Interactive Formal Verification
6: Sets

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

Set Notation in Isabelle

Set Notation in Isabelle

- Set notation is crucial to mathematical discourse.

Set Notation in Isabelle

- Set notation is crucial to mathematical discourse.
- Set-theoretic abstractions naturally express many complex constructions.

Set Notation in Isabelle

- Set notation is crucial to mathematical discourse.
- Set-theoretic abstractions naturally express many complex constructions.
- A set in higher-order logic is a *boolean-valued map*.

Set Notation in Isabelle

- Set notation is crucial to mathematical discourse.
- Set-theoretic abstractions naturally express many complex constructions.
- A set in higher-order logic is a *boolean-valued map*.
- Its elements must all have the *same type*

Set Theory Primitives

Set Theory Primitives

- The type α `set`, which abbreviates $\alpha \Rightarrow$ `bool`

Set Theory Primitives

- The type α `set`, which abbreviates $\alpha \Rightarrow$ `bool`
- The membership relation: $\in$

Set Theory Primitives

- The type α `set`, which abbreviates $\alpha \Rightarrow$`bool`
- The membership relation: $\in$
- The subset relation: $\subseteq$
  - Reflexive, anti-symmetric, transitive

Set Theory Primitives

- The type `α set`, which abbreviates `α⇒bool`
- The membership relation: $\in$
- The subset relation: $\subseteq$
  - Reflexive, anti-symmetric, transitive
- The empty set: { }

Set Theory Primitives

- The type `α set`, which abbreviates `α⇒bool`
- The membership relation: $\in$
- The subset relation: $\subseteq$
  - Reflexive, anti-symmetric, transitive
- The empty set: `{ }`
- The universal set: `UNIV`

Basic Set Theory Operations

Basic Set Theory Operations

$$e \in \{x.\, P(x)\} \iff P(e)$$

Basic Set Theory Operations

$$e \in \{x.\, P(x)\} \iff P(e)$$

$$e \in \{x \in A.\, P(x)\} \iff e \in A \land P(e)$$

Basic Set Theory Operations

$$
\begin{aligned}
e \in \{x.\, P(x)\} &\iff P(e) \\
e \in \{x \in A.\, P(x)\} &\iff e \in A \land P(e) \\
e \in -A &\iff e \notin A
\end{aligned}
$$

Basic Set Theory Operations

$$
\begin{aligned}
e \in \{x.\, P(x)\} &\iff P(e) \\
e \in \{x \in A.\, P(x)\} &\iff e \in A \wedge P(e) \\
e \in -A &\iff e \notin A \\
e \in A \cup B &\iff e \in A \vee e \in B
\end{aligned}
$$

Basic Set Theory Operations

$$
\begin{aligned}
e \in \{x.\, P(x)\} &\iff P(e) \\
e \in \{x \in A.\, P(x)\} &\iff e \in A \land P(e) \\
e \in -A &\iff e \notin A \\
e \in A \cup B &\iff e \in A \lor e \in B \\
e \in A \cap B &\iff e \in A \land e \in B
\end{aligned}
$$

Basic Set Theory Operations

$$\begin{aligned}
e \in \{x.\, P(x)\} &\iff P(e) \\
e \in \{x \in A.\, P(x)\} &\iff e \in A \land P(e) \\
e \in -A &\iff e \notin A \\
e \in A \cup B &\iff e \in A \lor e \in B \\
e \in A \cap B &\iff e \in A \land e \in B \\
e \in \texttt{Pow}(A) &\iff e \subseteq A
\end{aligned}$$

Big Union and Intersection

Big Union and Intersection

$$e \in \left(\bigcup x.\, B(x)\right) \iff \exists x.\, e \in B(x)$$

Big Union and Intersection

$$e \in \left(\bigcup x.\, B(x)\right) \iff \exists x.\, e \in B(x)$$

$$e \in \left(\bigcup x \in A.\, B(x)\right) \iff \exists x \in A.\, e \in B(x)$$

Big Union and Intersection

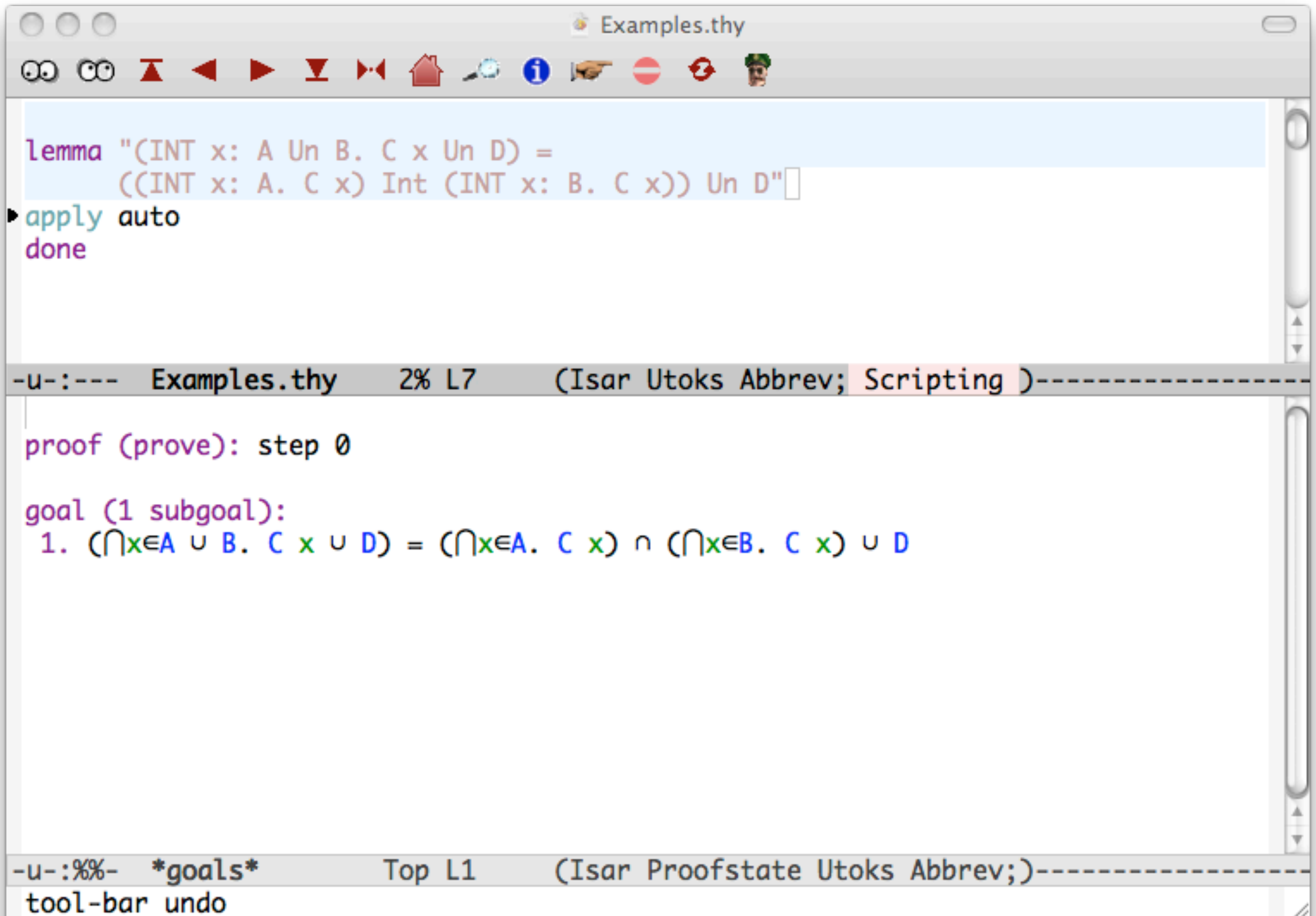
$$
\begin{aligned}
e \in \left(\bigcup x.\, B(x)\right) &\iff \exists x.\, e \in B(x) \\
e \in \left(\bigcup x \in A.\, B(x)\right) &\iff \exists x \in A.\, e \in B(x) \\
e \in \bigcup A &\iff \exists x \in A.\, e \in x
\end{aligned}
$$

Big Union and Intersection

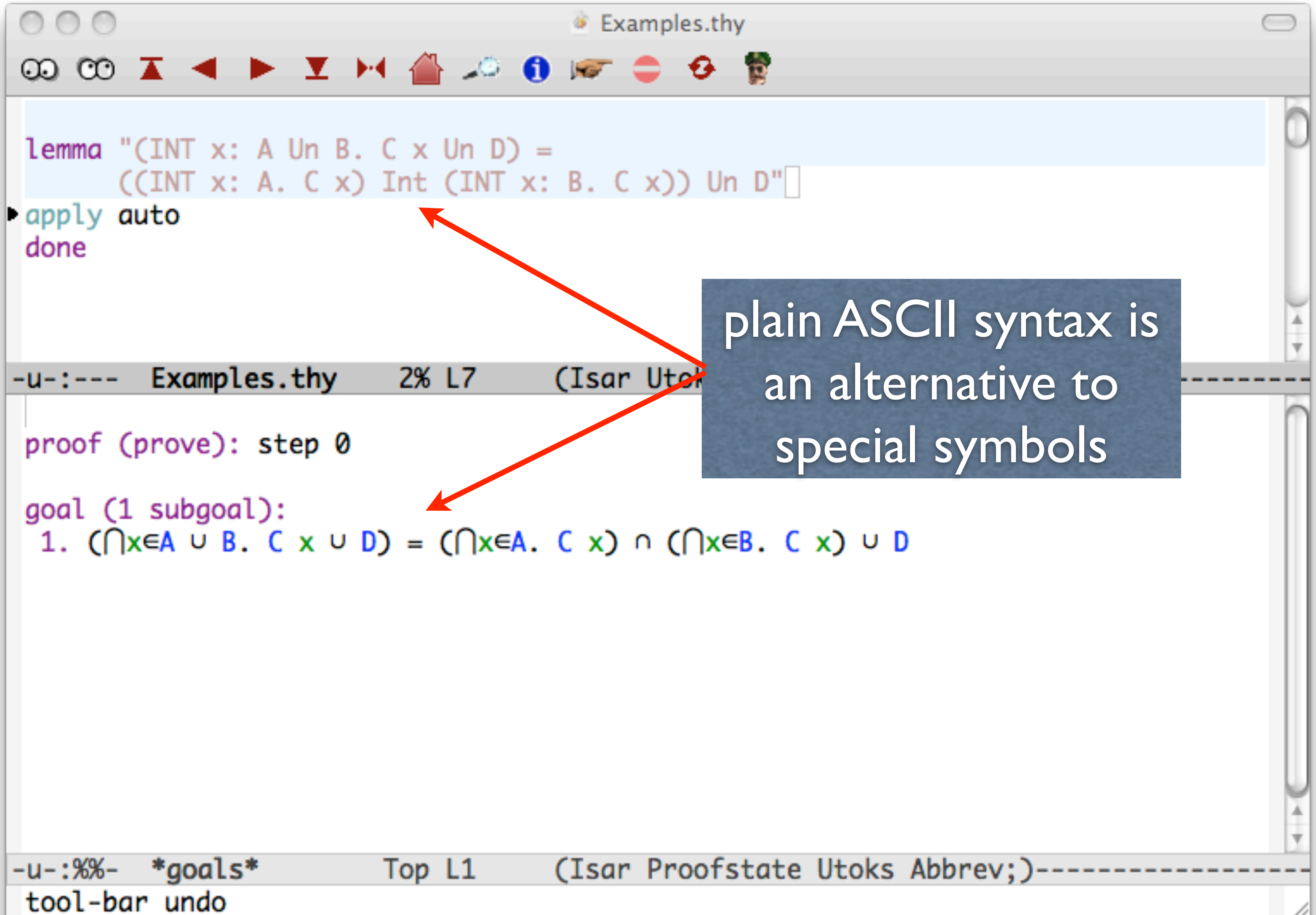
$$
\begin{aligned}
e \in \left(\bigcup x.\, B(x)\right) &\iff \exists x.\, e \in B(x) \\
e \in \left(\bigcup x \in A.\, B(x)\right) &\iff \exists x \in A.\, e \in B(x) \\
e \in \bigcup A &\iff \exists x \in A.\, e \in x
\end{aligned}
$$

And the analogous forms of intersections...

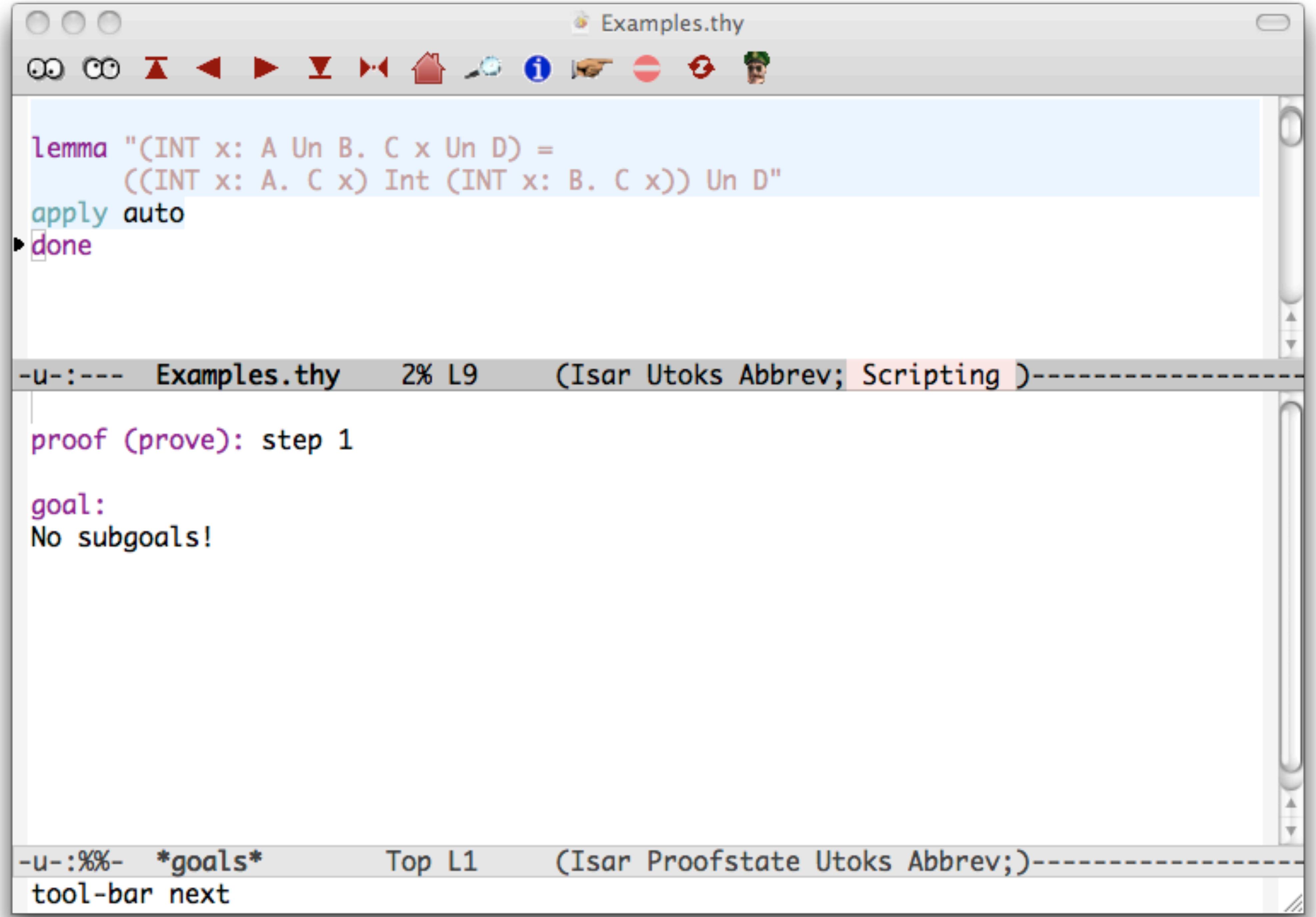
A Simple Set Theory Proof



A Simple Set Theory Proof



A Simple Set Theory Proof



Functions

Functions

$$e \in (f`A) \iff \exists x \in A.\, e = f(x)$$

Functions

$$e \in (f`A) \iff \exists x \in A.\, e = f(x)$$

$$e \in (f-`A) \iff f(e) \in A$$

Functions

$$e \in (f`A) \iff \exists x \in A.\, e = f(x)$$

$$e \in (f-`A) \iff f(e) \in A$$

$$f(x{:=}y) = (\lambda z.\ \texttt{if}\ z = x\ \texttt{then}\ y\ \texttt{else}\ f(z))$$

Functions

$$e \in (f`A) \iff \exists x \in A.\, e = f(x)$$

$$e \in (f-`A) \iff f(e) \in A$$

$$f(x{:=}y) = (\lambda z.\ \texttt{if}\, z = x\, \texttt{then}\, y\, \texttt{else}\, f(z))$$

- Also `inj`, `surj`, `bij`, `inv`, etc. (injective,...)

Functions

$$e \in (f`A) \iff \exists x \in A.\, e = f(x)$$

$$e \in (f-`A) \iff f(e) \in A$$

$$f(x{:=}y) = (\lambda z.\ \texttt{if}\ z = x\ \texttt{then}\ y\ \texttt{else}\ f(z))$$

- Also `inj`, `surj`, `bij`, `inv`, etc. (injective,...)
- Don't *re-invent* image and inverse image!!

Finite Set Notation

Finite Set Notation

$$\{a_1, \ldots, a_n\} = \mathtt{insert}(a_1, \ldots, \mathtt{insert}(a_n, \{\}))$$

Finite Set Notation

$$\{a_1, \ldots, a_n\} = \texttt{insert}(a_1, \ldots, \texttt{insert}(a_n, \{\}))$$

$$e \in \texttt{insert}(a, B) \iff e = a \vee e \in B$$

Finite Sets

A finite set is defined *inductively*
in terms of `{}` and `insert`

Finite Sets

A finite set is defined *inductively*
in terms of `{}` and `insert`

$$\mathtt{finite}(A \cup B) = (\mathtt{finite}\, A \wedge \mathtt{finite}\, B)$$

Finite Sets

A finite set is defined *inductively*
in terms of `{}` and `insert`

$$\texttt{finite}(A \cup B) = (\texttt{finite}\, A \wedge \texttt{finite}\, B)$$

$$\texttt{finite}\, A \Longrightarrow \texttt{card}(\texttt{Pow}\, A) = 2^{\texttt{card}\, A}$$

Intervals, Sums and Products

Intervals, Sums and Products

```
{..<u}  == {x. x < u}
 {..u}  == {x. x ≤ u}
{l<..}  == {x. l<x}
 {l..}  == {x. l≤x}
{l<..<u} == {l<..} ∩ {..<u}
 {l..<u} == {l..} ∩ {..<u}
```

Intervals, Sums and Products

```
{..<u}   == {x. x < u}
 {..u}   == {x. x ≤ u}
{l<..}   == {x. l<x}
 {l..}   == {x. l≤x}
{l<..<u} == {l<..} ∩ {..<u}
{l..<u}  == {l..} ∩ {..<u}
```

`setsum f A` and `setprod f A`

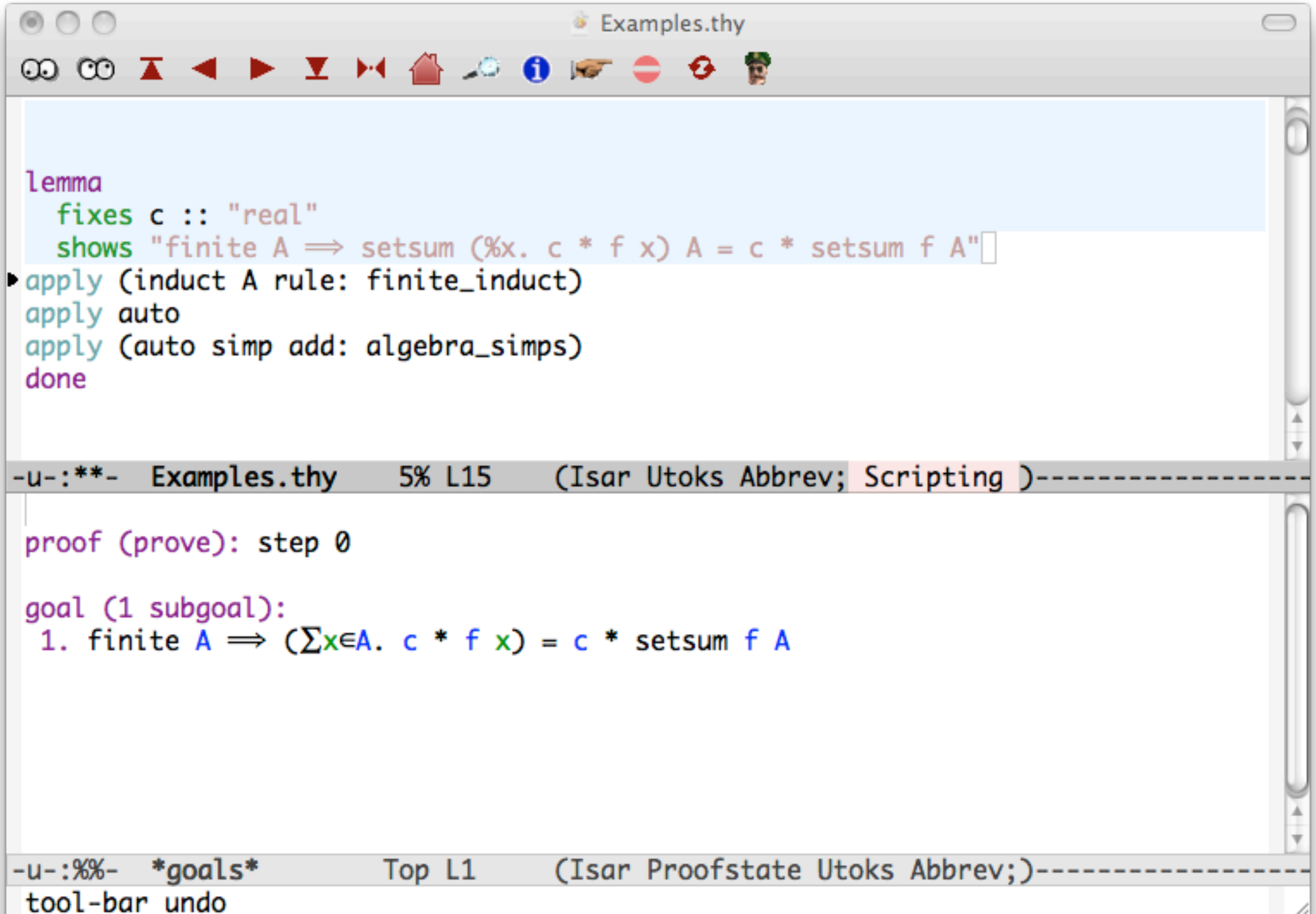
Intervals, Sums and Products

```
{..<u}   == {x. x < u}
{..u}    == {x. x ≤ u}
{l<..}   == {x. l<x}
{l..}    == {x. l≤x}
{l<..<u} == {l<..} ∩ {..<u}
{l..<u}  == {l..} ∩ {..<u}
```

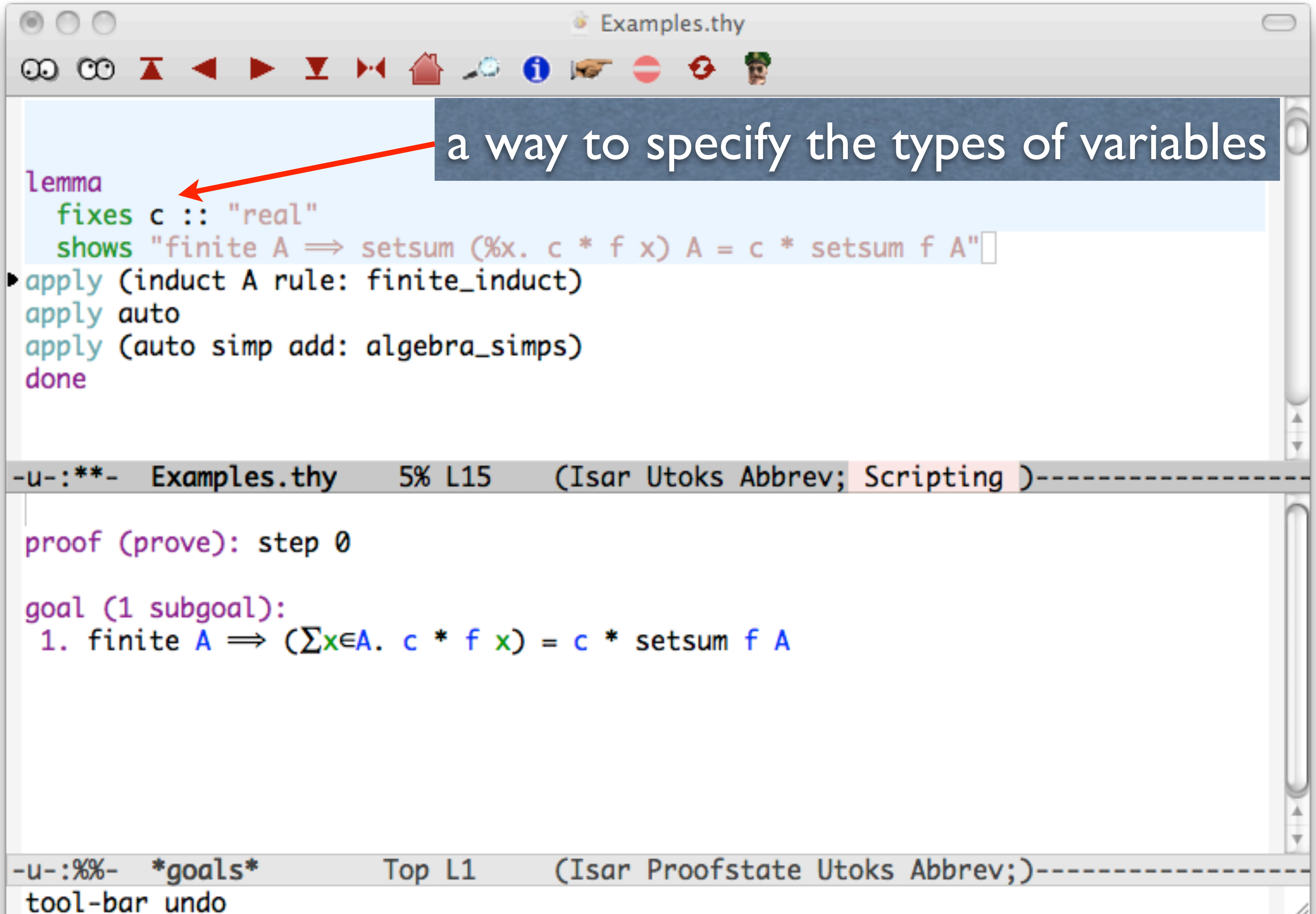
`setsum f A` and `setprod f A`

$\sum i \in I.\ f$ and $\prod i \in I.\ f$

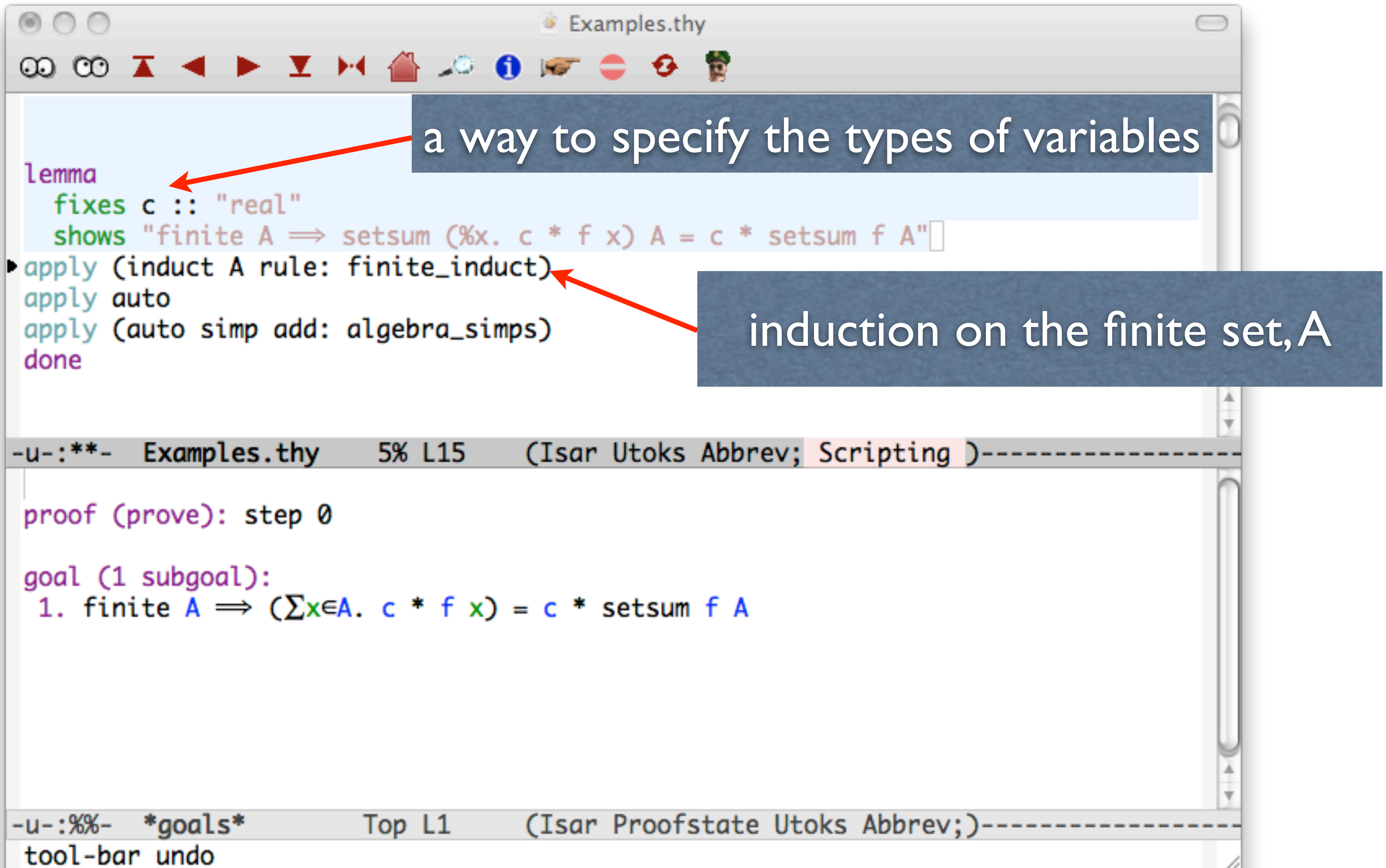
A Harder Proof Involving Sets



A Harder Proof Involving Sets



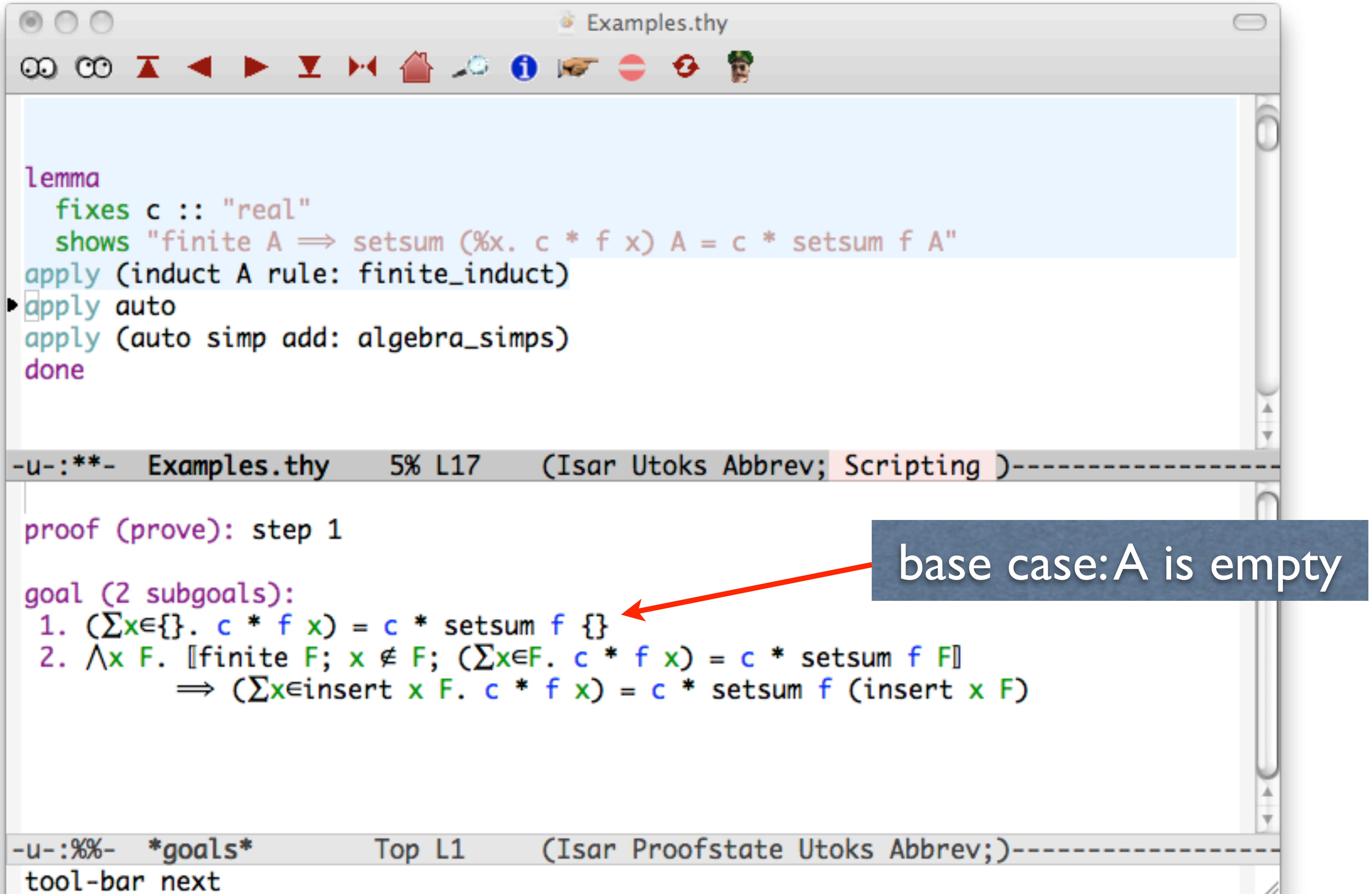
A Harder Proof Involving Sets



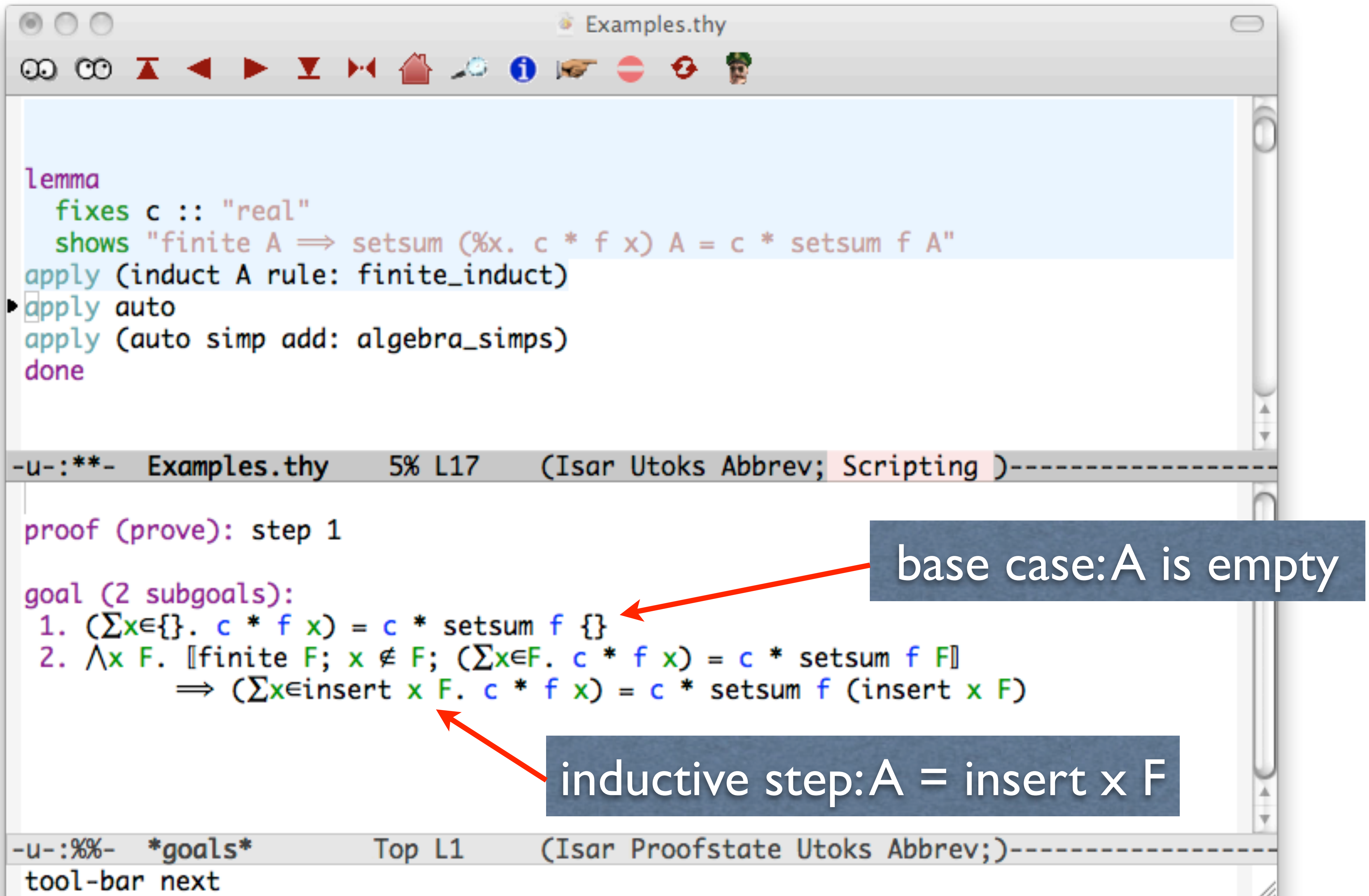
Outcome of the Induction

```
Examples.thy

lemma
  fixes c :: "real"
  shows "finite A ⟹ setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done

-u-:**-  Examples.thy     5% L17     (Isar Utoks Abbrev; Scripting )----------------

proof (prove): step 1

goal (2 subgoals):
 1. (∑x∈{}. c * f x) = c * setsum f {}
 2. ⋀x F. ⟦finite F; x ∉ F; (∑x∈F. c * f x) = c * setsum f F⟧
          ⟹ (∑x∈insert x F. c * f x) = c * setsum f (insert x F)

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

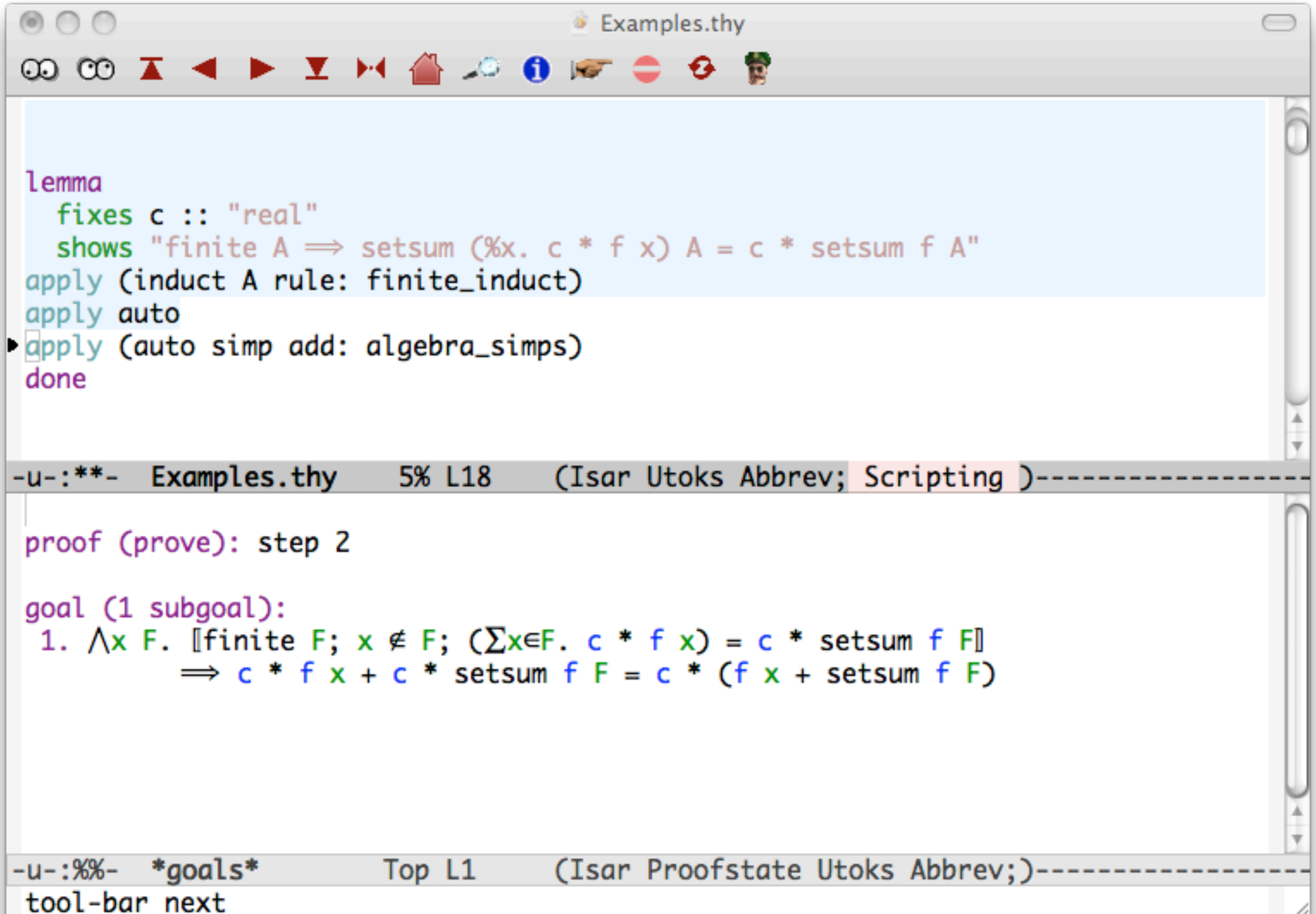
Outcome of the Induction



Outcome of the Induction



Almost There!



Almost There!

Examples.thy

```
lemma
  fixes c :: "real"
  shows "finite A ⟹ setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

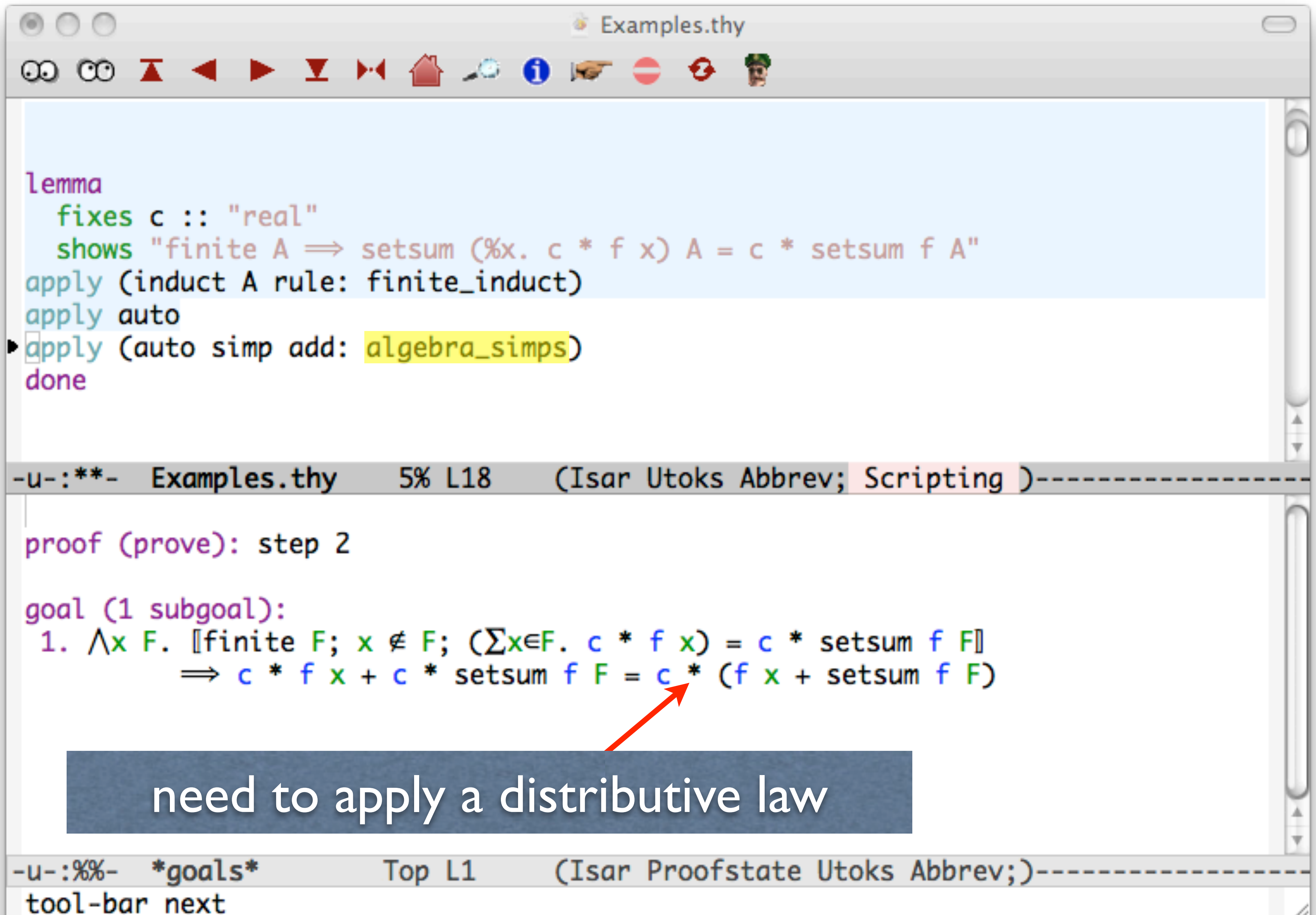
```
-u-:**-  Examples.thy     5% L18     (Isar Utoks Abbrev; Scripting )-----------------
```

```
proof (prove): step 2

goal (1 subgoal):
 1. ⋀x F. ⟦finite F; x ∉ F; (∑x∈F. c * f x) = c * setsum f F⟧
         ⟹ c * f x + c * setsum f F = c * (f x + setsum f F)
```

need to apply a distributive law

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

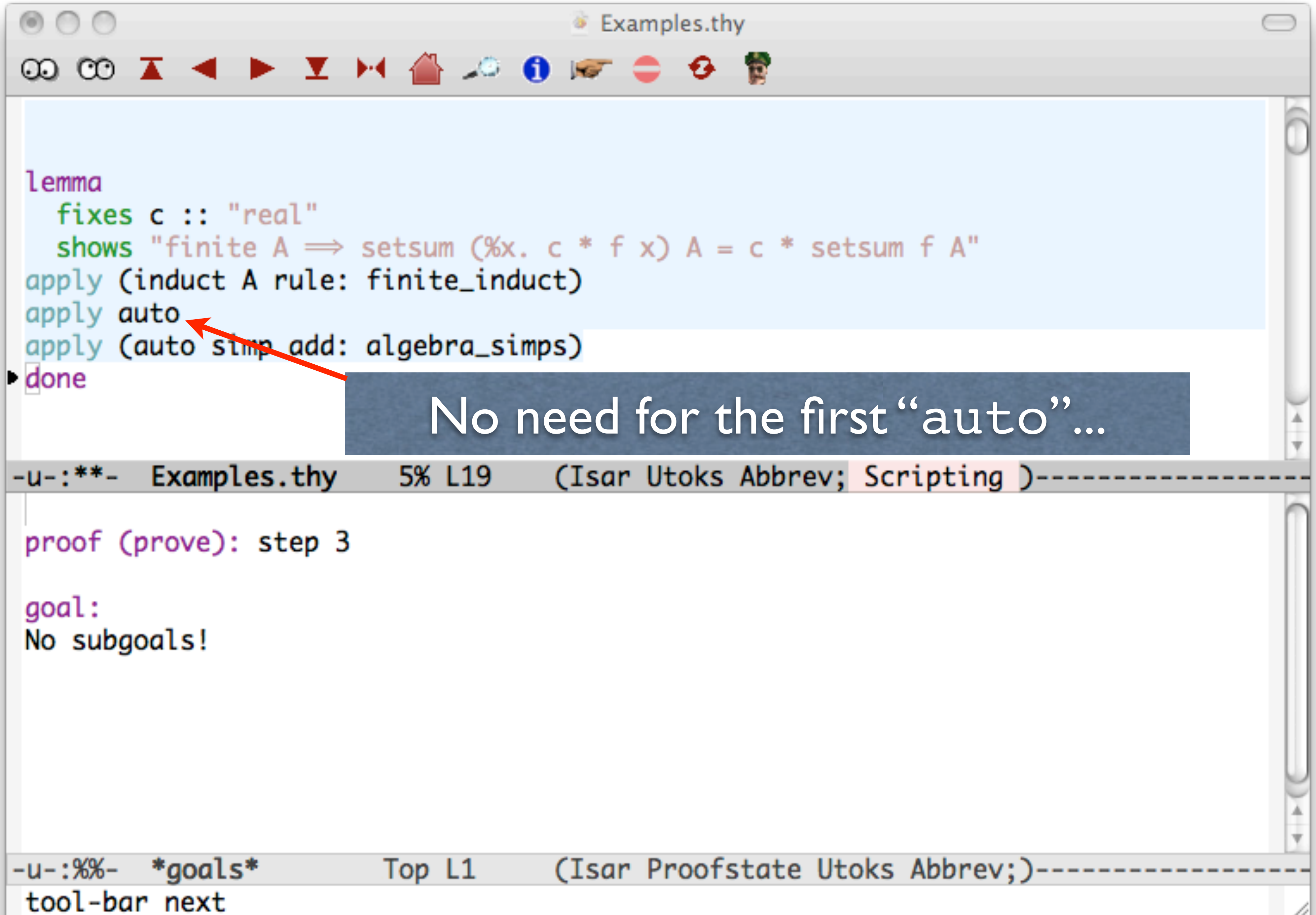
Almost There!



Finished!

```
Examples.thy

lemma
  fixes c :: "real"
  shows "finite A ⟹ setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done

-u-:**-  Examples.thy    5% L19    (Isar Utoks Abbrev; Scripting )----------------

proof (prove): step 3

goal:
No subgoals!

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

Finished!


Examples.thy
lemma
fixes c :: "real"
shows "finite A ⟹ setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
No need for the first "auto"...
-u-:**- Examples.thy 5% L19 (Isar Utoks Abbrev; Scripting)
proof (prove): step 3
goal:
No subgoals!
-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)
tool-bar next

Proving Theorems about Sets

Proving Theorems about Sets

- It is not practical to learn all the built-in lemmas.

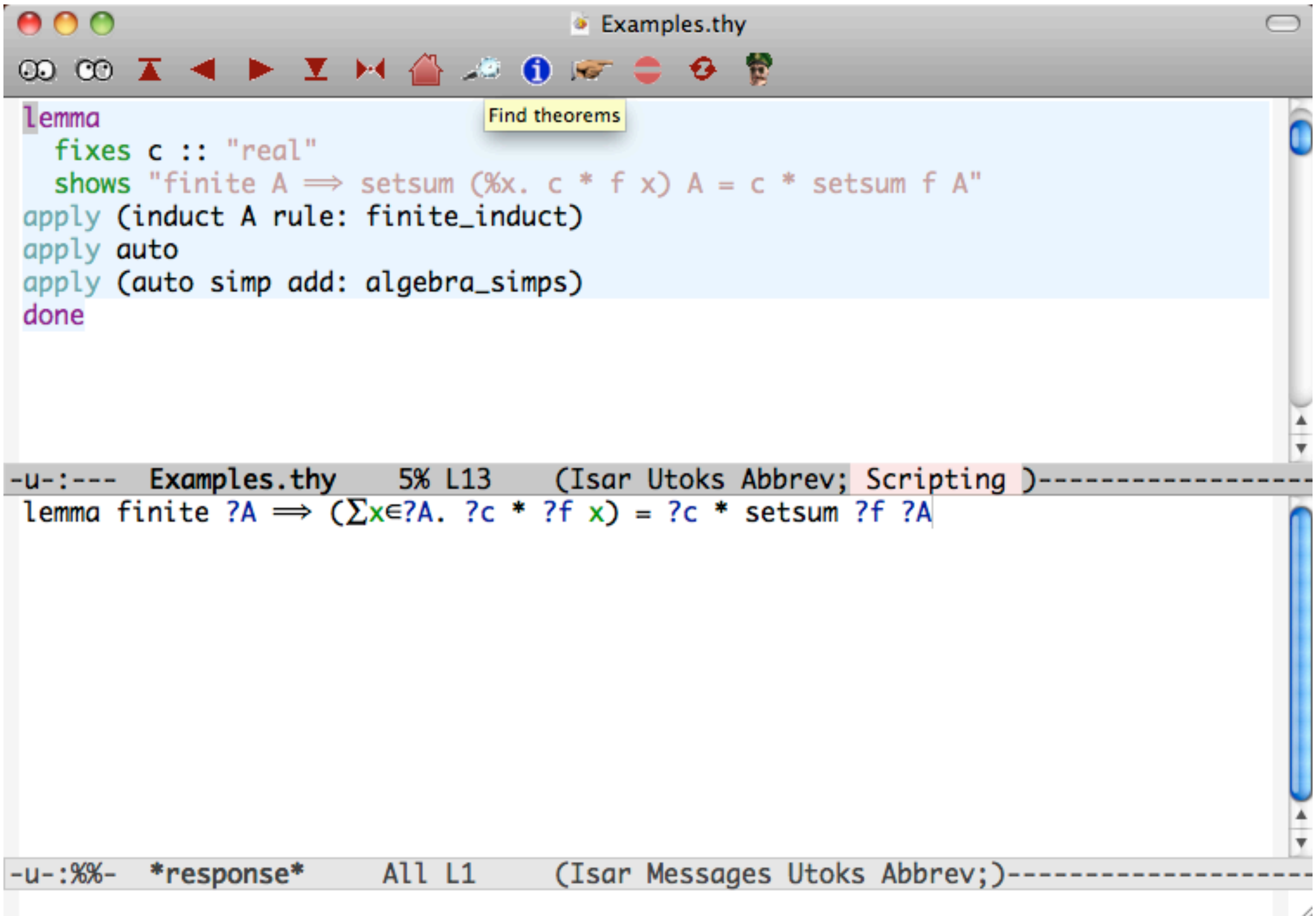
Proving Theorems about Sets

- It is not practical to learn all the built-in lemmas.
- Instead, try an automatic proof method:
  - `auto`
  - `force`
  - `blast`

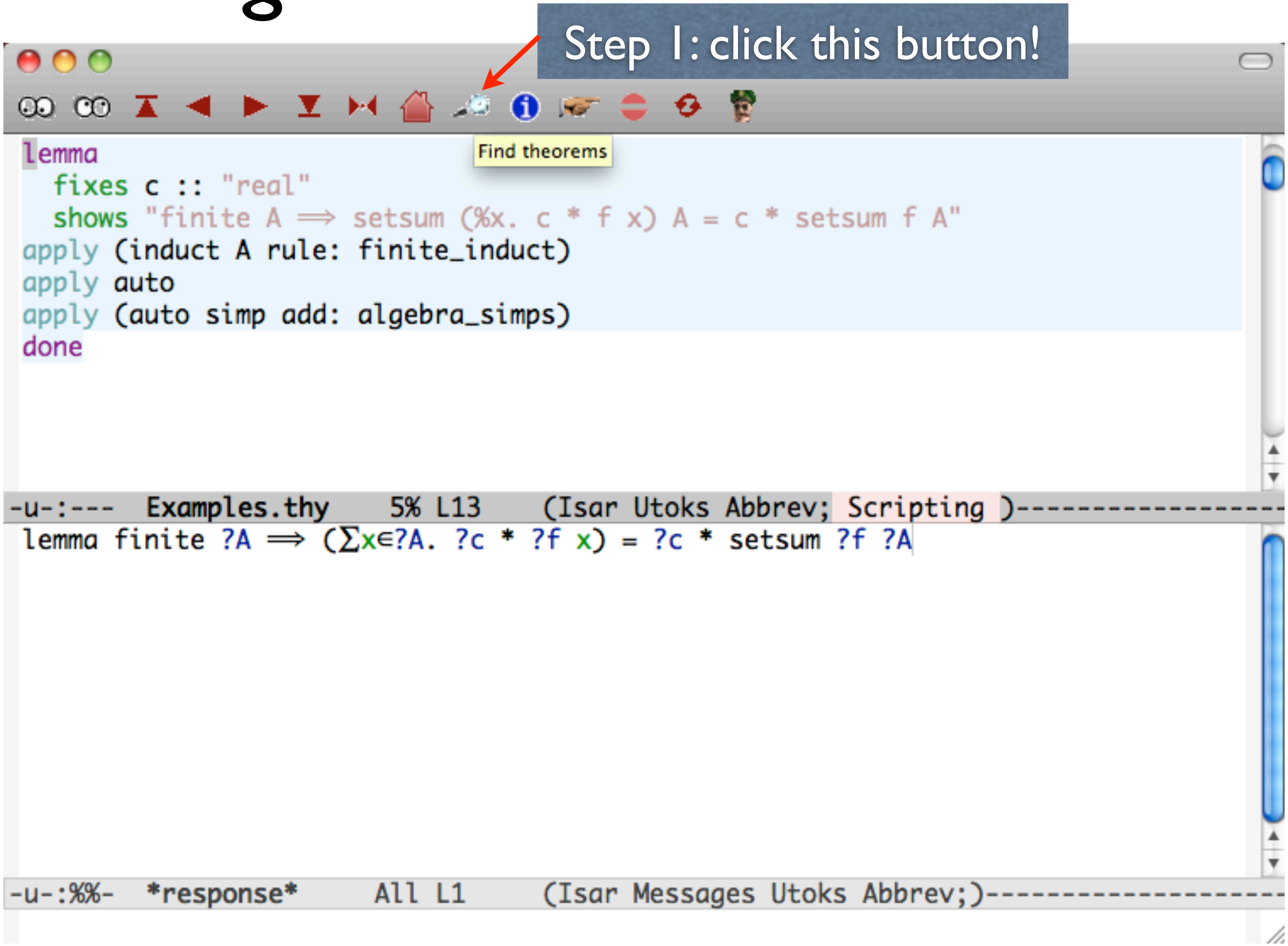
Proving Theorems about Sets

- It is not practical to learn all the built-in lemmas.
- Instead, try an automatic proof method:
  - `auto`
  - `force`
  - `blast`
- Each uses the built-in library, comprising hundreds of facts, with powerful heuristics.

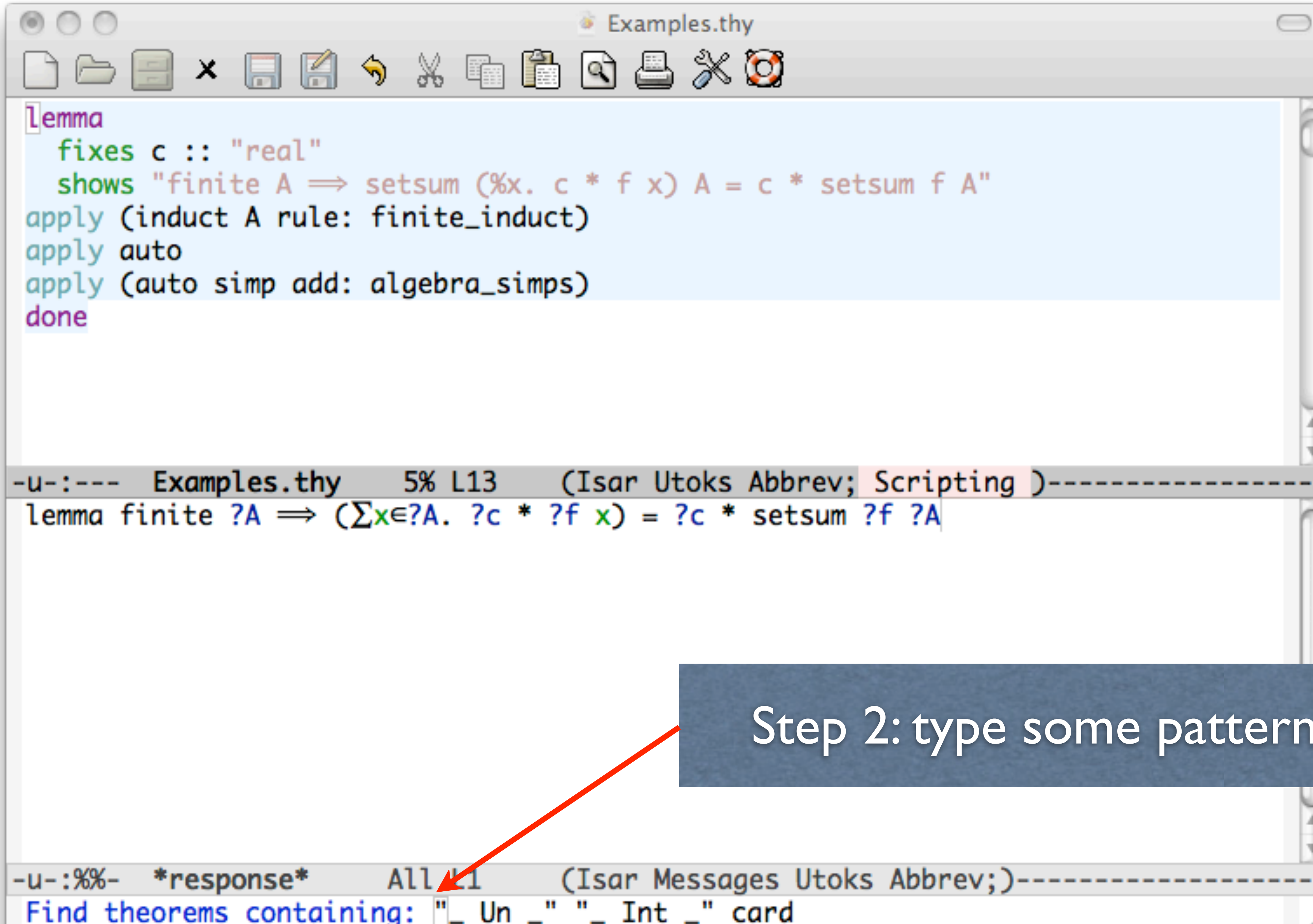
Finding Theorems about Sets



Finding Theorems about Sets



Finding Theorems about Sets



Step 2: type some patterns

What Theorems Were Found?

```
*response*

searched for:
    "_ ∪ _"
    "_ ∩ _"
    "card"

found 2 theorems in 0.120 secs:

Finite_Set.card_Un_Int:
  ⟦finite ?A; finite ?B⟧
  ⟹ card ?A + card ?B = card (?A ∪ ?B) + card (?A ∩ ?B)
Finite_Set.card_Un_disjoint:
  ⟦finite ?A; finite ?B; ?A ∩ ?B = {}⟧ ⟹ card (?A ∪ ?B) = card ?A + card ?B

-u-:%%-  *response*     All L2     (Isar Messages Utoks Abbrev;)--------------------
```