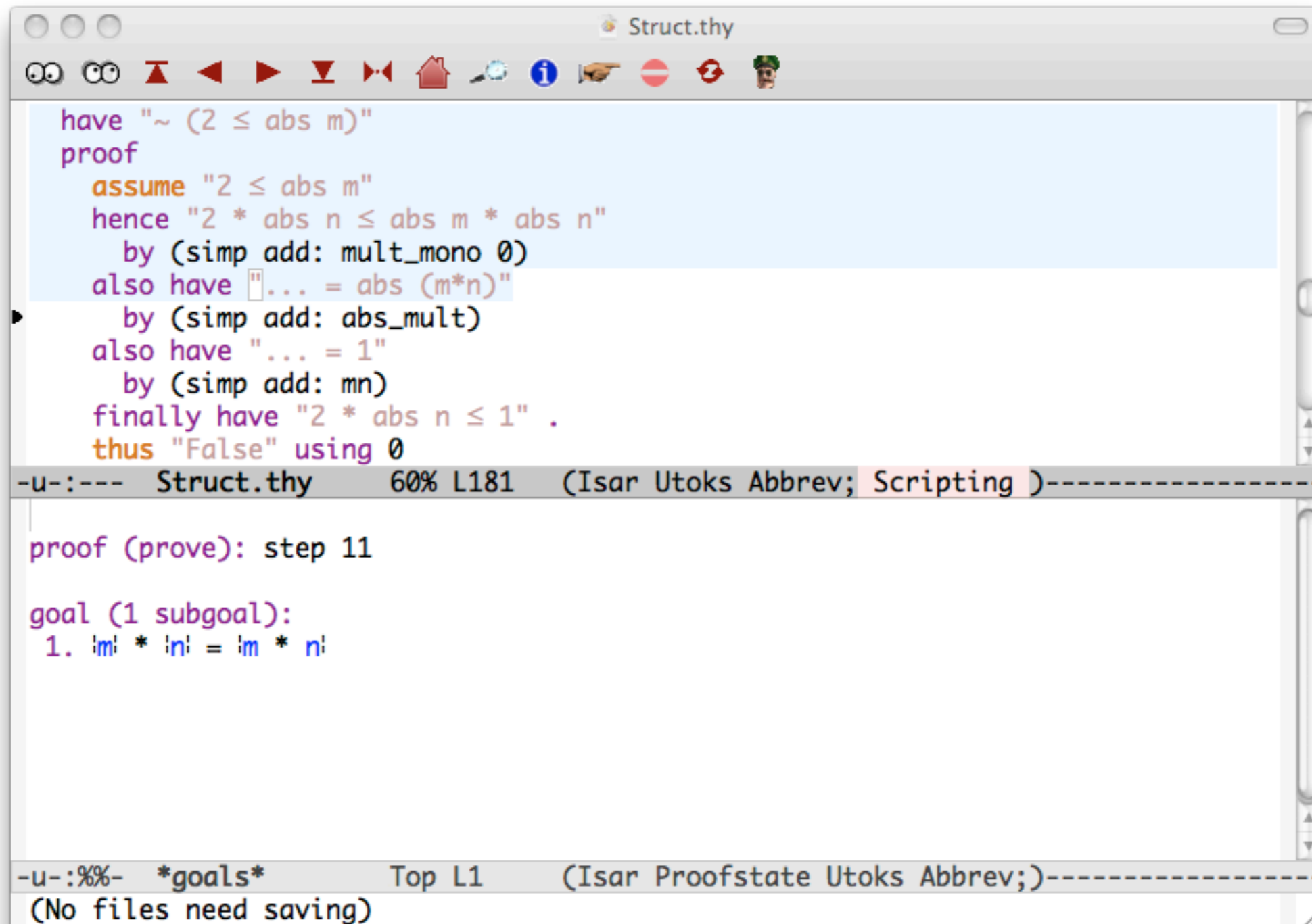
A Complete Proof

```
Struct.thy
[lemma abs_m_1:
  fixes m :: int
  assumes mn: "abs (m * n) = 1"
  shows      "abs m = 1"
proof -
  have 0: "m ≠ 0" "n ≠ 0" using mn
    by auto
  have "~ (2 ≤ abs m)"
  proof
    assume "2 ≤ abs m"
    hence "2 * abs n ≤ abs m * abs n"
      by (simp add: mult_mono 0)
    hence "2 * abs n ≤ abs (m*n)"
      by (simp add: abs_mult)
    hence "2 * abs n ≤ 1"
      by (auto simp add: mn)
    thus "False" using 0
      by auto
  qed
  thus "abs m = 1" using 0
    by auto
qed
```

a chain of steps leads to contradiction

-u:--- Struct.thy 43% L141 (Isar Utoks Abbrev; Scripting)-----

Calculational Proofs



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

```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
  thus "False" using 0
```

A status bar below the code shows: `-u-:--- Struct.thy 60% L181 (Isar Utoks Abbrev; Scripting)`

The bottom panel shows the current proof state:

```
proof (prove): step 11

goal (1 subgoal):
  1. |m| * |n| = |m * n|
```

A second status bar at the bottom reads: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)` and `(No files need saving)`

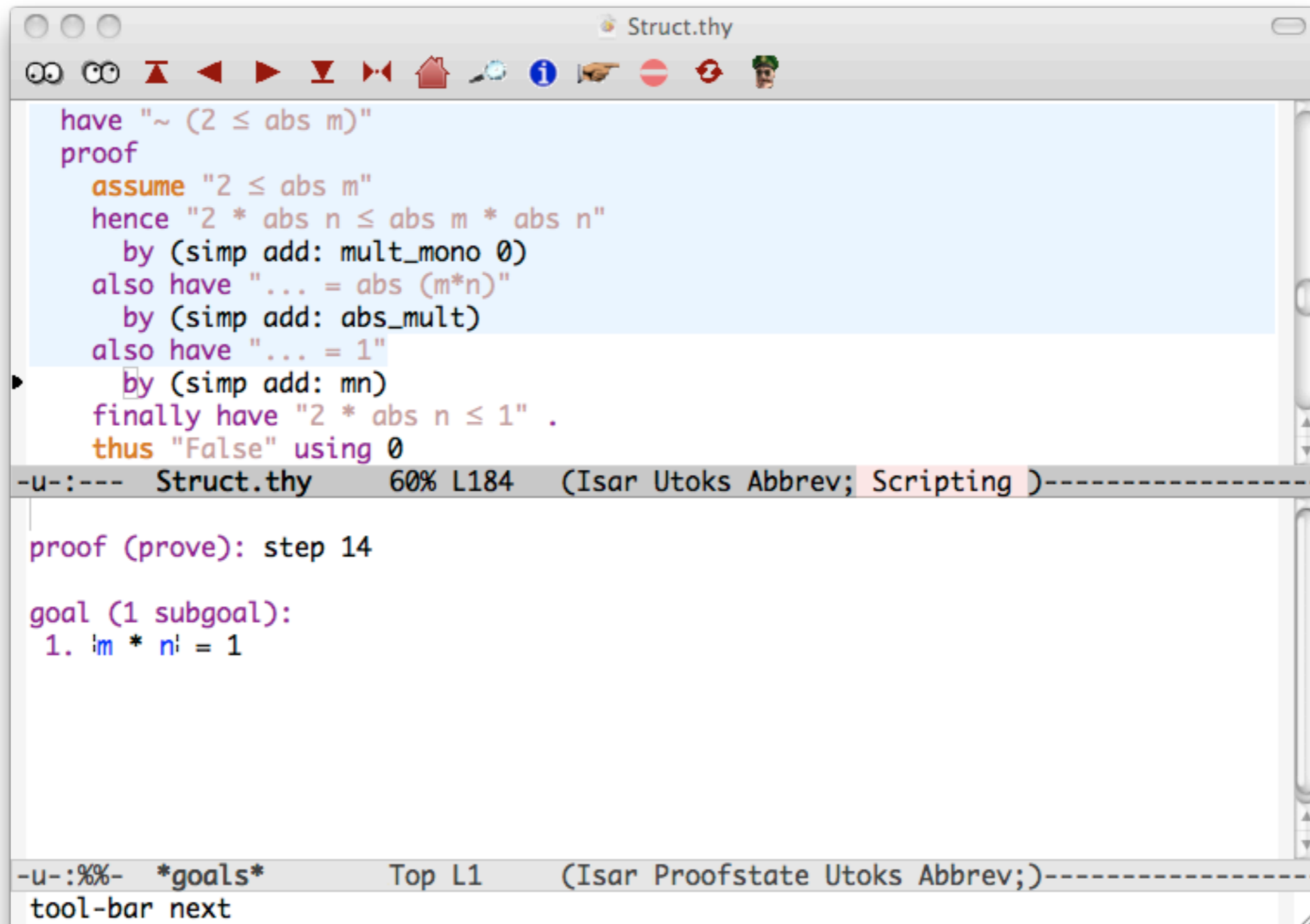
Calculational Proofs

```
Struct.thy
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
  thus "False" using 0
- u-:--- Struct.thy 60% L181 (Isar

proof (prove): step 11
goal (1 subgoal):
  1. |m| * |n| = |m * n|
- u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
(No files need saving)
```

form a series of equalities and inequalities

The Next Step



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

```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
  by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
  thus "False" using 0
```

A status bar below the script shows: `-u-:--- Struct.thy 60% L184 (Isar Utoks Abbrev; Scripting)`. Below this is a separate pane showing the current proof state:

```
proof (prove): step 14

goal (1 subgoal):
  1. |m| * |n| = 1
```

A second status bar at the bottom shows: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)`. At the very bottom, the text `tool-bar next` is visible.

The Next Step

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 expression `... = 1` in the script.

```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1"
  thus "False" using 0
```

proof (prove): step 14

goal (1 subgoal):

1. $|m * n| = 1$

tool-bar next

refers to the previous right-hand side

The Internal Calculation

The screenshot shows a window titled "Struct.thy" with a toolbar at the top. The main area contains a proof script in a light blue background:

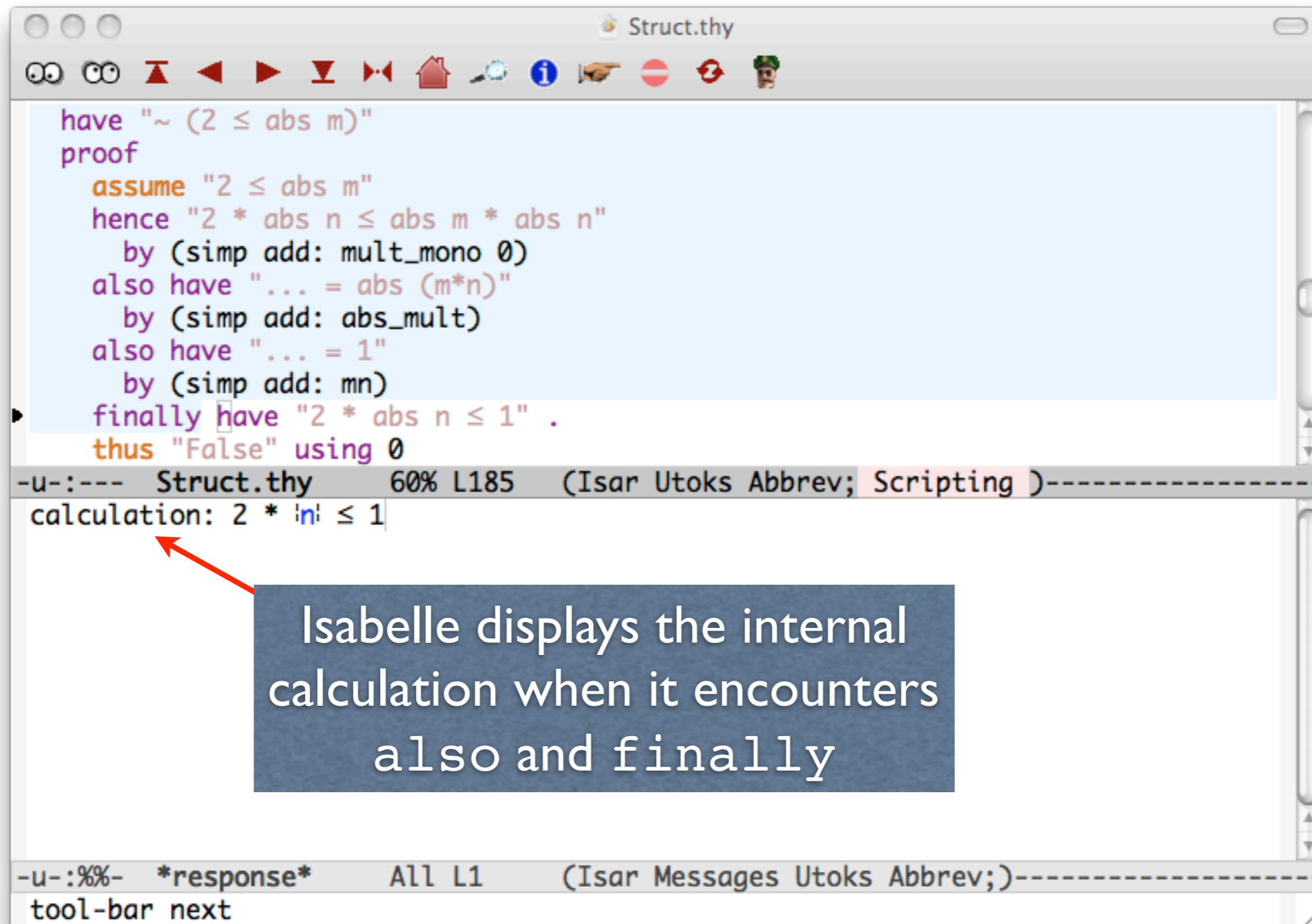
```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
  thus "False" using 0
```

Below the script is a status bar: "-u-:--- Struct.thy 60% L185 (Isar Utoks Abbrev; Scripting)-----".

The bottom section shows the internal calculation: "calculation: 2 * |n| ≤ 1".

At the very bottom, another status bar reads: "-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)-----" and the text "tool-bar next" is visible.

The Internal Calculation



```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
  thus "False" using 0

```

-u-:--- Struct.thy 60% L185 (Isar Utoks Abbrev; Scripting)-----

calculation: 2 * |n| ≤ 1

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

tool-bar next

Isabelle displays the internal calculation when it encounters also and finally

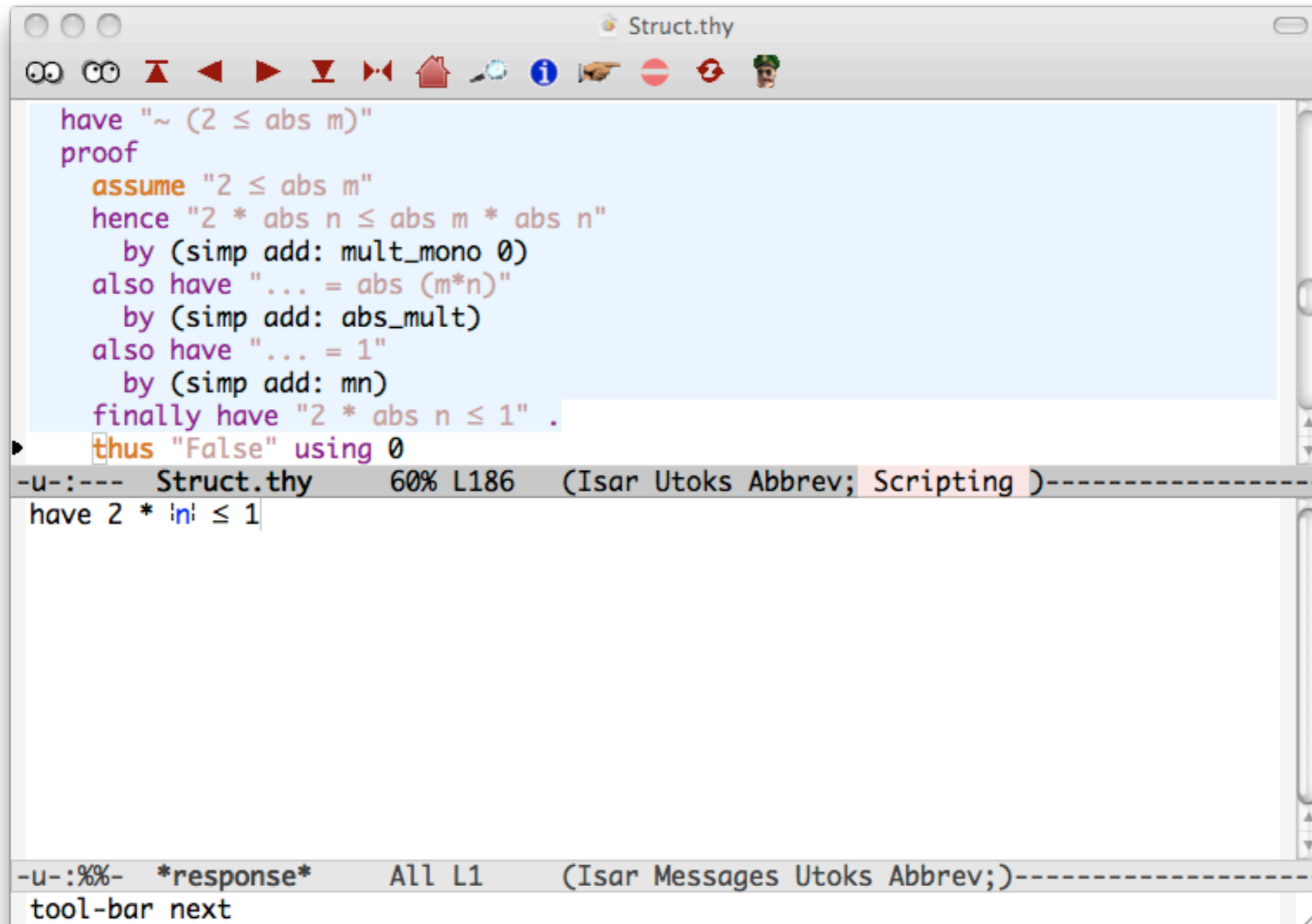
The Internal Calculation

structure
of a
calculation

```
Struct.thy
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
  thus "False" using 0
-u-:--- Struct.thy 60% L185 (Isar Utoks Abbrev; Scripting )-----
calculation: 2 * |n| ≤ 1
-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)-----
tool-bar next
```

Isabelle displays the internal
calculation when it encounters
also and finally

Ending the Calculation



```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
thus "False" using 0
```

-u-:--- Struct.thy 60% L186 (Isar Utoks Abbrev; Scripting)-----

```
have 2 * |n| ≤ 1
```

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

tool-bar next

Ending the Calculation

The screenshot shows a theorem prover interface with a proof script in the main editor and its execution output in a lower pane. The proof script is as follows:

```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
thus "False" using 0
```

The execution output pane shows the following message:

```
-u-:--- Struct.thy 60% L186 (Isar Utoks Abbrev; Scripting )-----
have 2 * |n| ≤ 1
```

Two red arrows point from a blue callout box to the `finally` line in the proof script and the `have 2 * |n| ≤ 1` line in the output pane. The callout box contains the text:

We have deduced $2 \times \text{abs } n \leq 1$

Ending the Calculation

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

```
have "~ (2 ≤ abs m)"
proof
  assume "2 ≤ abs m"
  hence "2 * abs n ≤ abs m * abs n"
    by (simp add: mult_mono 0)
  also have "... = abs (m*n)"
    by (simp add: abs_mult)
  also have "... = 1"
    by (simp add: mn)
  finally have "2 * abs n ≤ 1" .
thus "False" using 0
```

Below the script, a status bar indicates the current line: "-u-:--- Struct.thy 60% L186 (Isar Utoks Abbrev; Scripting)". The output area shows the result of the proof:

```
have 2 * |n| ≤ 1
```

Two callout boxes with arrows point to specific parts of the code:

- A box containing "indicates a trivial proof" points to the period at the end of the `finally have` line.
- A box containing "We have deduced $2 \times \text{abs } n \leq 1$ " points to the output line `have 2 * |n| ≤ 1`.

At the bottom of the window, another status bar shows "-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)" and "tool-bar next".

Structure of a Calculation

Structure of a Calculation

- The first line is *have/hence*

Structure of a Calculation

- The first line is *have/hence*
- Subsequent lines begin, also have "*... =*"

Structure of a Calculation

- The first line is *have/hence*
- Subsequent lines begin, also have "*... =*"
- *Any* transitive relation may be used. New ones may be declared.

Structure of a Calculation

- The first line is *have/hence*
- Subsequent lines begin, also have "*... =*"
- *Any* transitive relation may be used. New ones may be declared.
- The concluding line begins, *finally have/show*, repeats the calculation and terminates with *(.)*