

Interactive Formal Verification

2: Isabelle Theories

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

A Tiny Theory

```
theory BT imports Main begin


datatype 'a bt =
  Lf
| Br 'a "'a bt" "'a bt"

fun reflect :: "'a bt => 'a bt" where
  "reflect Lf = Lf"
| "reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)"

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

end
```

name of the
new theory



A Tiny Theory

```
theory BT imports Main begin

datatype 'a bt =
  Lf
| Br 'a "'a bt" "'a bt"

fun reflect :: "'a bt => 'a bt" where
  "reflect Lf = Lf"
| "reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)"

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

end
```

name of the
new theory

A Tiny Theory

```
theory BT imports Main begin
```

```
datatype 'a bt =  
  Lf  
| Br 'a "'a bt" "'a bt"
```

```
fun reflect :: "'a bt => 'a bt" where  
  "reflect Lf = Lf"  
| "reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)"
```

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

```
end
```

the theory it builds upon

name of the
new theory

A Tiny Theory

```
theory BT imports Main begin
```

```
datatype 'a bt =
```

```
  Lf
```

```
  | Br 'a "'a bt" "'a bt"
```

```
fun reflect :: "'a bt => 'a bt" where
```

```
  "reflect Lf = Lf"
```

```
  | "reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)"
```

```
lemma reflect_reflect_ident: "reflect (reflect t) = t"
```

```
  apply (induct t)
```

```
    apply auto
```

```
  done
```

```
end
```

the theory it builds upon

declarations of types,
constants, etc

name of the
new theory

A Tiny Theory

```
theory BT imports Main begin
```

```
datatype 'a bt =  
  Lf  
| Br 'a "'a bt" "'a bt"
```

```
fun reflect :: "'a bt => 'a bt" where  
  "reflect Lf = Lf"  
| "reflect (Br a t1 t2) = Br a (reflect t2) (reflect t1)"
```

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

```
end
```

the theory it builds upon

declarations of types,
constants, etc

proving a theorem

Notes on Theory Structure

Notes on Theory Structure

- A theory can *import* any existing theories.

Notes on Theory Structure

- A theory can *import* any existing theories.
- Types, constants, etc., must be *declared before use*.

Notes on Theory Structure

- A theory can *import* any existing theories.
- Types, constants, etc., must be *declared before use*.
- The various declarations and proofs may otherwise appear in any order.

Notes on Theory Structure

- A theory can *import* any existing theories.
- Types, constants, etc., must be *declared before use*.
- The various declarations and proofs may otherwise appear in any order.
- Many declarations can be confined to *local scopes*.

Notes on Theory Structure

- A theory can *import* any existing theories.
- Types, constants, etc., must be *declared before use*.
- The various declarations and proofs may otherwise appear in any order.
- Many declarations can be confined to *local scopes*.
- A finished theory can be imported by others.

Some Fancy Type Declarations

```
typedecl loc    -- "an unspecified type of locations"

type_synonym val    = nat -- "values"
type_synonym state  = "loc => val"
type_synonym aexp   = "state => val"
type_synonym bexp   = "state => bool" -- "functions on states"

datatype
  com = SKIP
      | Assign loc aexp          ("_ ::= _" 60)
      | Semi    com com         ("_ ; _" [60, 60] 10)
      | Cond    bexp com com    ("IF _ THEN _ ELSE _" 60)
      | While   bexp com        ("WHILE _ DO _" 60)
```

Some Fancy Type Declarations

```
typedecl loc -- "an unspecified type of locations"
```

```
type_synonym val = nat -- "values"
```

```
type_synonym state = "loc => val"
```

```
type_synonym aexp = "state => val"
```

```
type_synonym bexp = "state => bool" -- "functions on states"
```



new basic types

```
datatype
```

```
  com = SKIP
```

```
    | Assign loc aexp      ("_ ::= _" 60)
    | Semi   com com      ("_ ; _" [60, 60] 10)
    | Cond   bexp com com ("IF _ THEN _ ELSE _" 60)
    | While  bexp com      ("WHILE _ DO _" 60)
```

Some Fancy Type Declarations

```
typedecl loc -- "an unspecified type of locations"
```

```
type_synonym val = nat -- "values"
```

```
type_synonym state = "loc => val"
```

```
type_synonym aexp = "state => val"
```

```
type_synonym bexp = "state => bool" -- "functions on states"
```

new basic types

```
datatype
```

```
  com = SKIP
```

```
    | Assign loc aexp
```

```
    | Semi    com com
```

```
    | Cond   bexp com com
```

```
    | While  bexp com
```

```
( "_ ::= _ " 60 )
```

```
( "_; _" [60, 60] 10 )
```

```
( "IF _ THEN _ ELSE _" 60 )
```

```
( "WHILE _ DO _" 60 )
```

concrete syntax for commands

Some Fancy Type Declarations

```
typedecl loc -- "an unspecified type of locations"
```

```
type_synonym val = nat -- "values"
```

```
type_synonym state = "loc => val"
```

```
type_synonym aexp = "state => val"
```

```
type_synonym bexp = "state => bool" -- "functions on states"
```

new basic types

concrete syntax for commands

```
datatype
```

```
com = SKIP
```

```
| Assign loc aexp  
| Semi com com  
| Cond bexp com com  
| While bexp com
```

```
("_ ::= _" 60)  
("_; _" [60, 60] 10)  
("IF _ THEN _ ELSE _" 60)  
("WHILE _ DO _" 60)
```

recursive type of commands

Notes on Type Declarations

Notes on Type Declarations

- Type synonyms merely introduce *abbreviations*.

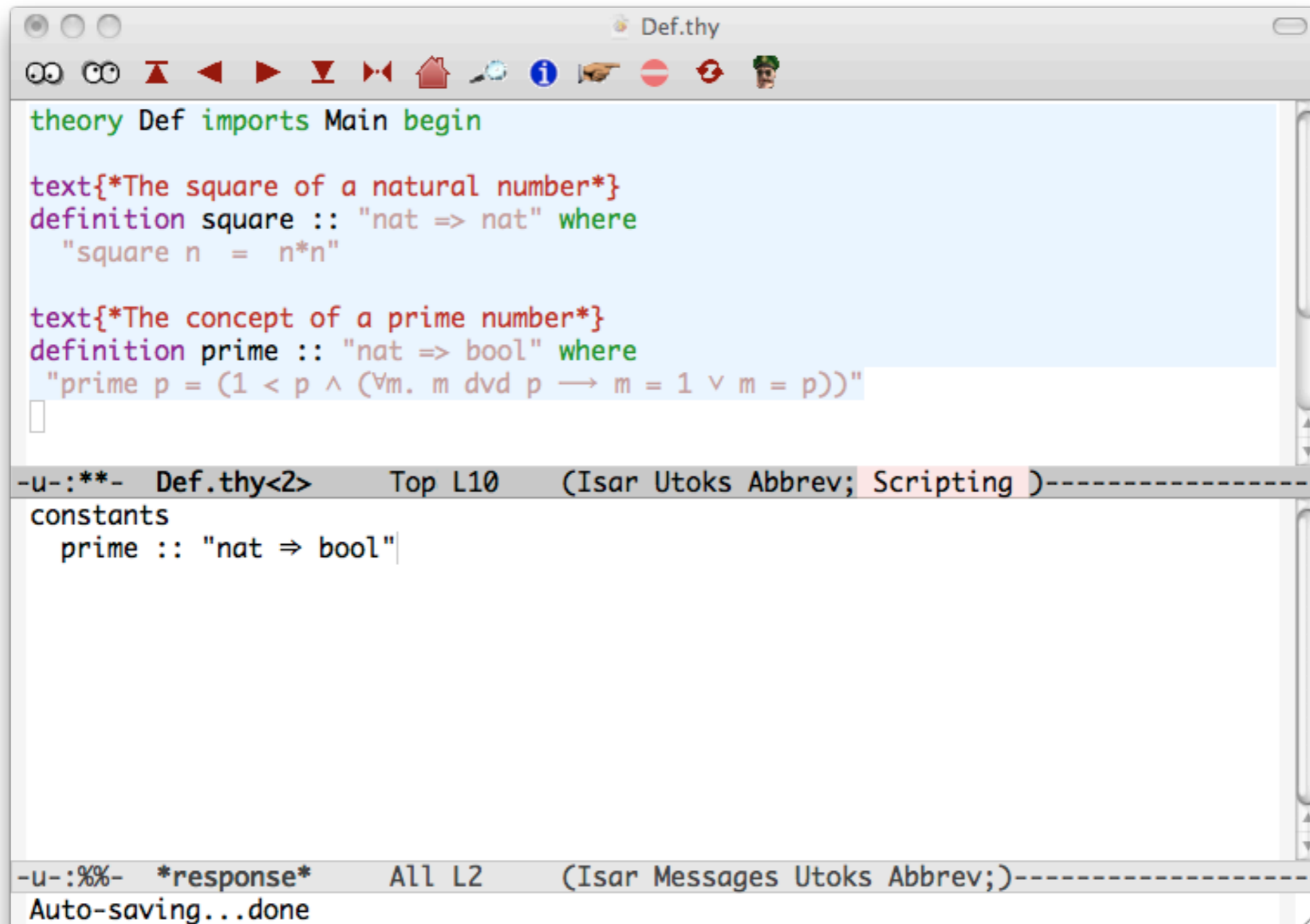
Notes on Type Declarations

- Type synonyms merely introduce *abbreviations*.
- Recursive data types are less general than in functional programming languages.
 - No recursion into the domain of a function.
 - Mutually recursive definitions can be tricky.

Notes on Type Declarations

- Type synonyms merely introduce *abbreviations*.
- Recursive data types are less general than in functional programming languages.
 - No recursion into the domain of a function.
 - Mutually recursive definitions can be tricky.
- Recursive types are equipped with proof methods for *induction* and *case analysis*.

Basic Constant Definitions



```
theory Def imports Main begin

text{*The square of a natural number*}
definition square :: "nat => nat" where
  "square n = n*n"

text{*The concept of a prime number*}
definition prime :: "nat => bool" where
  "prime p = (1 < p ^ (∀m. m dvd p → m = 1 ∨ m = p))"

□

-u-:***- Def.thy<2>      Top L10      (Isar Utoks Abbrev; Scripting )-----
constants
  prime :: "nat ⇒ bool"

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

Notes on Constant Definitions

Notes on Constant Definitions

- Basic definitions are *not* recursive.

Notes on Constant Definitions

- Basic definitions are *not* recursive.
- Every variable on the right-hand side must also appear on the left.

Notes on Constant Definitions

- Basic definitions are *not* recursive.
- Every variable on the right-hand side must also appear on the left.
- In proofs, definitions are *not* expanded by default!
 - Defining the constant C to denote t yields the theorem C_def , asserting $C=t$.
 - Abbreviations can be declared through a separate mechanism.

Lists in Isabelle

Lists in Isabelle

- We illustrate data types and functions using a reduced Isabelle theory that lacks lists.

Lists in Isabelle

- We illustrate data types and functions using a reduced Isabelle theory that lacks lists.
- The standard Isabelle environment has a *comprehensive list library*:
 - Functions `#` (cons), `@` (append), `map`, `filter`, `nth`, `take`, `drop`, `takeWhile`, `dropWhile`, ...
 - Cases: `(case xs of [] => [] | x#xs => ...)`
 - Over 600 theorems!

List Induction Principle

List Induction Principle

To show $\varphi(xs)$, it suffices to show the *base case* and *inductive step*:

- $\varphi(\text{Nil})$
- $\varphi(xs) \Rightarrow \varphi(\text{Cons}(x, xs))$

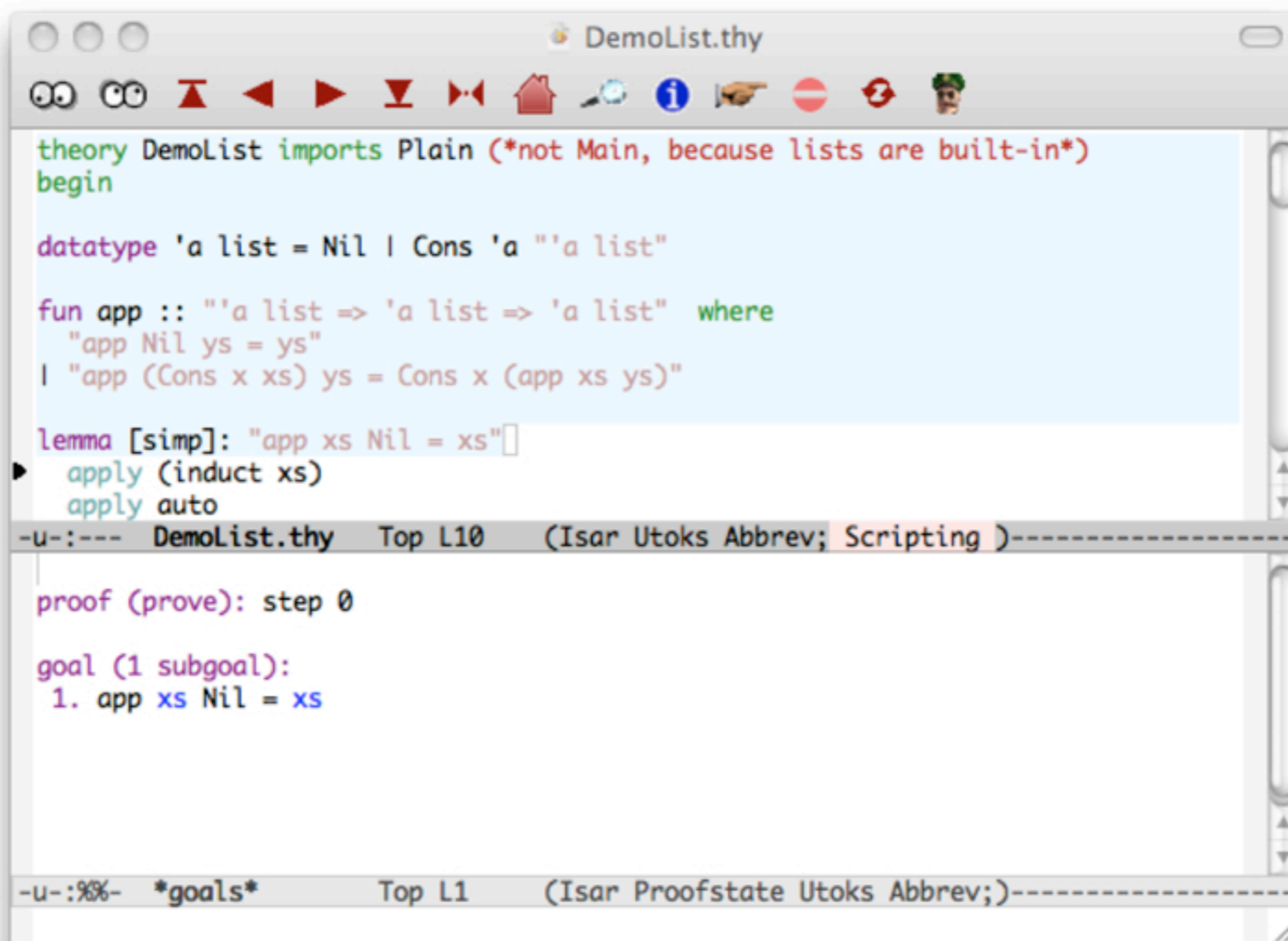
List Induction Principle

To show $\varphi(xs)$, it suffices to show the *base case* and *inductive step*:

- $\varphi(\text{Nil})$
- $\varphi(xs) \Rightarrow \varphi(\text{Cons}(x, xs))$

The principle of case analysis is similar, expressing that any list has one of the forms Nil or $\text{Cons}(x, xs)$ (for some x and xs).

Proof General



```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
- u-:--- DemoList.thy   Top L10   (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 0

goal (1 subgoal):
  1. app xs Nil = xs
- u-:%%-   *goals*       Top L1   (Isar Proofstate Utoks Abbrev; )-----
```


Proof General

The screenshot shows the Proof General IDE interface. The top window, titled "DemoList.thy", contains the following code:

```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
```

The code is color-coded: keywords like `theory`, `imports`, `begin`, `datatype`, `fun`, `where`, `lemma`, `apply`, and `auto` are in green; identifiers and literals are in red; and the function arguments and return types are in purple. A blue highlight covers the `lemma` and its `apply` steps. A red arrow points from a text box to the `app xs Nil = xs` line.

The bottom window shows the current proof state:

```
-u-:--- DemoList.thy Top L10 (Isar Utoks Abbrev; Scripting )-----
proof (prove): step 0
goal (1 subgoal):
  1. app xs Nil = xs
```

The bottom status bar indicates the current state: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)`.

processed material
highlighted in blue

Proof General

```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto

proof (prove): step 0
goal (1 subgoal):
  1. app xs Nil = xs
```

The screenshot shows the Proof General IDE with a file named 'DemoList.thy'. The top window displays the source code, with the lemma and its proof steps highlighted in blue. A red arrow points from a callout box to this highlighted area. The bottom window shows the Isabelle output, including the current goal and the step number. A red arrow points from another callout box to the 'step 0' output.

processed material highlighted in blue

Isabelle's output shown in a separate window

Proof General

```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto

proof (prove): step 0
goal (1 subgoal):
  1. app xs Nil = xs
```

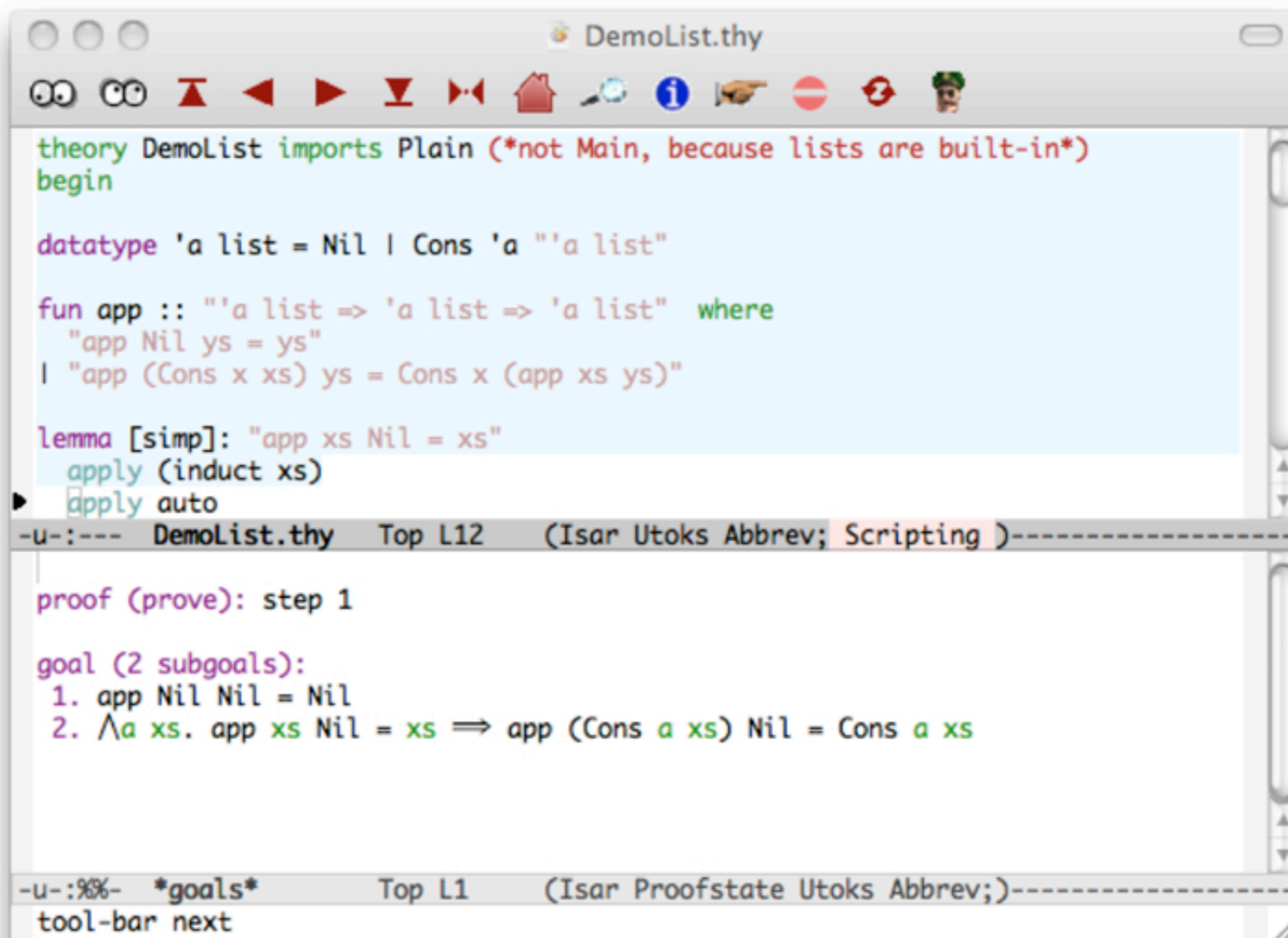
The screenshot shows the Proof General IDE with a file named 'DemoList.thy'. The top window displays the source code, with the first two lines of the 'lemma' block highlighted in blue. The bottom window shows the Isabelle output, including the start of a proof attempt. Three red arrows point from text boxes to specific parts of the code and output.

processed material highlighted in blue

Isabelle's output shown in a separate window

the very start of a proof attempt

Proof by Induction



```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

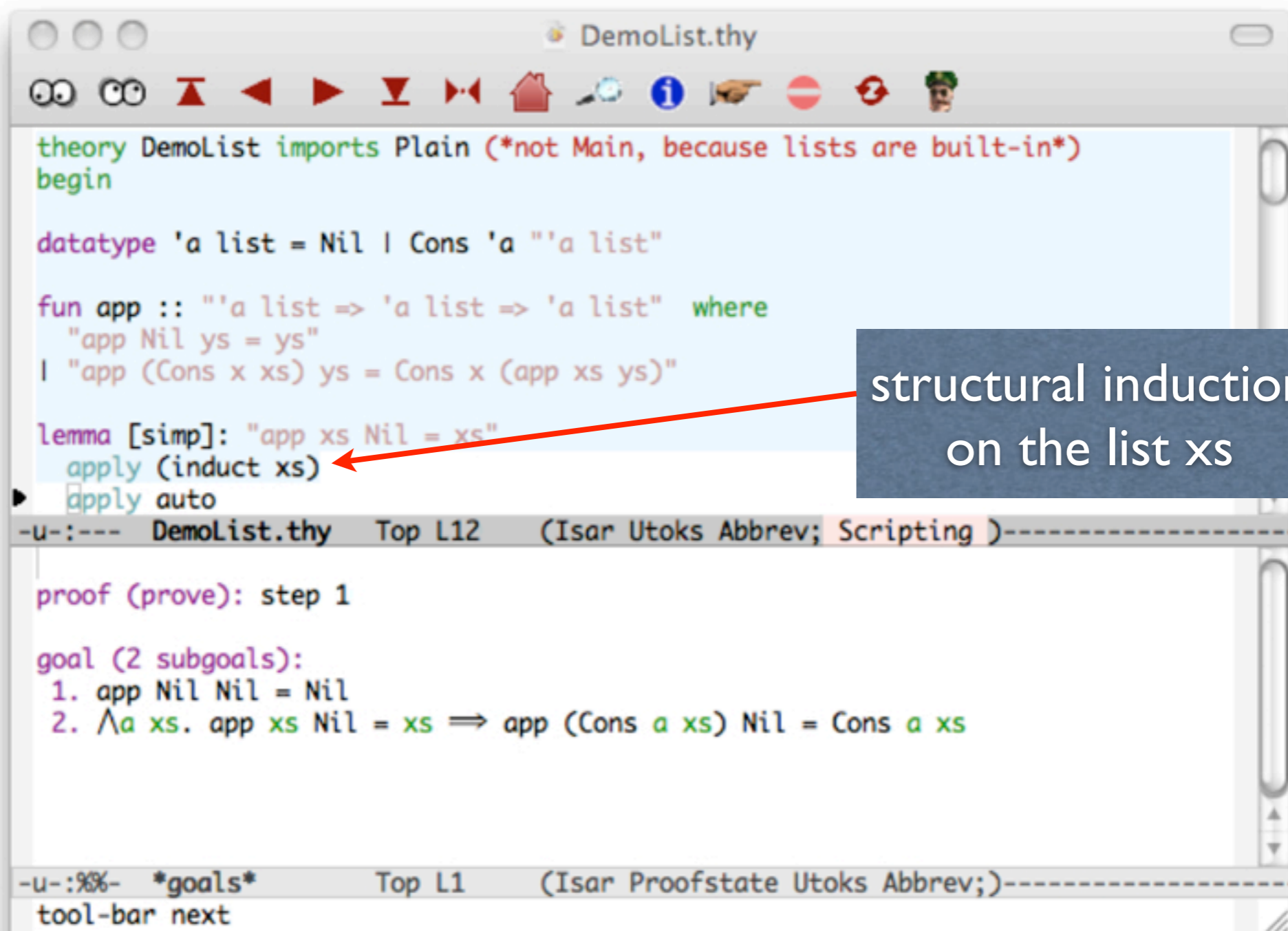
fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
- u-:--- DemoList.thy   Top L12   (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 1

goal (2 subgoals):
  1. app Nil Nil = Nil
  2.  $\wedge a xs. \text{app } xs \text{ Nil} = xs \implies \text{app } (\text{Cons } a \text{ xs}) \text{ Nil} = \text{Cons } a \text{ xs}$ 
- u-:%%- *goals*       Top L1   (Isar Proofstate Utoks Abbrev;)-----
tool-bar next
```

Proof by Induction



The screenshot shows a theorem prover interface with a code editor and a proof state window. The code editor contains the following text:

```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
```

An annotation box with the text "structural induction on the list xs" has a red arrow pointing to the `induct xs` line in the code.

The proof state window shows the following text:

```
-u-:--- DemoList.thy   Top L12   (Isar Utoks Abbrev; Scripting )-----
|
| proof (prove): step 1
| goal (2 subgoals):
| 1. app Nil Nil = Nil
| 2.  $\wedge a\ xs. \text{app } xs\ Nil = xs \implies \text{app } (\text{Cons } a\ xs)\ Nil = \text{Cons } a\ xs$ 
|
```

The bottom status bar shows: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----` and `tool-bar next`.

Proof by Induction

```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto

proof (prove): step 1

goal (2 subgoals):
  1. app Nil Nil = Nil
  2.  $\wedge a xs. \text{app } xs \text{ Nil} = xs \implies \text{app } (\text{Cons } a \text{ xs}) \text{ Nil} = \text{Cons } a \text{ xs}$ 

tool-bar next
```

structural induction
on the list xs

base case and
inductive step

Proof by Induction

```
theory DemoList imports Plain (*not Main, because lists are built-in*)
begin

datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto

proof (prove): step 1

goal (2 subgoals):
  1. app Nil Nil = Nil
  2.  $\wedge a xs. \text{app } xs \text{ Nil} = xs \implies \text{app } (\text{Cons } a \text{ xs}) \text{ Nil} = \text{Cons } a \text{ xs}$ 

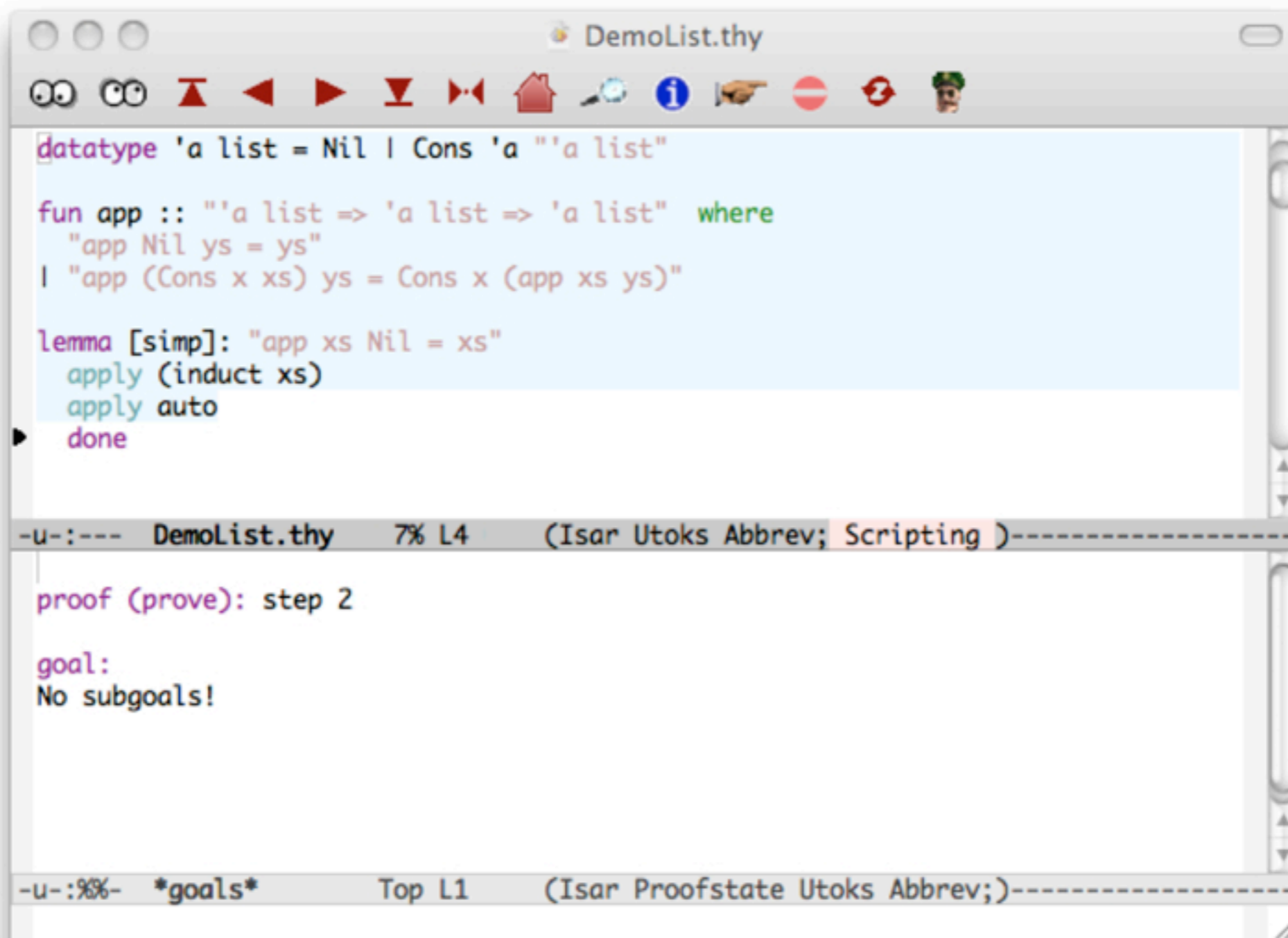
-u-:--- DemoList.thy Top L12 (Isar Utoks Abbrev: Scripting )
-u-:%%- *goals* Top L1 (Isar
tool-bar next
```

structural induction on the list xs

base case and inductive step

induction hypothesis

Finishing a Proof



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

```
datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
  done
```

Below the code is a status bar: `-u-:--- DemoList.thy 7% L4 (Isar Utoks Abbrev; Scripting)-----`

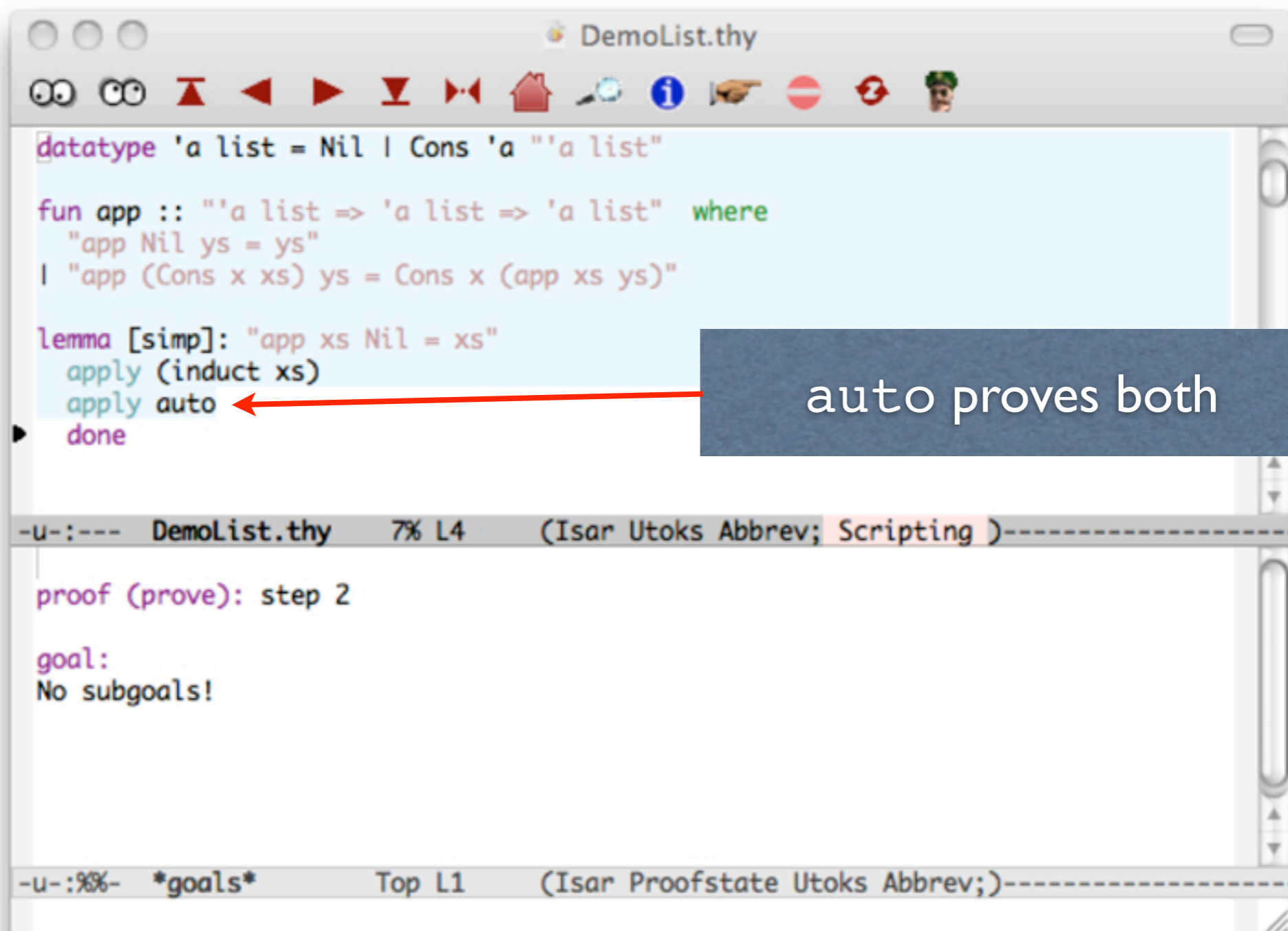
The lower pane shows the proof state:

```
proof (prove): step 2

goal:
No subgoals!
```

The bottom status bar reads: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----`

Finishing a Proof



The screenshot shows a theorem prover interface with a code editor and a proof state window. The code editor contains the following text:

```
datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
done
```

A red arrow points from a dark blue box containing the text "auto proves both" to the `apply auto` line in the code editor.

The proof state window shows the following text:

```
-u-:--- DemoList.thy 7% L4 (Isar Utoks Abbrev; Scripting )-----
proof (prove): step 2
goal:
No subgoals!
```

The bottom status bar of the proof state window shows: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)`

Finishing a Proof

The screenshot shows a theorem prover interface with a code editor and a proof state window. The code editor contains the following text:

```
datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list" where
  "app Nil ys = ys"
| "app (Cons x xs) ys = Cons x (app xs ys)"

lemma [simp]: "app xs Nil = xs"
  apply (induct xs)
  apply auto
done
```

A red arrow points from the text "auto proves both" to the `apply auto` line in the code editor.

The proof state window shows the following text:

```
proof (prove): step 2

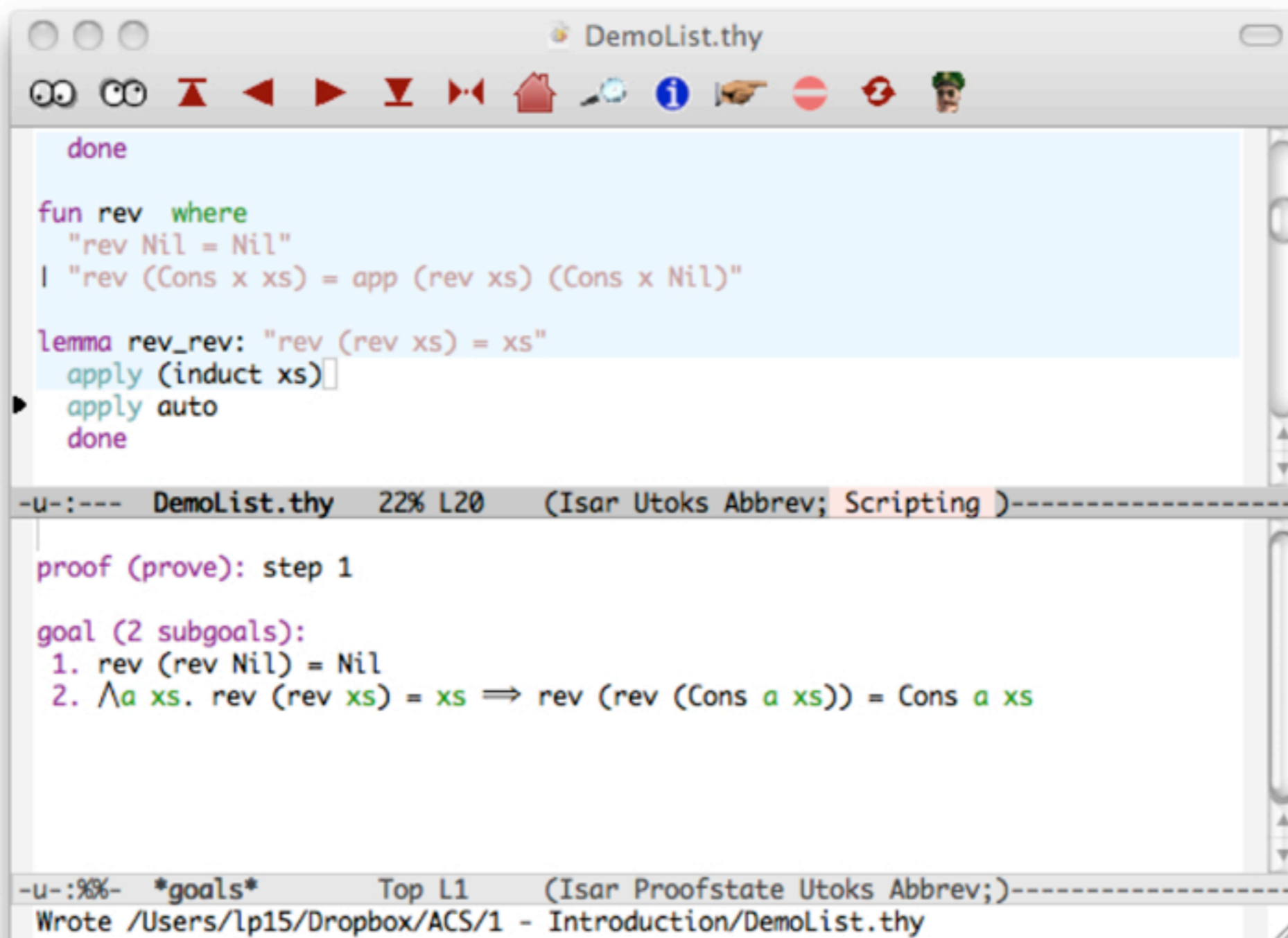
goal:
No subgoals!
```

A red arrow points from the text "We must still issue 'done' to register the theorem" to the `No subgoals!` line in the proof state window.

auto proves both

We must still issue "done"
to register the theorem

Another Proof Attempt



```
done

fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

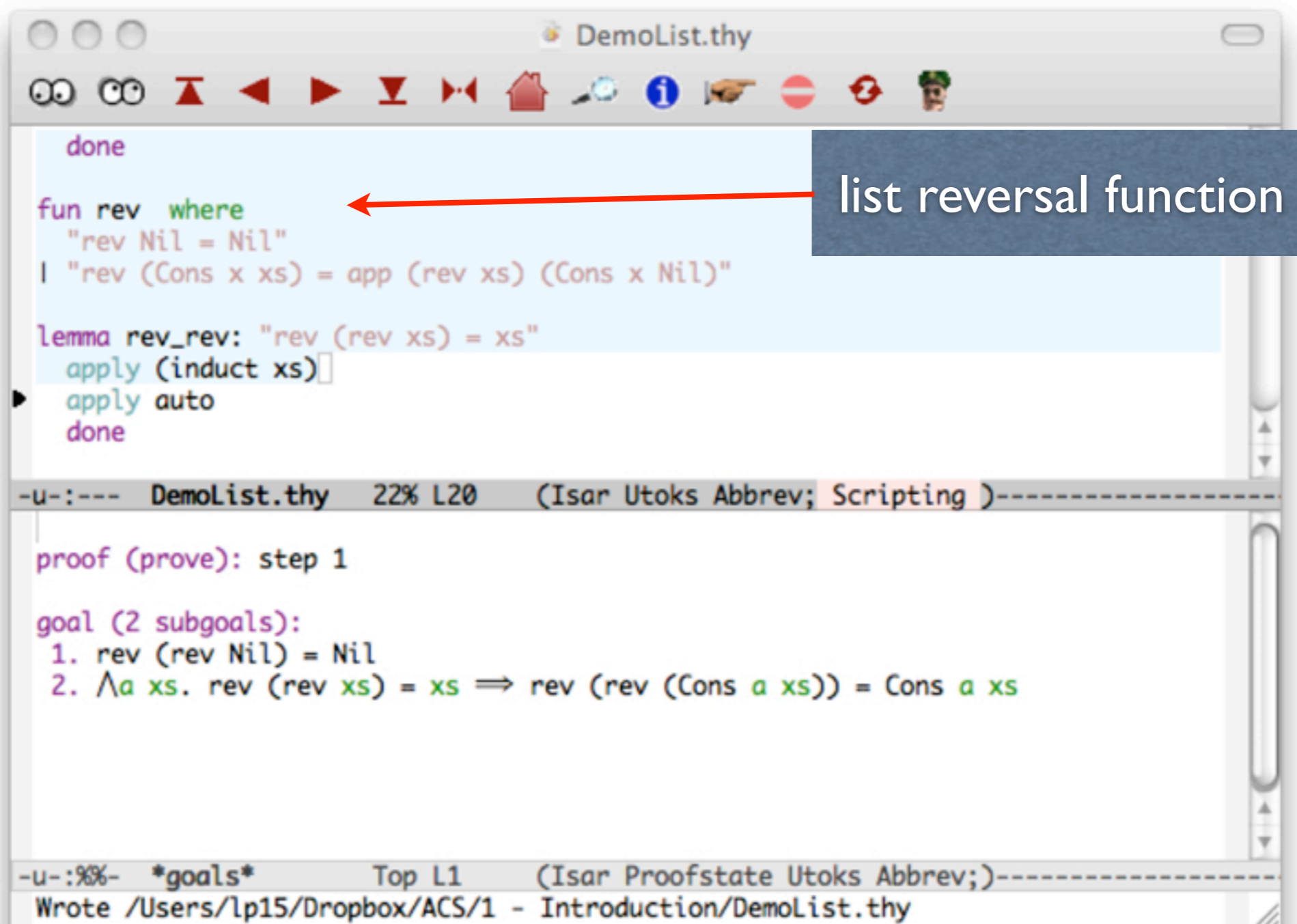
lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
done

-u-:--- DemoList.thy 22% L20 (Isar Utoks Abbrev; Scripting )-----
proof (prove): step 1

goal (2 subgoals):
  1. rev (rev Nil) = Nil
  2.  $\wedge a xs. \text{rev (rev xs) = xs} \implies \text{rev (rev (Cons a xs)) = Cons a xs}$ 

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

Another Proof Attempt



The screenshot shows a theorem prover interface with a window titled "DemoList.thy". The main editor contains the following code:

```
done

fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
done
```

A red arrow points from a blue callout box labeled "list reversal function" to the `fun rev` definition. Below the code, the proof state is shown:

```
-u-:--- DemoList.thy 22% L20 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 1

goal (2 subgoals):
1. rev (rev Nil) = Nil
2.  $\wedge a\ xs. \text{rev (rev xs) = xs} \implies \text{rev (rev (Cons a xs)) = Cons a xs}$ 
```

The bottom status bar shows: `-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----` and `Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy`.

Another Proof Attempt

The screenshot shows a theorem prover interface with a file named "DemoList.thy". The code defines a list reversal function and a lemma to be proved. The proof attempt is shown below the code, with two subgoals. Red arrows point from text boxes to the function definition and the second subgoal.

```
done

fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
done
```

-u-:--- DemoList.thy 22% L20 (Isar Utoks Abbrev; Scripting)-----

```
proof (prove): step 1

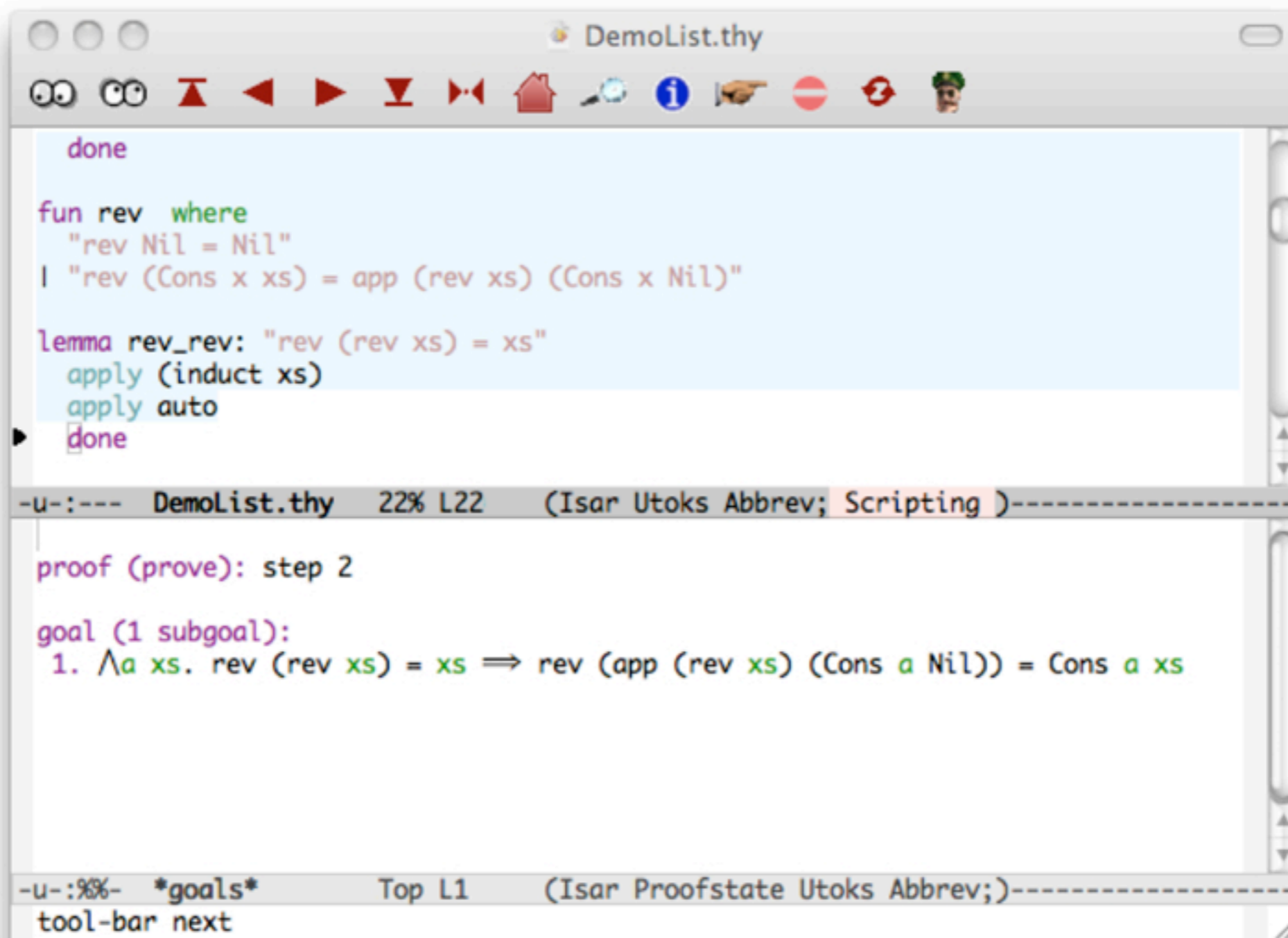
goal (2 subgoals):
  1. rev (rev Nil) = Nil
  2.  $\wedge a xs. \text{rev (rev xs) = xs} \implies \text{rev (rev (Cons a xs)) = Cons a xs}$ 
```

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy

list reversal function

Can we prove both subgoals?

Stuck!



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

```
done

fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
done
```

Below the code is a status bar: "-u-:--- DemoList.thy 22% L22 (Isar Utoks Abbrev; Scripting)-----".

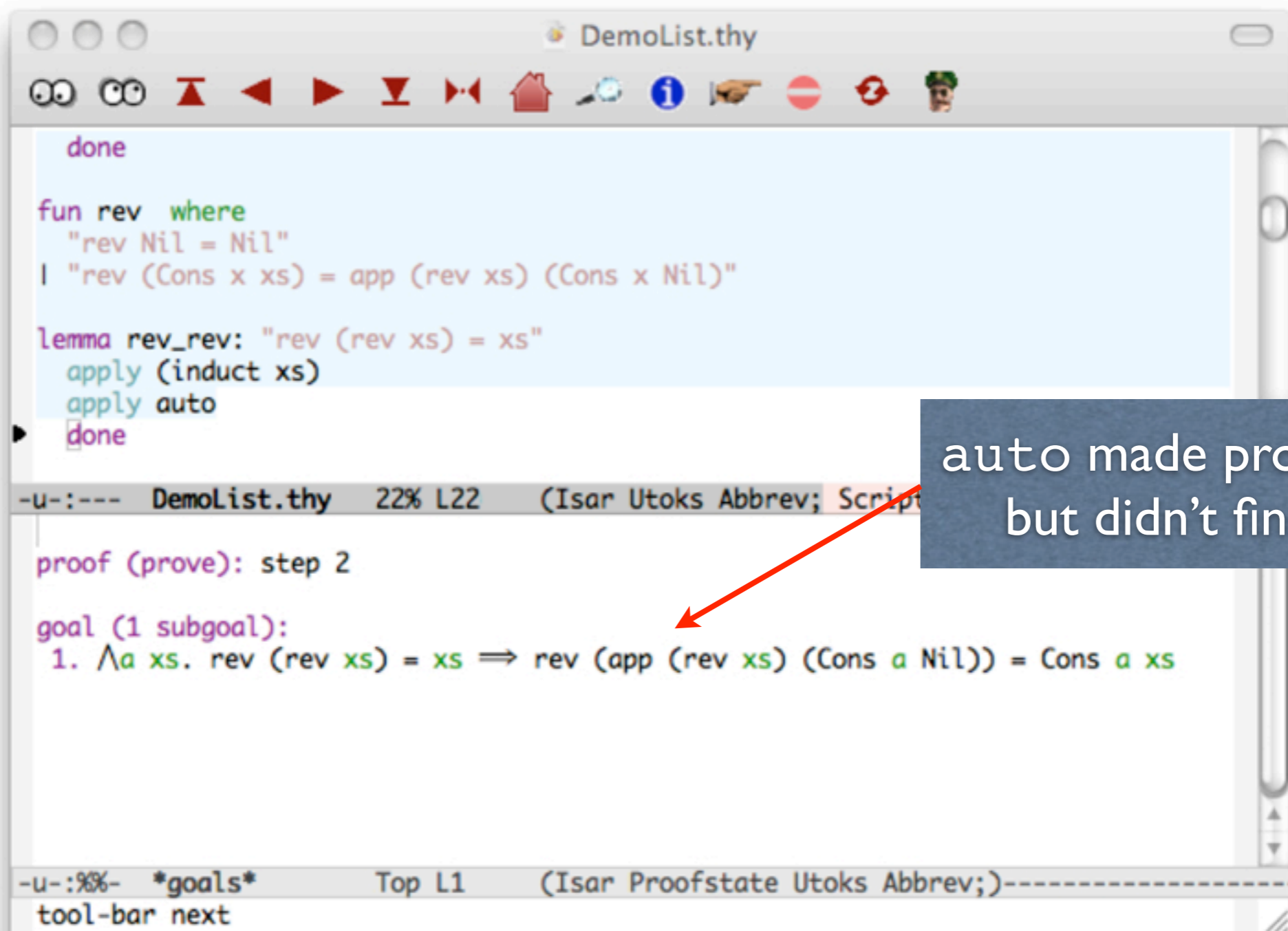
The next section shows the proof state:

```
proof (prove): step 2

goal (1 subgoal):
  1.  $\Lambda a xs. \text{rev (rev xs) = xs} \implies \text{rev (app (rev xs) (Cons a Nil)) = Cons a xs}$ 
```

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

Stuck!



```
done

fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
  done

proof (prove): step 2

goal (1 subgoal):
  1.  $\wedge a \ xs. \text{rev (rev xs) = xs} \implies \text{rev (app (rev xs) (Cons a Nil)) = Cons a xs}$ 
```

-u-:--- DemoList.thy 22% L22 (Isar Utoks Abbrev; Script

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

tool-bar next

auto made progress
but didn't finish

Stuck!

```
done

fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
  done

proof (prove): step 2

goal (1 subgoal):
  1.  $\Lambda a xs. \text{rev (rev xs) = xs} \Rightarrow \text{rev (app (rev xs) (Cons a Nil)) = Cons a xs}$ 

-u-:--- DemoList.thy 22% L22 (Isar Utoks Abbrev; Script
-u-:%%- *goals* Top L1 (Isar
tool-bar next
```

auto made progress but didn't finish

looks like we need a lemma relating rev and app!

Stuck Again!

```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
  done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)

-u-:--- DemoList.thy 21% L24 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 2

goal (1 subgoal):
  1.  $\wedge a xs.$ 
      rev (app xs ys) = app (rev ys) (rev xs)  $\implies$ 
      app (app (rev ys) (rev xs)) (Cons a Nil) =
      app (rev ys) (app (rev xs) (Cons a Nil))

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

Stuck Again!

```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)

-u-:--- DemoList.thy 21% L24 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 2

goal (1 subgoal):
1.  $\wedge a xs.$ 
   rev (app xs ys) = app (rev ys) (rev xs)  $\implies$ 
   app (app (rev ys) (rev xs)) (Cons a Nil) =
   app (rev ys) (app (rev xs) (Cons a Nil))

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

we dreamt up a lemma...

Stuck Again!

```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
  done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)

-u-:--- DemoList.thy 21% L24 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 2

goal (1 subgoal):
1.  $\forall a xs.$ 
   rev (app xs ys) = app (rev ys) (rev xs)  $\implies$ 
   app (app (rev ys) (rev xs)) (Cons a Nil) =
   app (rev ys) (app (rev xs) (Cons a Nil))

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

we dreamt up a lemma...

But it needs another lemma!

Stuck Again!

```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)

-u-:--- DemoList.thy 21% L24 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 2

goal (1 subgoal):
1.  $\forall a xs.$ 
   rev (app xs ys) = app (rev ys) (rev xs)  $\implies$ 
   app (app (rev ys) (rev xs)) (Cons a Nil) =
   app (rev ys) (app (rev xs) (Cons a Nil))

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

we dreamt up a lemma...

But it needs another lemma!

Stuck Again!

```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
  done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)

-u-:--- DemoList.thy 21% L24 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 2

goal (1 subgoal):
1.  $\forall a xs.$ 
   rev (app xs ys) = app (rev ys) (rev xs)  $\implies$ 
   app (app (rev ys) (rev xs)) (Cons a Nil) =
   app (rev ys) (app (rev xs) (Cons a Nil))

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

we dreamt up a lemma...

But it needs another lemma!

Stuck Again!

```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
  done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)

-u-:--- DemoList.thy 21% L24 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 2

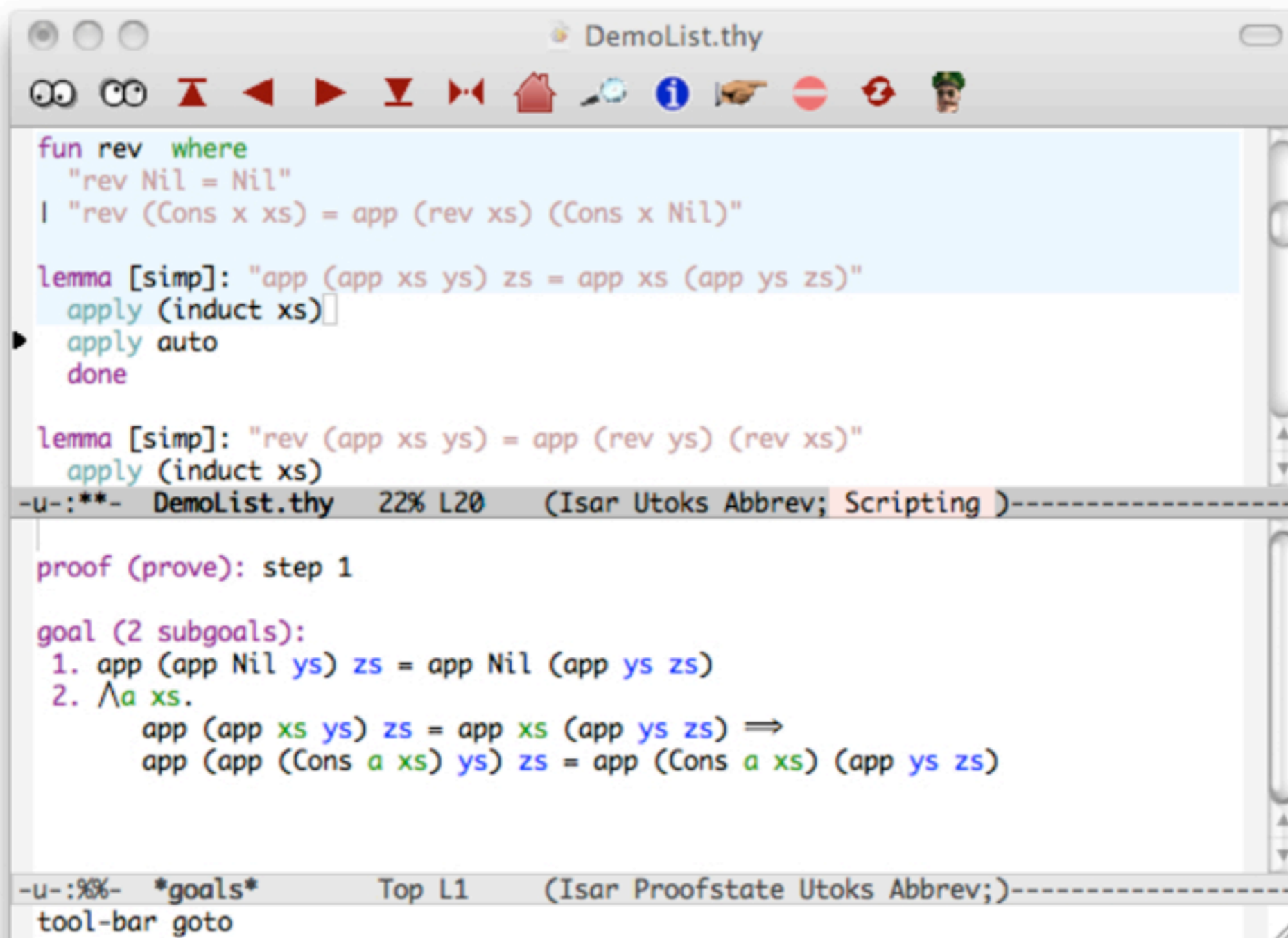
goal (1 subgoal):
1.  $\forall a xs.$ 
   rev (app xs ys) = app (rev ys) (rev xs)  $\Rightarrow$ 
   app (app (rev ys) (rev xs)) (Cons a Nil) =
   app (rev ys) (app (rev xs) (Cons a Nil))

-u-:%%- *goals* Top L1 (Isar Proofstate Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy
```

we dreamt up a lemma...

But it needs another lemma!

The Final Piece of the Jigsaw



```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "app (app xs ys) zs = app xs (app ys zs)"
  apply (induct xs)
  apply auto
  done

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  done

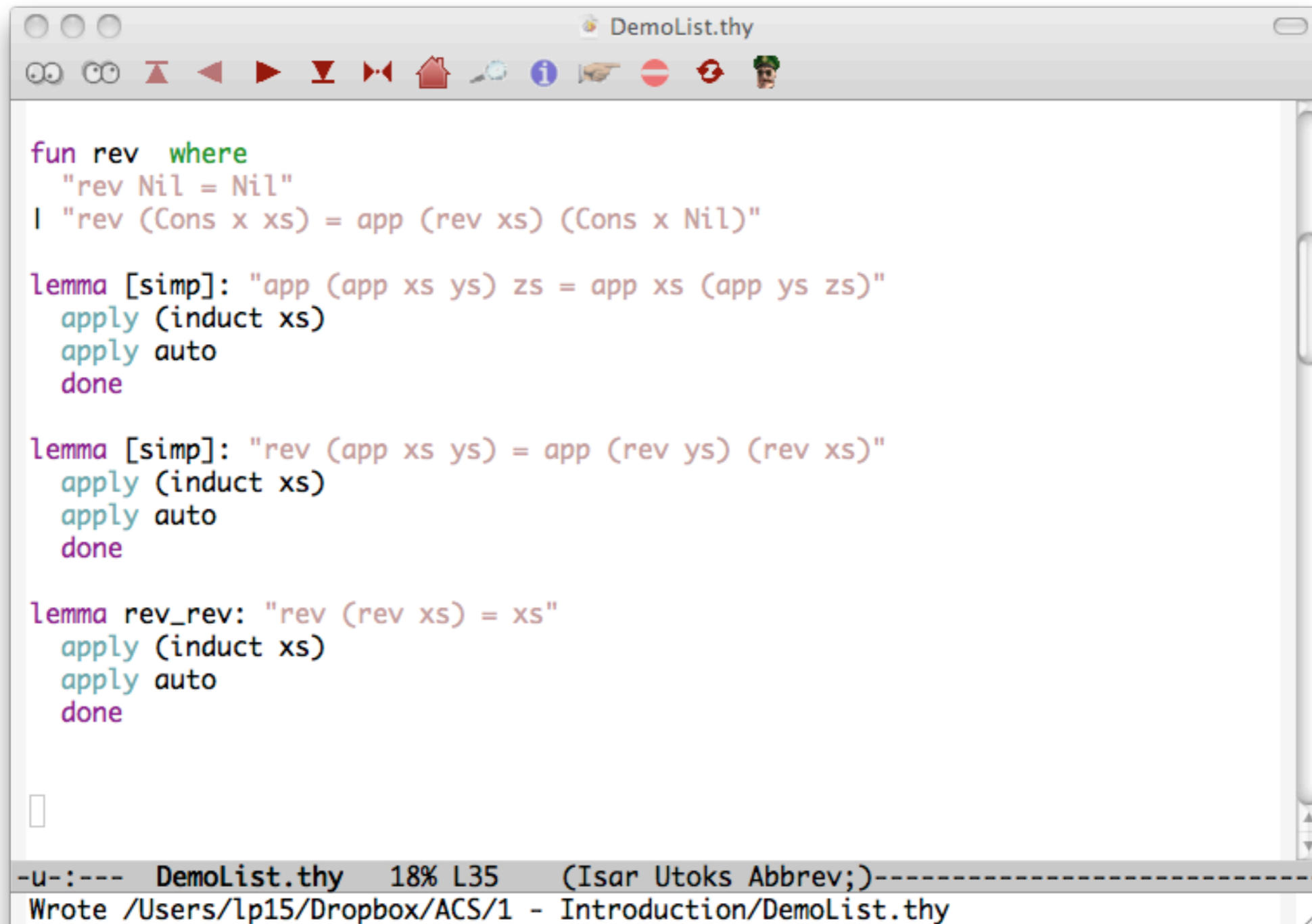
-u-:**- DemoList.thy 22% L20 (Isar Utoks Abbrev; Scripting )-----

proof (prove): step 1

goal (2 subgoals):
1. app (app Nil ys) zs = app Nil (app ys zs)
2.  $\wedge a xs.$ 
   app (app xs ys) zs = app xs (app ys zs)  $\implies$ 
   app (app (Cons a xs) ys) zs = app (Cons a xs) (app ys zs)

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

The Finished Proof



```
fun rev where
  "rev Nil = Nil"
| "rev (Cons x xs) = app (rev xs) (Cons x Nil)"

lemma [simp]: "app (app xs ys) zs = app xs (app ys zs)"
  apply (induct xs)
  apply auto
  done

lemma [simp]: "rev (app xs ys) = app (rev ys) (rev xs)"
  apply (induct xs)
  apply auto
  done

lemma rev_rev: "rev (rev xs) = xs"
  apply (induct xs)
  apply auto
  done
```

-u-:--- DemoList.thy 18% L35 (Isar Utoks Abbrev;)-----
Wrote /Users/lp15/Dropbox/ACS/1 - Introduction/DemoList.thy