

Isabelle Tutorial 2: Inductively Defined Sets

Lawrence C Paulson
Computer Laboratory
University of Cambridge

Set Theory Primitives

- The type α `set`, which is similar to $\alpha \Rightarrow \text{bool}$
- The membership relation: $\in$
- The subset relation: $\subseteq$
 - Reflexive, anti-symmetric, transitive
- The empty set: $\{\}$
- The universal set: UNIV

Basic Set Theory Operations

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

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

$$e \in -A \iff e \notin A$$

$$e \in A \cup B \iff e \in A \vee e \in B$$

$$e \in A \cap B \iff e \in A \wedge e \in B$$

$$e \in \text{Pow}(A) \iff e \subseteq A$$

Big Union and Intersection

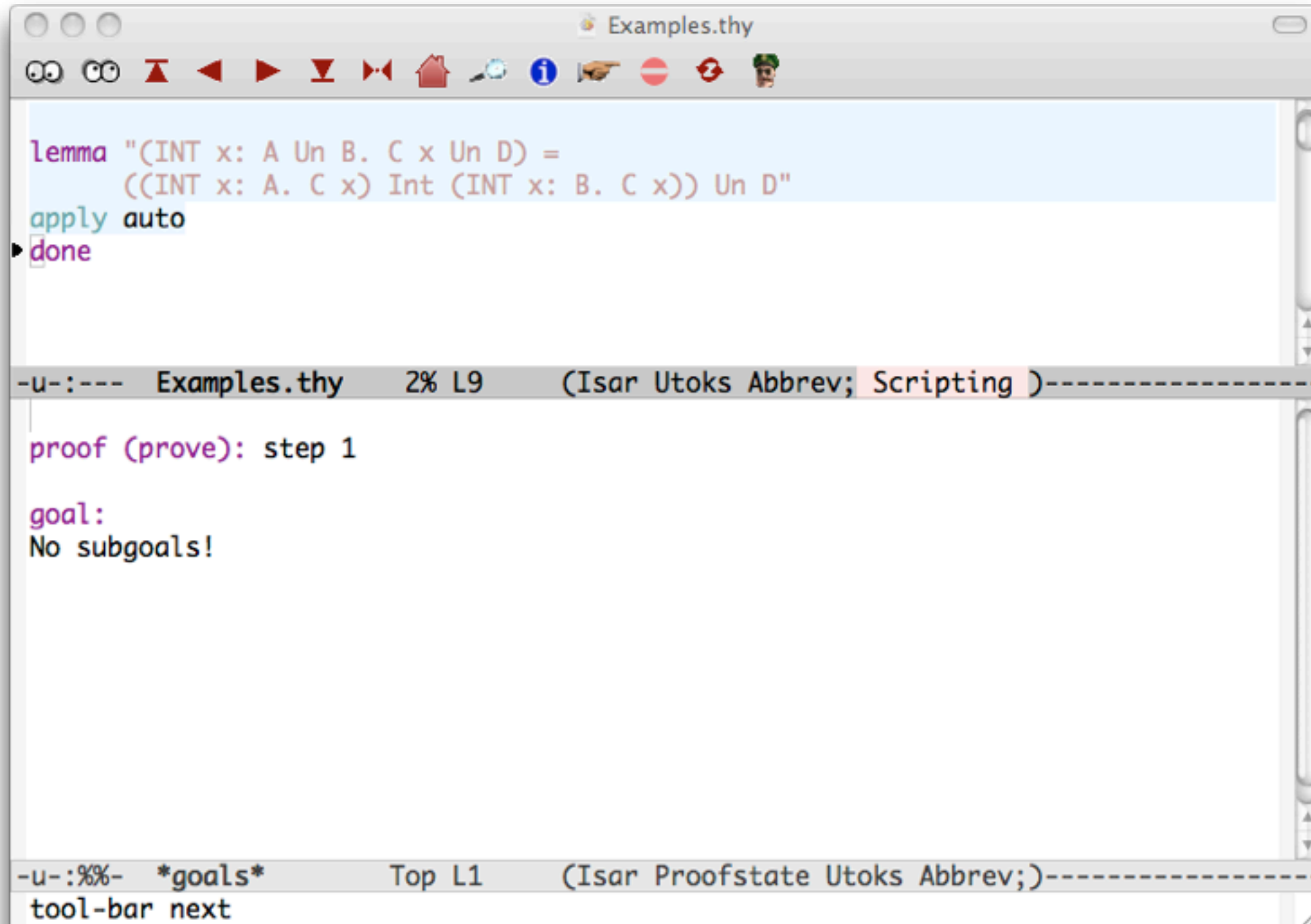
$$e \in \left(\bigcup x. B(x) \right) \iff \exists x. e \in B(x)$$

$$e \in \left(\bigcup_{x \in A} B(x) \right) \iff \exists x \in A. e \in B(x)$$

$$e \in \bigcup A \iff \exists x \in A. e \in x$$

And the analogous forms of intersections...

A Simple Set Theory Proof



The screenshot shows the Isabelle/Isar IDE interface. The main editor window displays a lemma and its proof. The lemma states that the union of two sets, A and B, intersected with a set C, is equal to the union of the intersection of A and C, and the intersection of B and C. The proof is currently at the first step, and the goal is to prove the lemma. The status bar at the bottom indicates the current position in the proof and the available toolbars.

```
lemma "(INT x: A Un B. C x Un D) =  
      ((INT x: A. C x) Int (INT x: B. C x)) Un D"  
apply auto  
done
```

-u-:--- Examples.thy 2% L9 (Isar Utoks Abbrev; Scripting)-----

```
proof (prove): step 1  
goal:  
No subgoals!
```

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

Finite Set Notation

$$\{a_1, \dots, a_n\} = \text{insert } a_1 \text{ (... (insert } a_n \{\})...)$$

where

$$x \in \text{insert } a \ B \iff x=a \vee x \in B$$

Finite Sets

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

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

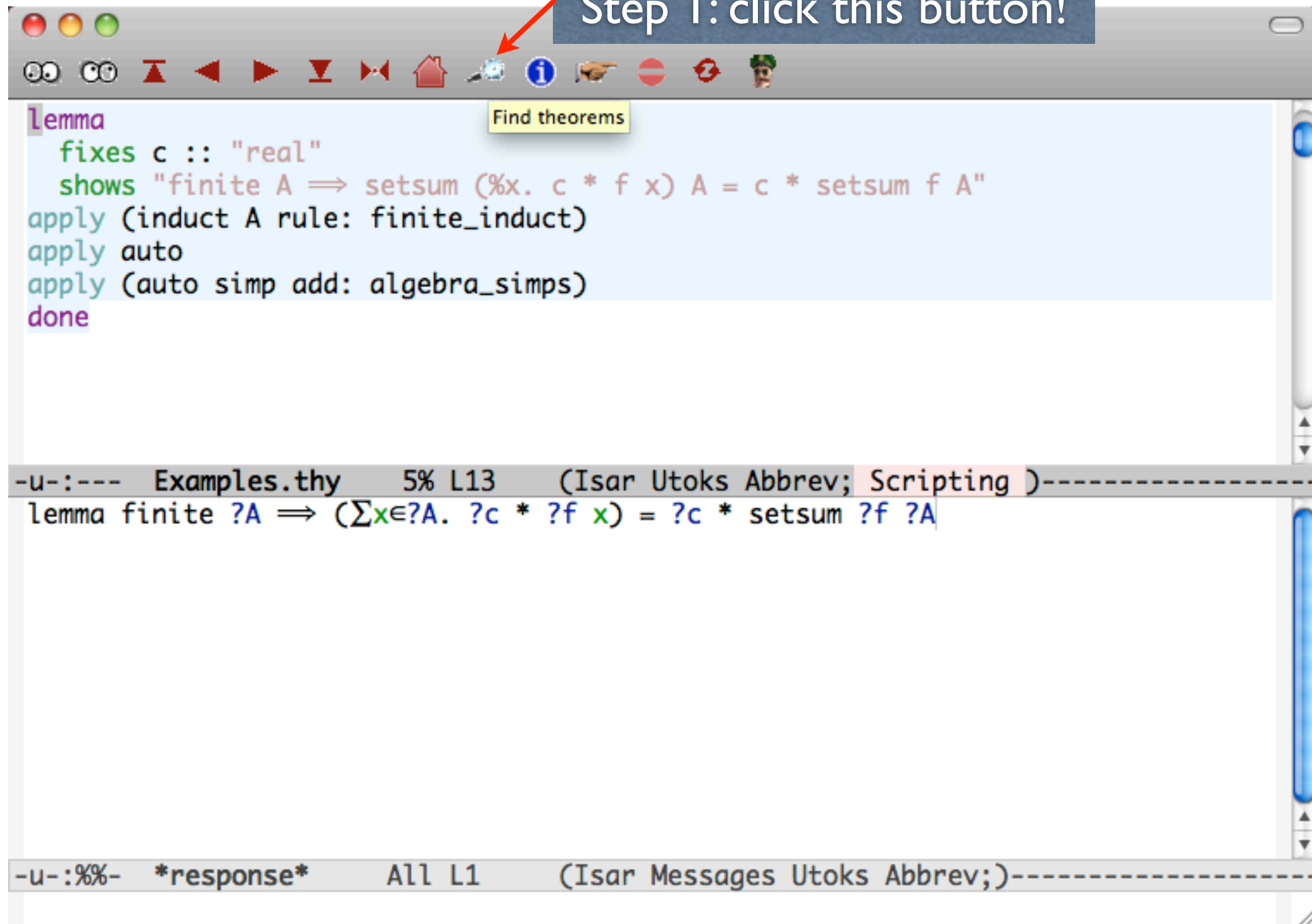
$$\text{finite } A \implies \text{card}(\text{Pow } A) = 2^{\text{card } A}$$

Proving Theorems about Sets

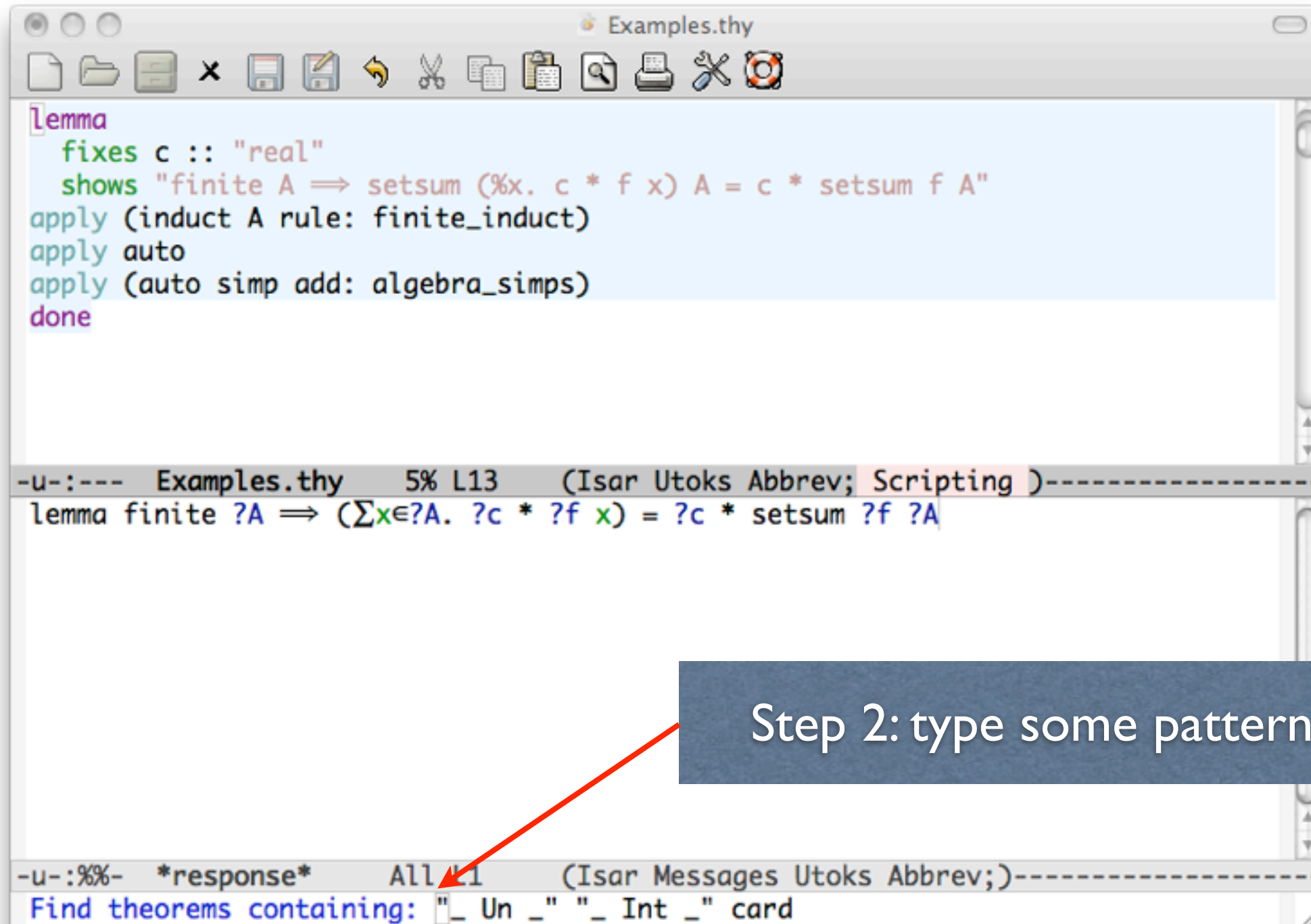
- It is not practical to learn all the built-in lemmas.
- Instead, try an automatic proof method:
 - `auto`
 - `force`
 - `blast`
- Each uses the built-in library, comprising hundreds of facts, with powerful heuristics.

Finding Theorems about Sets

Step 1: click this button!



Finding Theorems about Sets



The screenshot shows the Isabelle/Isar IDE interface. The main editor window, titled 'Examples.thy', contains a lemma proof:

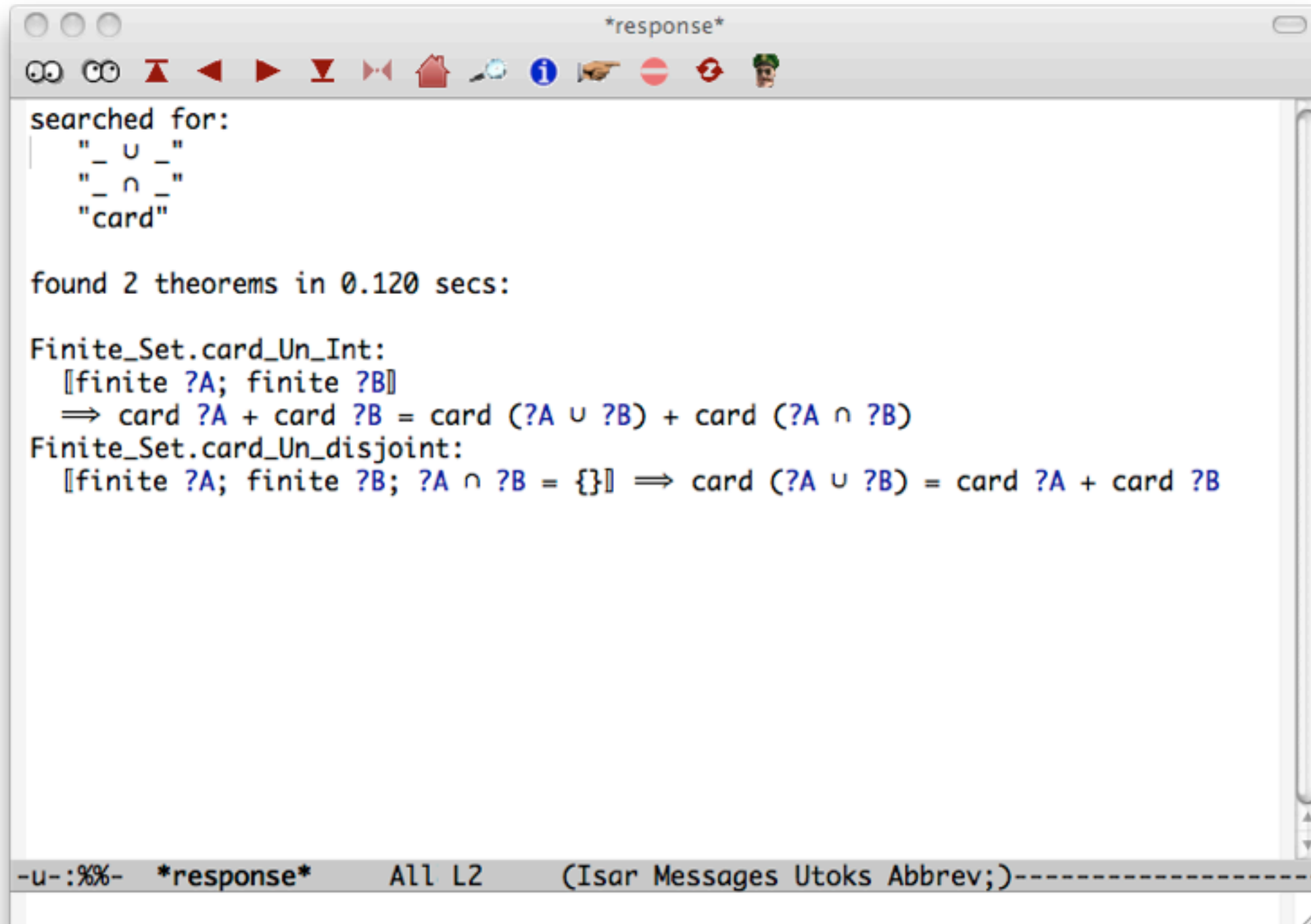
```
lemma
  fixes c :: "real"
  shows "finite A  $\Rightarrow$  setsum (%x. c * f x) A = c * setsum f A"
apply (induct A rule: finite_induct)
apply auto
apply (auto simp add: algebra_simps)
done
```

Below the editor is a status bar showing the current line and column: '-u-:--- Examples.thy 5% L13 (Isar Utoks Abbrev; Scripting)-----'. Below this is a search bar with the text: 'lemma finite ?A $\Rightarrow$ ($\sum x \in ?A. ?c * ?f x$) = ?c * setsum ?f ?A'.

At the bottom of the IDE is a search results window titled '-u-:%%- *response* All 11 (Isar Messages Utoks Abbrev;)-----'. It contains the text: 'Find theorems containing: "_ Un _" "_ Int _" card'. A red arrow points from the search results window to the search bar.

Step 2: type some patterns

Which Theorems Were Found?



The screenshot shows a window titled `*response*` with a toolbar containing icons for search, navigation, and other functions. The main text area displays the following content:

```
searched for:
  " _ u _ "
  " _ n _ "
  "card"

found 2 theorems in 0.120 secs:

Finite_Set.card_Un_Int:
  [[finite ?A; finite ?B]
    $\Rightarrow$  card ?A + card ?B = card (?A  $\cup$  ?B) + card (?A  $\cap$  ?B)
Finite_Set.card_Un_disjoint:
  [[finite ?A; finite ?B; ?A  $\cap$  ?B = {}]]  $\Rightarrow$  card (?A  $\cup$  ?B) = card ?A + card ?B
```

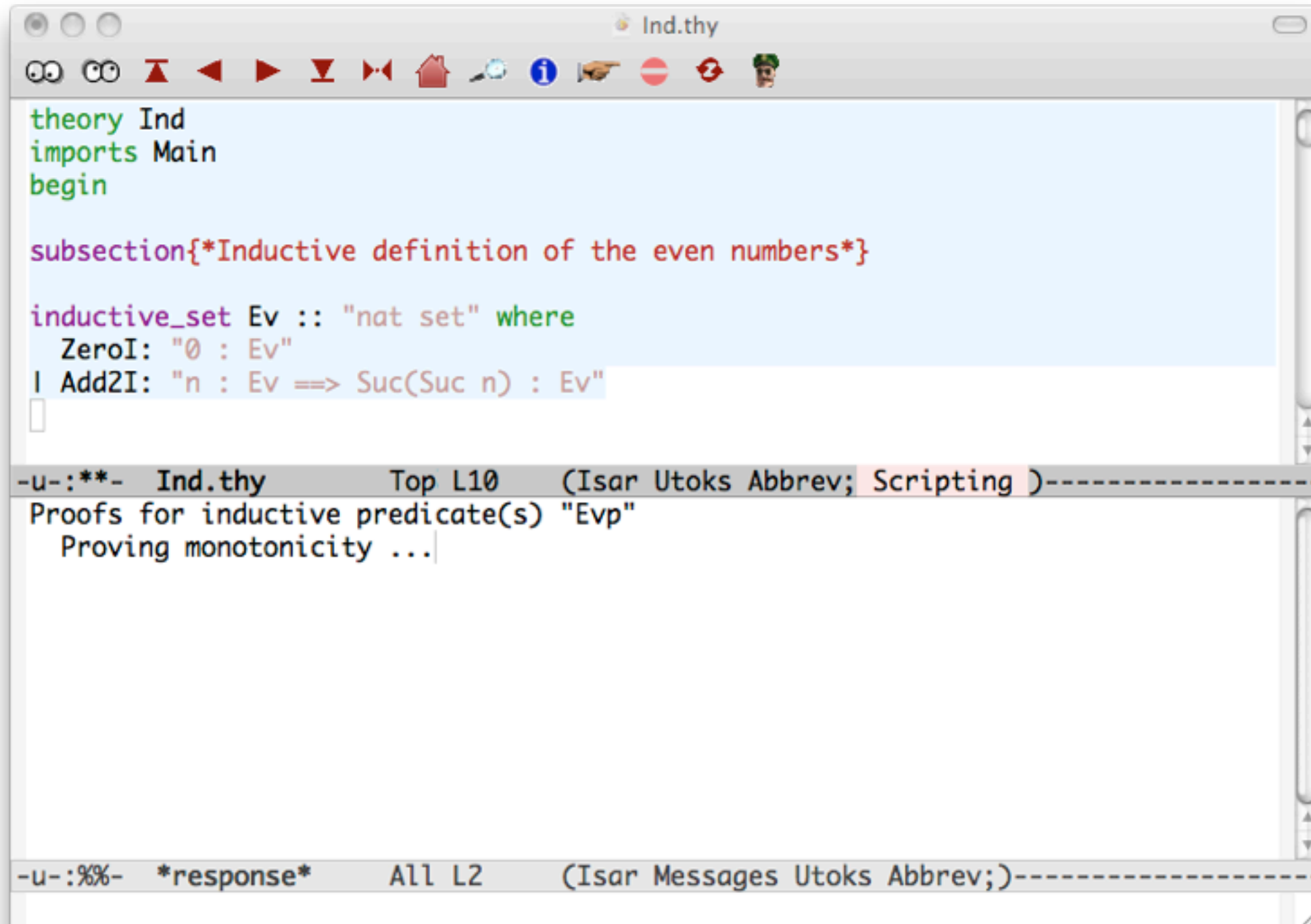
The status bar at the bottom shows the command `-u-:%%- *response* All L2 (Isar Messages Utoks Abbrev;)`.

Inductively Defined Sets

Defining a Set Inductively

- The set of even numbers is the least set such that
 - 0 is even.
 - If n is even, then $n+2$ is even.
- These can be viewed as *introduction rules*.
- We get an *induction principle* to express that no other numbers are even.
- Induction is used throughout mathematics, and to express the semantics of programming languages.

Inductive Definitions in Isabelle



The screenshot shows the Isabelle IDE interface. The main window displays the following code:

```
theory Ind
imports Main
begin

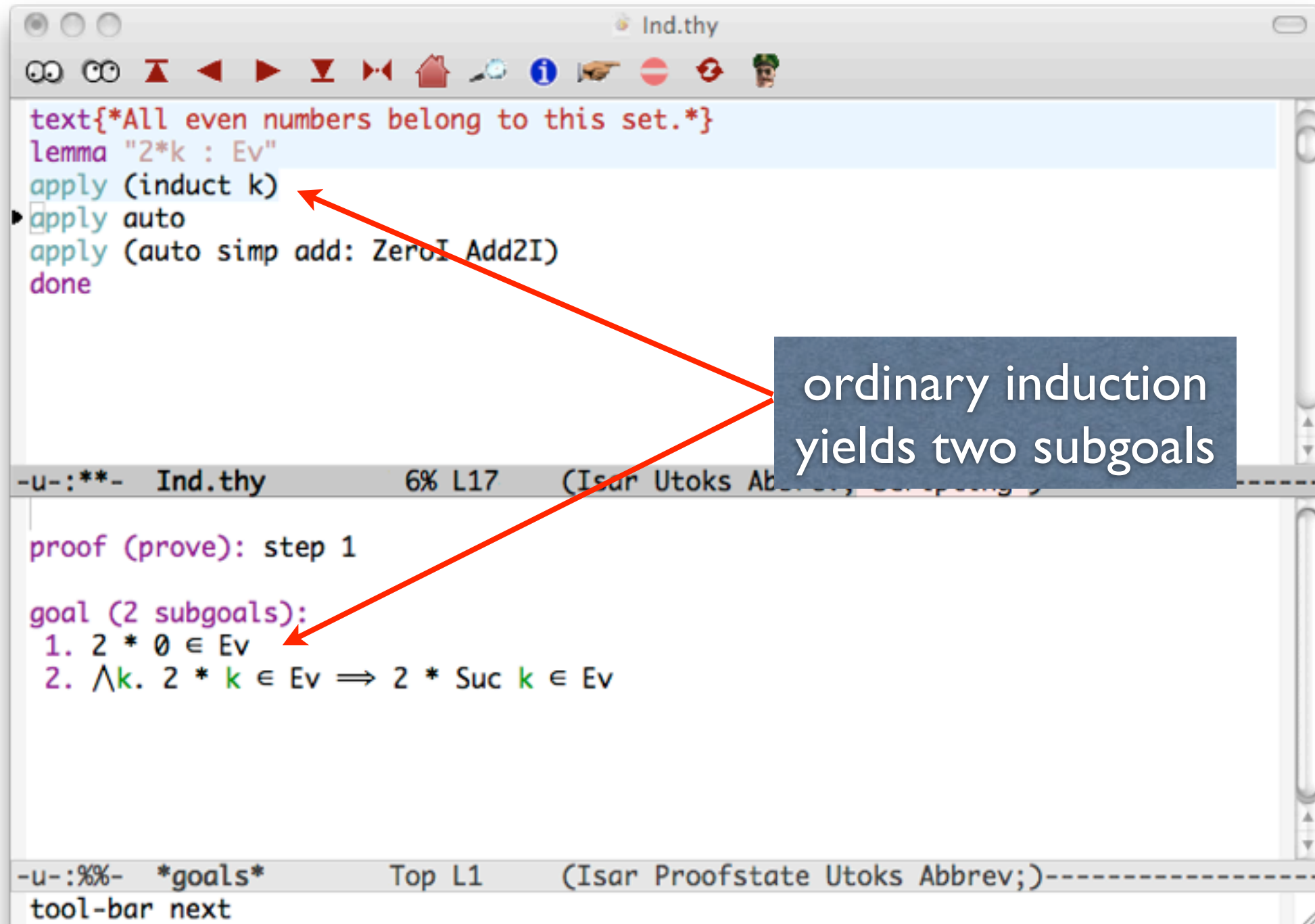
subsection{*Inductive definition of the even numbers*}

inductive_set Ev :: "nat set" where
  ZeroI: "0 : Ev"
| Add2I: "n : Ev ==> Suc(Suc n) : Ev"

```

Below the code editor, there are two panels. The top panel shows the current location in the document: `-u-:**- Ind.thy Top L10 (Isar Utoks Abbrev; Scripting )-----`. The bottom panel shows the current response: `-u-:%%- *response* All L2 (Isar Messages Utoks Abbrev; )-----`.

Even Numbers Belong to Ev



```
Ind.thy
text{*All even numbers belong to this set.*}
lemma "2*k : Ev"
  apply (induct k)
  apply auto
  apply (auto simp add: ZeroI Add2I)
  done

proof (prove): step 1
goal (2 subgoals):
1. 2 * 0 ∈ Ev
2.  $\wedge k. 2 * k \in \text{Ev} \Rightarrow 2 * \text{Suc } k \in \text{Ev}$ 
```

ordinary induction yields two subgoals

-u-:***- Ind.thy 6% L17 (Isar Utoks Abbrev;)

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

tool-bar next

Proving Set Membership

```
Ind.thy
text{*All even numbers belong to this set.*}
lemma "2*k : Ev"
  apply (induct k)
  apply auto
  apply (auto simp add: ZeroI Add2I)
  done

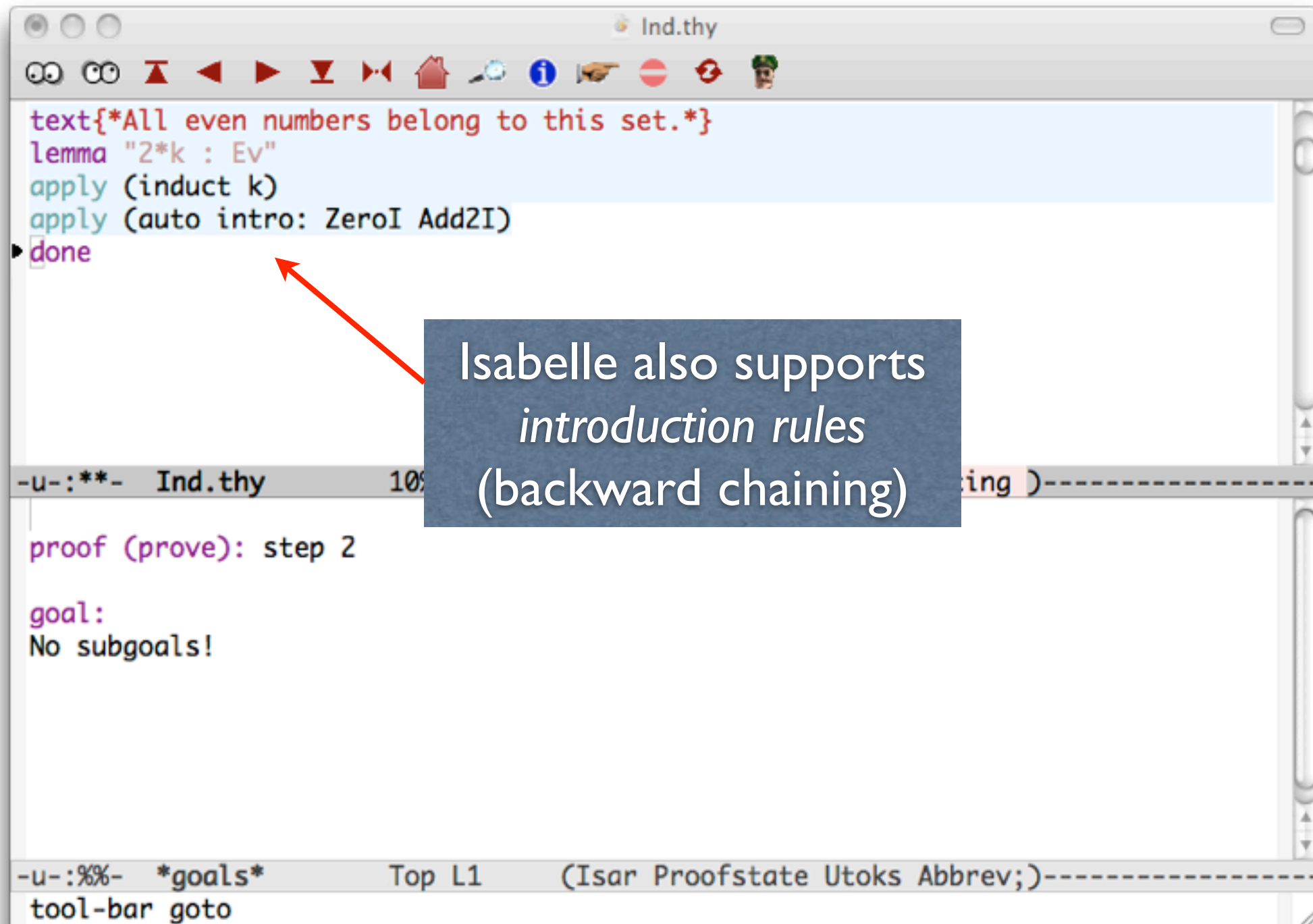
-u-:**- Ind.thy 6% L18

proof (prove): step 2
goal (2 subgoals):
  1. 0 ∈ Ev
  2.  $\wedge k. 2 * k \in \text{Ev} \Rightarrow \text{Suc} (\text{Suc} (2 * k)) \in \text{Ev}$ 

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

after simplification, the subgoals resemble the introduction rules

Finishing the Proof



```
text{*All even numbers belong to this set.*}
lemma "2*k : Ev"
apply (induct k)
apply (auto intro: ZeroI Add2I)
done

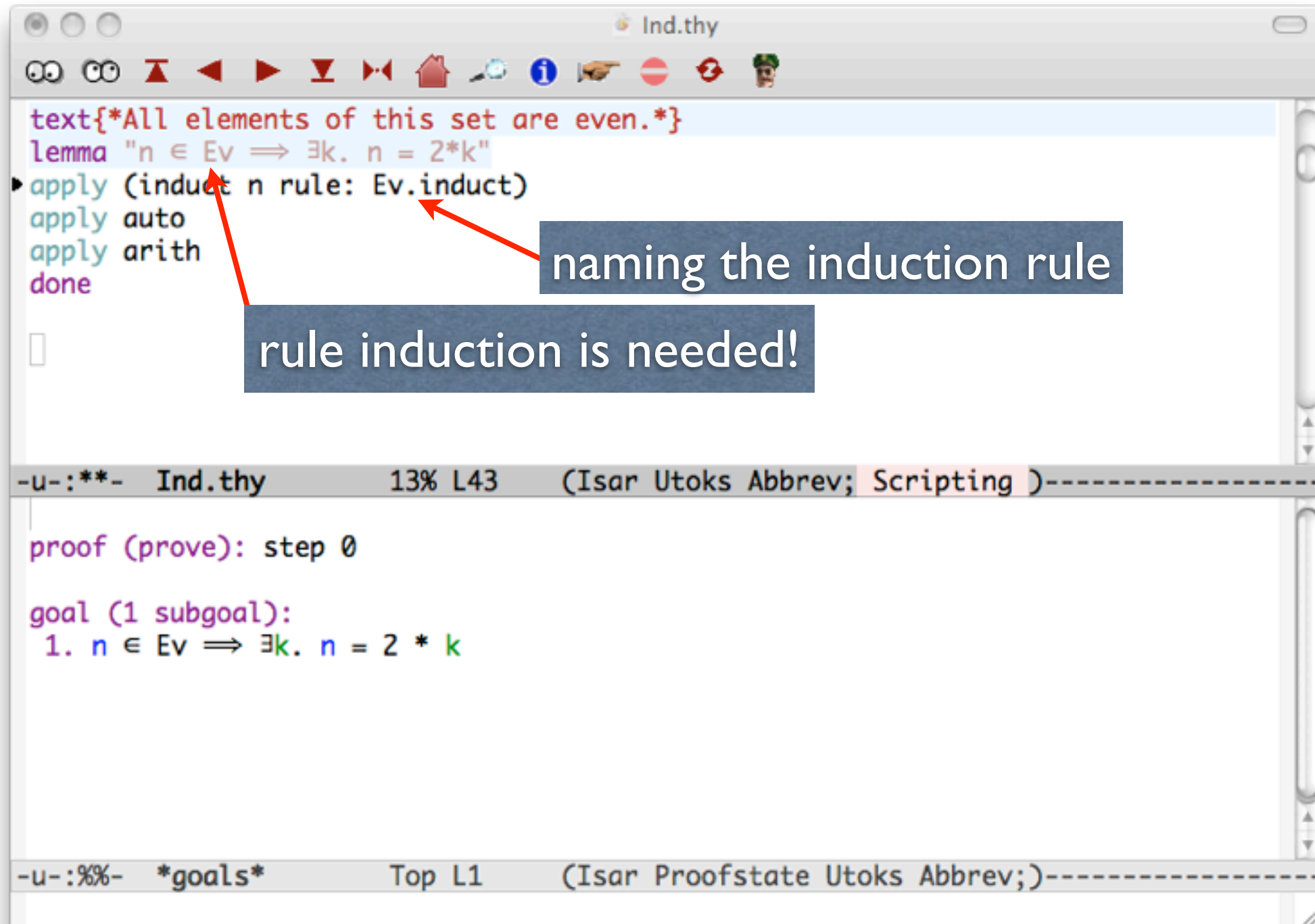
proof (prove): step 2
goal:
No subgoals!
```

Isabelle also supports
introduction rules
(backward chaining)

Rule Induction

- Proving something about *every* element of the set.
- It expresses that the inductive set is *minimal*.
- It is sometimes called “induction on derivations”
- There is a *base case* for every non-recursive introduction rule
- ...and an *inductive step* for the other rules.

Ev Has only Even Numbers



The screenshot shows a theorem prover interface with a file named 'Ind.thy'. The main editor contains the following code:

```
text{*All elements of this set are even.*}  
lemma "n ∈ Ev ⇒ ∃k. n = 2*k"  
• apply (induct n rule: Ev.induct)  
  apply auto  
  apply arith  
done
```

Two red arrows point from text boxes to the code. One arrow points to 'Ev.induct' with the text 'naming the induction rule'. The other arrow points to '(induct n rule: Ev.induct)' with the text 'rule induction is needed!'. Below the main editor, there is a status bar showing '-u-:**- Ind.thy 13% L43 (Isar Utoks Abbrev; Scripting)'. Below that, there is a goal window showing:

```
proof (prove): step 0  
goal (1 subgoal):  
1. n ∈ Ev ⇒ ∃k. n = 2 * k
```

The goal window has a status bar showing '-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)'. The interface includes a toolbar with various icons for navigation and editing.

An Example of Rule Induction

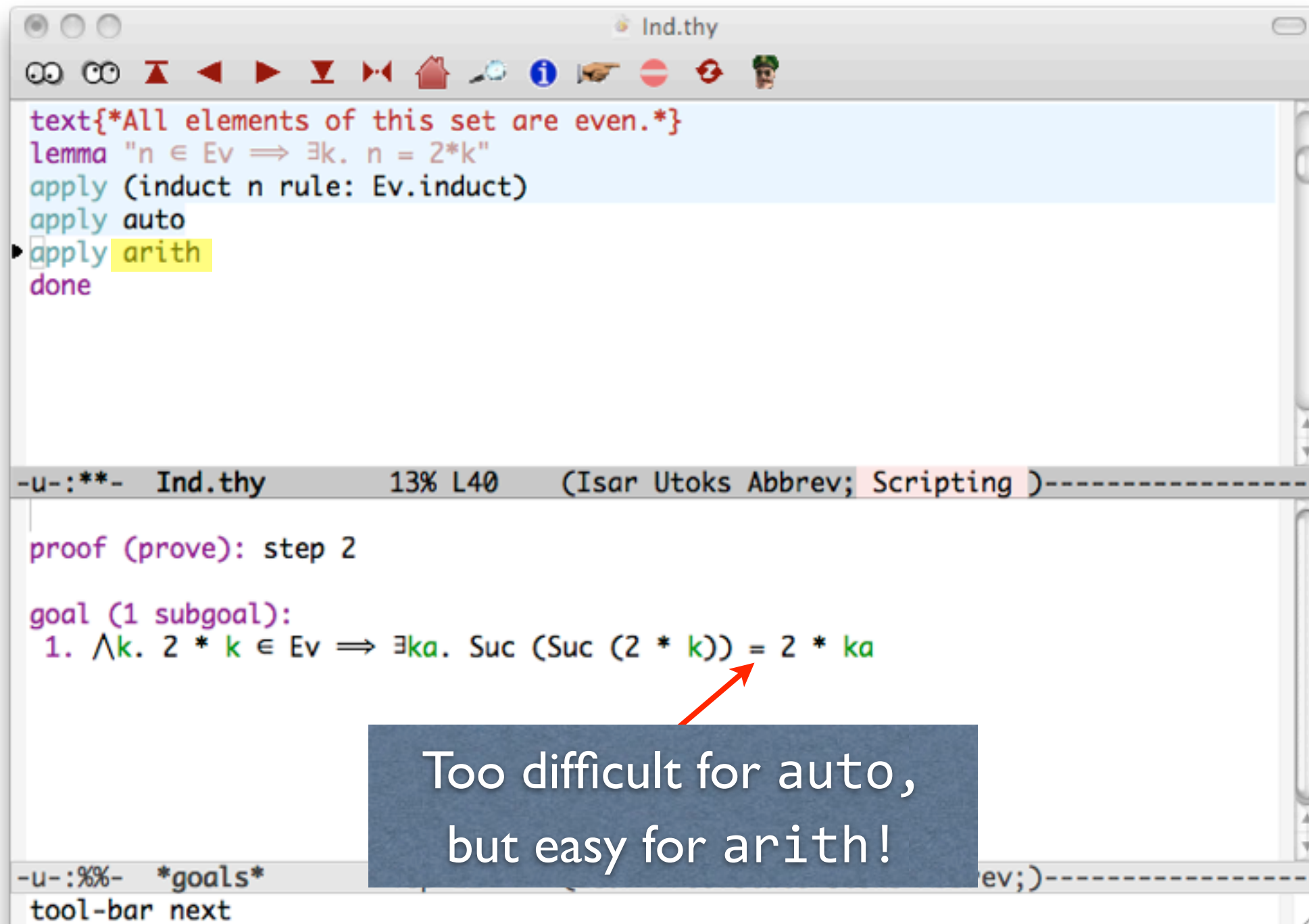
```
Ind.thy
text{*All elements of this set are even.*}
lemma "n ∈ Ev ⇒ ∃k. n = 2*k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done

-u-:**- Ind.thy 13% L39 (Isar Utoks Abbrev; Scripting )-----
proof (prove): step 1
goal (2 subgoals):
1. ∃k. 0 = 2 * k
2. ∧n. [n ∈ Ev; ∃k. n = 2 * k] ⇒ ∃k. Suc (Suc n) = 2 * k
-u-:%%-
tool-bar next
```

base case: n replaced by 0

induction step: n replaced by $\text{Suc} (\text{Suc } n)$

One Tricky Goal Left!



The screenshot shows the Isabelle/Isar IDE interface. The top window, titled 'Ind.thy', contains a proof script. The bottom window shows the current goal state. A red arrow points from a text box to the goal expression.

```
text{*All elements of this set are even.*}
lemma "n ∈ Ev ⇒ ∃k. n = 2*k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done
```

proof (prove): step 2

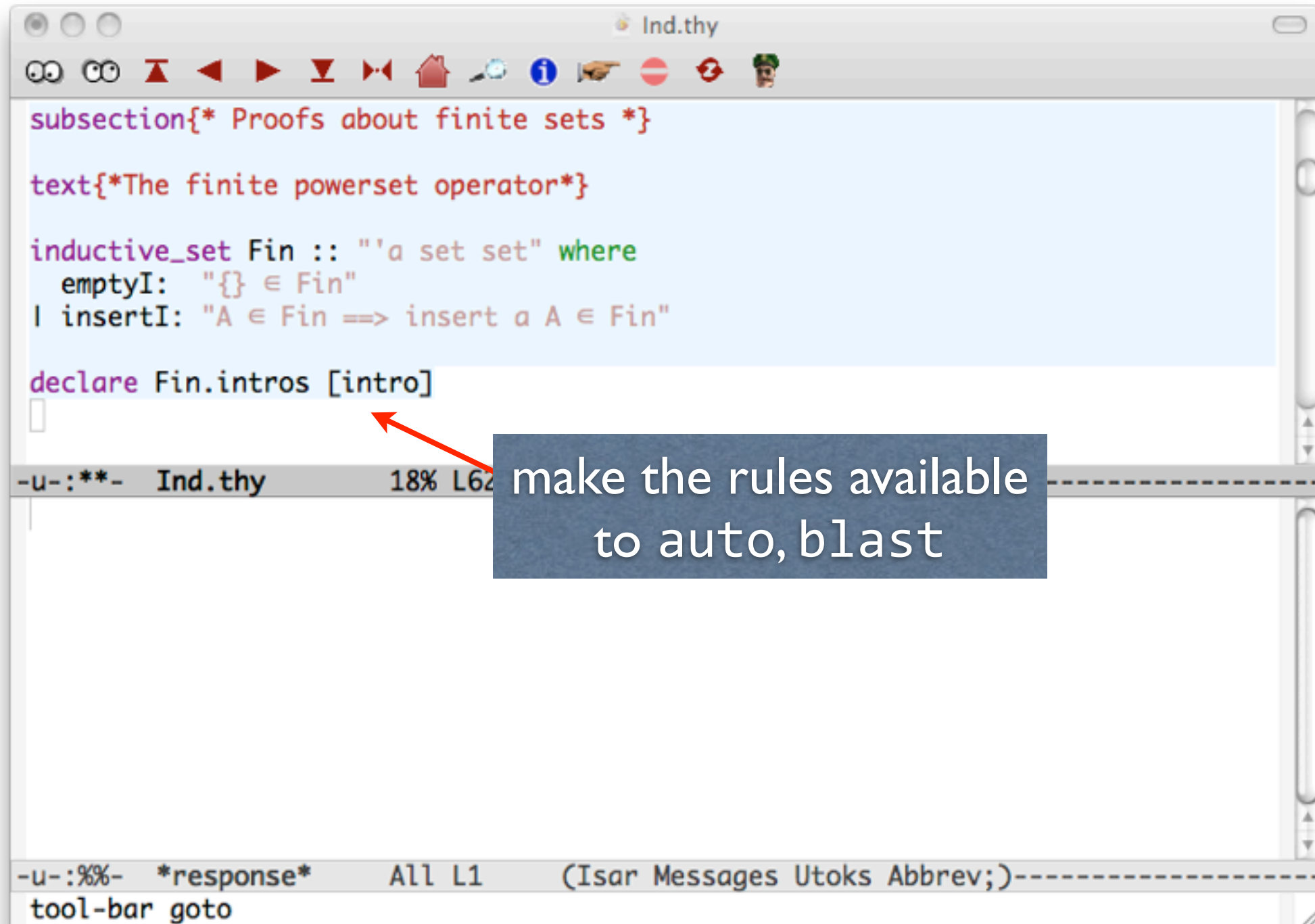
goal (1 subgoal):

1. $\wedge k. 2 * k \in \text{Ev} \Rightarrow \exists ka. \text{Suc} (\text{Suc} (2 * k)) = 2 * ka$

Too difficult for auto,
but easy for arith!

A Final Example

Defining Finiteness



The screenshot shows a theorem prover interface with a file named `Ind.thy`. The main text area contains the following code:

```
subsection{* Proofs about finite sets *}

text{*The finite powerset operator*}

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

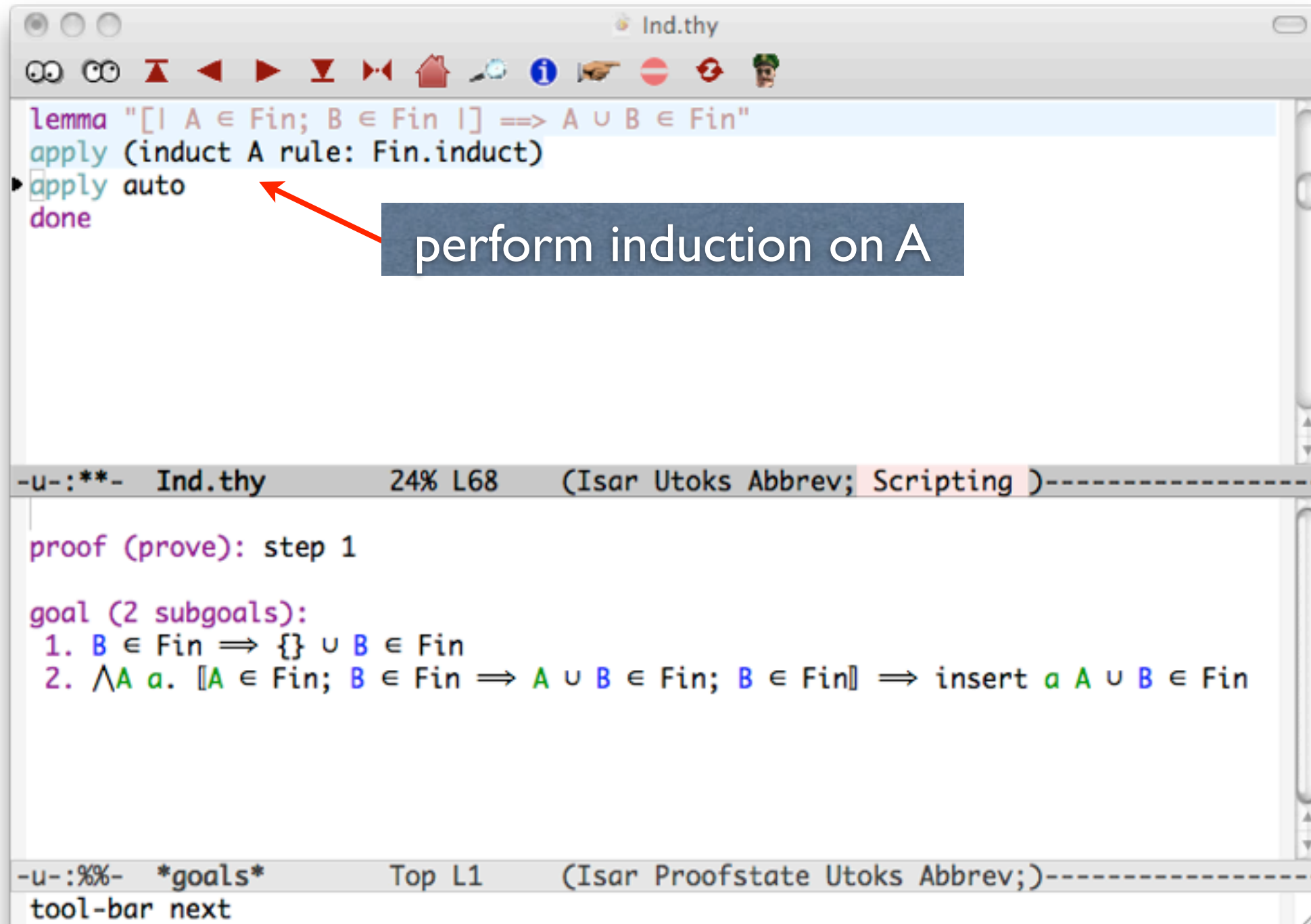
declare Fin.intros [intro]
```

A red arrow points from a text box to the `declare` line. The text box contains the text "make the rules available to auto, blast".

The status bar at the bottom shows the following information:

- `-u-:**-`
- `Ind.thy`
- `18% L62`
- `*response*`
- `All L1`
- `(Isar Messages Utoks Abbrev;)`
- `tool-bar goto`

The Union of Two Finite Sets



The screenshot shows a theorem prover interface with a file named 'Ind.thy'. The main editor contains the following code:

```
lemma "[| A ∈ Fin; B ∈ Fin |] ==> A ∪ B ∈ Fin"
  apply (induct A rule: Fin.induct)
  apply auto
done
```

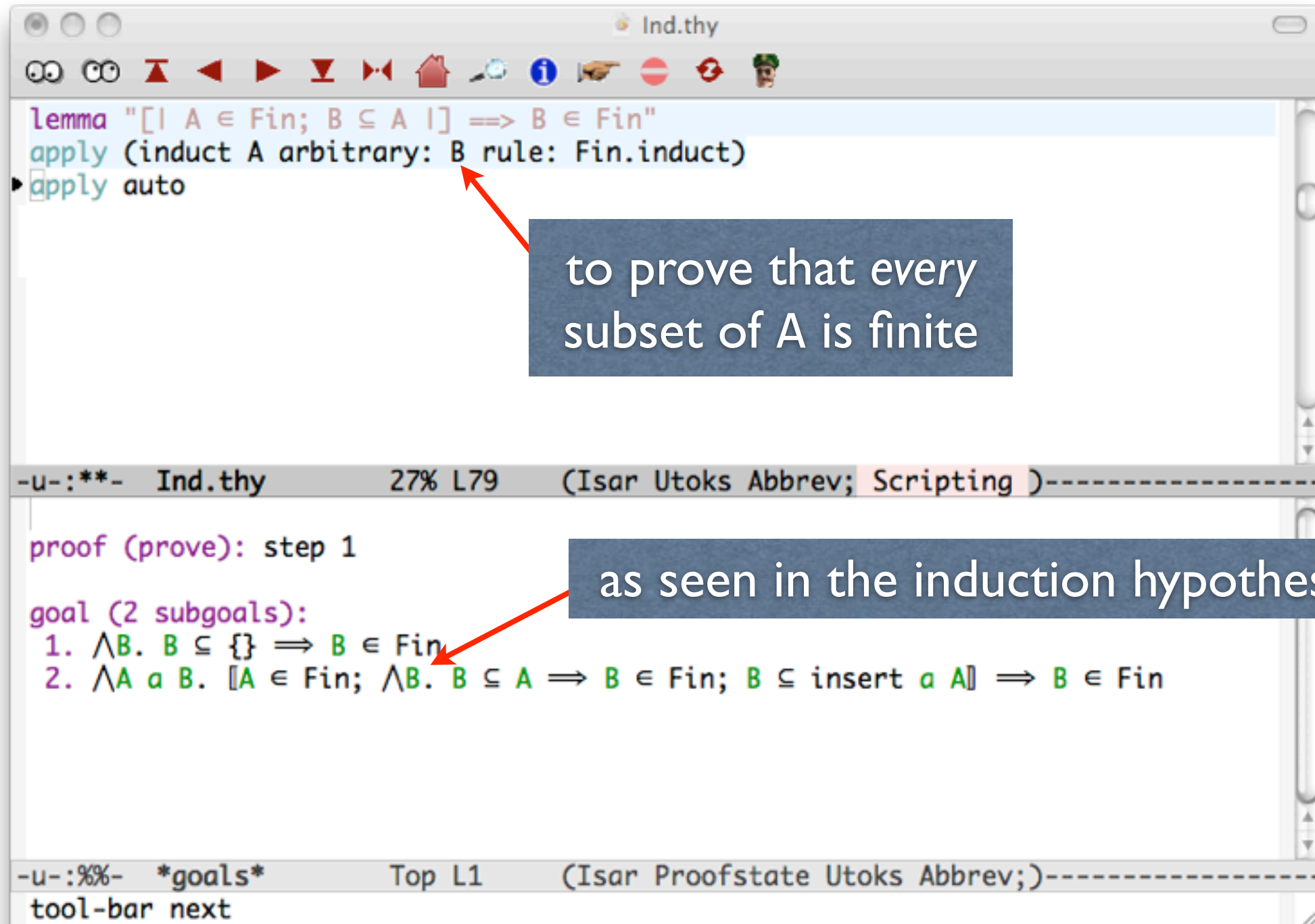
A red arrow points from the text 'perform induction on A' to the 'induct A' part of the second line of code.

The bottom of the interface shows a goal state with two subgoals:

```
proof (prove): step 1
goal (2 subgoals):
  1. B ∈ Fin ==> {} ∪ B ∈ Fin
  2. ∧A a. [| A ∈ Fin; B ∈ Fin ==> A ∪ B ∈ Fin; B ∈ Fin |] ==> insert a A ∪ B ∈ Fin
```

The bottom status bar indicates the current goal is 'Top L1' and provides a 'tool-bar next' button.

A Subset of a Finite Set



The screenshot shows a theorem prover interface with a file named `Ind.thy`. The main text area contains the following code:

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
  apply (induct A arbitrary: B rule: Fin.induct)
  apply auto
```

A red arrow points from the text box "to prove that every subset of A is finite" to the `rule: Fin.induct` part of the `apply` command.

Below the main text area, a status bar shows `-u-:**- Ind.thy 27% L79 (Isar Utoks Abbrev; Scripting )-----`.

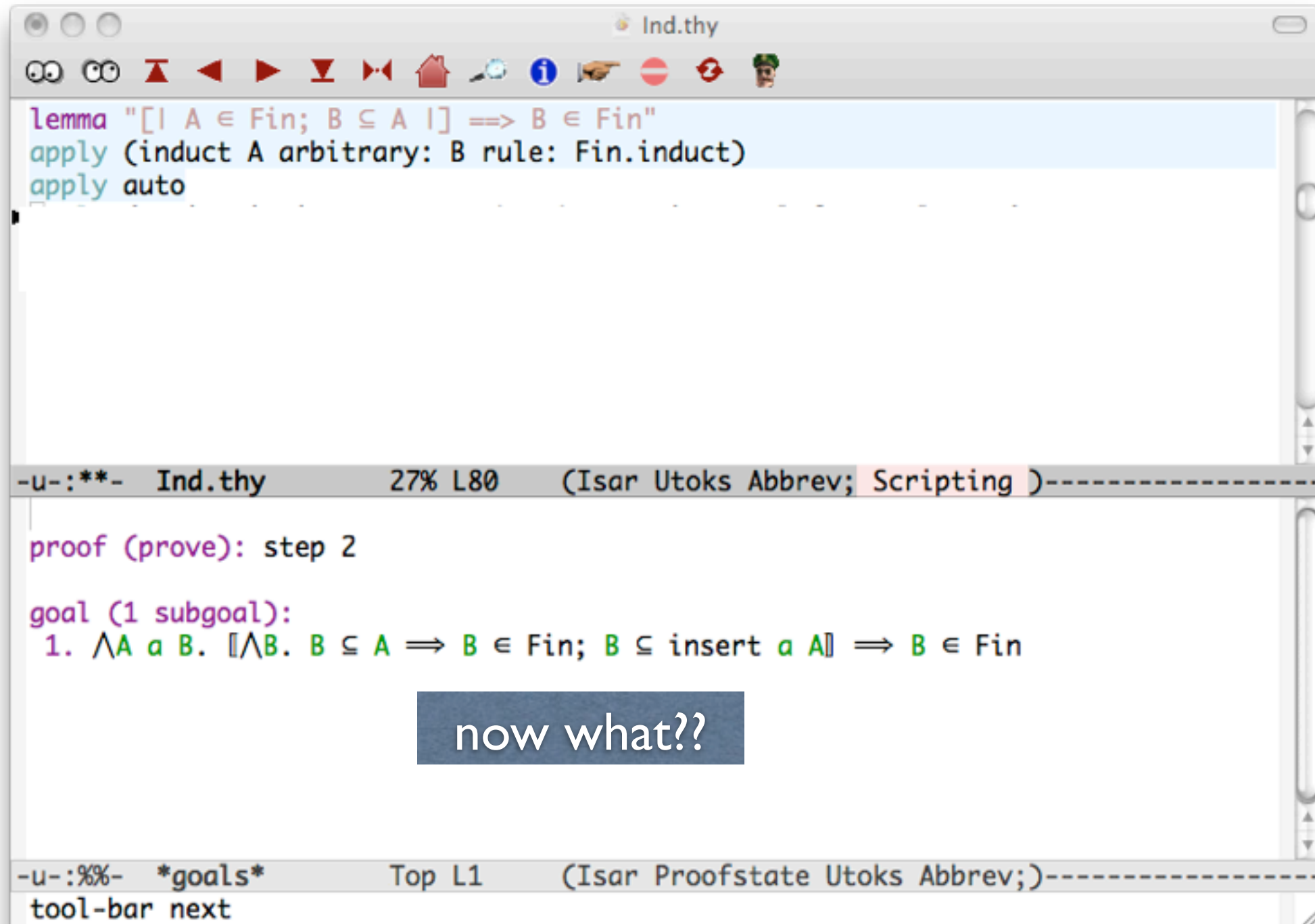
The proof steps are shown in a separate pane:

```
proof (prove): step 1
  goal (2 subgoals):
    1.  $\bigwedge B. B \subseteq \{\} \Rightarrow B \in \text{Fin}$ 
    2.  $\bigwedge A \ a \ B. [A \in \text{Fin}; \bigwedge B. B \subseteq A \Rightarrow B \in \text{Fin}; B \subseteq \text{insert } a \ A] \Rightarrow B \in \text{Fin}$ 
```

A red arrow points from the text box "as seen in the induction hypothesis" to the second subgoal, which represents the induction hypothesis.

At the bottom, another status bar shows `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----` and a `tool-bar next` button.

A Critical Point in the Proof



```
Ind.thy

lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
  apply (induct A arbitrary: B rule: Fin.induct)
  apply auto
  .
  .
  .

-u-:**- Ind.thy 27% L80 (Isar Utoks Abbrev; Scripting )-----

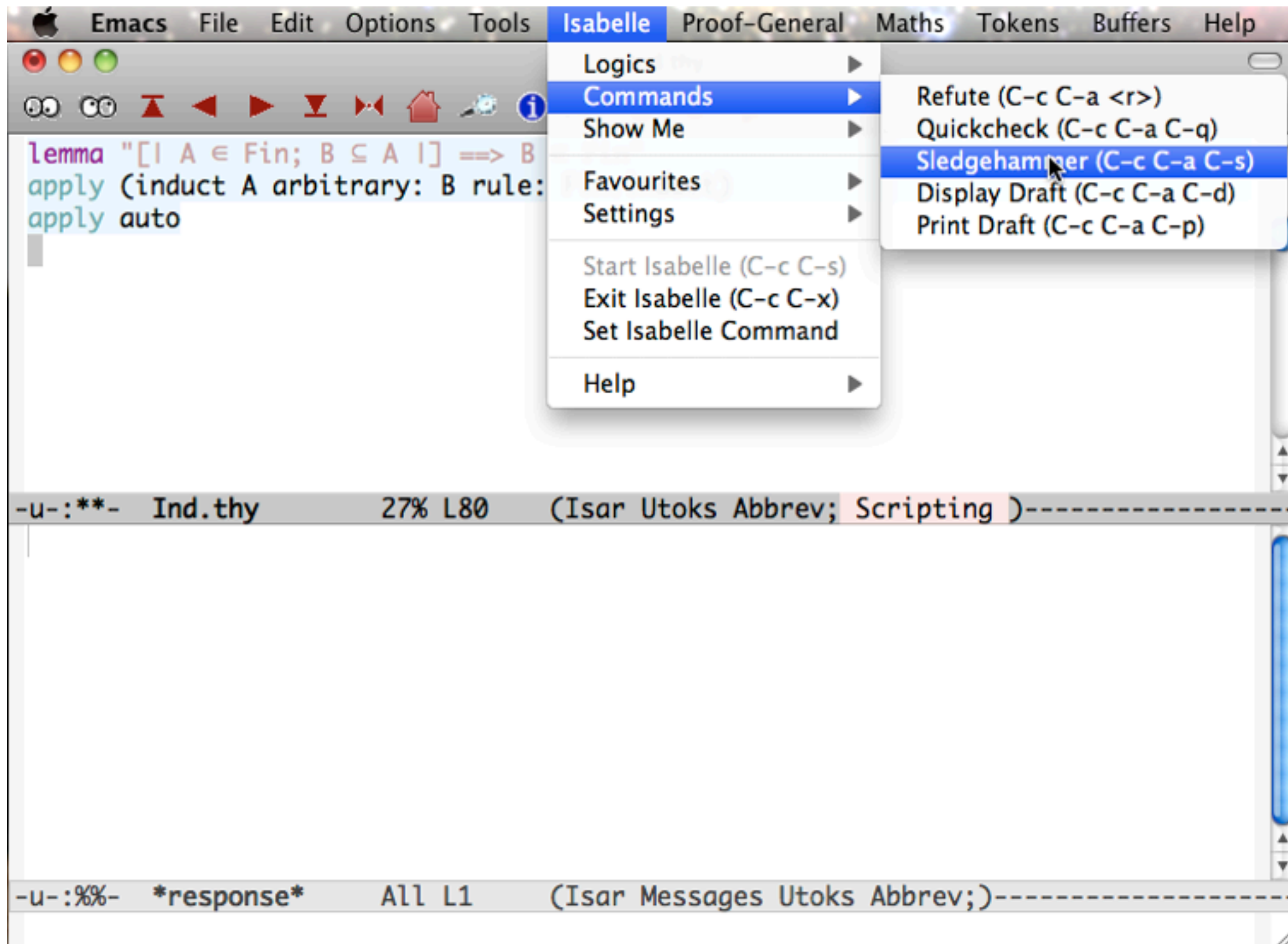
proof (prove): step 2
goal (1 subgoal):
  1.  $\bigwedge A\ a\ B. [\bigwedge B. B \subseteq A \implies B \in \text{Fin}; B \subseteq \text{insert } a\ A] \implies B \in \text{Fin}$ 

```

now what??

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

Time to Try Sledgehammer!



Success!

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
  apply (induct A arbitrary: B rule: Fin.induct)
  apply auto
  []

-u-:**- Ind.thy 27% L80 (Isar Utoks Abbrev)
Sledgehammer: external prover "spass" for subgoal 1

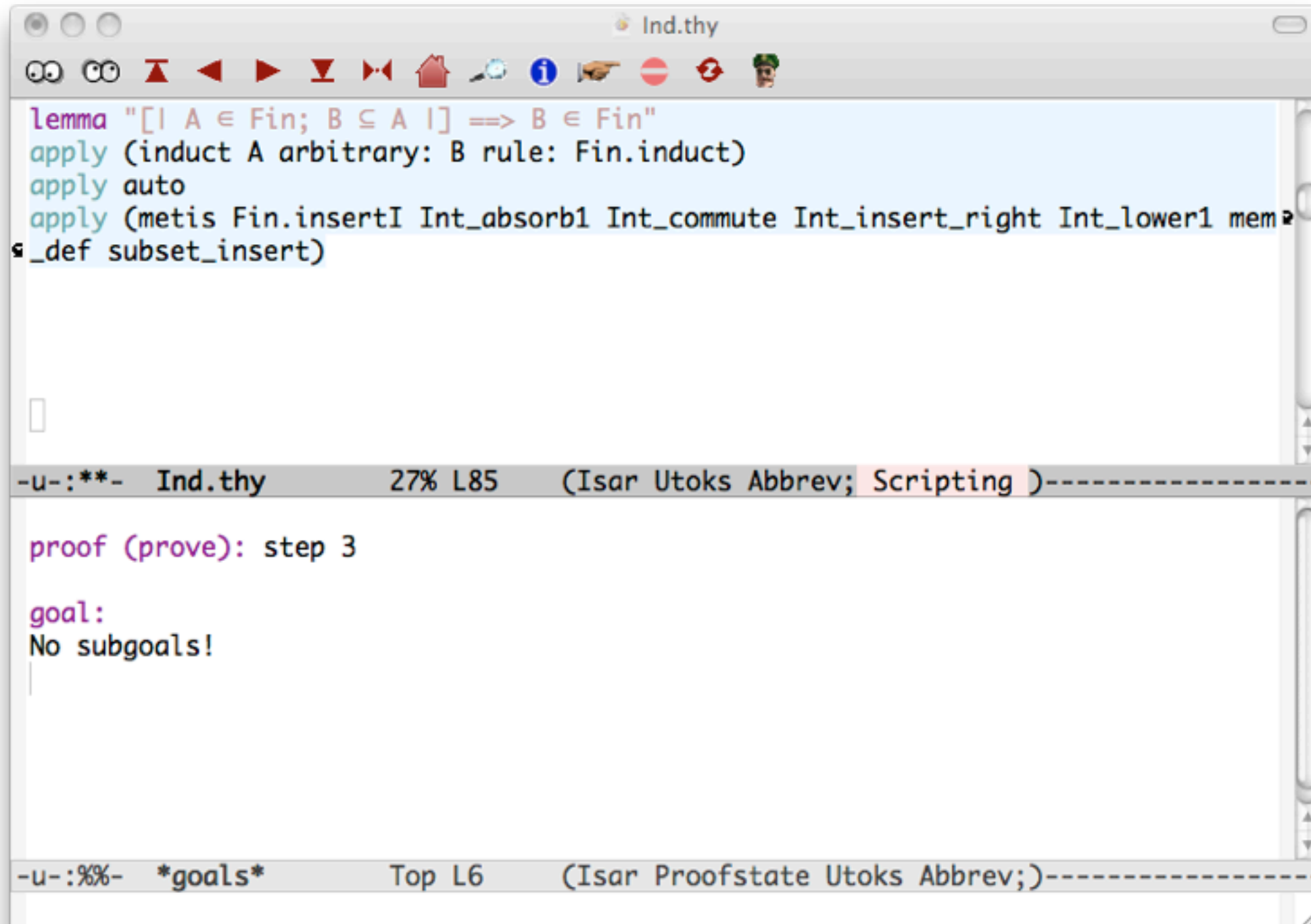
$$\wedge A a B. [\wedge B. B \subseteq A \Rightarrow B \in \text{Fin}; B \subseteq \text{insert } a A] \Rightarrow B \in \text{Fin}$$

Try this command: apply (metis Fin.insertI Int_absorb1 Int_commute Int_insert_right Int_lower1 mem_def subset_insert)
For minimizing the number of lemmas try this command:
atp_minimize [atp=spass] Fin.insertI Int_absorb1 Int_commute Int_insert_right Int_lower1 mem_def subset_insert
-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)-----
menu-bar Isabelle Commands Sledgehammer
```

this command should prove the goal

this one may return a more compact command

The Completed Proof

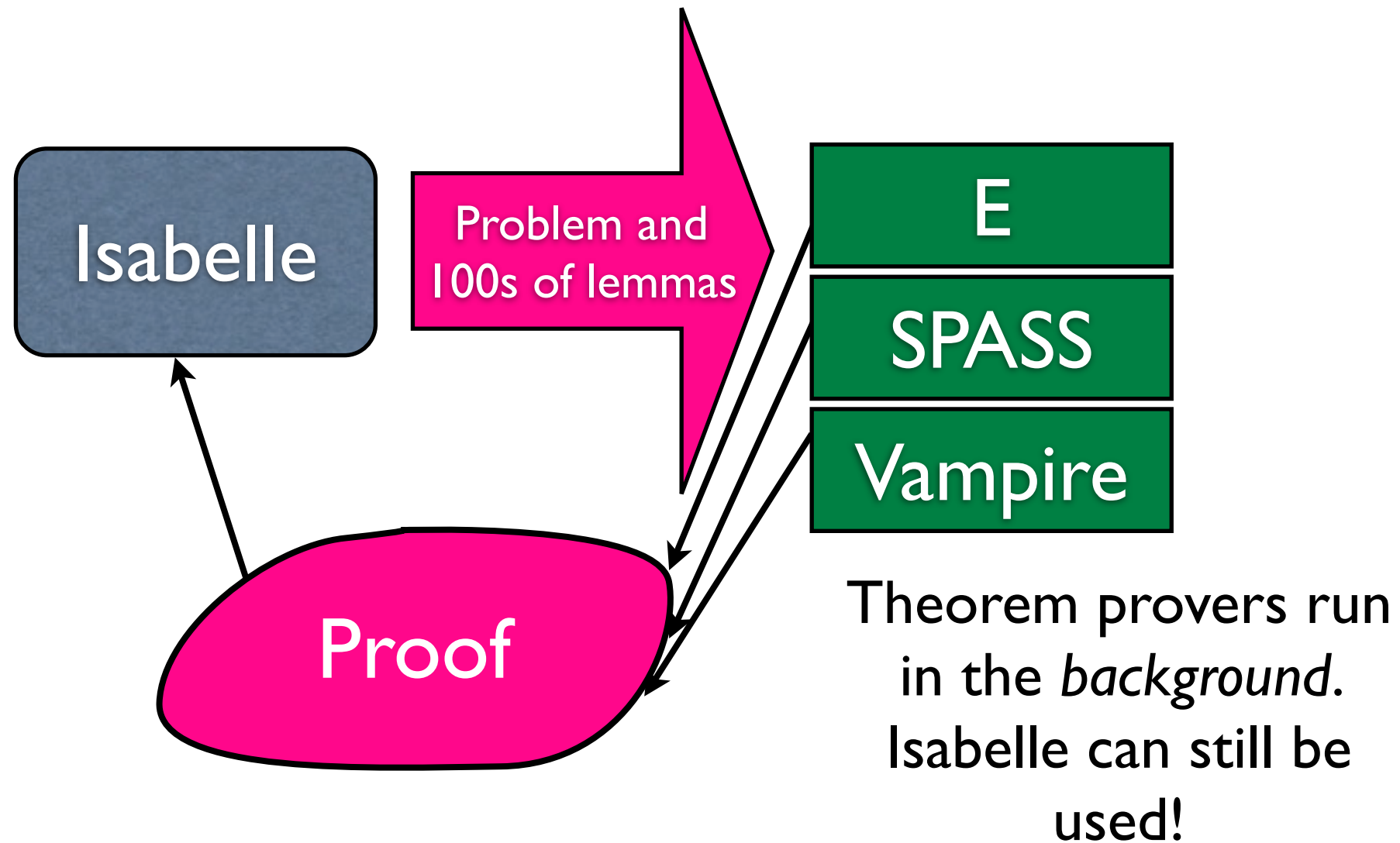


The screenshot shows the Isabelle/Isar IDE interface. The main editor window displays a lemma and its proof. The lemma states that if a set A is finite and B is a subset of A, then B is finite. The proof is completed using the `induct` tactic, followed by `auto` and `metis` with several lemmas. The status bar indicates the proof is at line 85, 27% complete. The bottom panel shows the goal state, which is now empty, indicating the proof is complete.

```
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
  apply (induct A arbitrary: B rule: Fin.induct)
  apply auto
  apply (metis Fin.insertI Int_absorb1 Int_commute Int_insert_right Int_lower1 mem_
  _def subset_insert)

proof (prove): step 3
goal:
No subgoals!
```

How Sledgehammer Works



Notes on Sledgehammer

- It is always available, but it cannot work miracles.
- It does not prove the goal, but returns a call to `metis`. This command *usually* works...
- The `minimise` option removes redundant theorems, increasing the likelihood of success.
- Calling `metis` directly is difficult unless you know exactly which lemmas are needed.

Counterexample Finding

- Don't waste time trying to prove impossible statements!
- Isabelle can find counterexamples quickly...
 - *quickcheck*: random testing of executable specifications (broadly interpreted)
 - *nitpick*: a more general, SAT-based counterexample finder
- Consider switching on “auto quickcheck” or “auto nitpick”, although they can be slow!

Nitpick Example

The screenshot shows the Isabelle proof assistant interface. The top menu bar includes 'Aquamacs', 'File', 'Edit', 'Options', 'Tools', 'Isabelle', 'Proof-General', 'Tokens', 'Maths', 'Window', and 'Help'. The 'Isabelle' menu is currently selected. Below the menu bar is a toolbar with various icons for navigation and editing. The main window displays two tabs: 'Def.thy' and 'BT.thy'. The 'BT.thy' tab is active, showing the following code:

```
lemma reflect_reflect_ident: "reflect (reflect t) = t"
  apply (induct t)
  apply auto
  done

lemma "reflect t = t"
```

Below the code editor, there is a status bar showing 'u-:**- BT.thy 62% (17,1) (Isar Utoks Abbrev; Scripting)'. Below this, there are two tabs: '*response*' and '*goals*'. The '*response*' tab is active, showing the following text:

```
Nitpicking formula...

Nitpick found a counterexample for card 'a = 4:

Free variable:
  t = Br a1 (Br a2 Lf Lf) Lf
```

At the bottom of the window, there is a status bar showing 'uU:%%- *response* All (6,30) (Isar Messages Utoks Abbrev)'.

Other Things to Learn

- structural operational semantics (SOS):
definitions and proofs
- the Isar structured proof language
- axiomatic type classes
- locales and contexts
- code generation