Interactive Formal Verification
7: Inductive Definitions

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

Defining a Set Inductively

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.

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*.

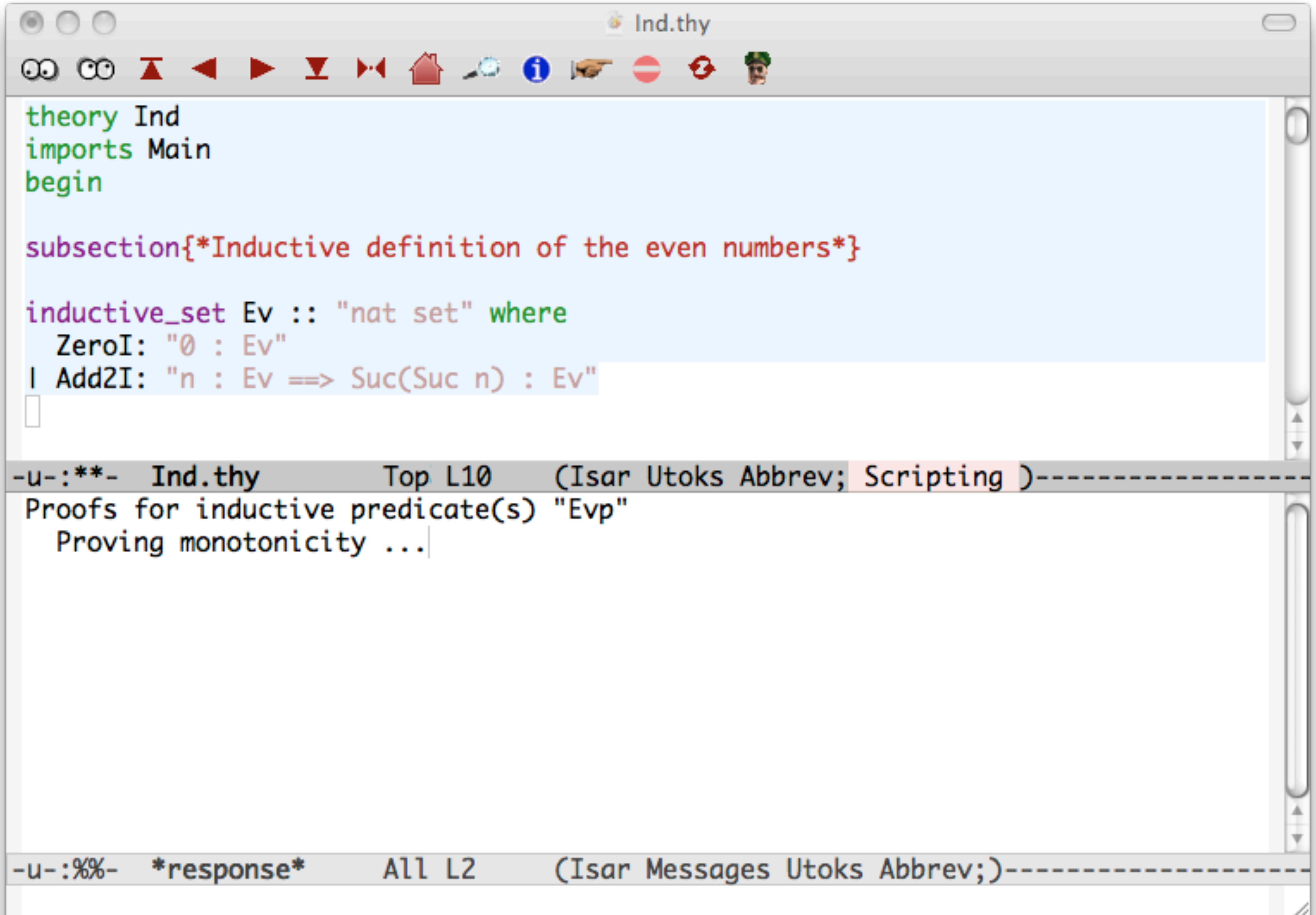
Defining a Set Inductively

- 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.

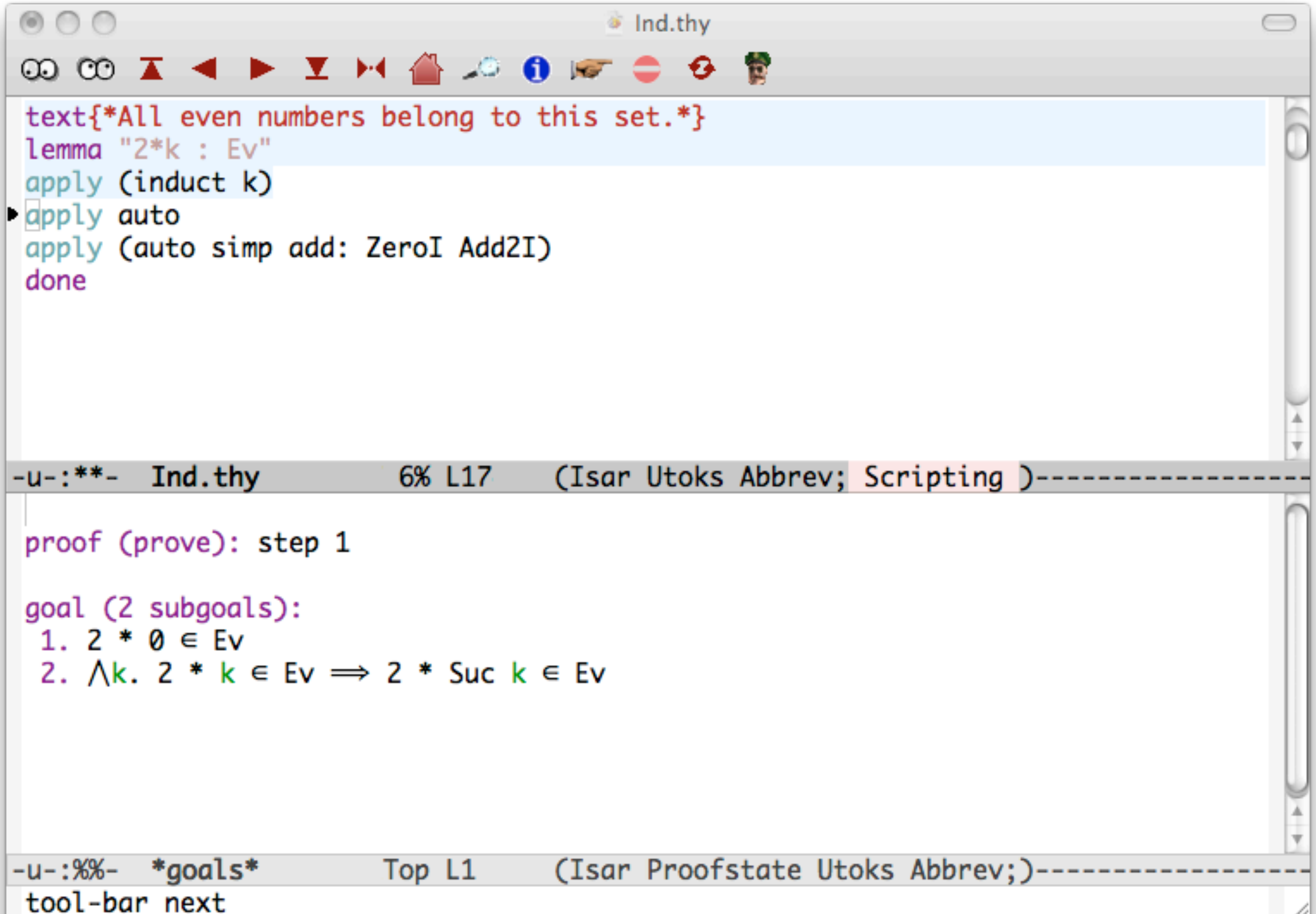
Defining a Set Inductively

- 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.

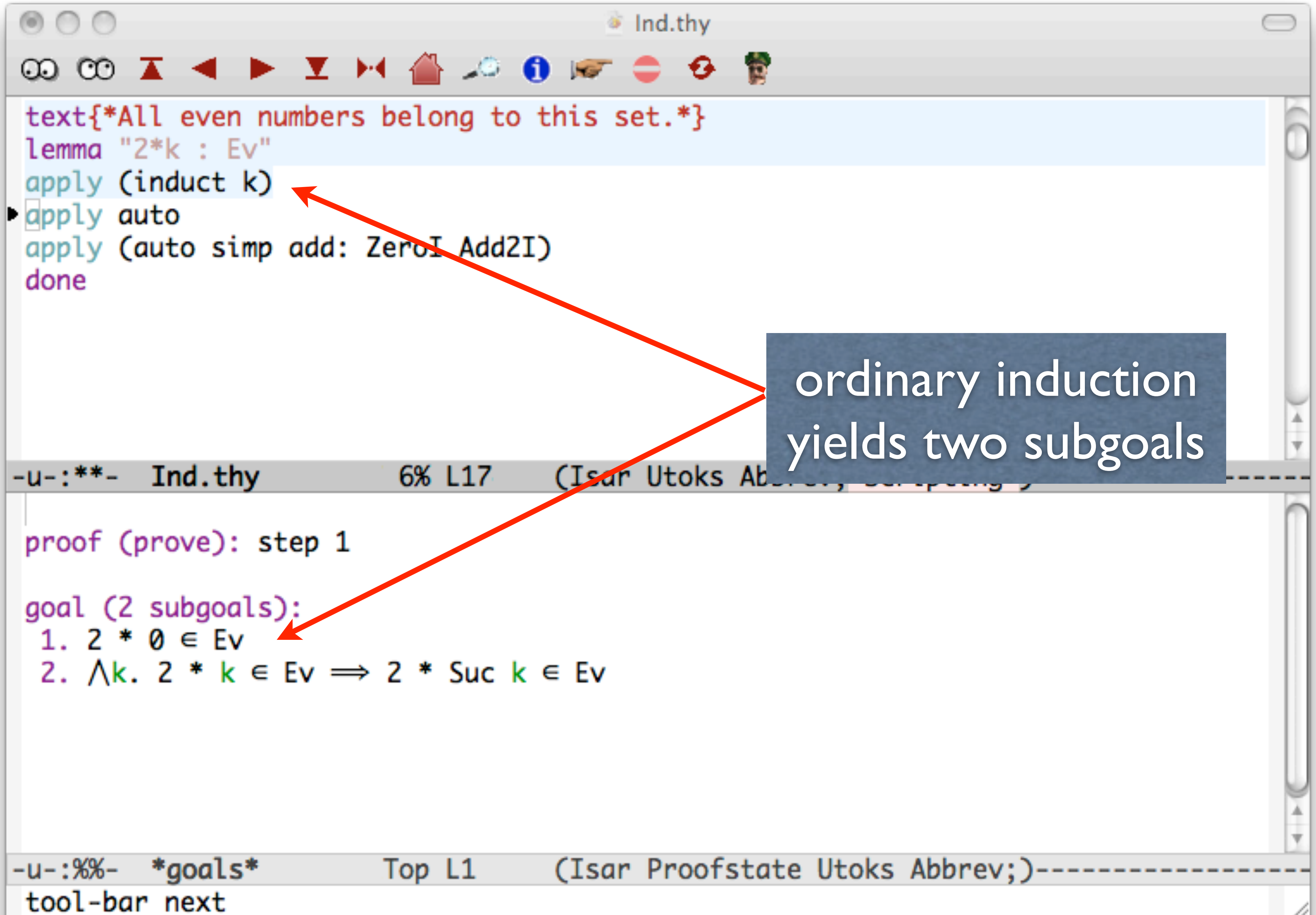
Inductive Definitions in Isabelle



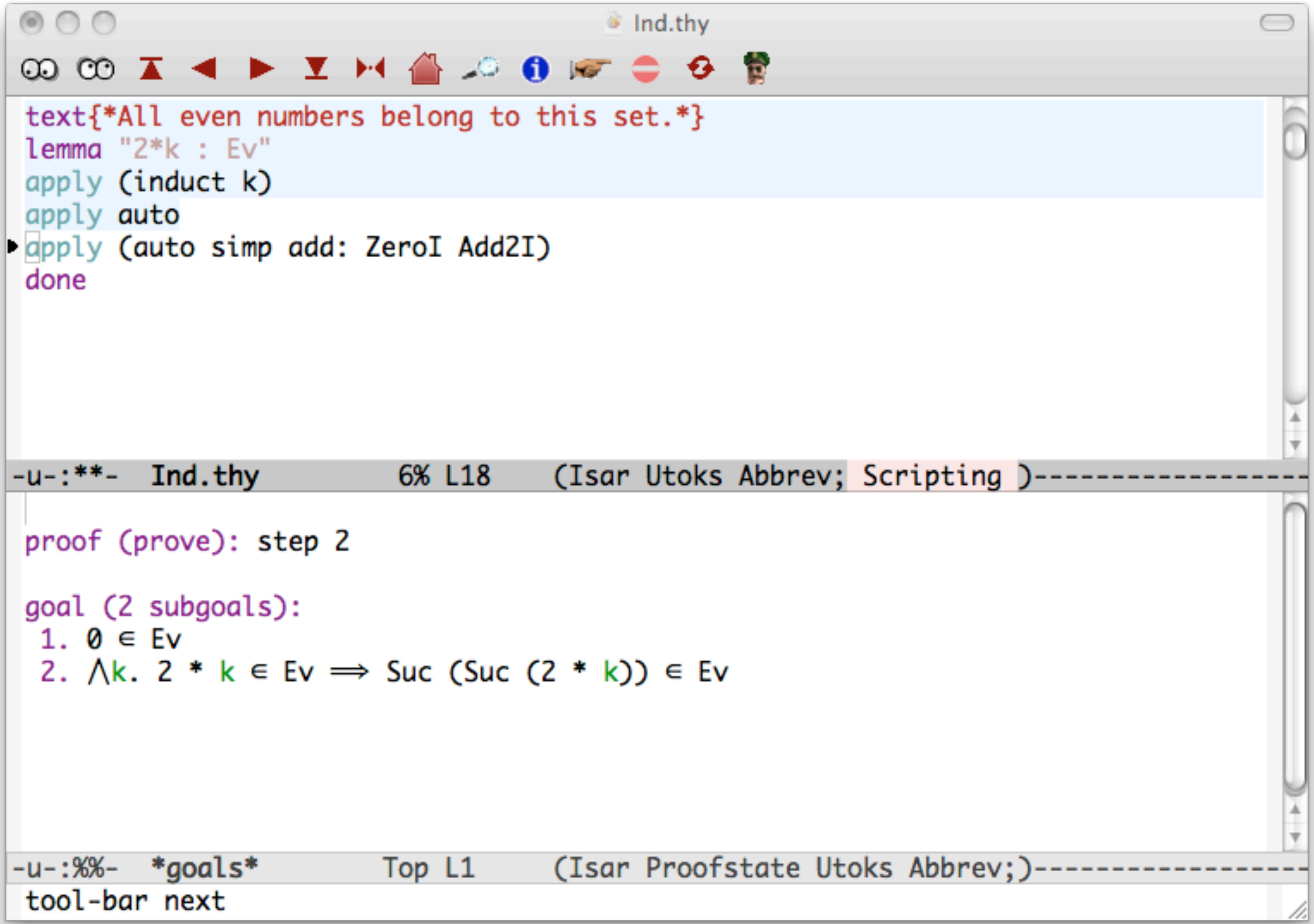
Even Numbers Belong to `Ev`



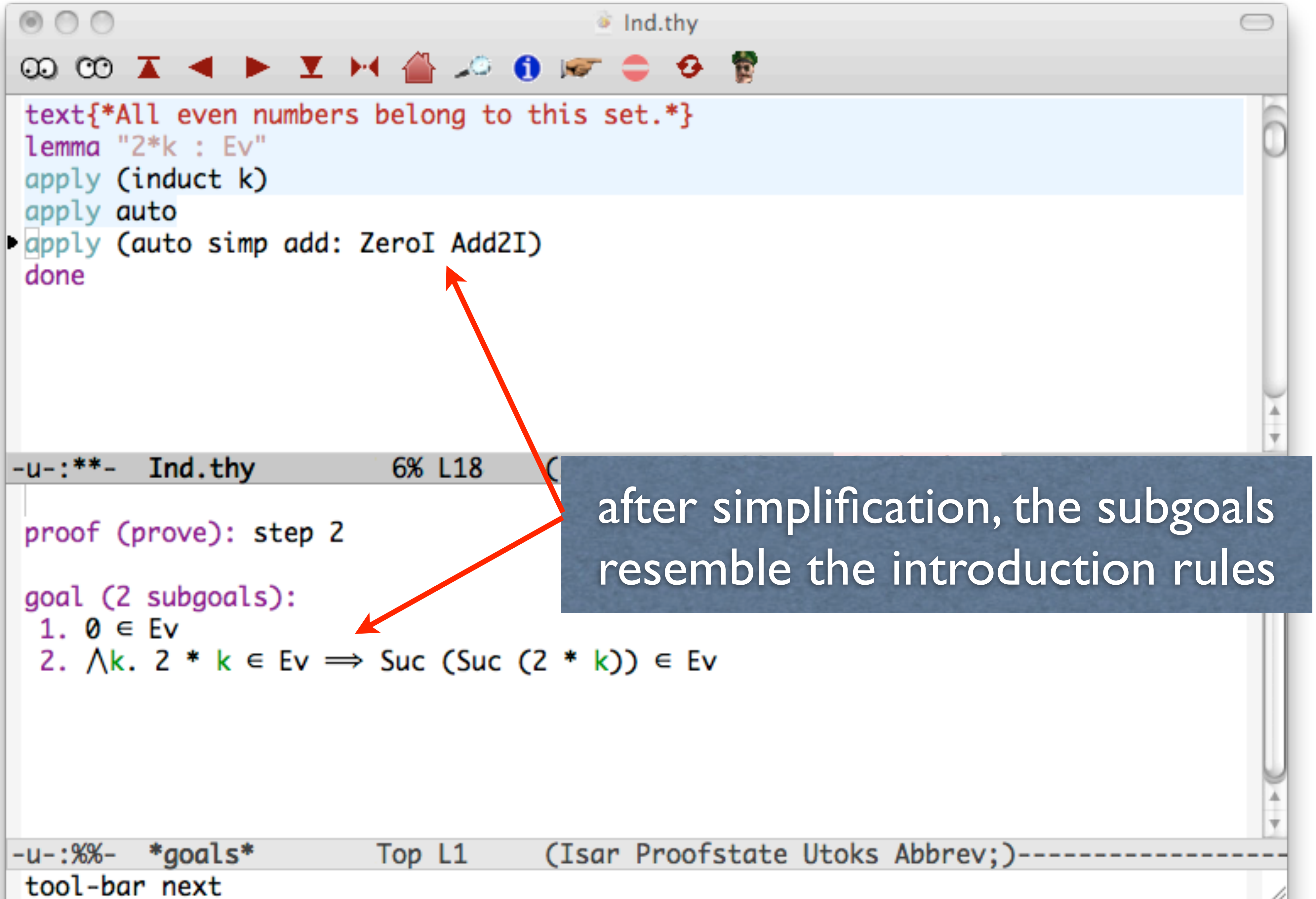
Even Numbers Belong to `Ev`



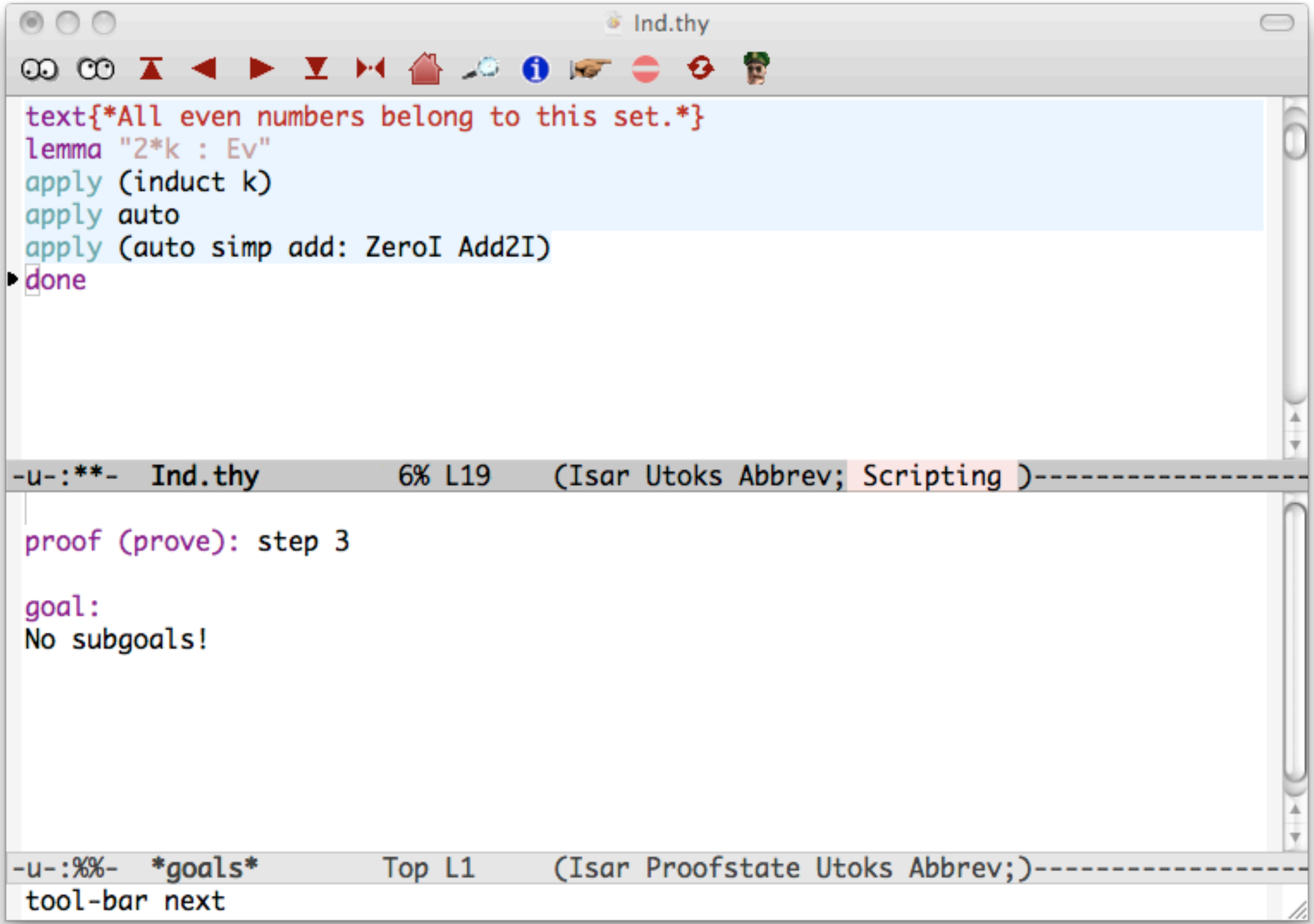
Proving Set Membership



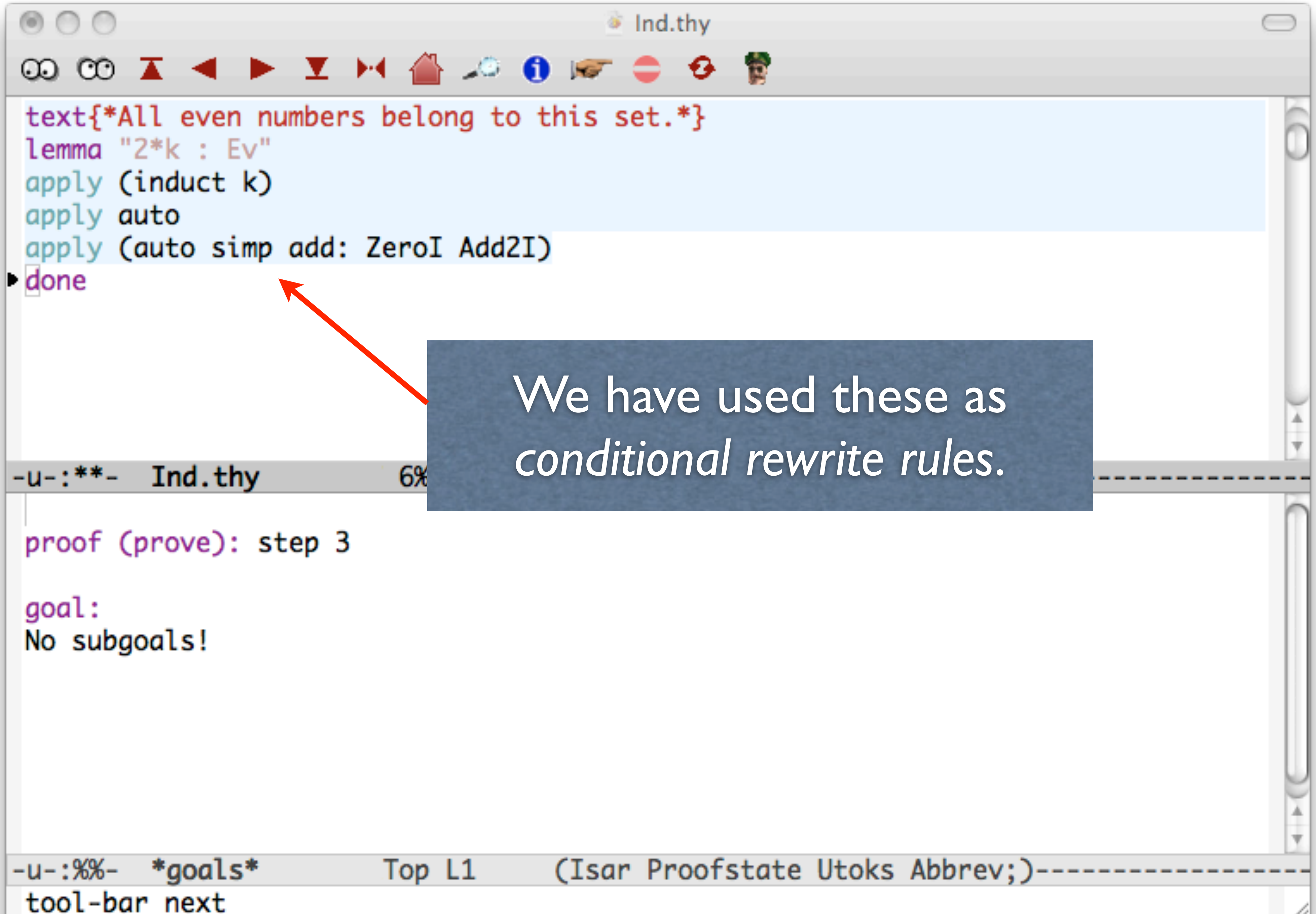
Proving Set Membership



Finishing the Proof

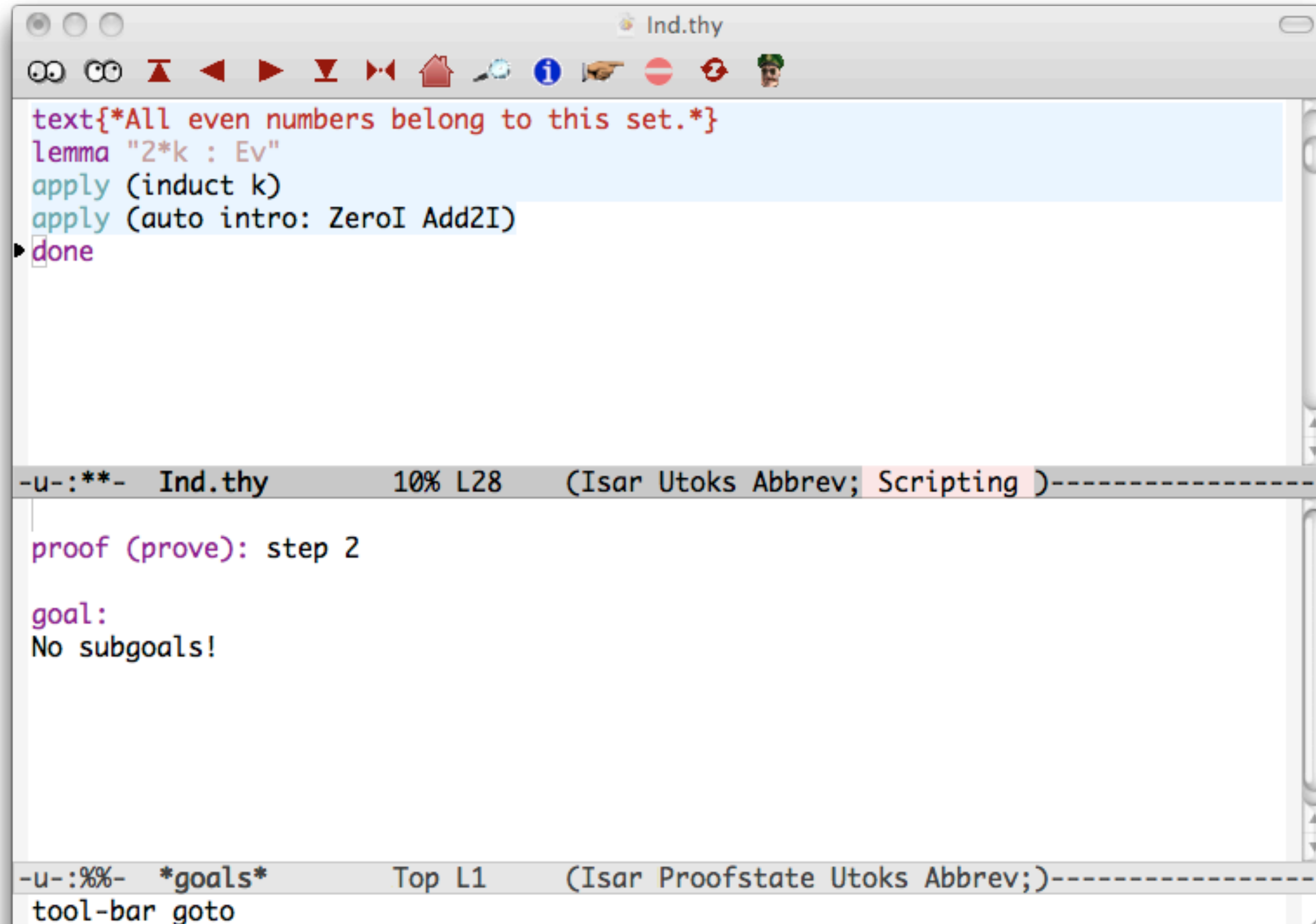


Finishing the Proof

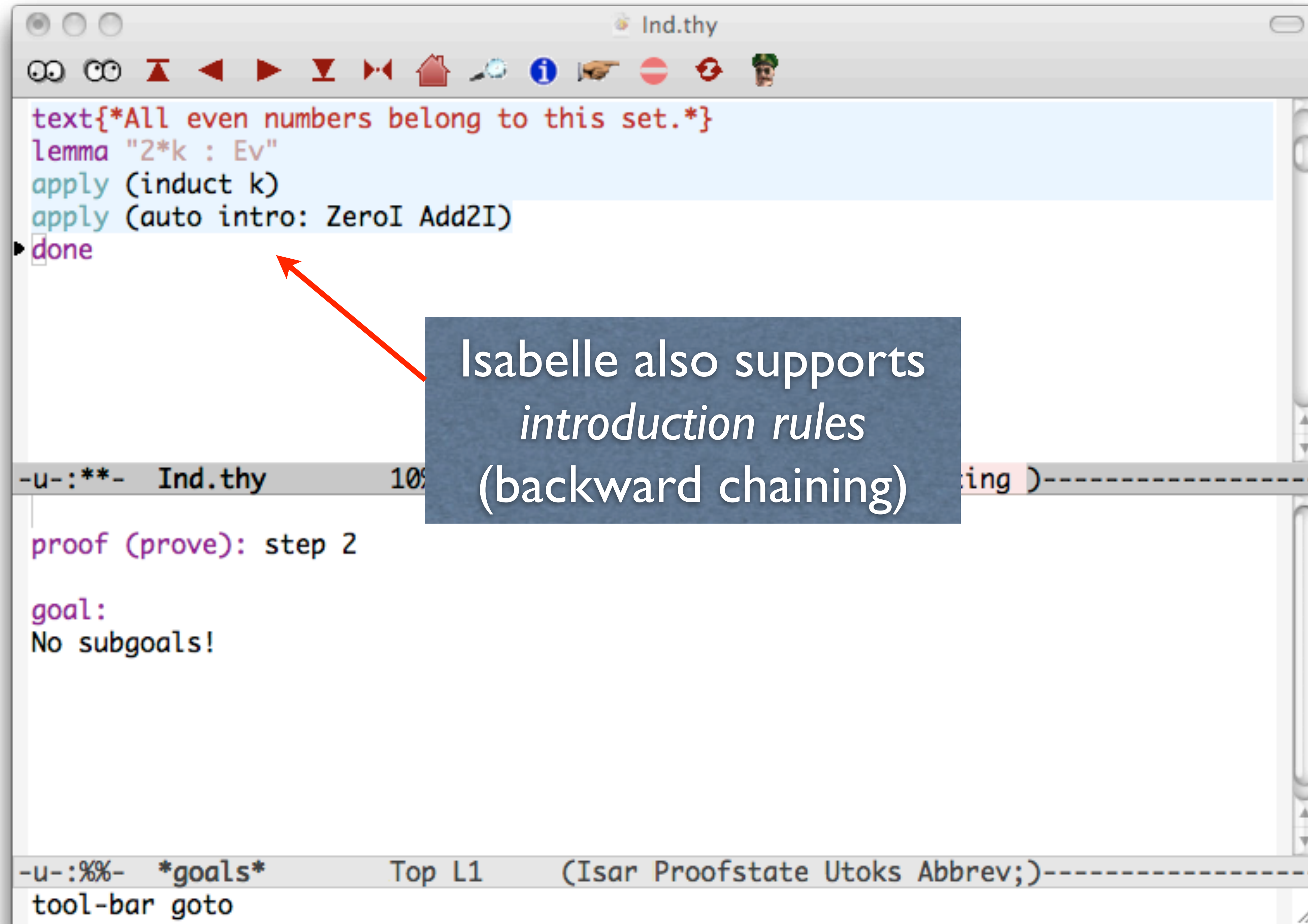


We have used these as *conditional rewrite rules.*

Finishing the Proof



Finishing the Proof



Rule Induction

Rule Induction

- Proving something about *every* element of the set.

Rule Induction

- Proving something about *every* element of the set.
- It expresses that the inductive set is *minimal*.

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”

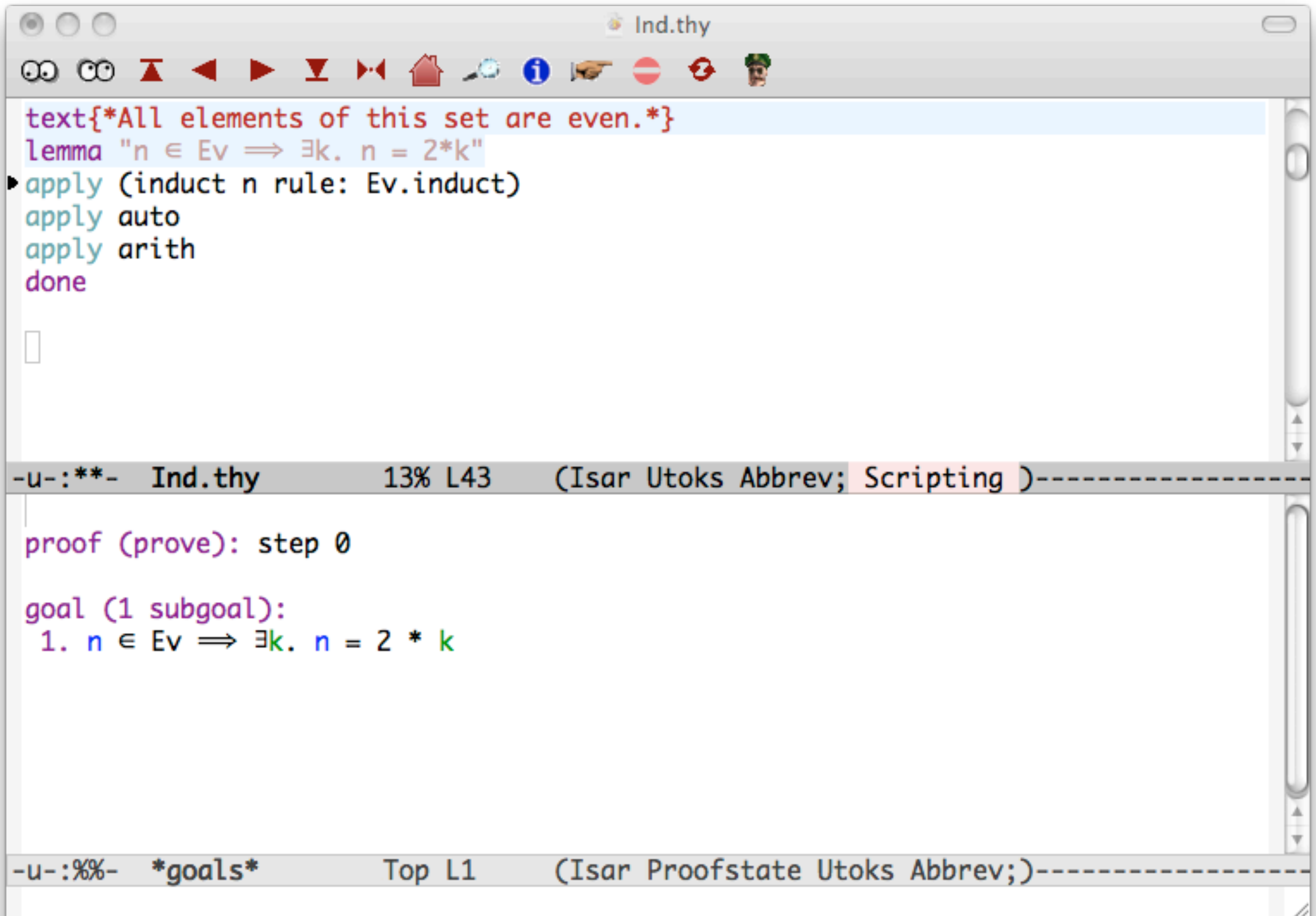
Rule Induction

- 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

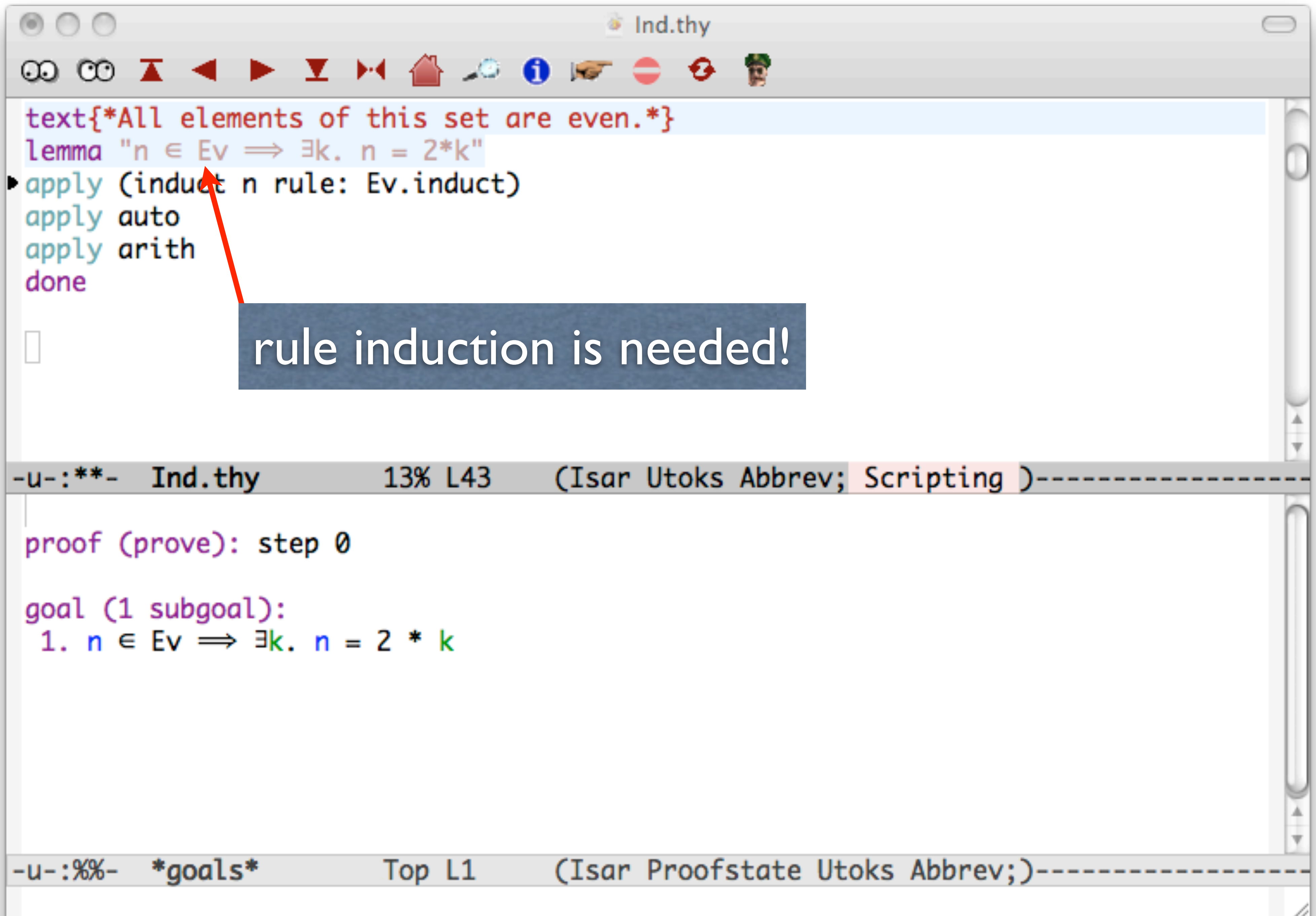
Rule Induction

- 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.

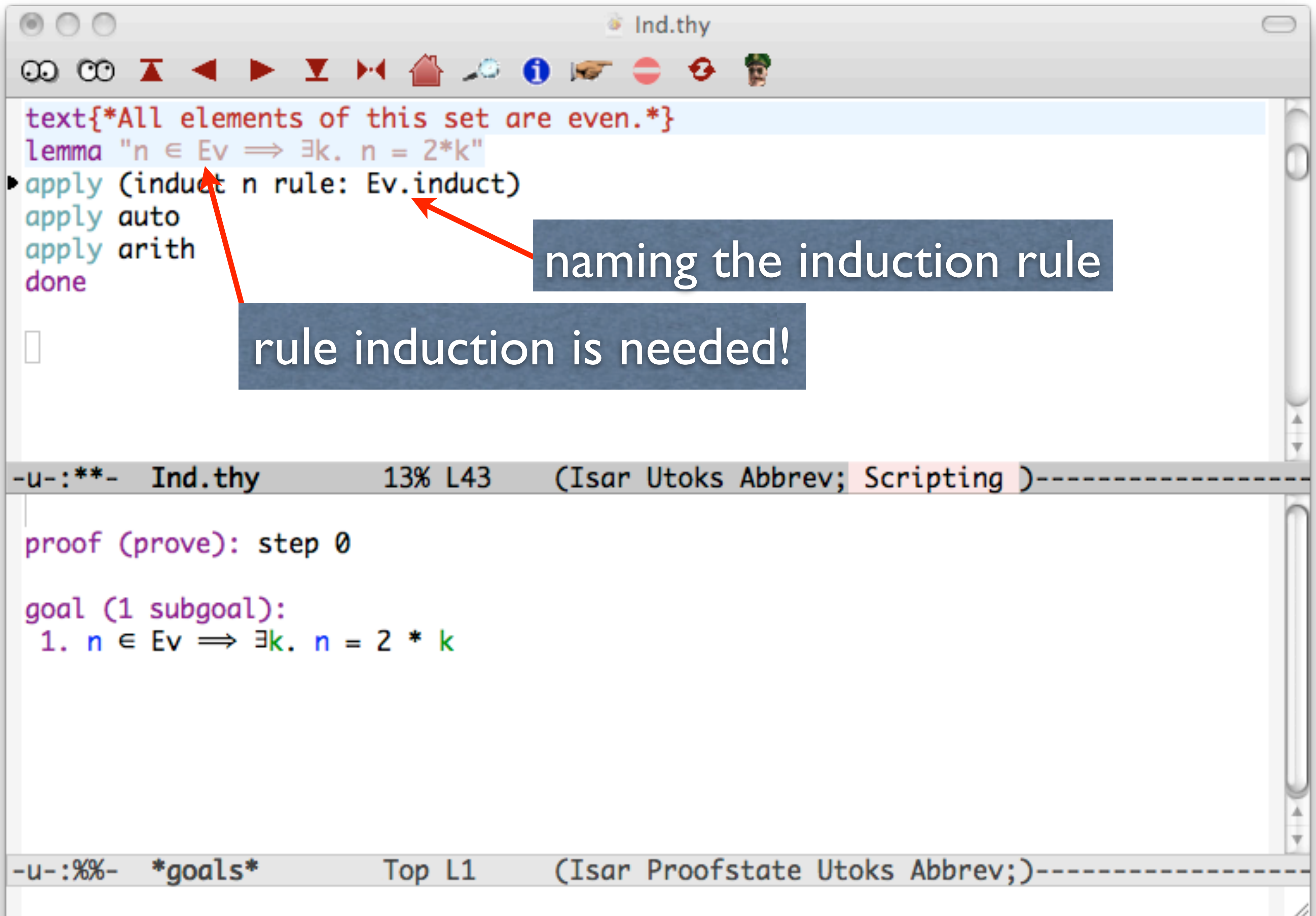
Ev Has only Even Numbers



Ev Has only Even Numbers



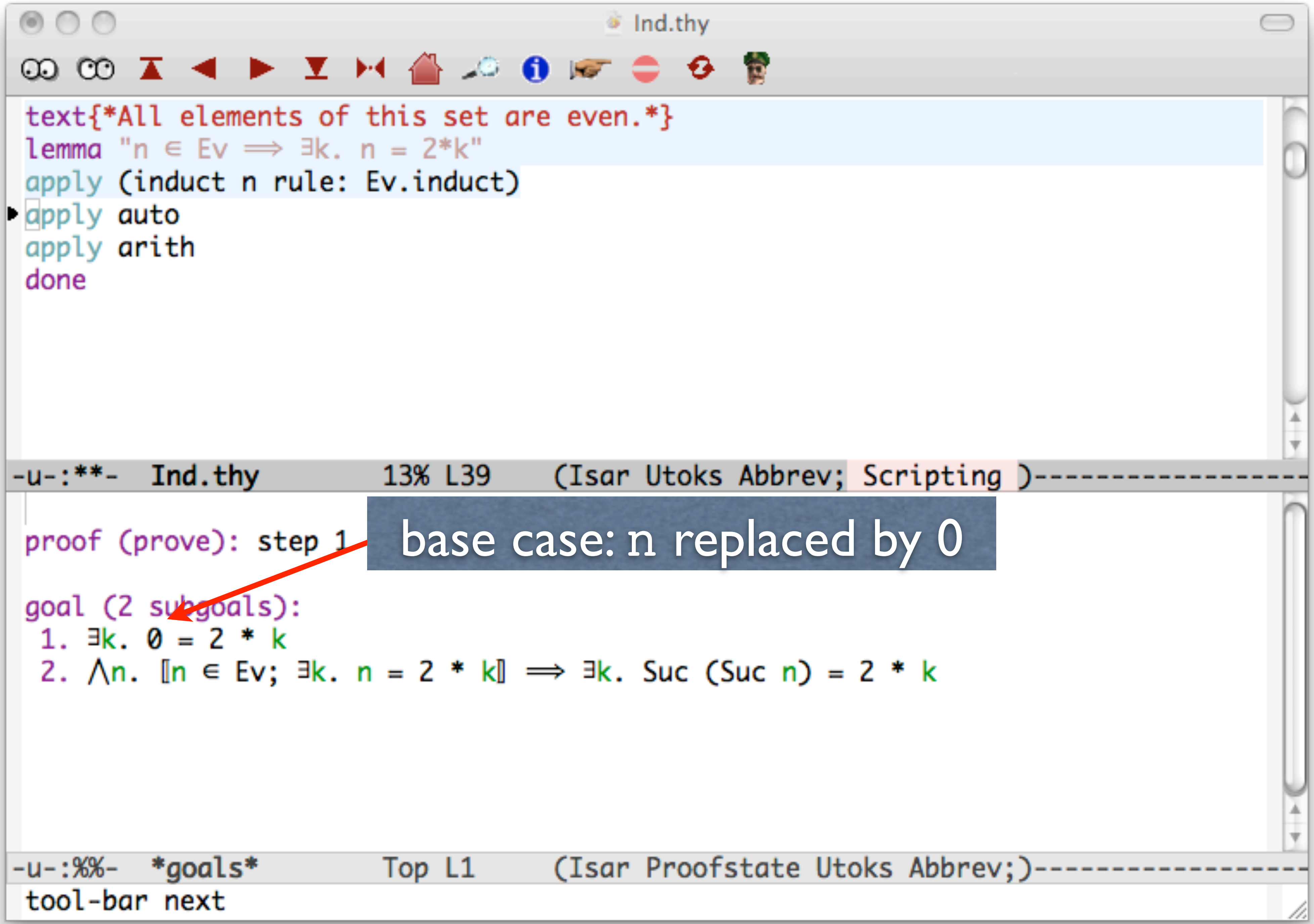
Ev Has only Even Numbers



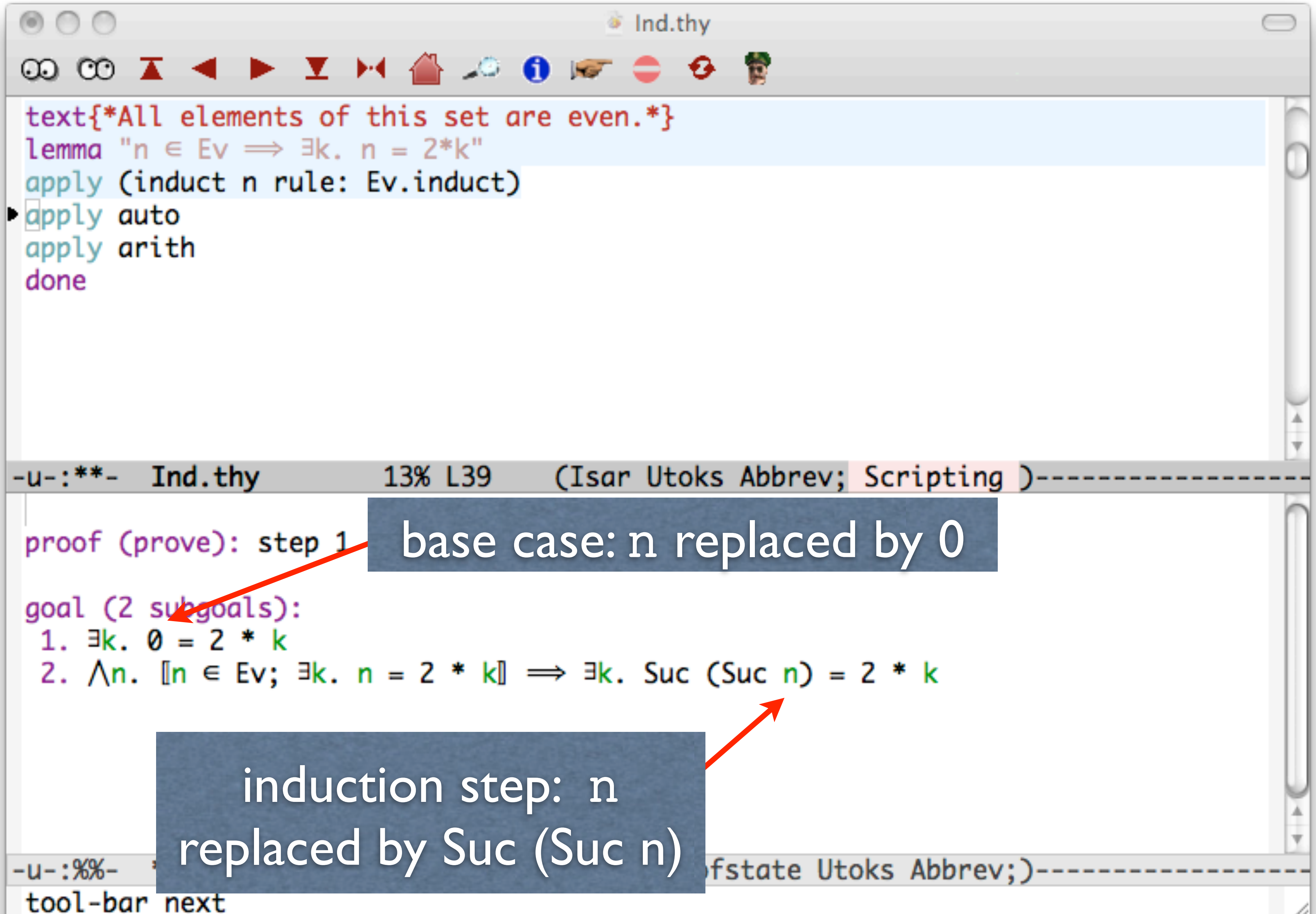
An Example of Rule Induction

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

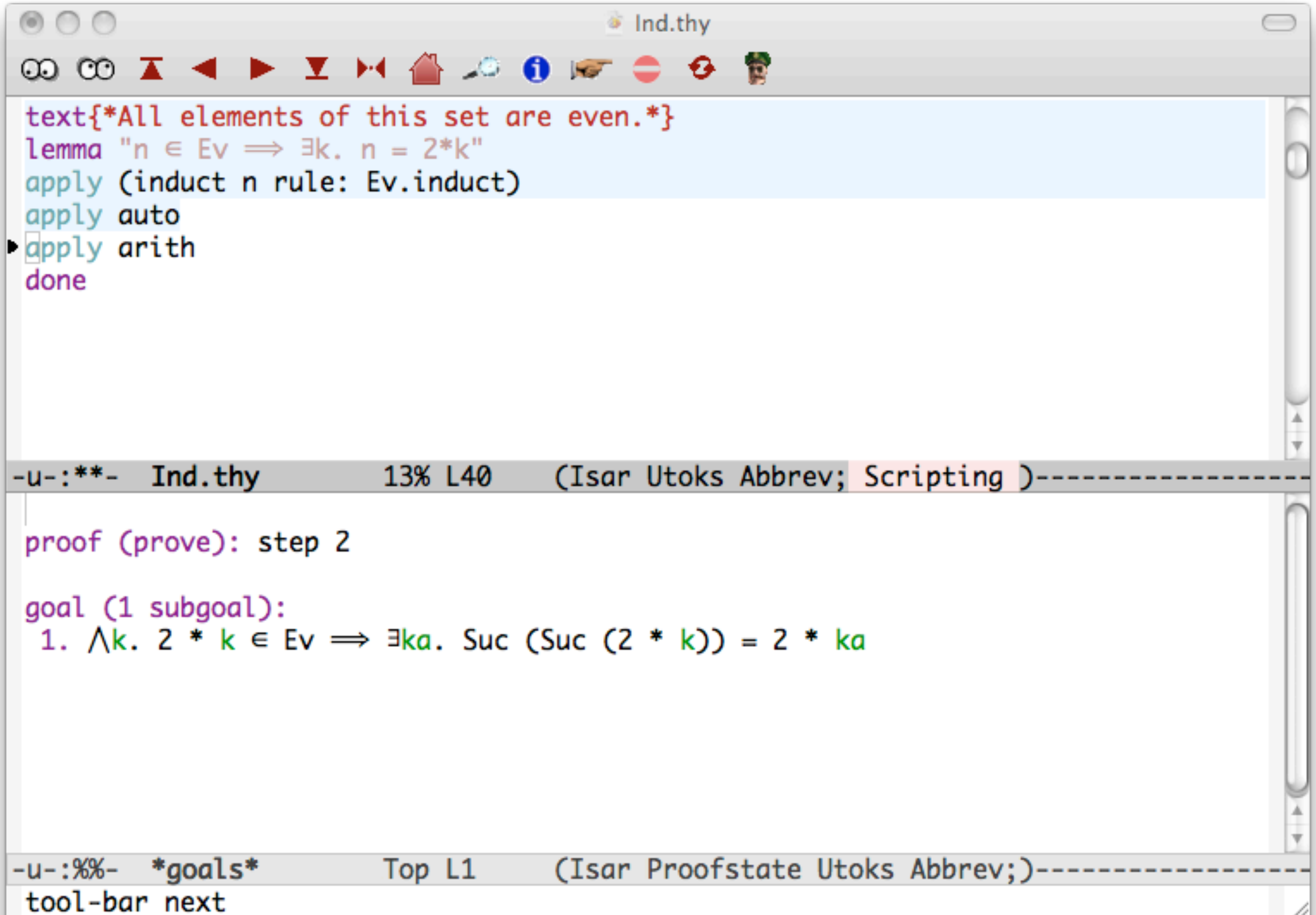
An Example of Rule Induction



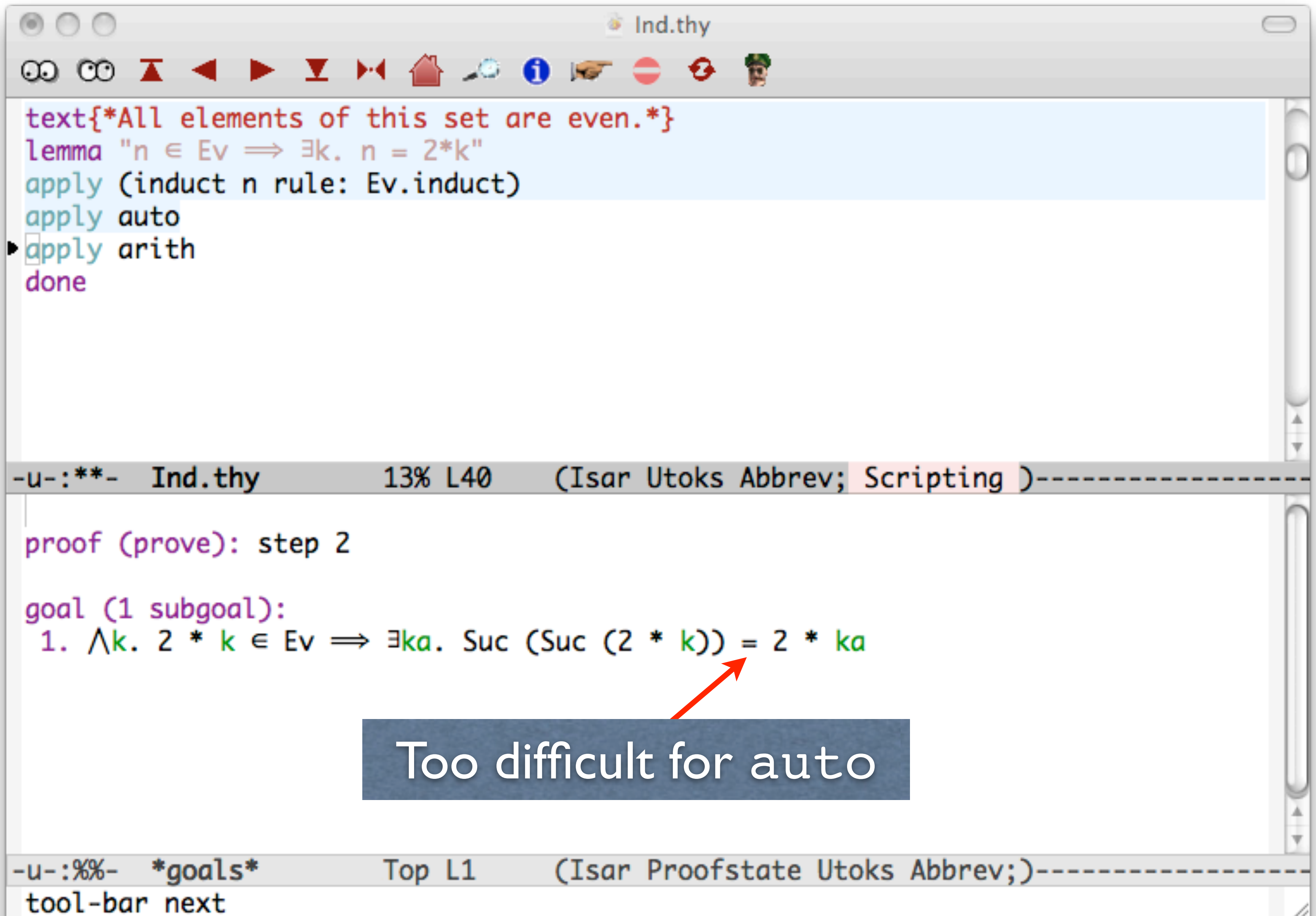
An Example of Rule Induction



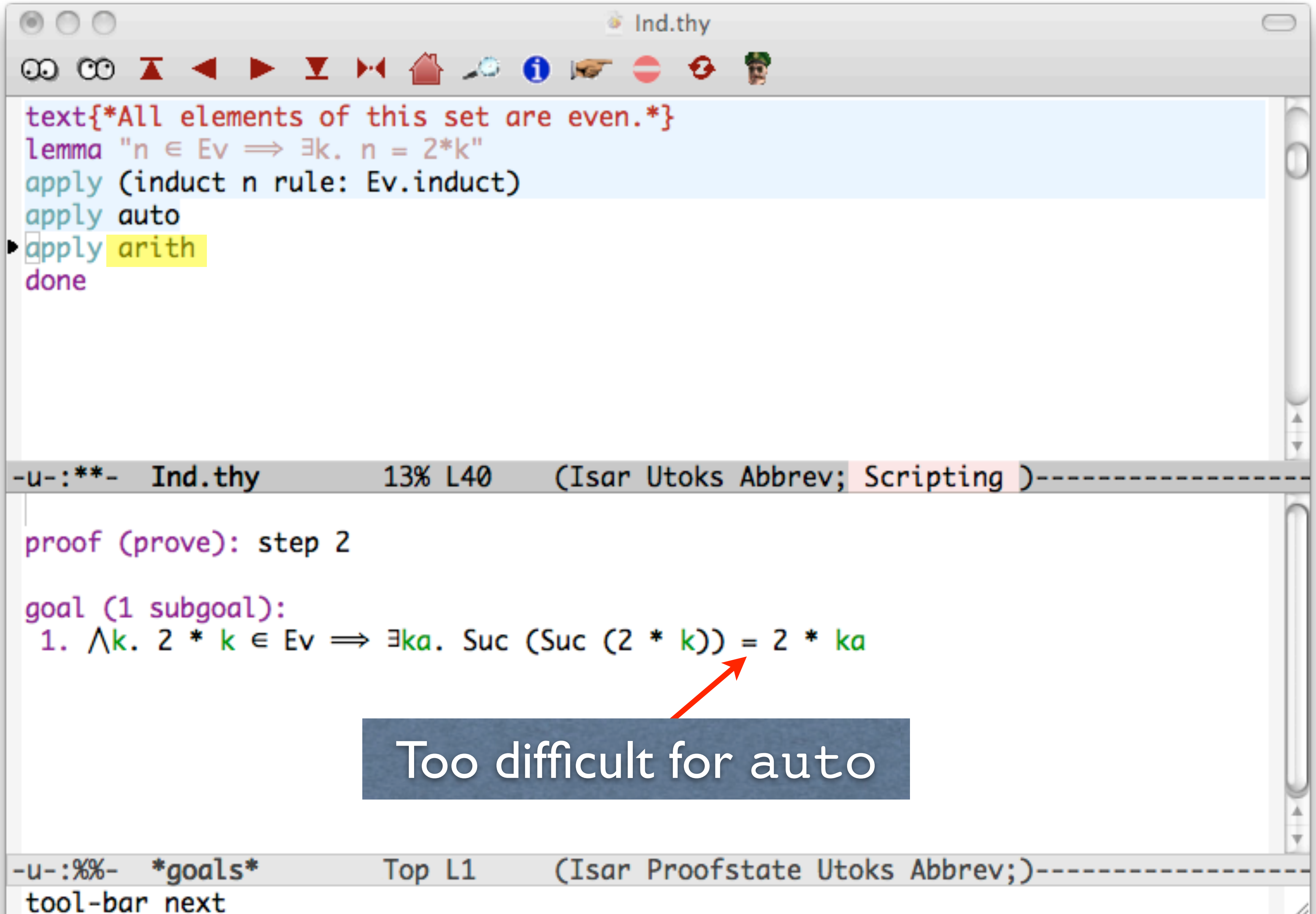
Nearly There!



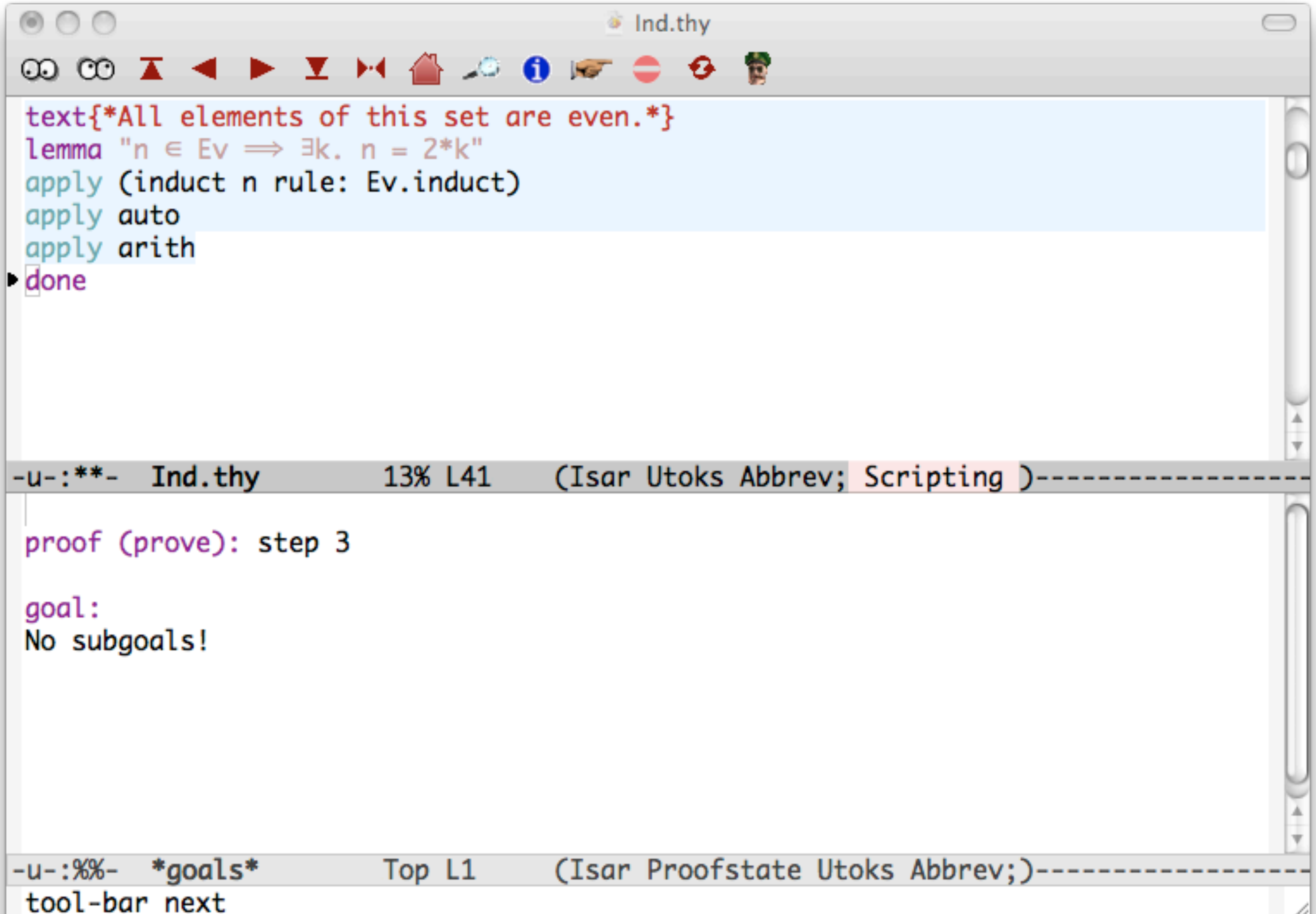
Nearly There!



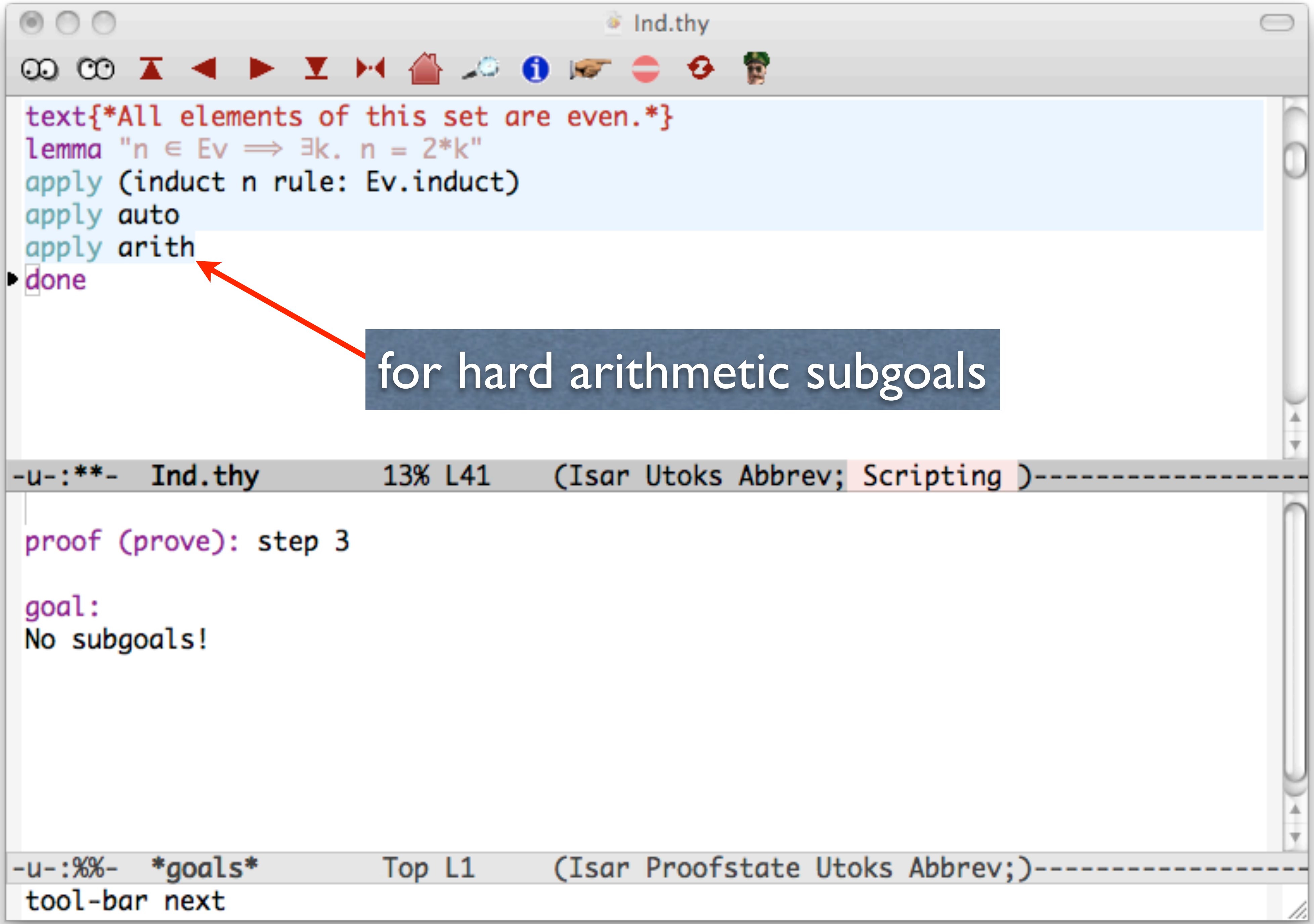
Nearly There!



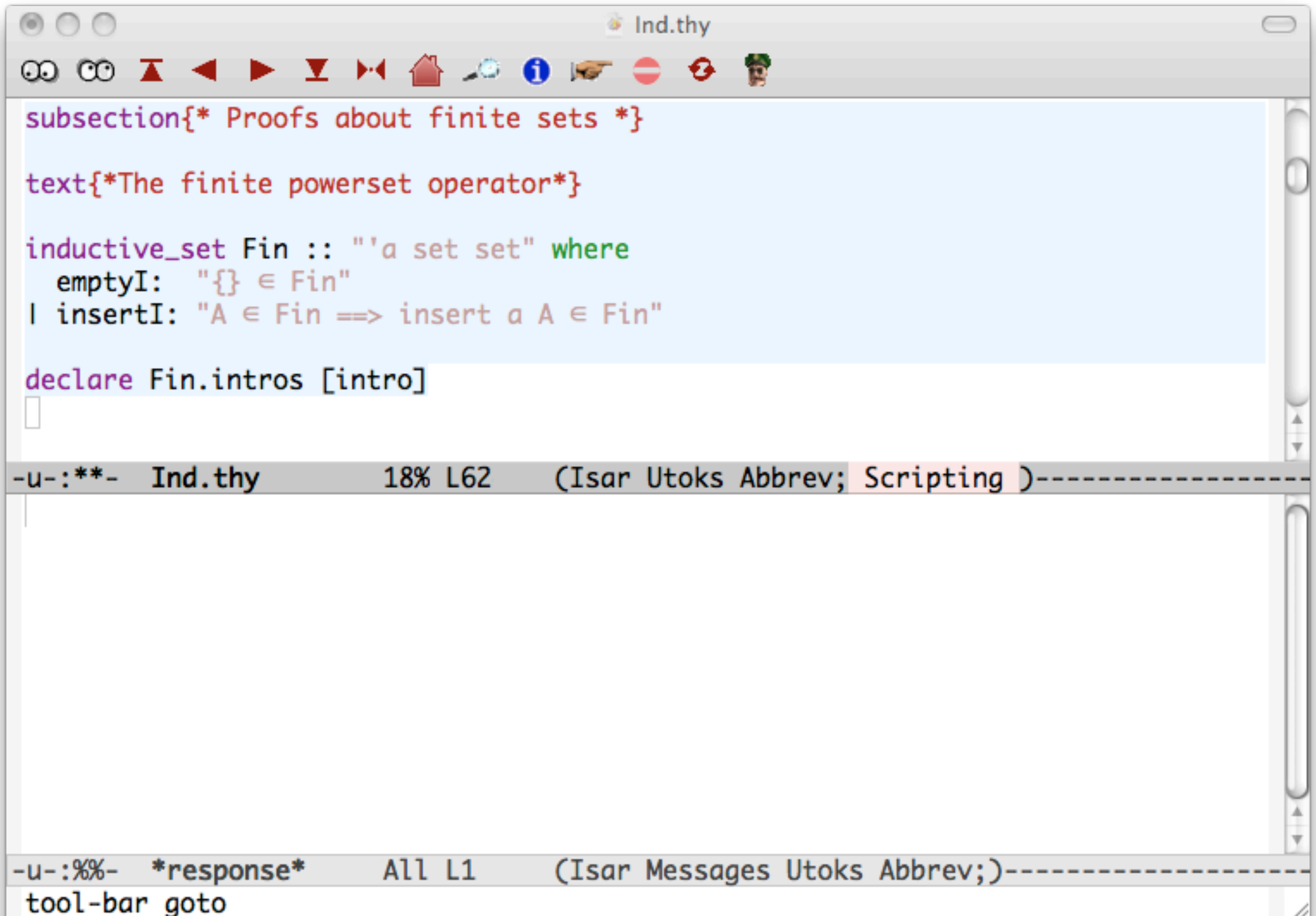
The `arith` Proof Method



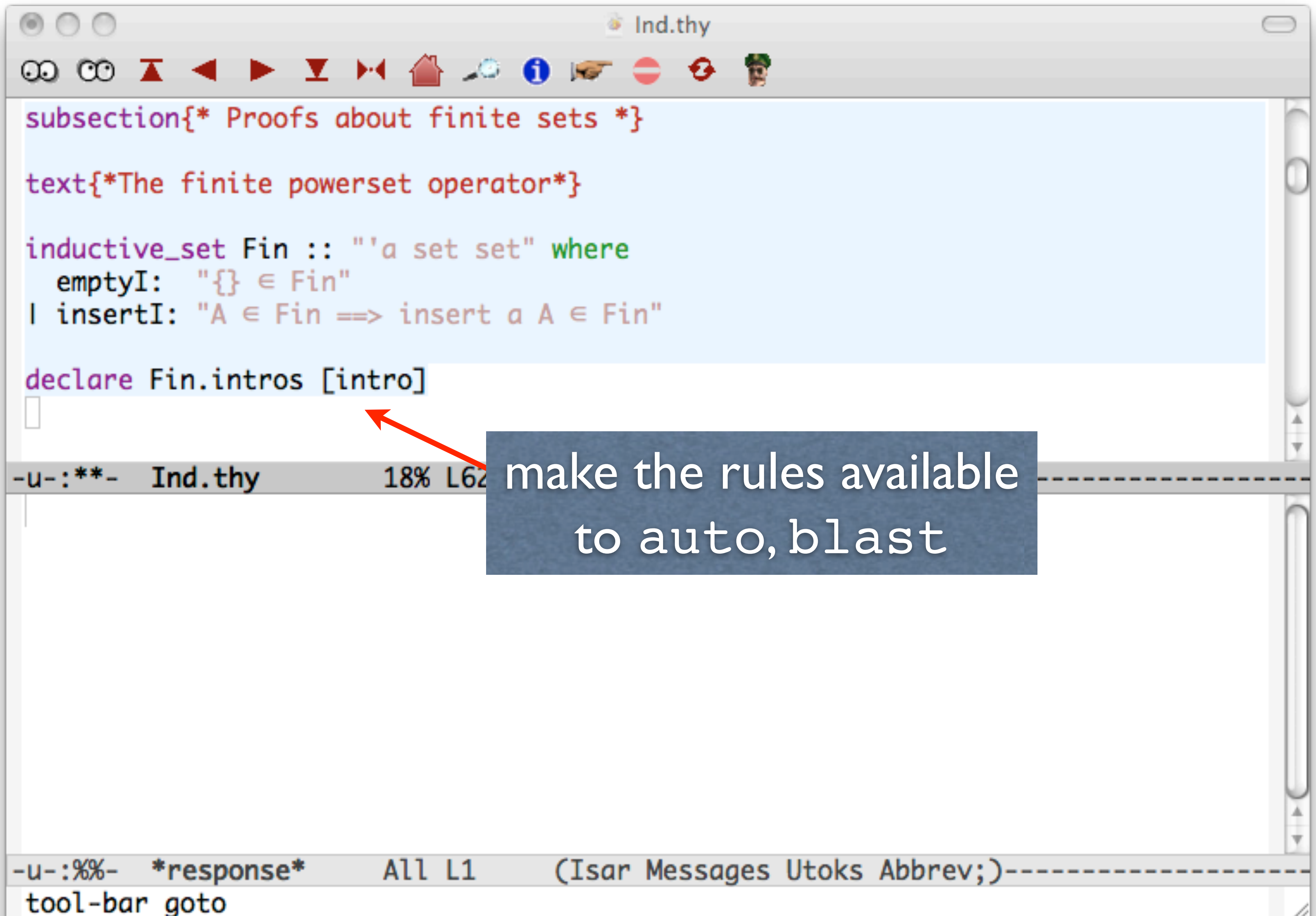
The `arith` Proof Method



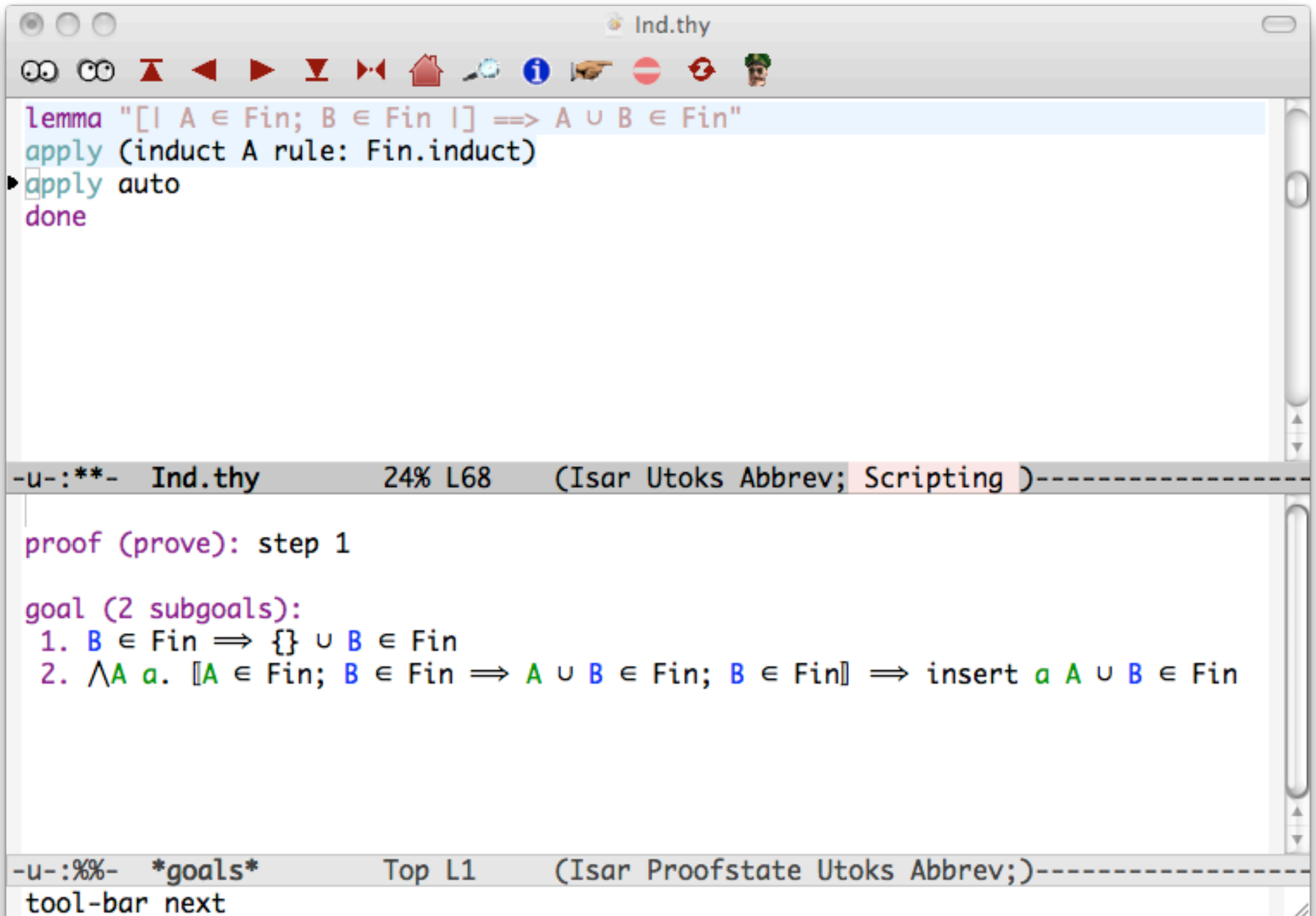
Defining Finiteness



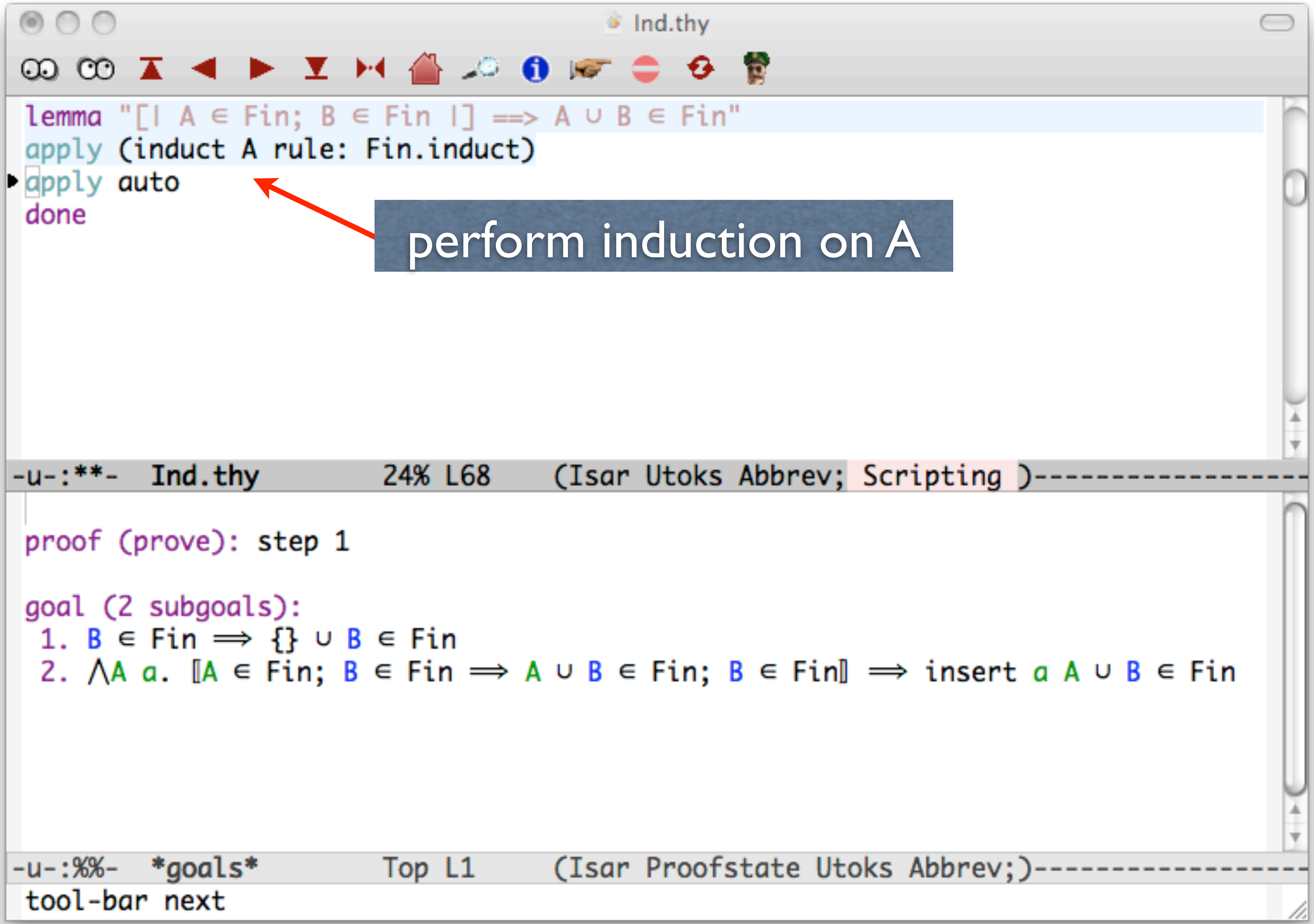
Defining Finiteness



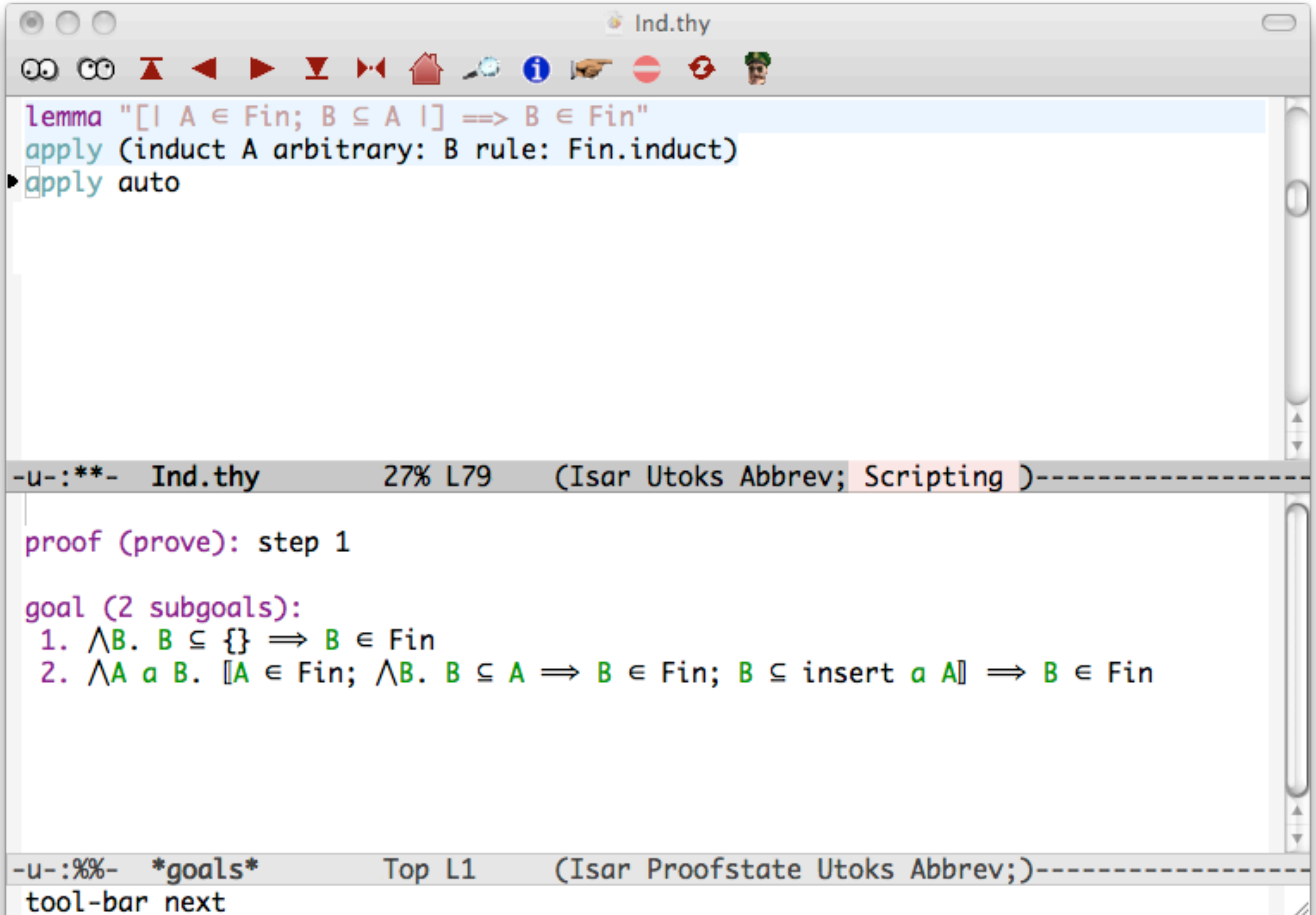
The Union of Two Finite Sets



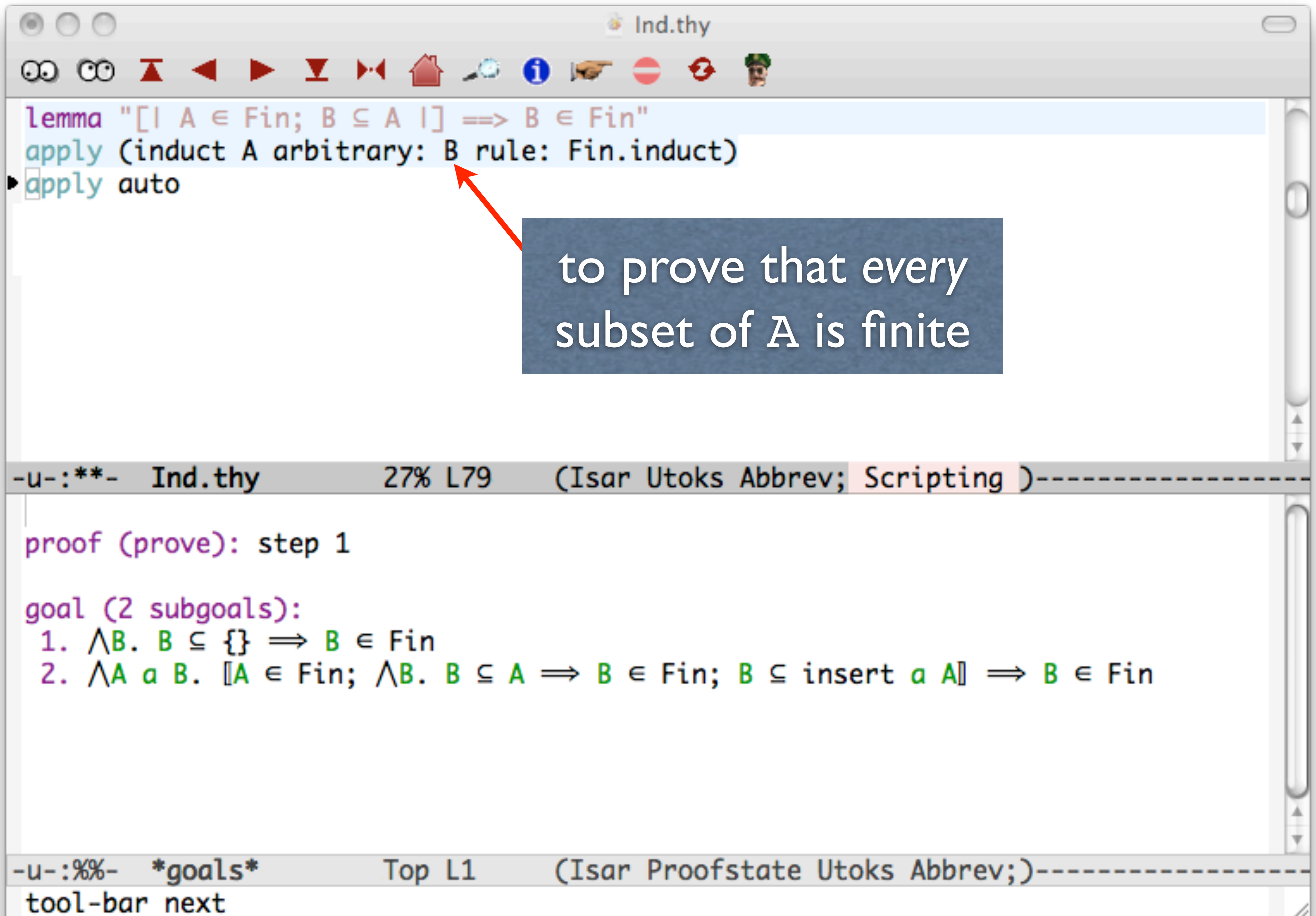
The Union of Two Finite Sets



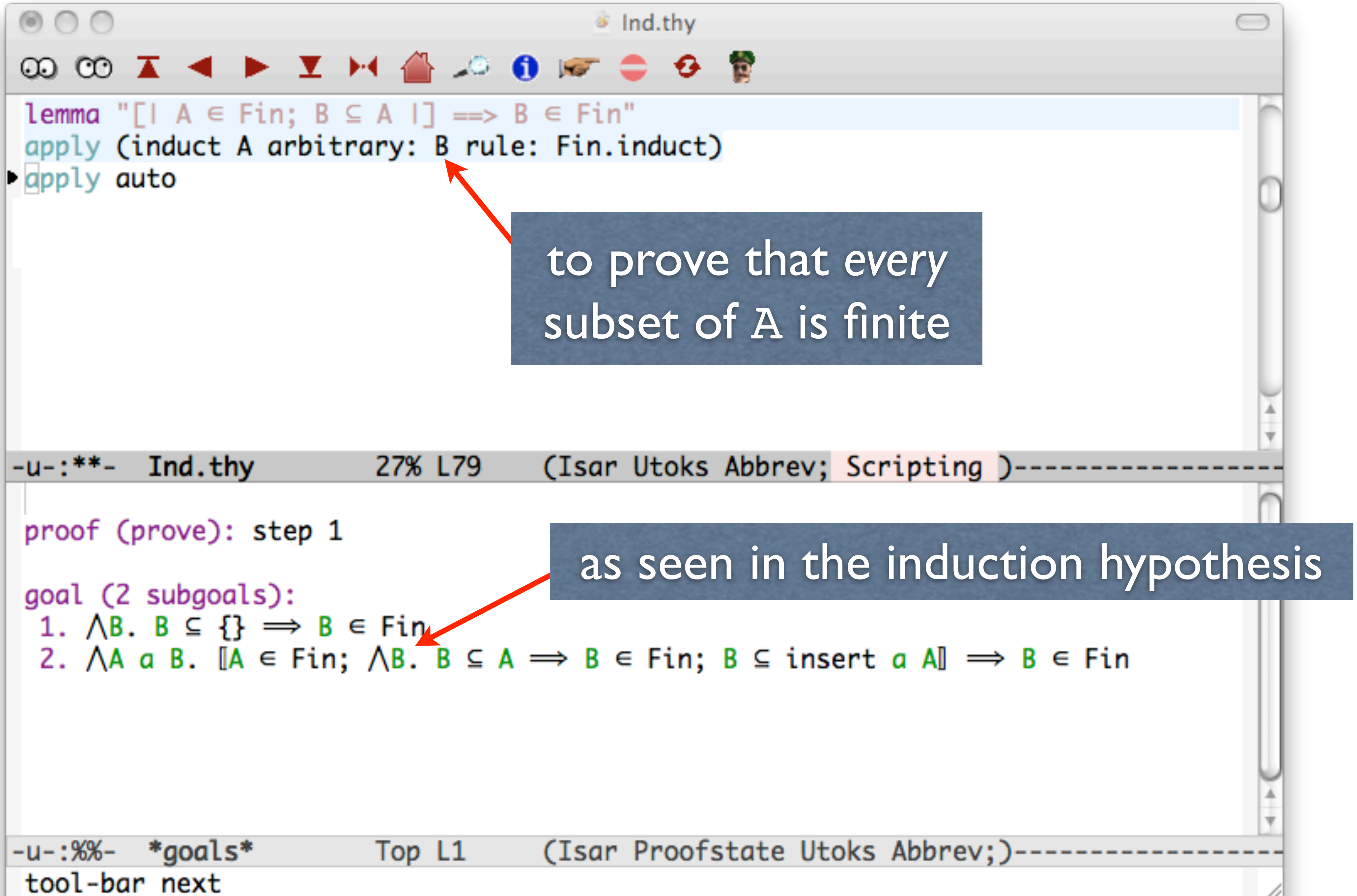
A Subset of a Finite Set



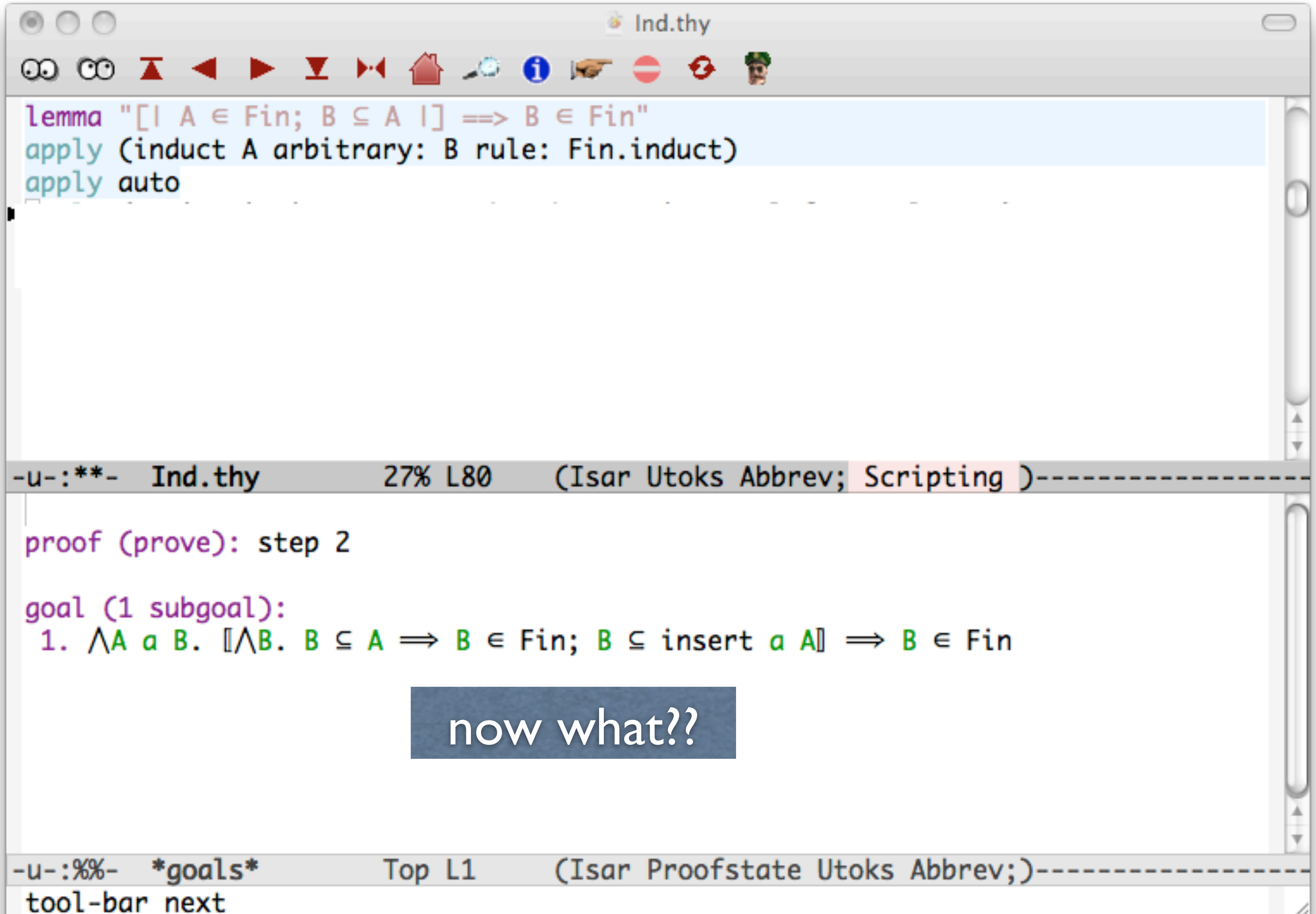
A Subset of a Finite Set



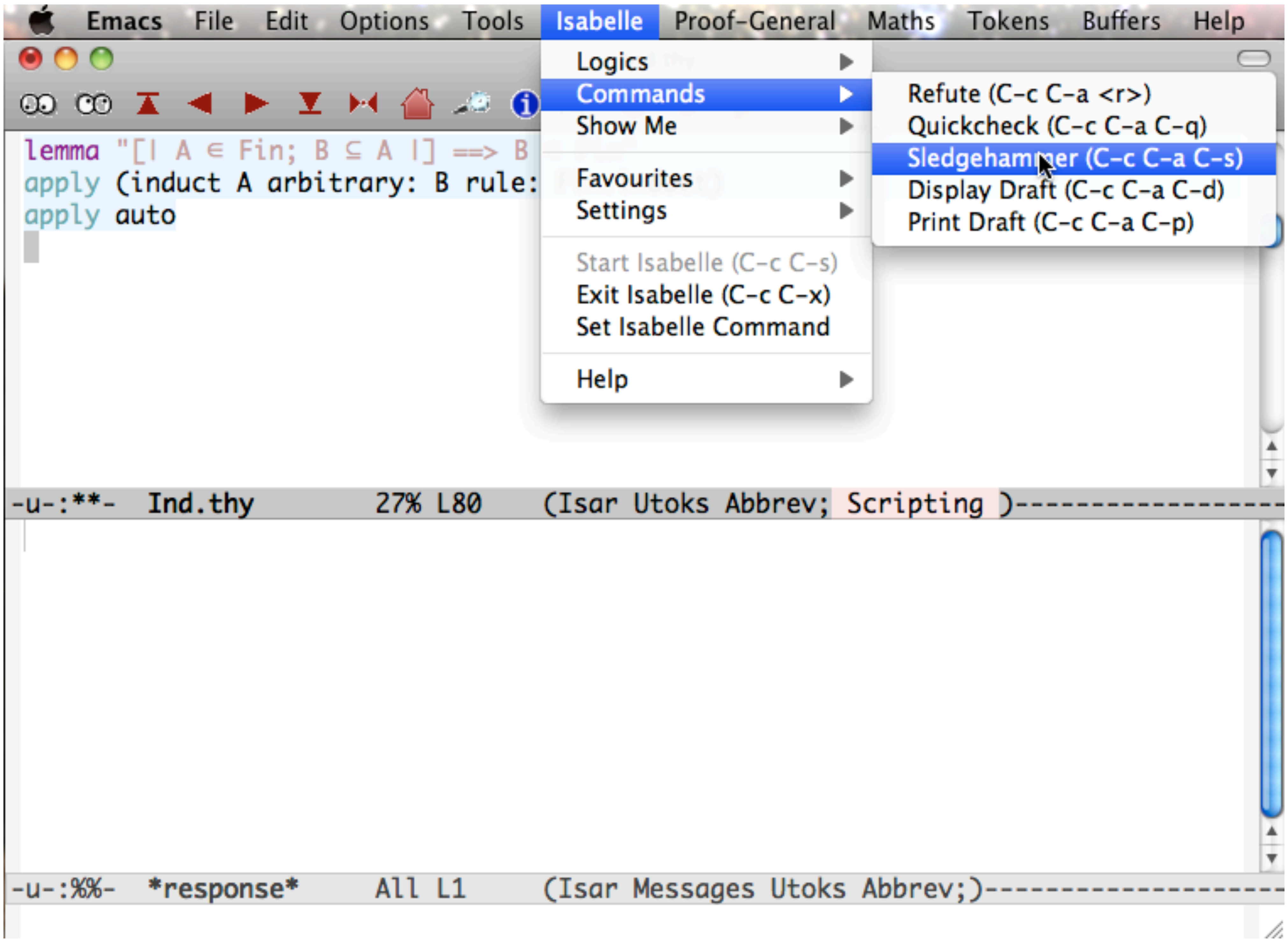
A Subset of a Finite Set



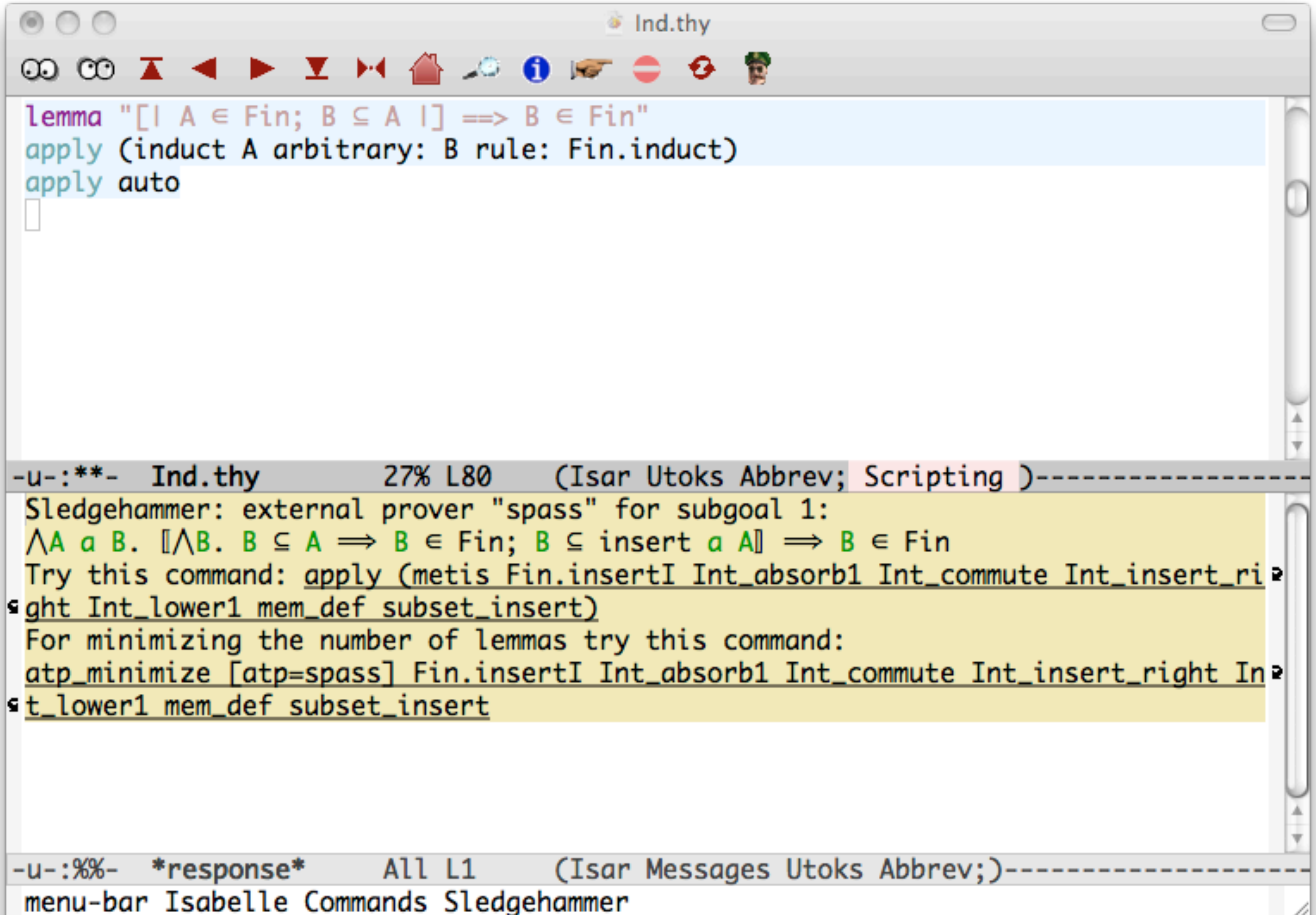
A Crucial Point in the Proof



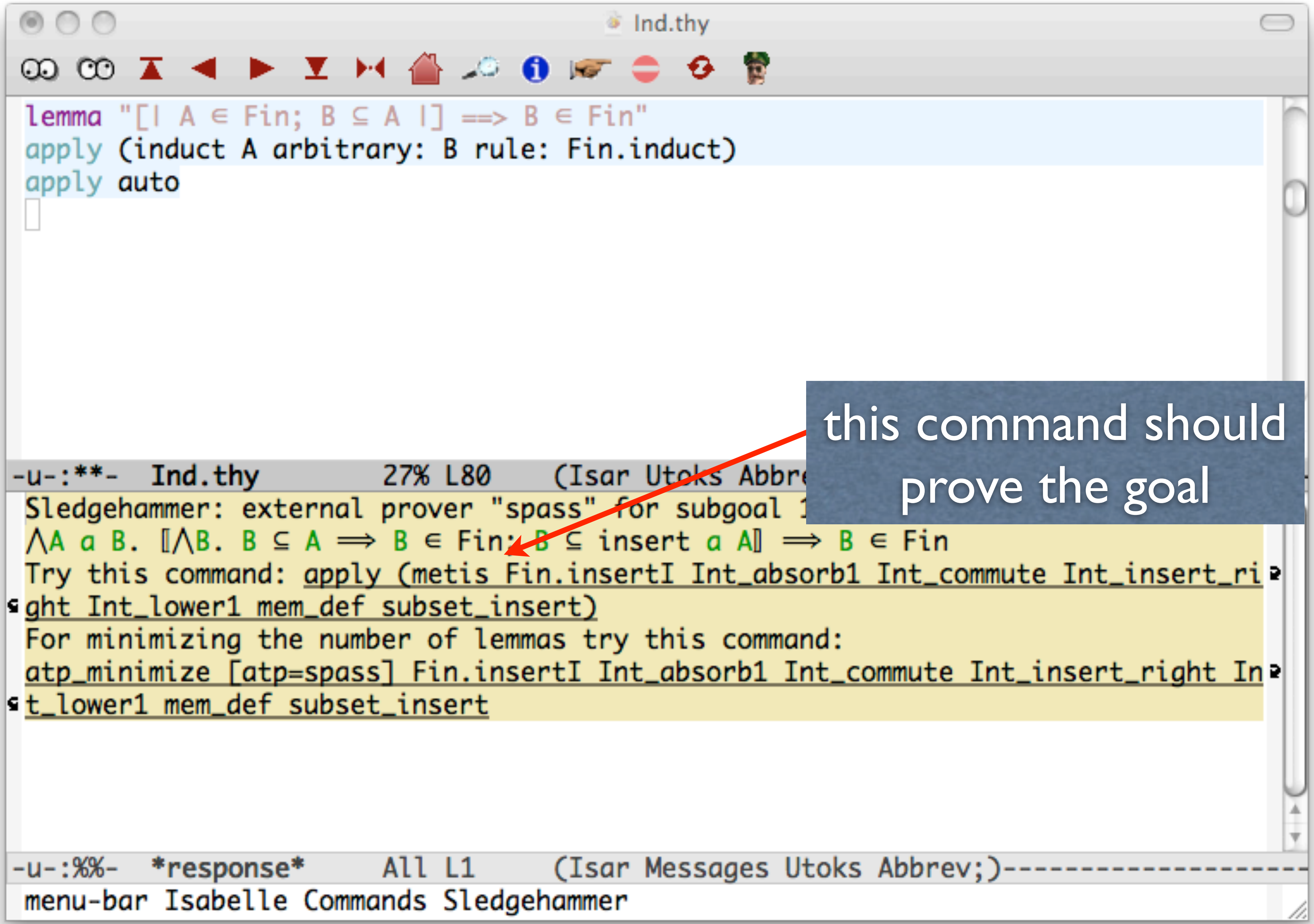
Time to Try Sledgehammer!



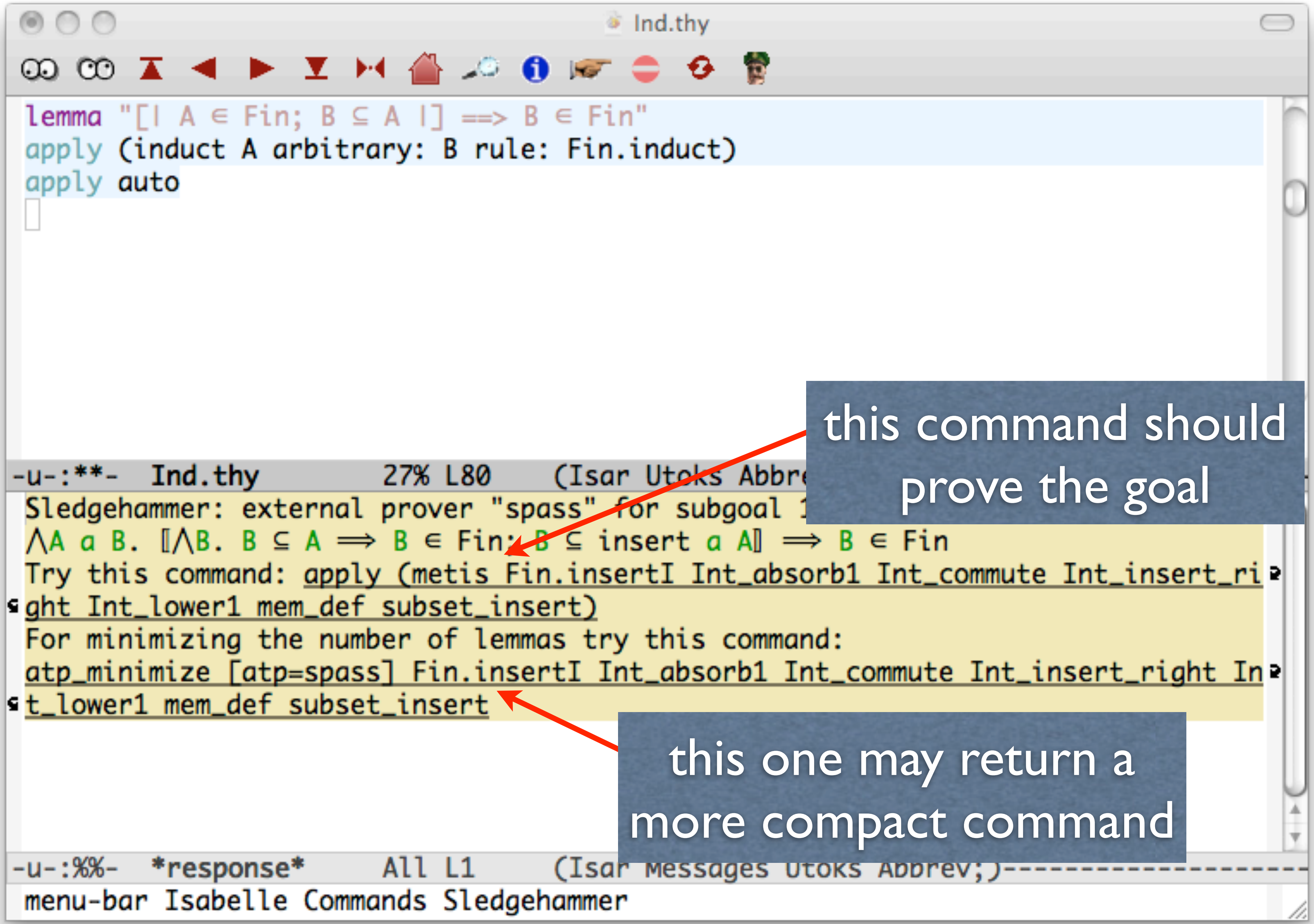
Success!


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)
Sledgehammer: external prover "spass" for subgoal 1:
⋀A a B. ⟦⋀B. B ⊆ A ⟹ B ∈ Fin; B ⊆ insert a A⟧ ⟹ B ∈ 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

Success!

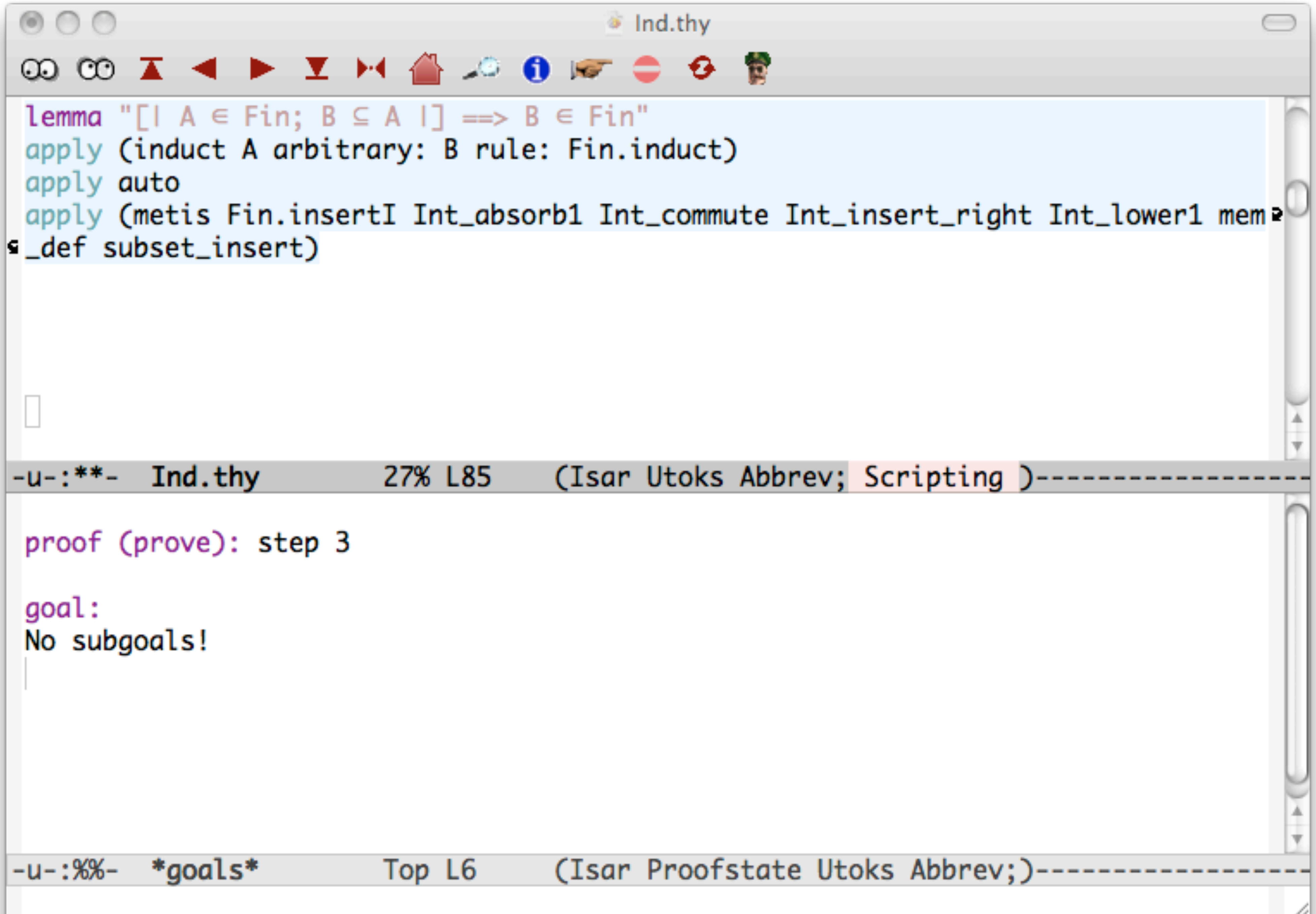

Ind.thy
lemma "[| A ∈ Fin; B ⊆ A |] ==> B ∈ Fin"
apply (induct A arbitrary: B rule: Fin.induct)
apply auto
-u-:**- Ind.thy 27% L80 (Isar Utoks Abbre
Sledgehammer: external prover "spass" for subgoal 1
⋀A a B. ⟦⋀B. B ⊆ A ⟹ B ∈ Fin; B ⊆ insert a A⟧ ⟹ B ∈ Fin
Try this command: apply (metis Fin.insertI Int_absorb1 Int_commute Int_insert_ri
ght 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 In
t_lower1 mem_def subset_insert
-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)
menu-bar Isabelle Commands Sledgehammer
this command should prove the goal

Success!



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 Abbre
Sledgehammer: external prover "spass" for subgoal 1
⋀A a B. ⟦⋀B. B ⊆ A ⟹ B ∈ Fin; B ⊆ insert a A⟧ ⟹ B ∈ 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

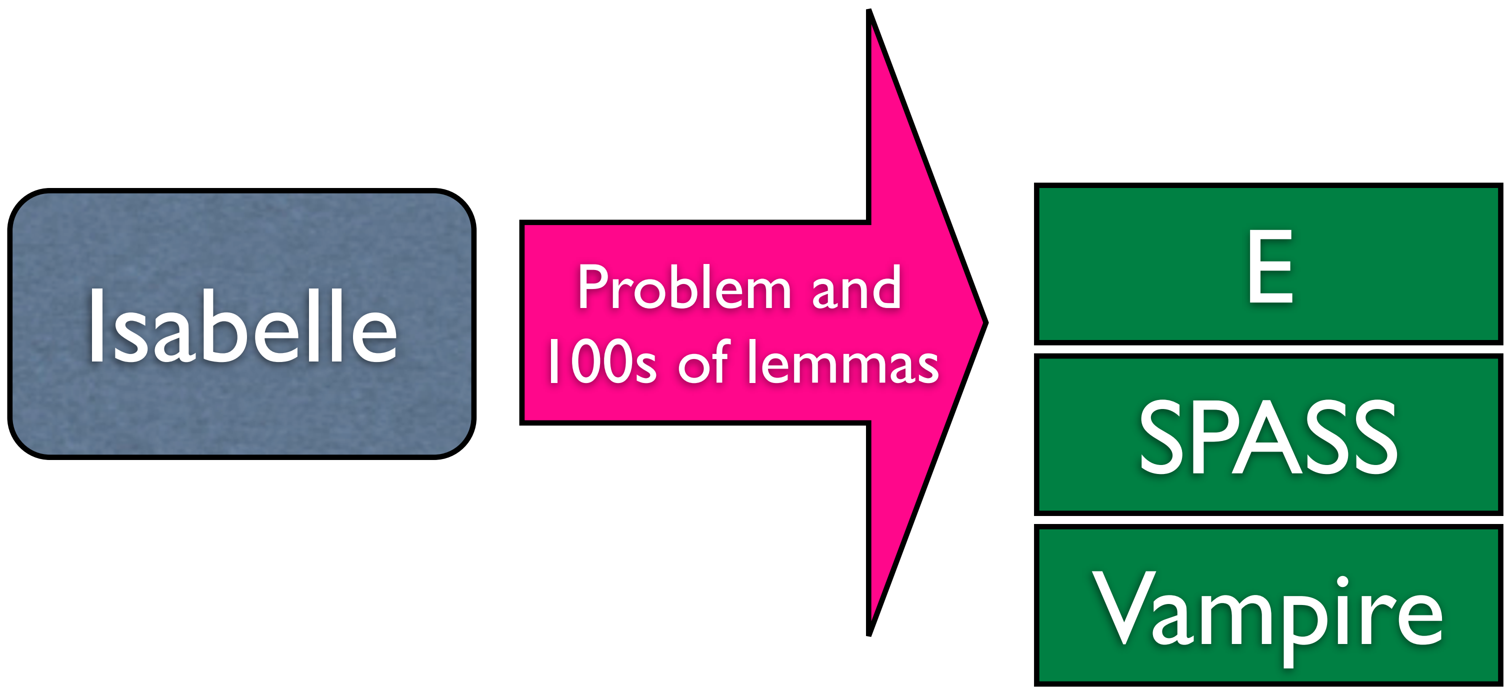
The Completed Proof



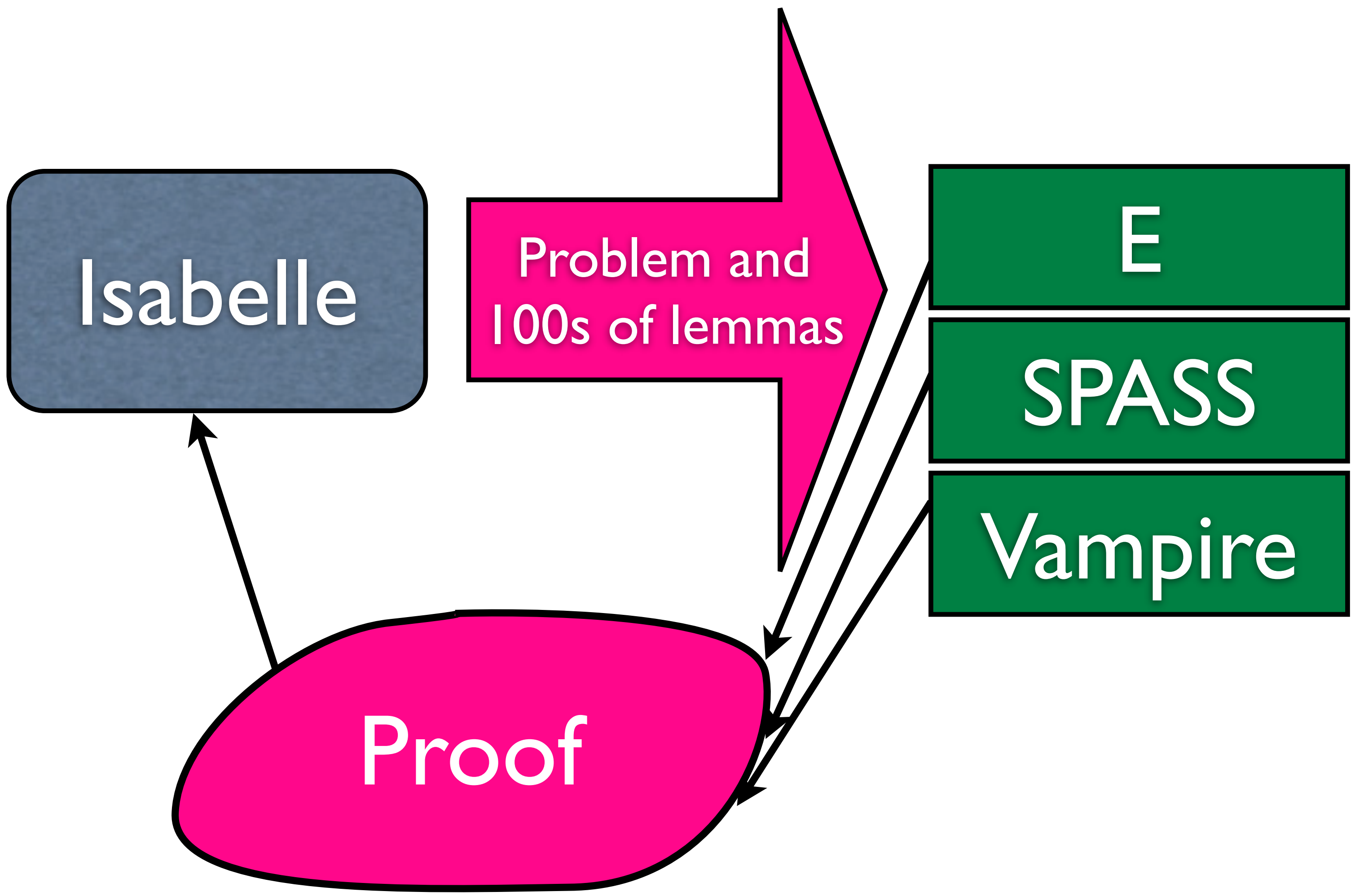
How Sledgehammer Works

Isabelle

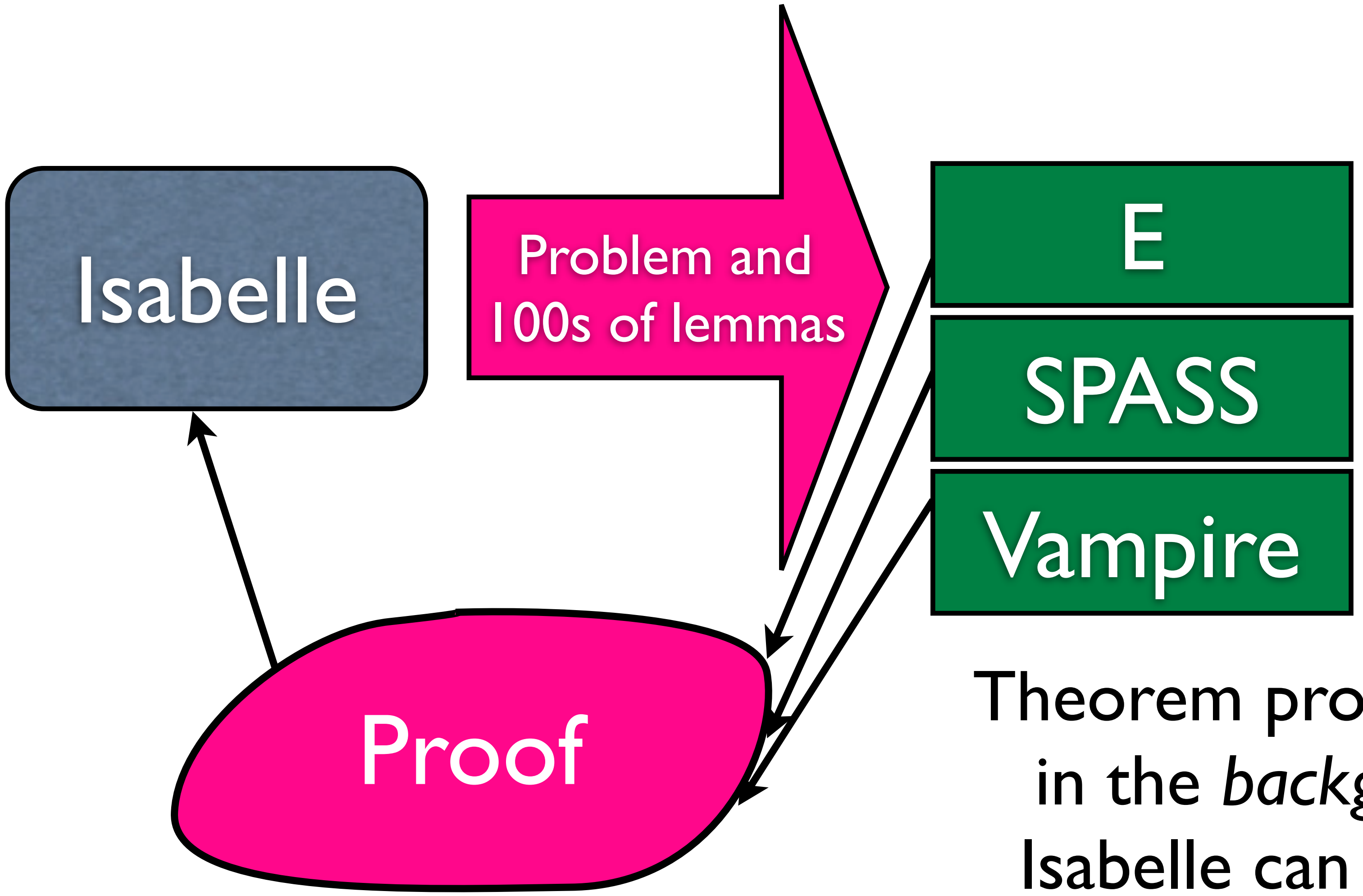
How Sledgehammer Works



How Sledgehammer Works



How Sledgehammer Works



Theorem provers run in the *background*. Isabelle can still be used!

Notes on Sledgehammer

Notes on Sledgehammer

- It is always available, though it usually fails...

Notes on Sledgehammer

- It is always available, though it usually fails...
- It does not prove the goal, but returns a call to `metis`. This command *usually* works...

Notes on Sledgehammer

- It is always available, though it usually fails...
- 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.

Notes on Sledgehammer

- It is always available, though it usually fails...
- 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.