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 `α⇒bool`
- The membership relation: ∈
- The subset relation: ⊆
  - Reflexive, anti-symmetric, transitive
- The empty set: { }
- The universal set: `UNIV`

Basic Set Theory Operations

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

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

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

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

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

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

Big Union and Intersection

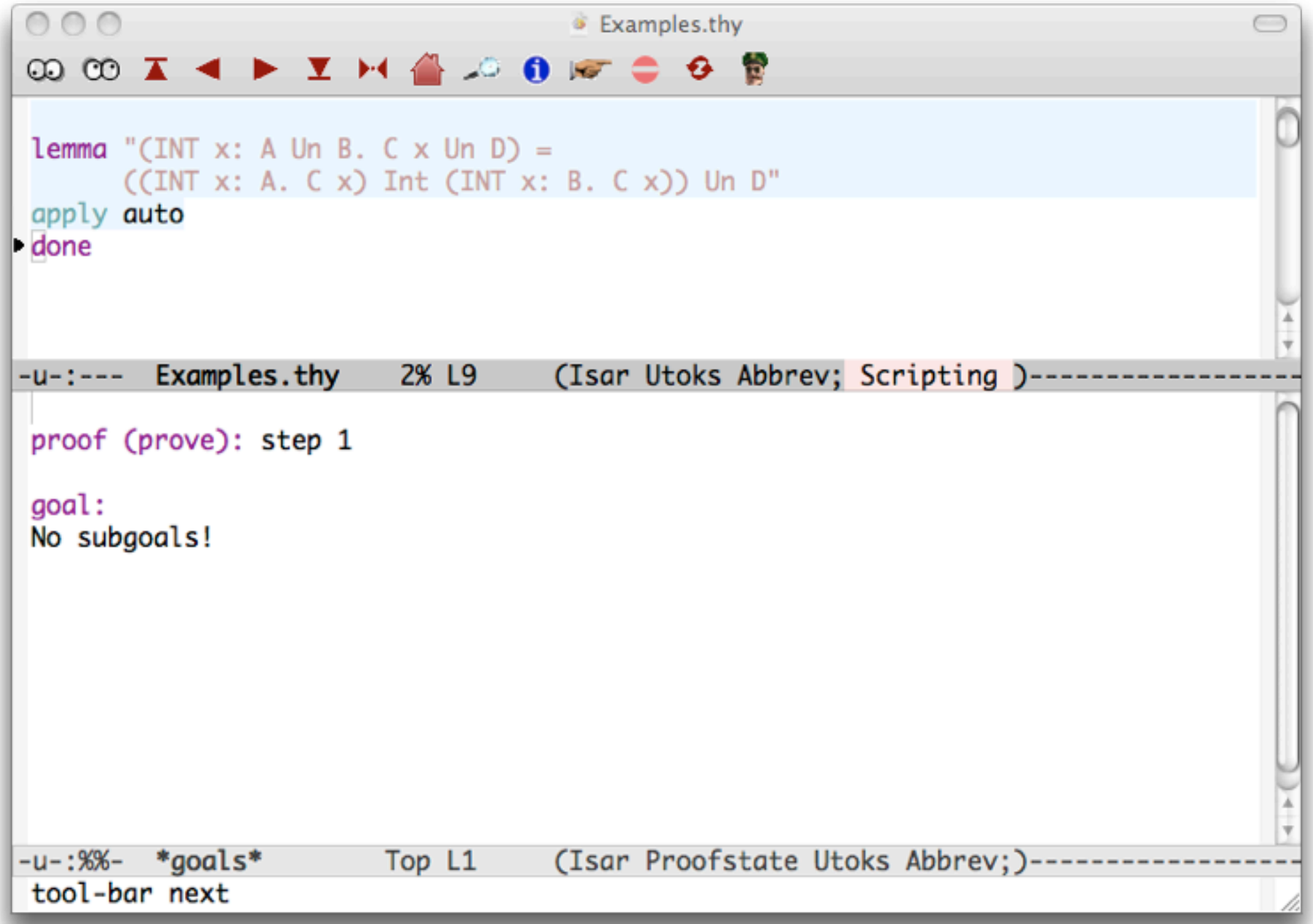
$$e \in \left(\bigcup x.\, B(x)\right) \iff \exists x.\, e \in B(x)$$

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

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

And the analogous forms of intersections...

A Simple Set Theory Proof



Finite Set Notation

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

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`

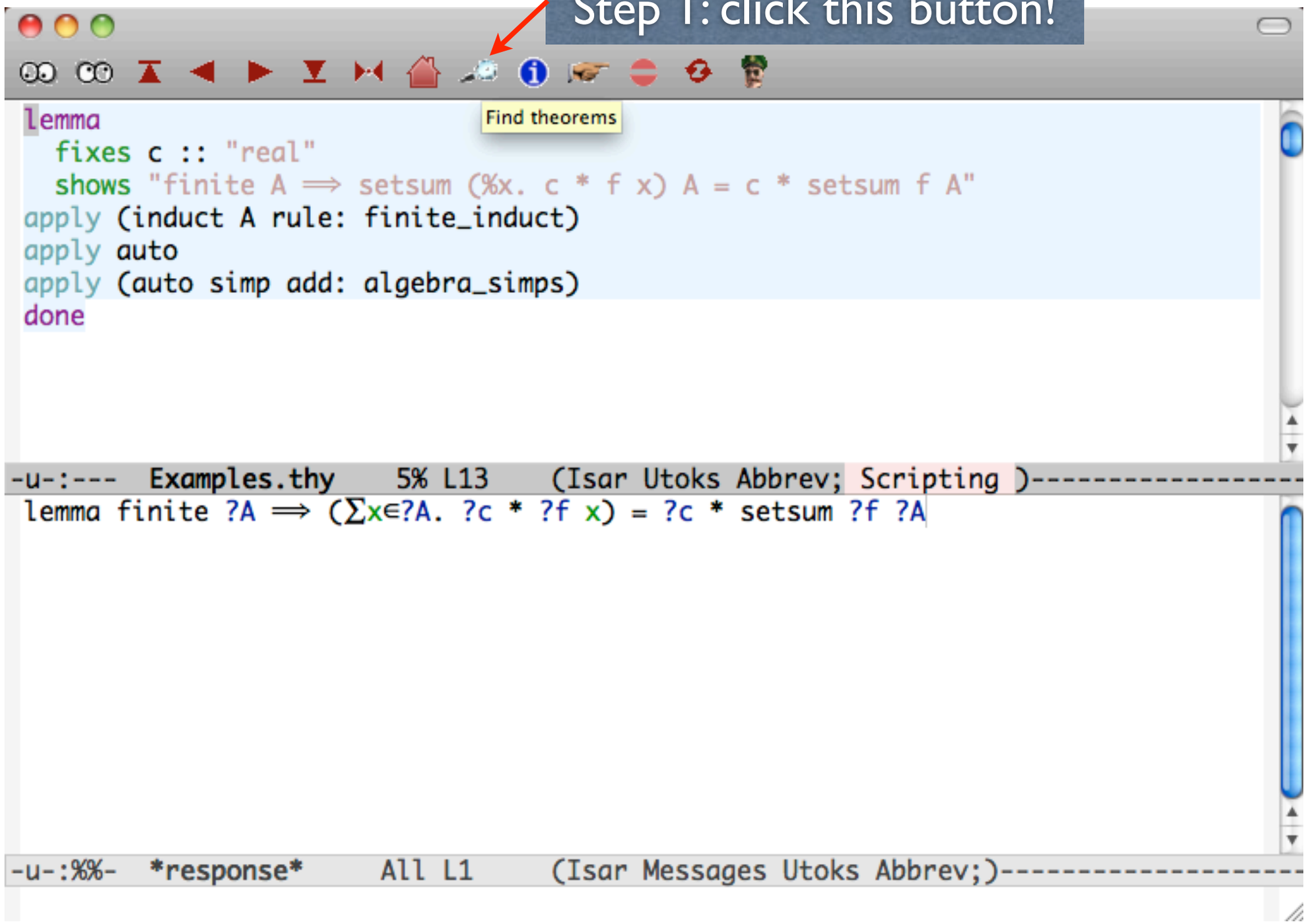
$$\texttt{finite}(A \cup B) = (\texttt{finite}\, A \wedge \texttt{finite}\, B)$$

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

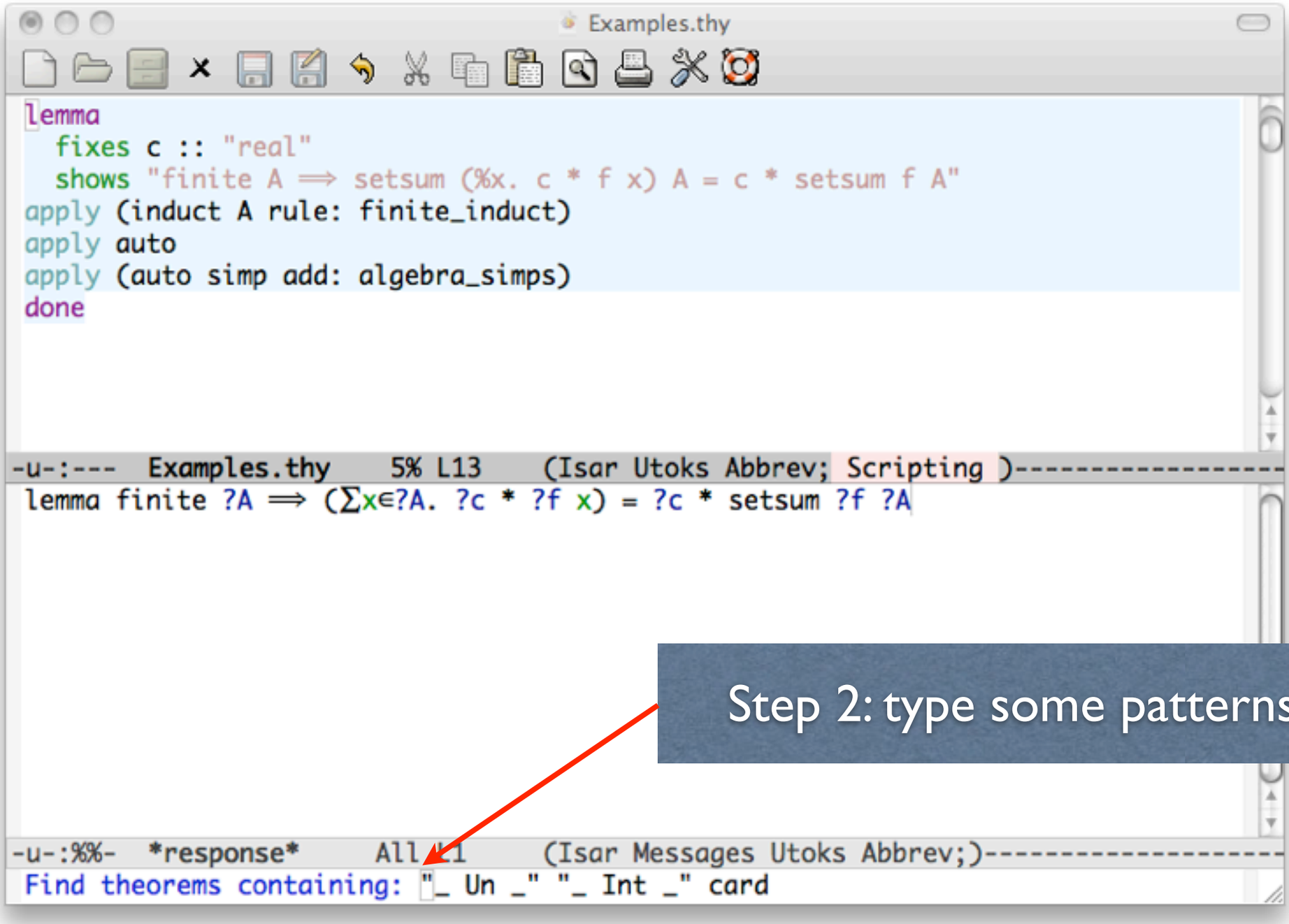
Proving Theorems about Sets

- 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.

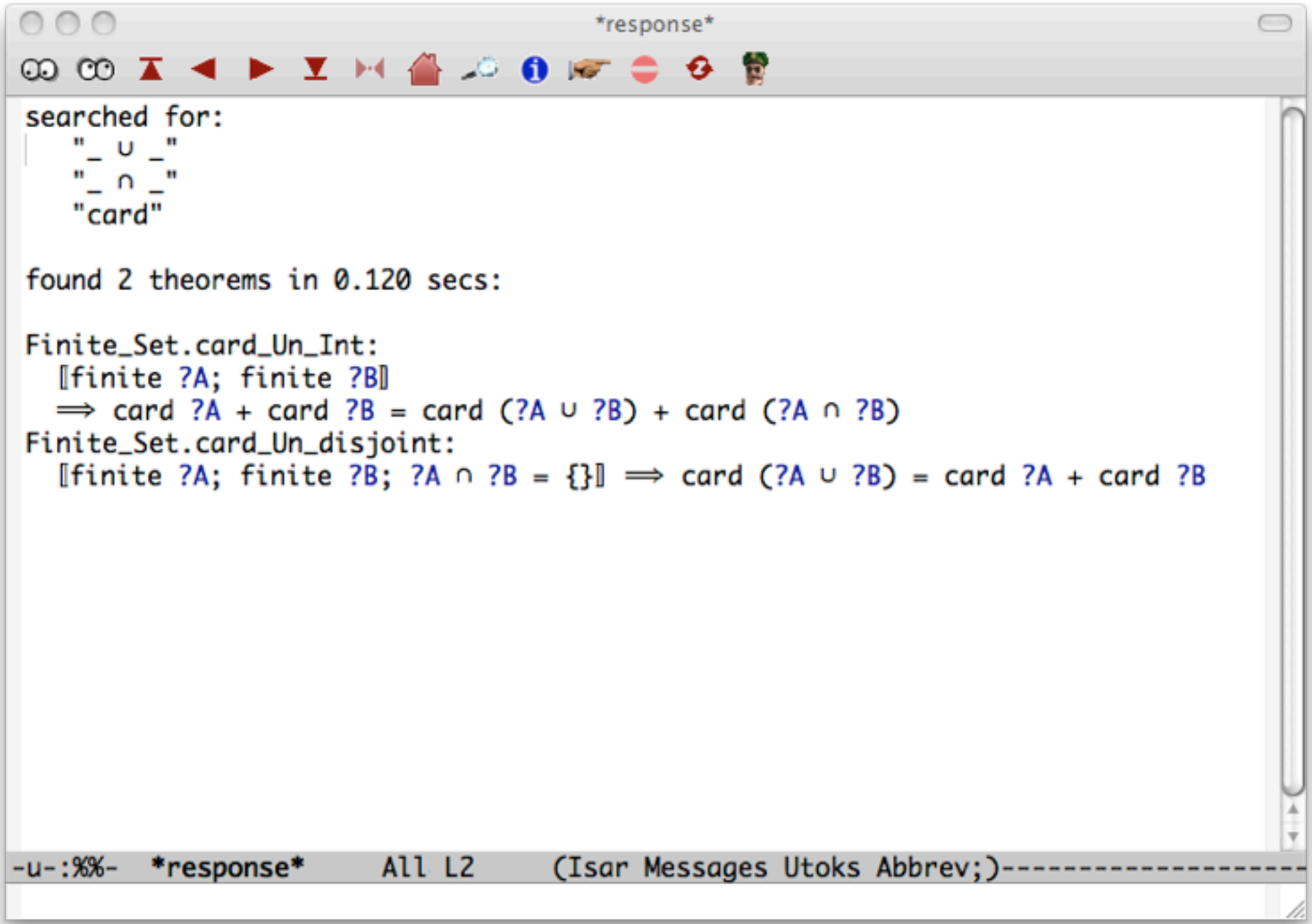
Finding Theorems about Sets



Finding Theorems about Sets



Which Theorems Were Found?

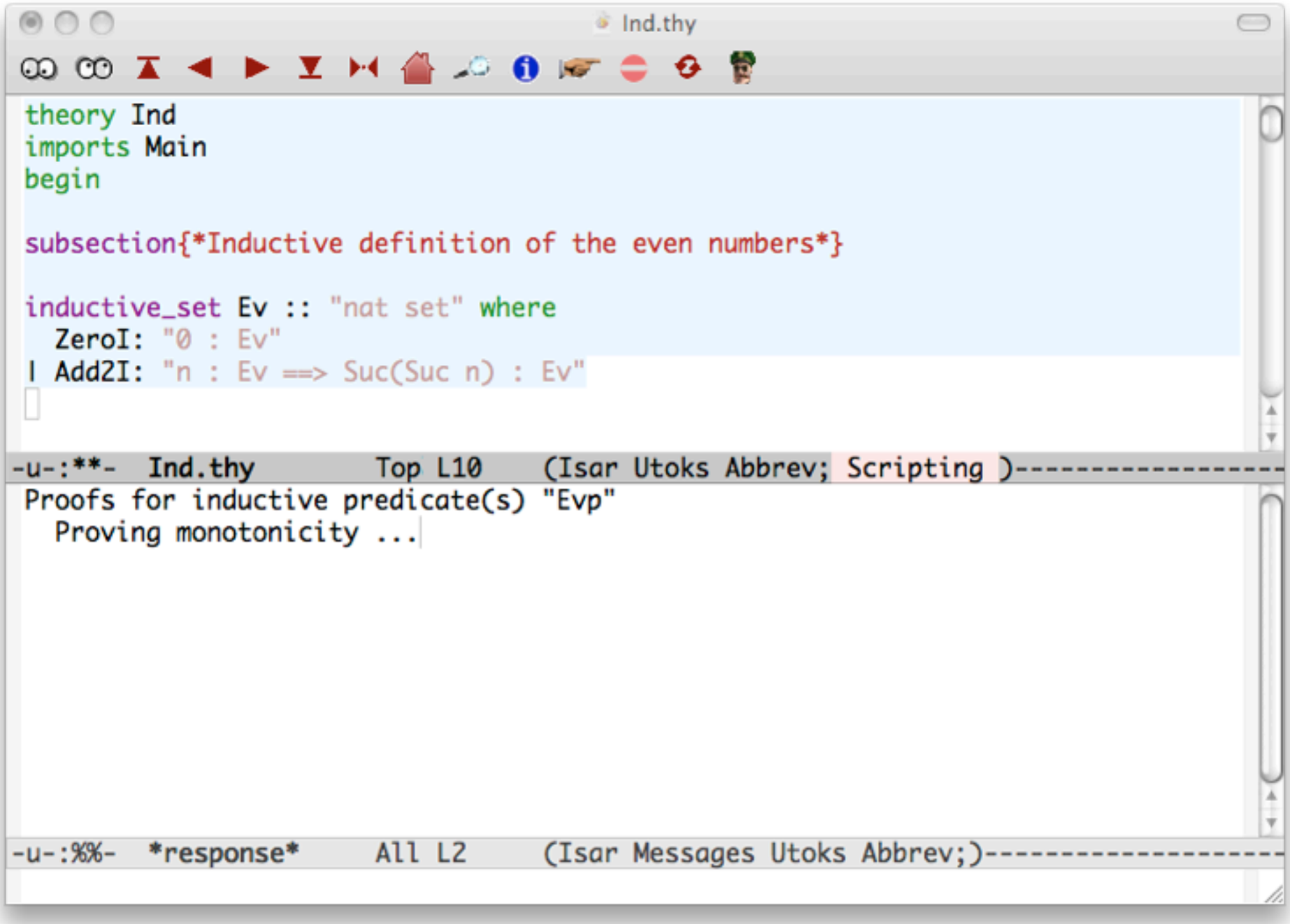


Inductively Defined Sets

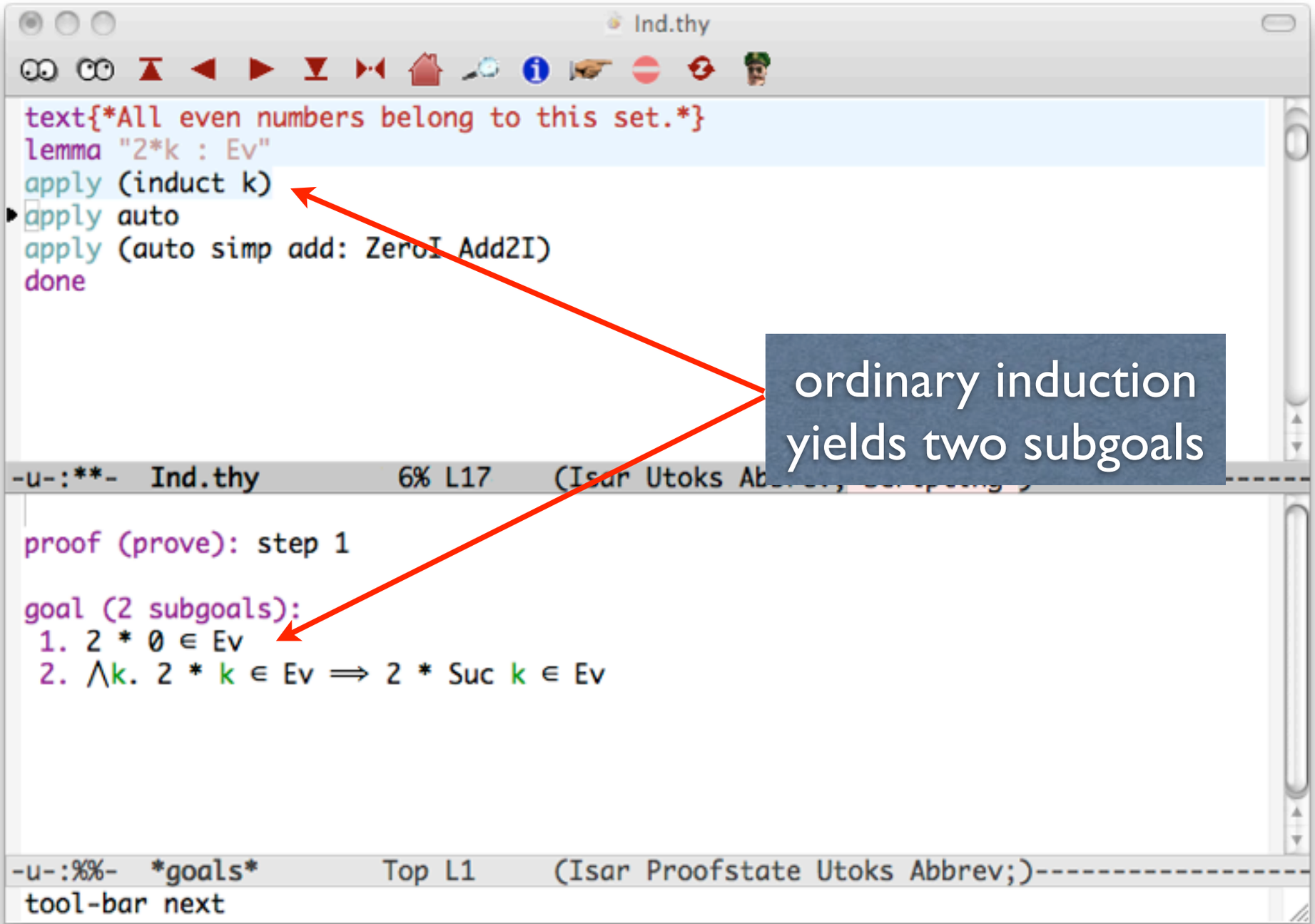
Defining a Set Inductively

- 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.

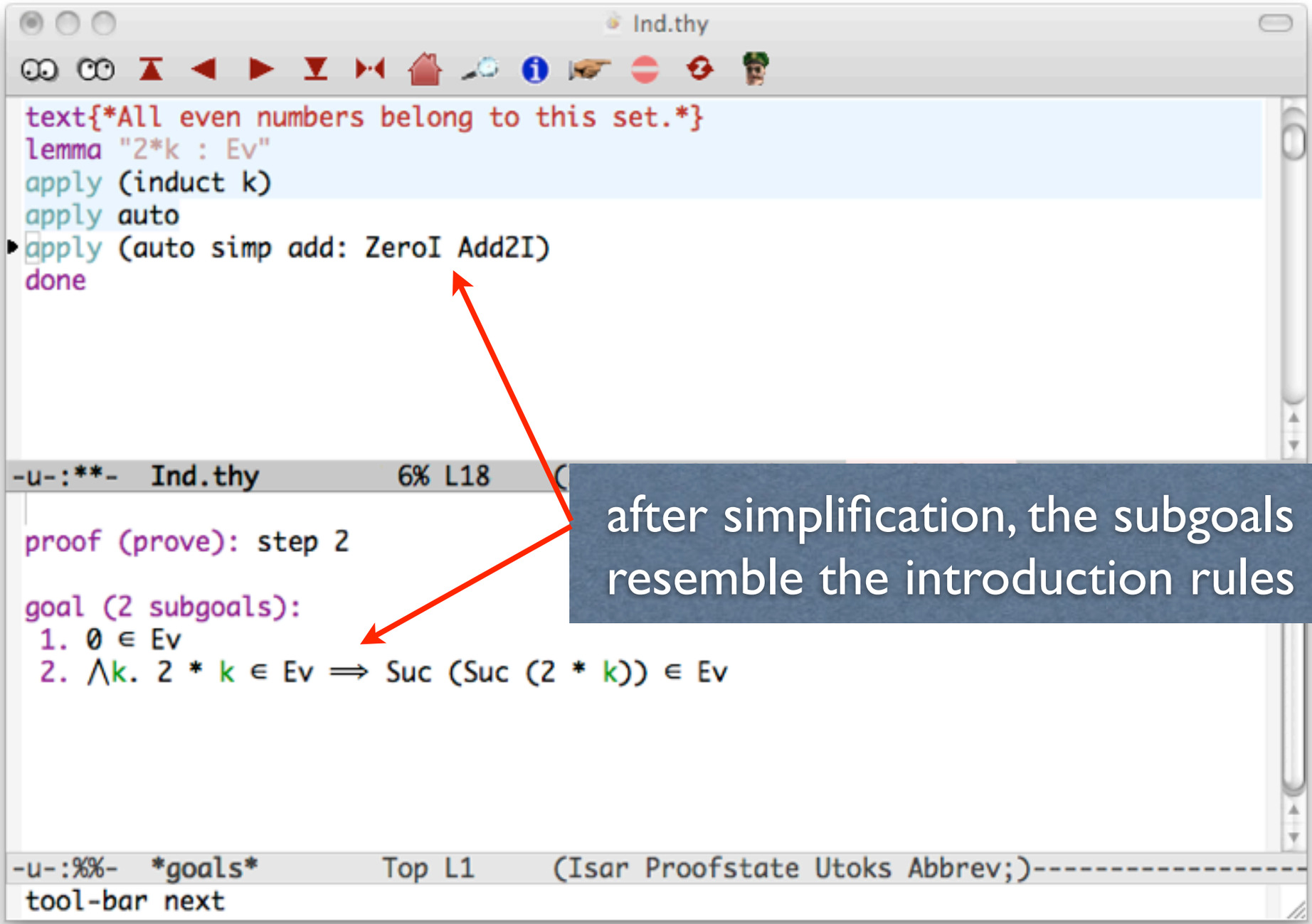
Inductive Definitions in Isabelle



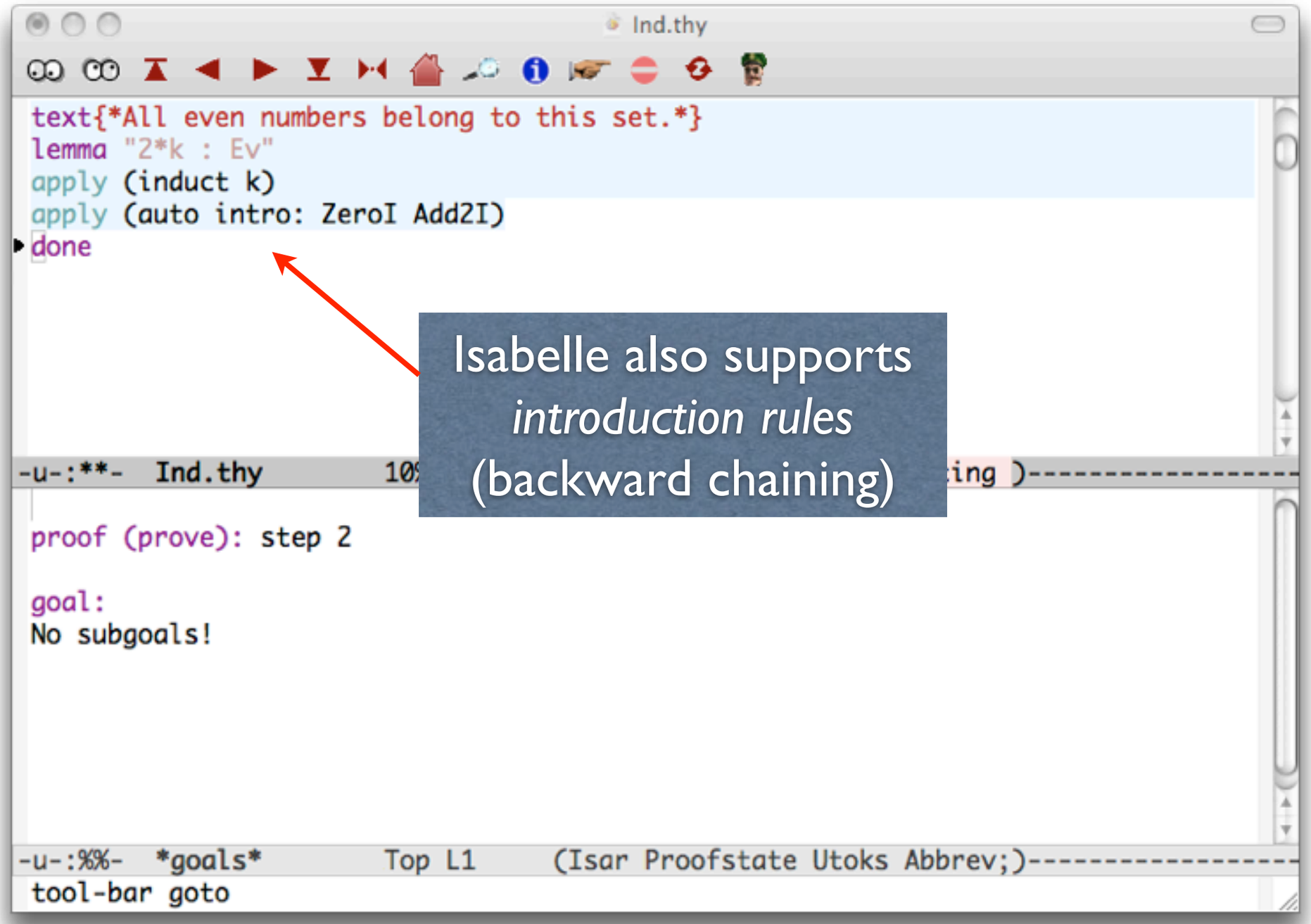
Even Numbers Belong to Ev



Proving Set Membership



Finishing the Proof

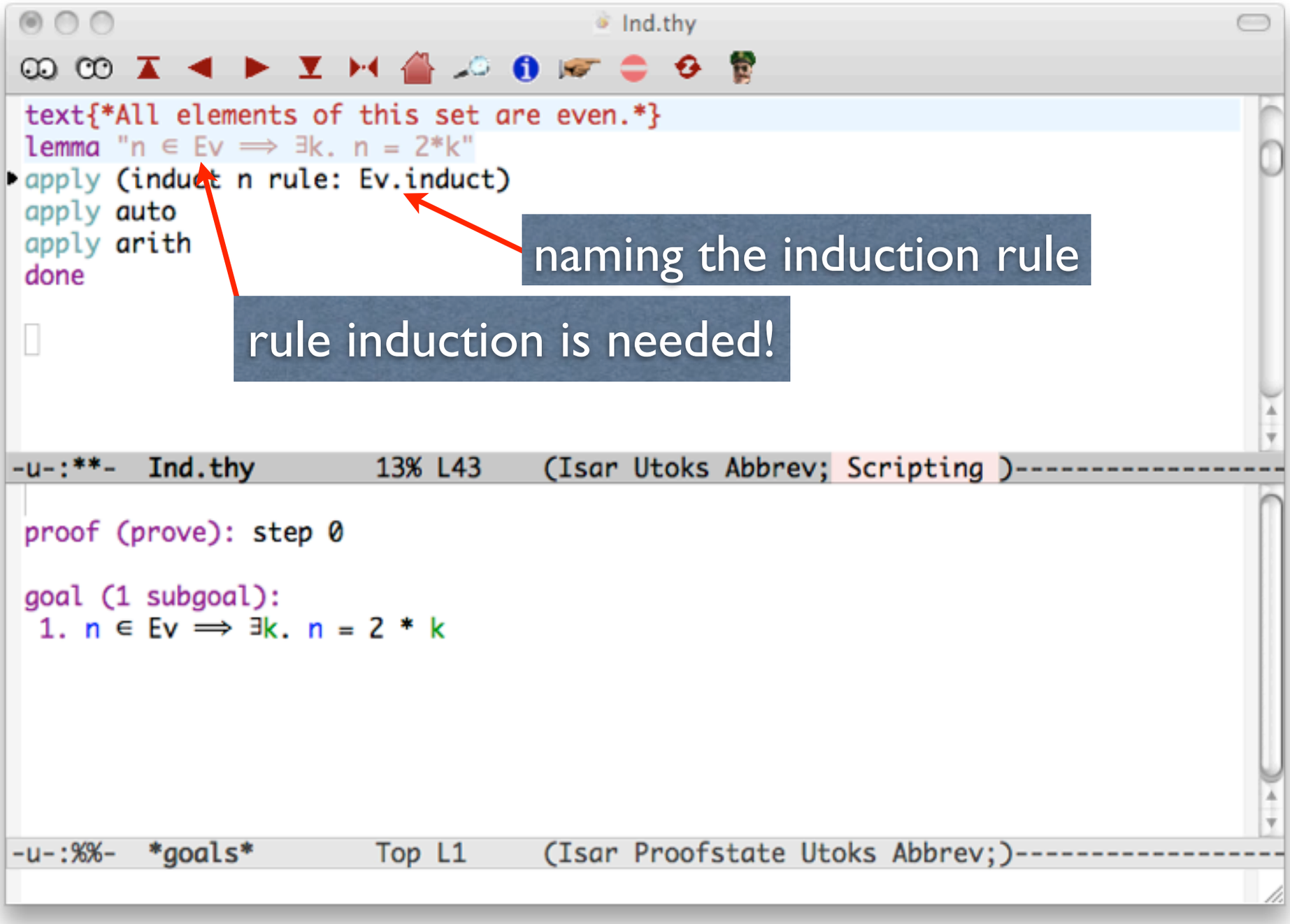


Isabelle also supports *introduction rules* (backward chaining)

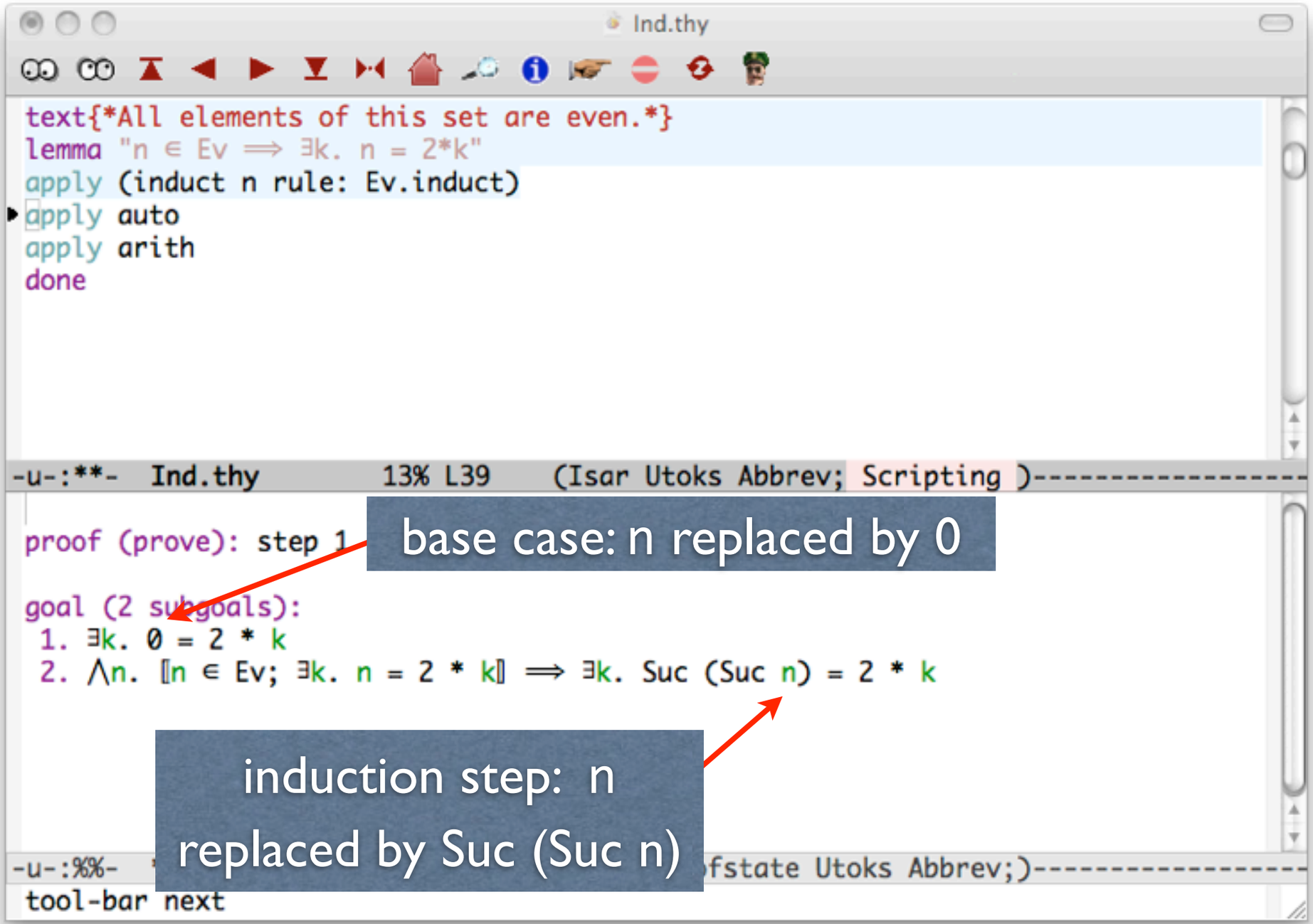
Rule Induction

- 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.

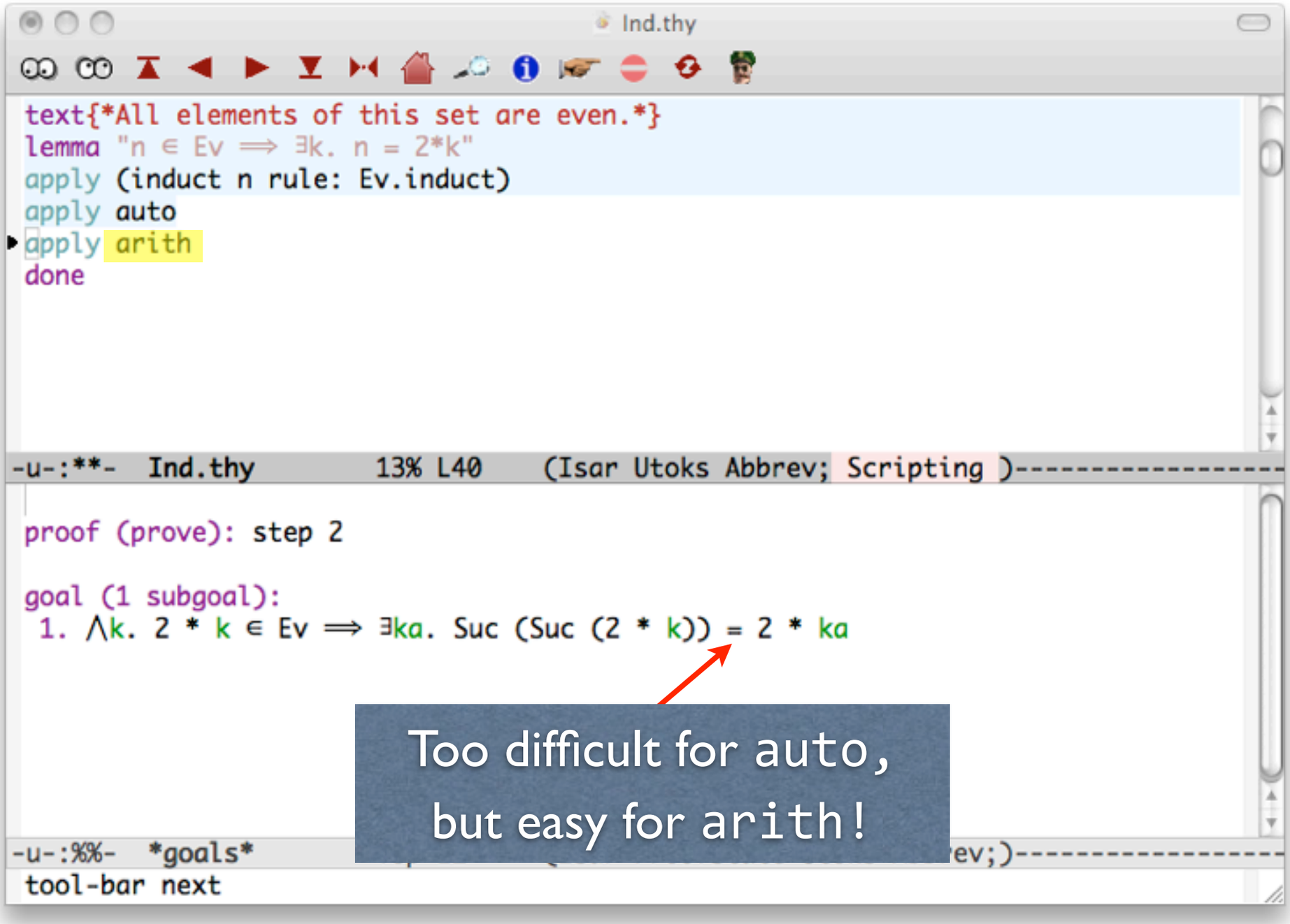
Ev Has only Even Numbers



An Example of Rule Induction

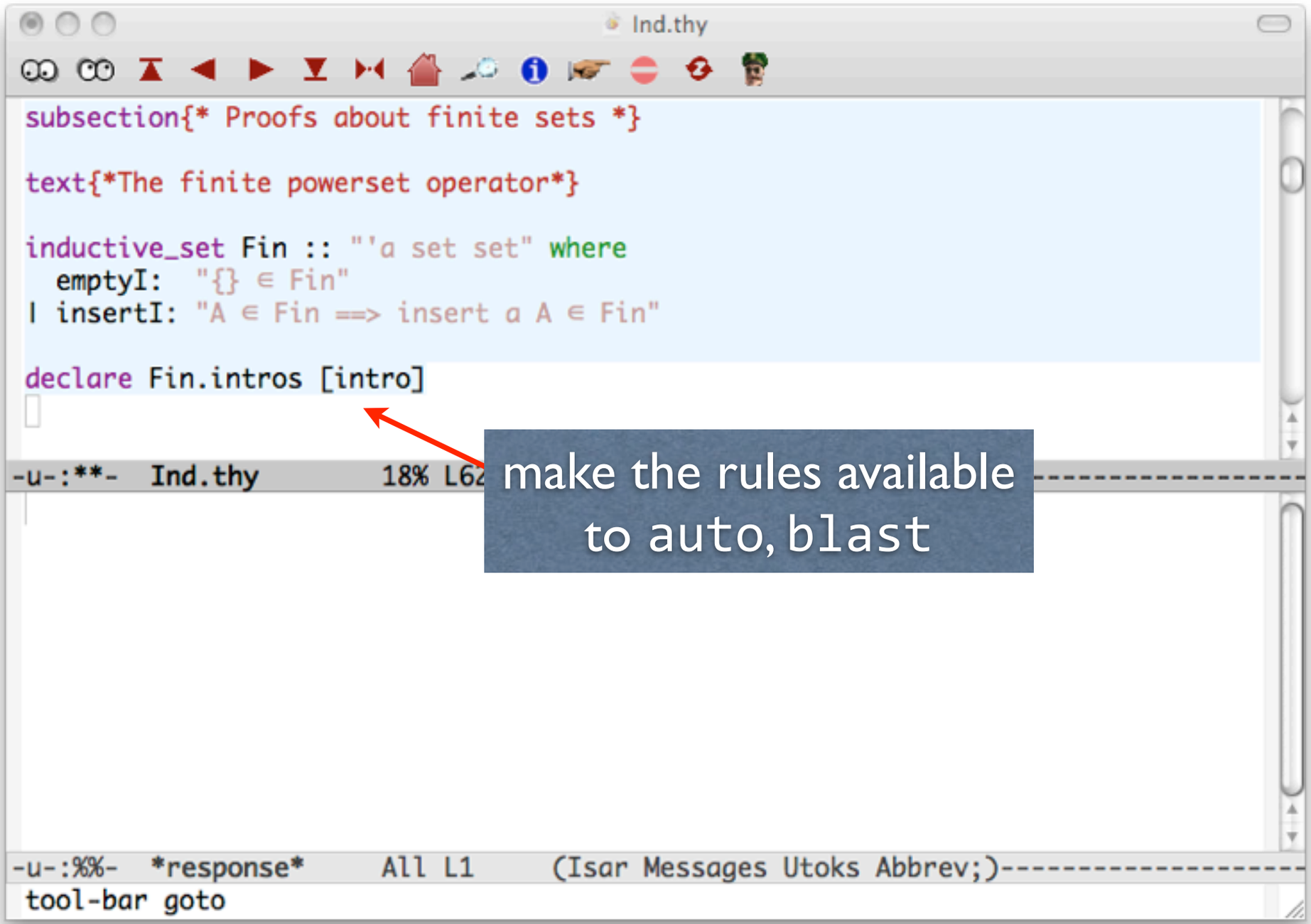


One Tricky Goal Left!

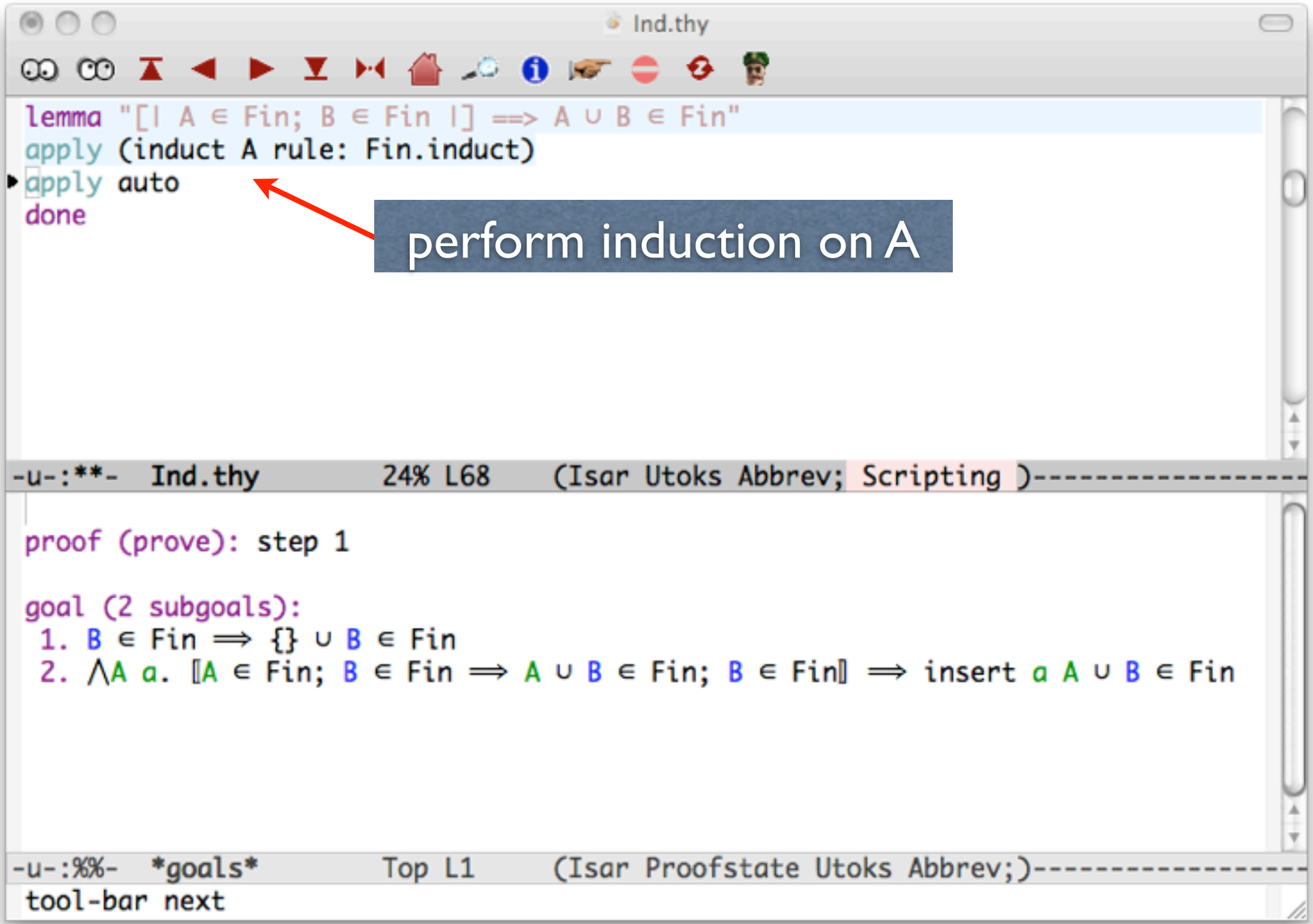


A Final Example

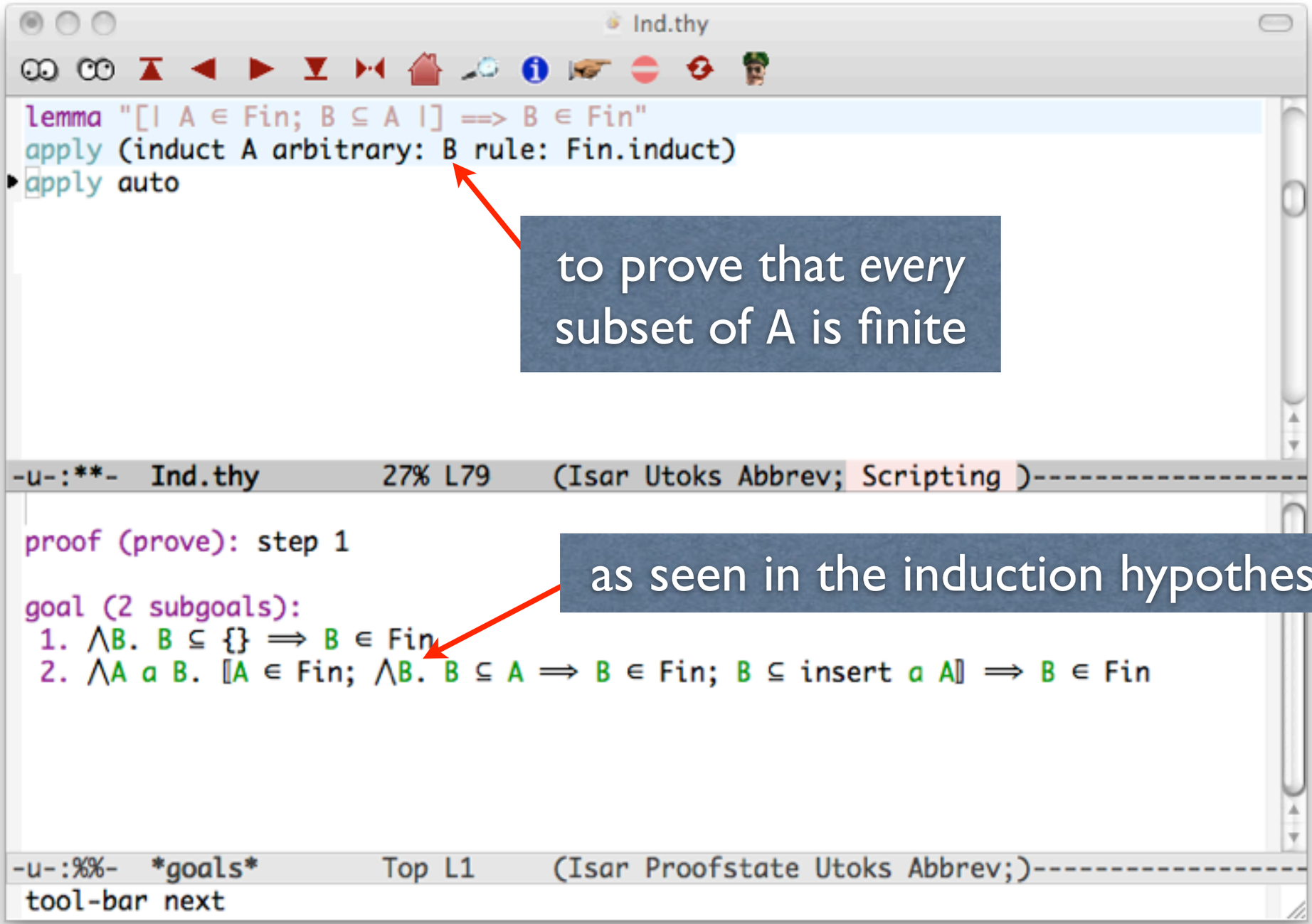
Defining Finiteness



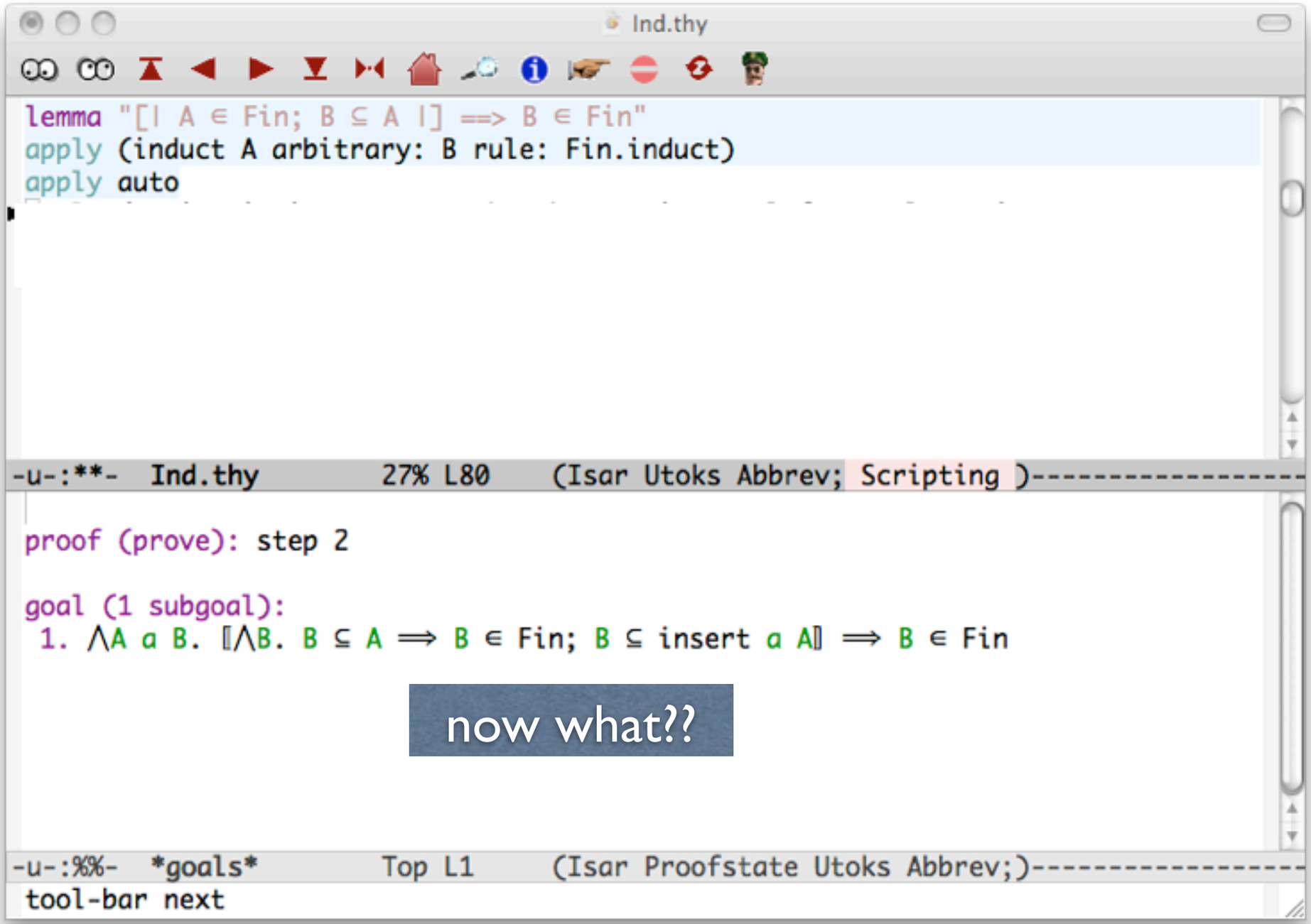
The Union of Two Finite Sets



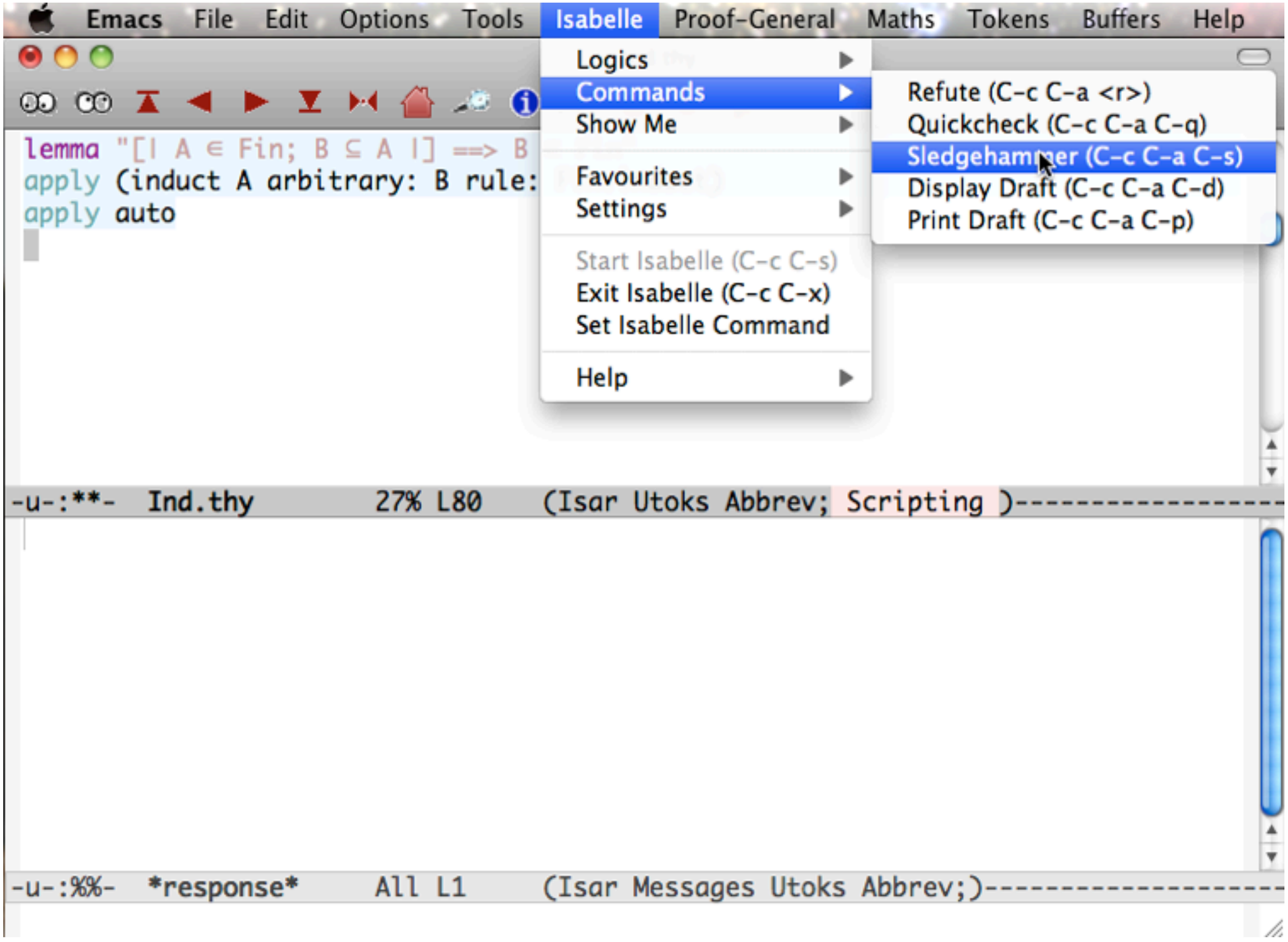
A Subset of a Finite Set



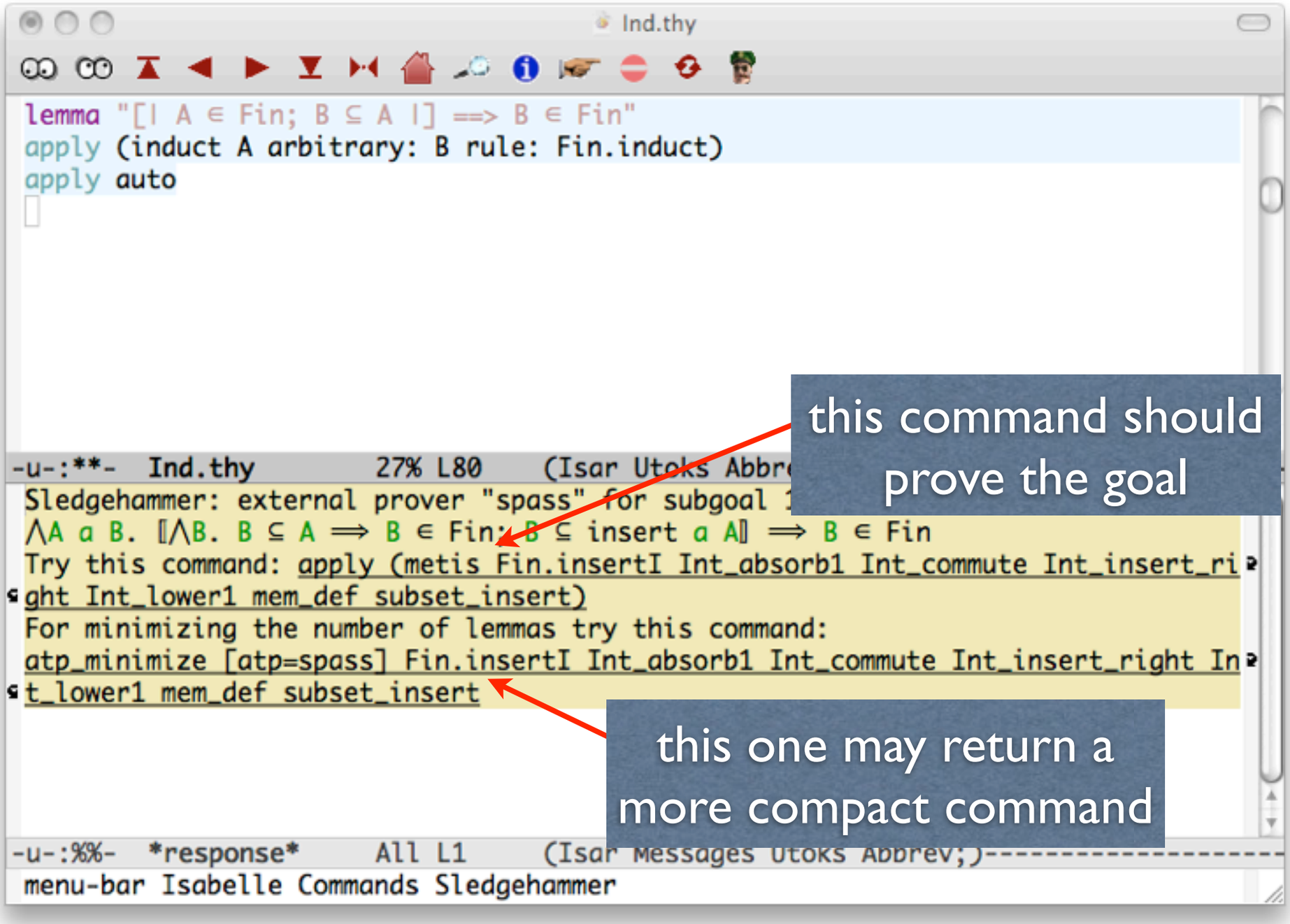
A Critical Point in the Proof



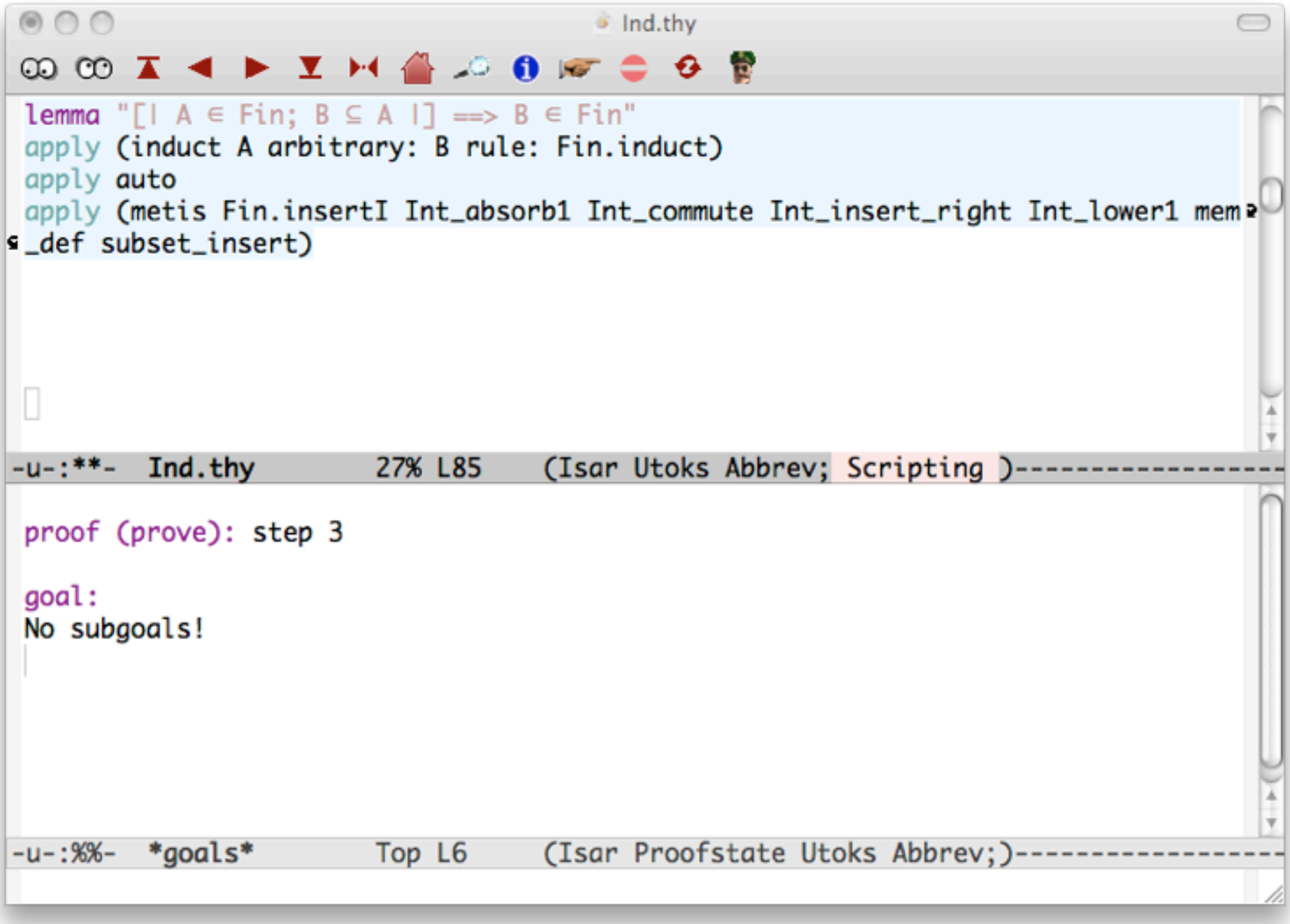
Time to Try Sledgehammer!



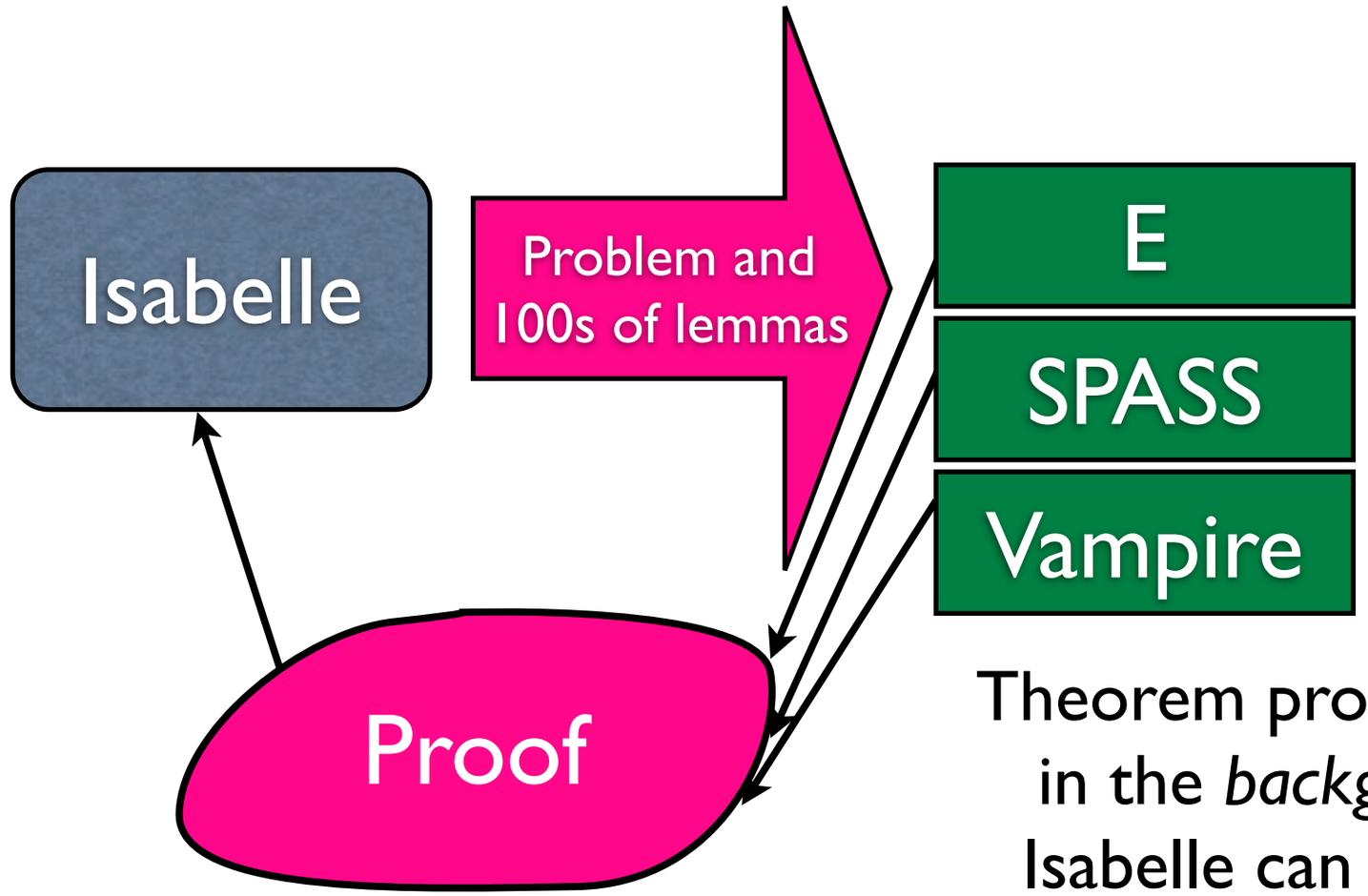
Success!


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 Abbr
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
this command should prove the goal
this one may return a more compact command
-u-:%%- *response* All L1 (Isar Messages Utoks Abbrev;)
menu-bar Isabelle Commands Sledgehammer

The Completed Proof



How Sledgehammer Works



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

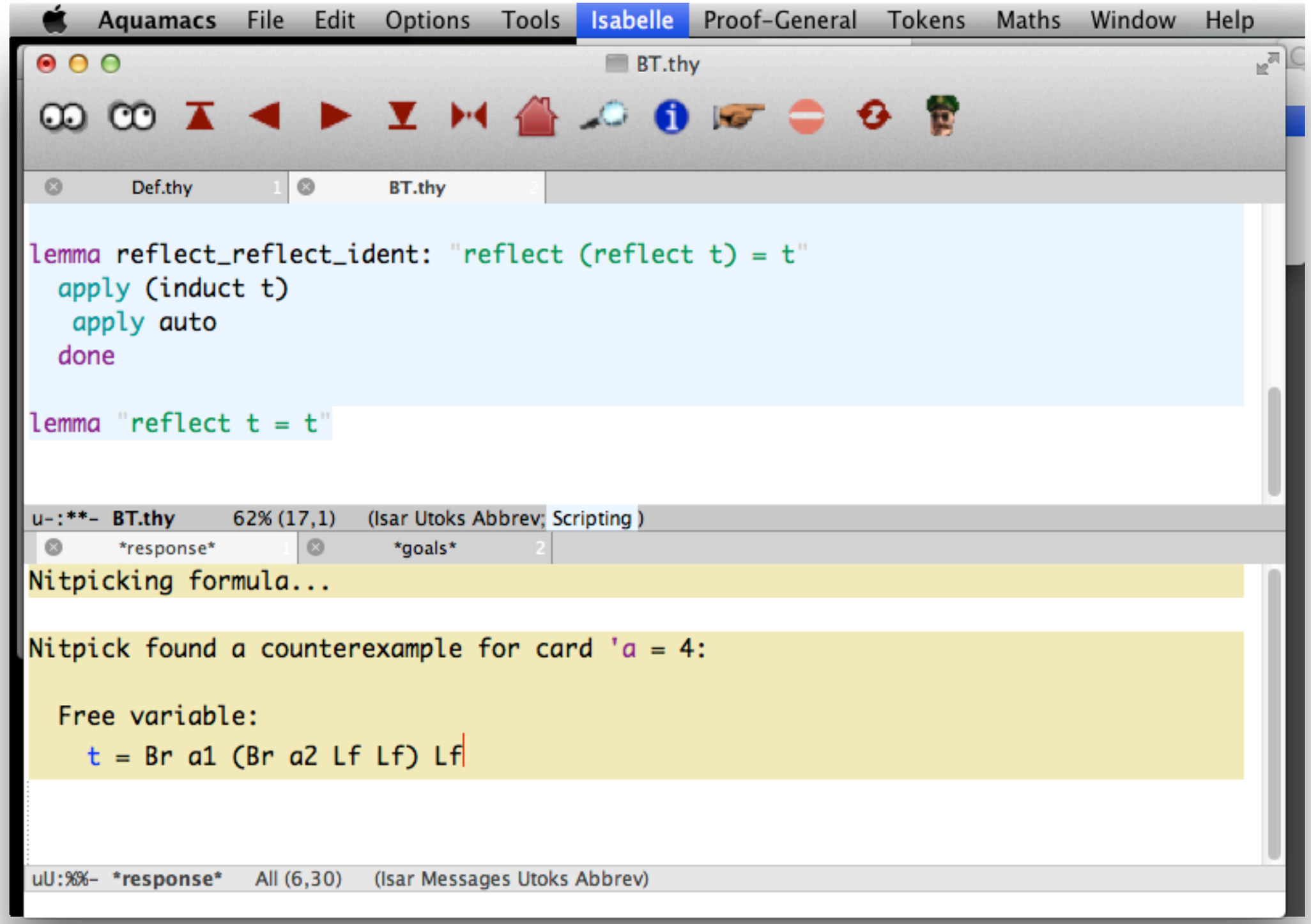
Notes on Sledgehammer

- 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



Other Things to Learn

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