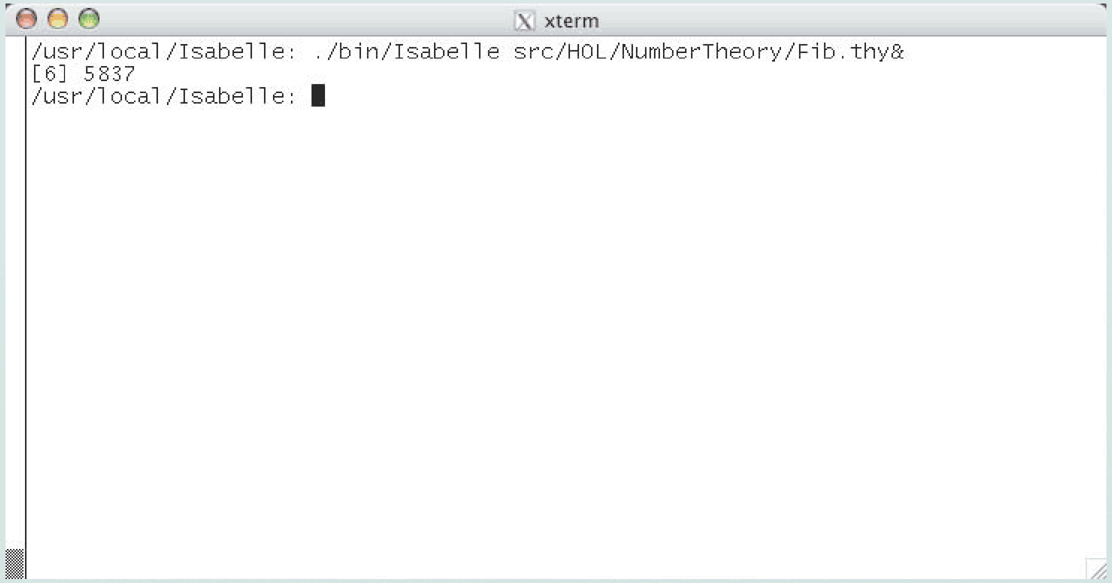
Isabelle and Proof General: Preview

Lawrence C. Paulson
University of Cambridge

Getting Started

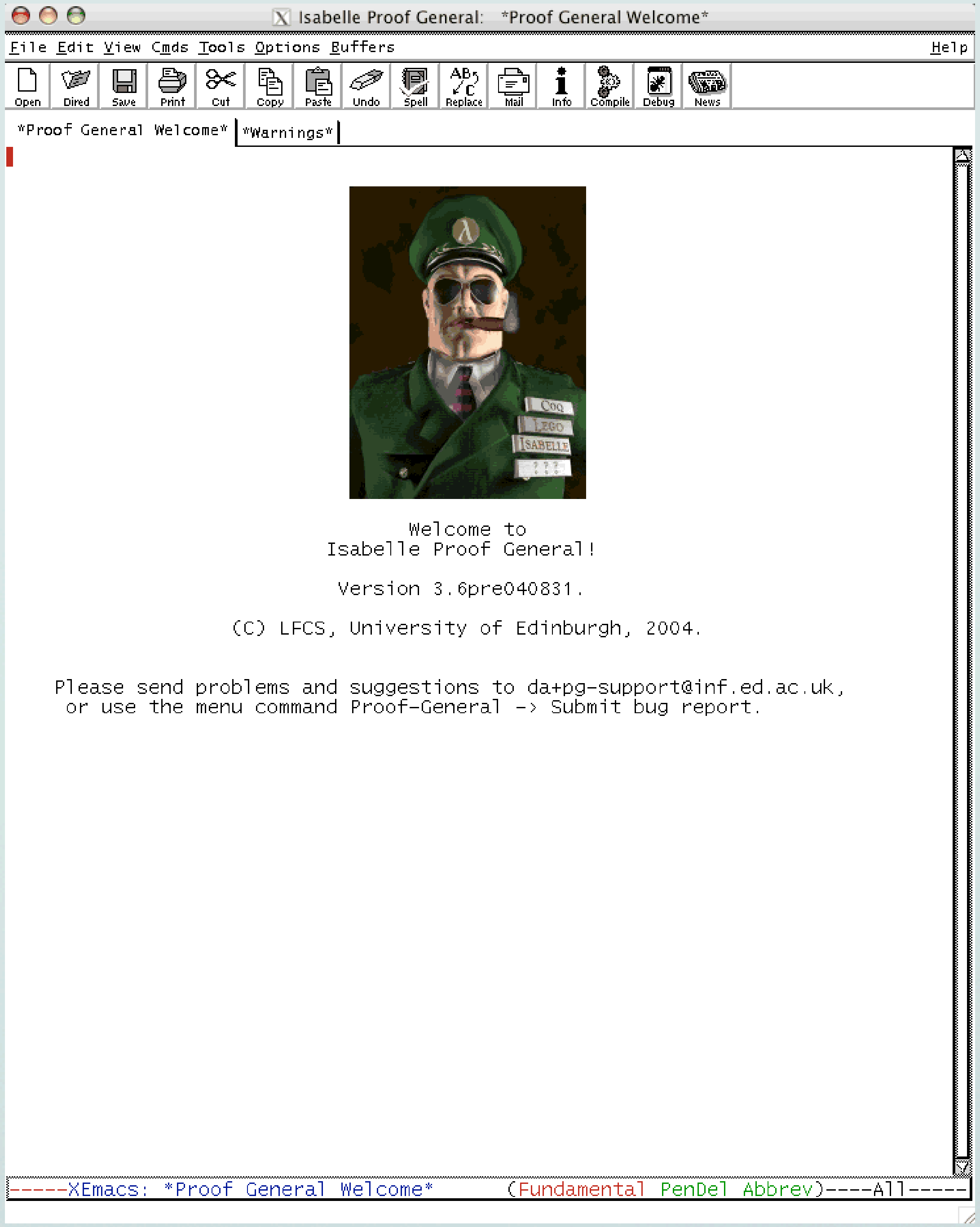
- Install Isabelle, following instructions on the download page.
- Install Proof General.
- Proof General requires the editor XEmacs to be installed.
- If you have not used XEmacs before, practice on plain text files before attempting proofs!
- Launch Isabelle from the command line.
- Here, Isabelle has been installed at `/usr/local` and is used to open one of the standard theories.

Launching Isabelle at the UNIX command line



```
/usr/local/Isabelle: ./bin/Isabelle src/HOL/NumberTheory/Fib.thy&
[6] 5837
/usr/local/Isabelle: 
```

Proof General

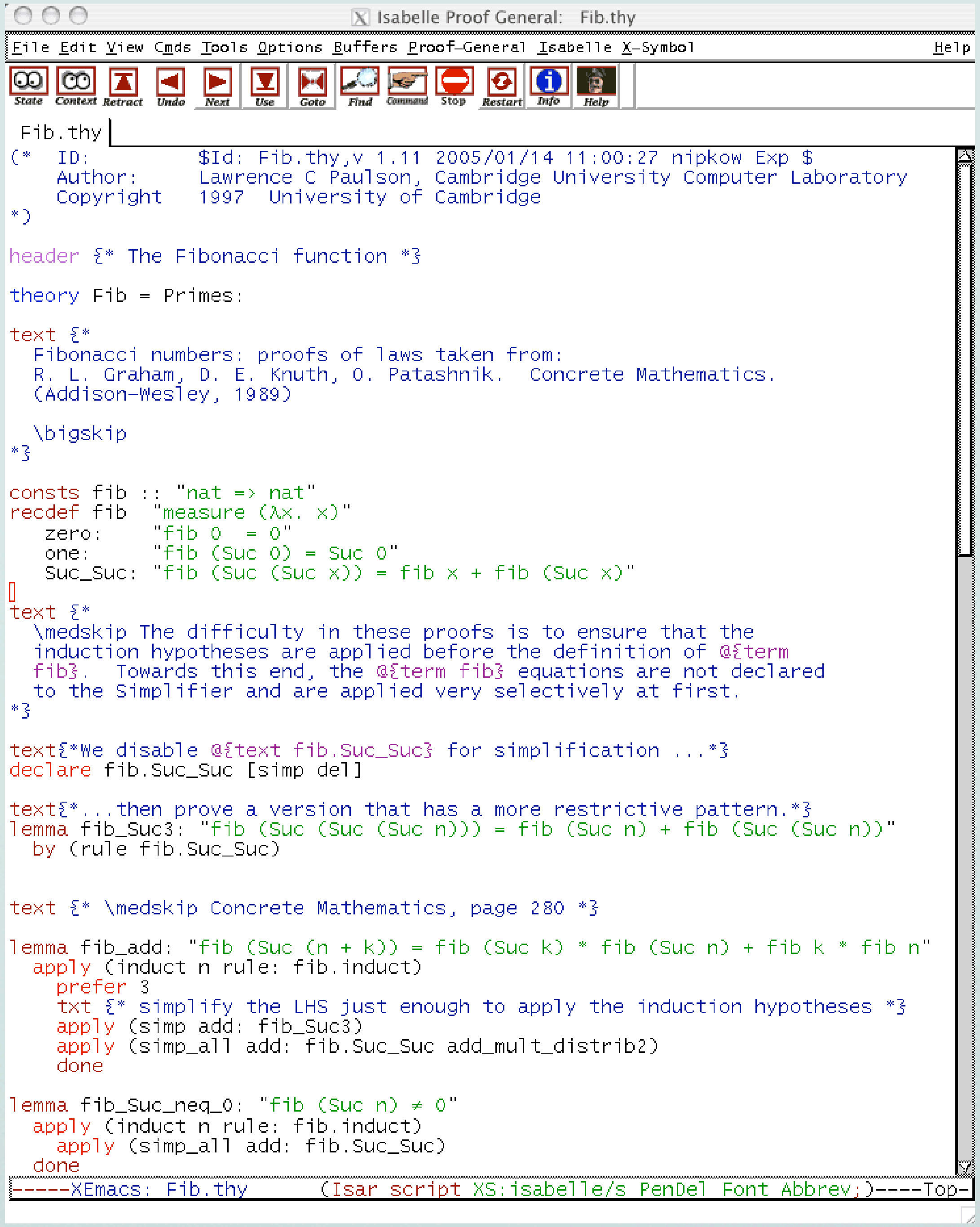

Isabelle Proof General: *Proof General Welcome*
File Edit View Cmds Tools Options Buffers
Help
Open
Dired
Save
Print
Cut
Copy
Paste
Undo
Spell
Replace
Mail
Info
Compile
Debug
News
Proof General Welcome
Warnings
Coq
LEGO
ISABELLE
???
-----XEmacs: *Proof General Welcome*   (Fundamental PenDel Abbrev)----All-----

```
Welcome to
Isabelle Proof General!

Version 3.6pre040831.

(C) LFCS, University of Edinburgh, 2004.

Please send problems and suggestions to da+pg-support@inf.ed.ac.uk,
 or use the menu command Proof-General -> Submit bug report.
```

- Proof General launches within XEmacs.
- If you don't see this splash screen, Proof General is not correctly installed.

The Theory File

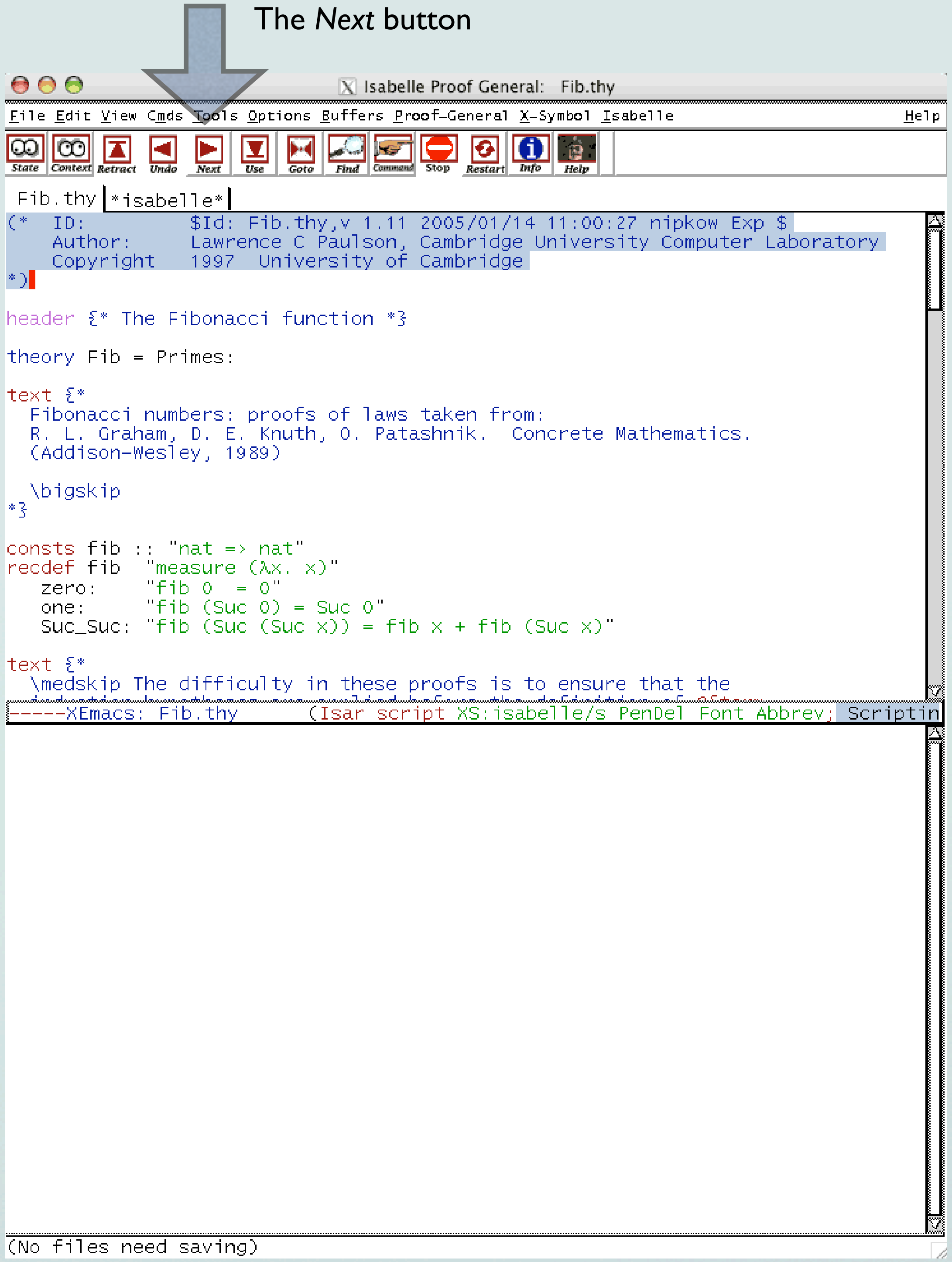
- The theory opens in Proof General.
- Theory files visited from XEmacs also open in Proof General.
- Syntax colouring distinguishes constants, types, keywords, etc.
- The toolbar gives quick access to basic proof operations.
- This theory defines the Fibonacci function and proves theorems about it.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General Isabelle X-Symbol Help
State Context Retract Undo Next Use Goto Find Command Stop Restart Info Help
Fib.thy

```
(*  ID:         $Id: Fib.thy,v 1.11 2005/01/14 11:00:27 nipkow Exp $
    Author:     Lawrence C Paulson, Cambridge University Computer Laboratory
    Copyright   1997  University of Cambridge
*)

header {* The Fibonacci function *}

theory Fib = Primes:

text {*
  Fibonacci numbers: proofs of laws taken from:
  R. L. Graham, D. E. Knuth, O. Patashnik.  Concrete Mathematics.
  (Addison-Wesley, 1989)

  \bigskip
*}

consts fib :: "nat => nat"
recdef fib  "measure (λx. x)"
   zero:    "fib 0  = 0"
   one:     "fib (Suc 0) = Suc 0"
   Suc_Suc: "fib (Suc (Suc x)) = fib x + fib (Suc x)"

text {*
  \medskip The difficulty in these proofs is to ensure that the
  induction hypotheses are applied before the definition of @{term
  fib}.  Towards this end, the @{term fib} equations are not declared
  to the Simplifier and are applied very selectively at first.
*}

text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) ≠ 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done
-----XEmacs: Fib.thy      (Isar script XS:isabelle/s PenDel Font Abbrev;)----Top-
```

Basic Navigation

- A theory file contains definitions, proofs, LaTeX markup, and general commands.
- Clicking on *Next* starts Isabelle and processes the first item: a comment.
- Repeated clicks on *Next* step through the definitions.
- Proof General highlights material that has been processed in blue.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

The *Next* button

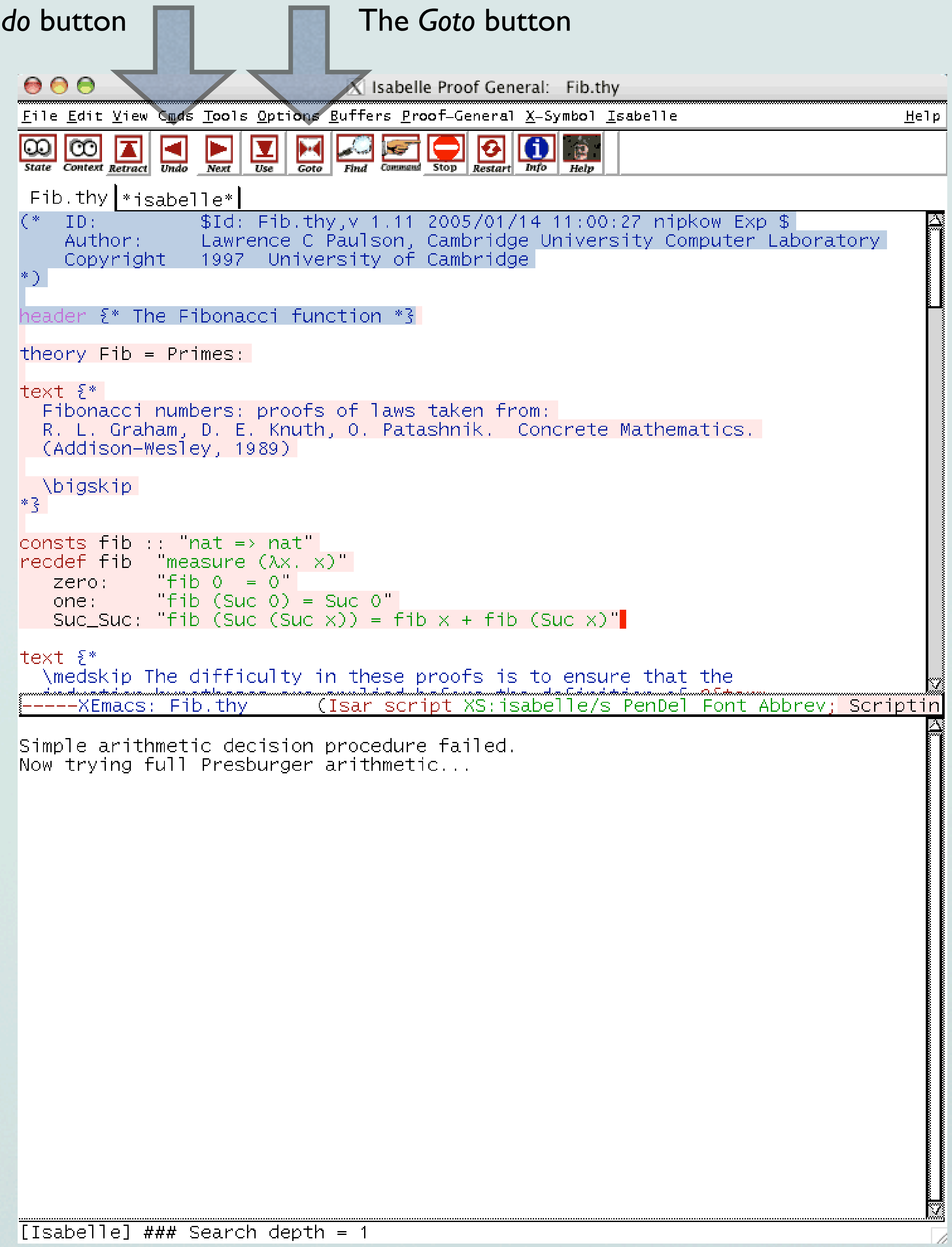
```
(*  ID:         $Id: Fib.thy,v 1.11 2005/01/14 11:00:27 nipkow Exp $
    Author:     Lawrence C Paulson, Cambridge University Computer Laboratory
    Copyright   1997  University of Cambridge
*)

header {* The Fibonacci function *}

theory Fib = Primes:

text {*
  Fibonacci numbers: proofs of laws taken from:
  R. L. Graham, D. E. Knuth, O. Patashnik.  Concrete Mathematics.
  (Addison-Wesley, 1989)

  \bigskip
*}

consts fib :: "nat => nat"
recdef fib  "measure (λx. x)"
   zero:    "fib 0  = 0"
   one:     "fib (Suc 0) = Suc 0"
   Suc_Suc: "fib (Suc (Suc x)) = fib x + fib (Suc x)"

text {*
  \medskip The difficulty in these proofs is to ensure that the
```

```
-----XEmacs: Fib.thy      (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
(No files need saving)
```

Jumping Ahead

- You can click anywhere in the theory and then click on *Goto*.
- *Goto* can even go backward, undoing declarations and commands. (To undo a single command, use the *Undo* button.)
- The `header` command is processed quickly, but the `theory` command refers to another theory.
- While Isabelle is working, Proof General highlights the corresponding text in pink.



The *Undo* button

The *Goto* button

Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

```
(*  ID:         $Id: Fib.thy,v 1.11 2005/01/14 11:00:27 nipkow Exp $
    Author:     Lawrence C Paulson, Cambridge University Computer Laboratory
    Copyright   1997  University of Cambridge
*)

header {* The Fibonacci function *}

theory Fib = Primes:

text {*
  Fibonacci numbers: proofs of laws taken from:
  R. L. Graham, D. E. Knuth, O. Patashnik.  Concrete Mathematics.
  (Addison-Wesley, 1989)

  \bigskip
*}

consts fib :: "nat => nat"
recdef fib  "measure (λx. x)"
   zero:    "fib 0  = 0"
   one:     "fib (Suc 0) = Suc 0"
   Suc_Suc: "fib (Suc (Suc x)) = fib x + fib (Suc x)"

text {*
  \medskip The difficulty in these proofs is to ensure that the
```

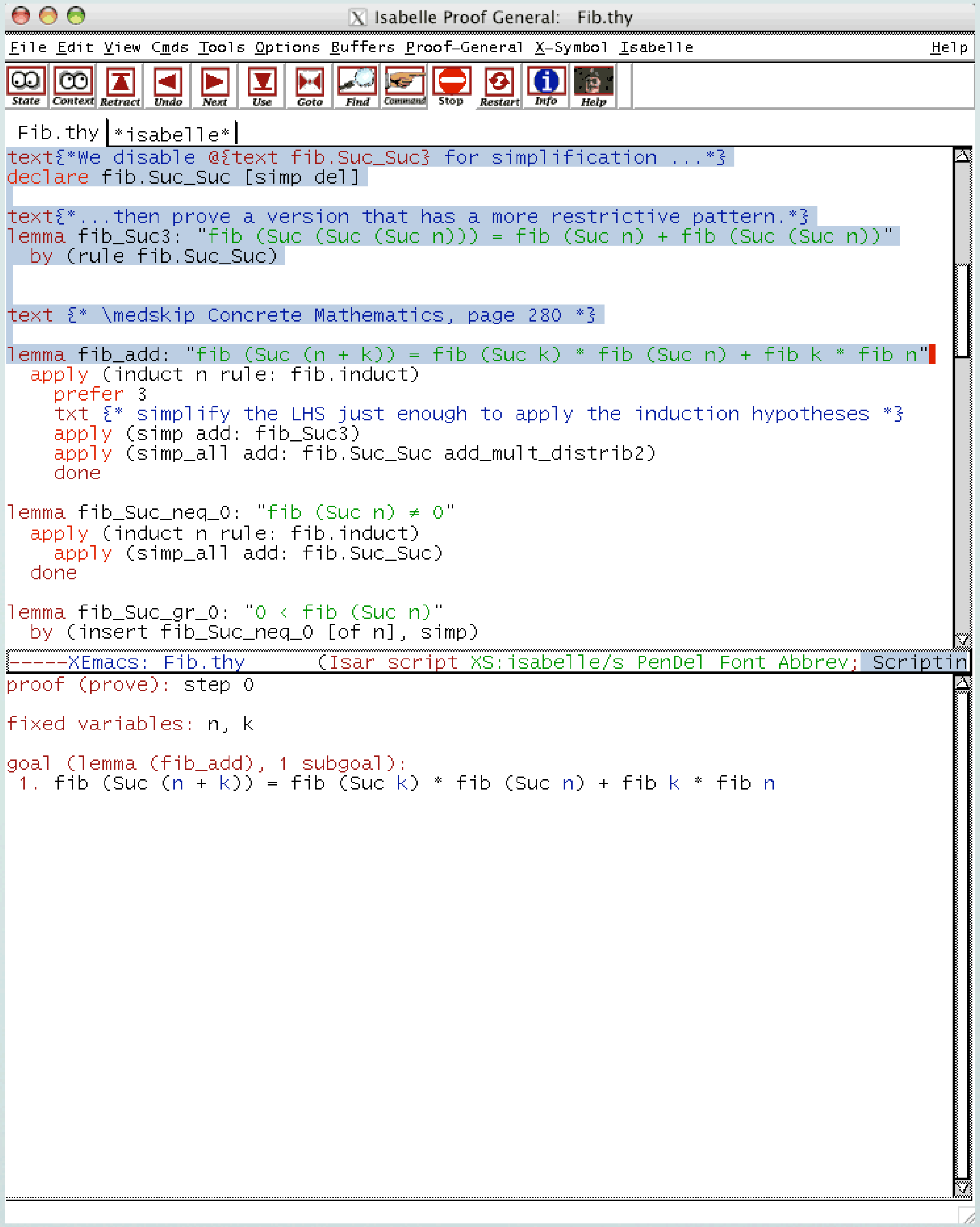
```
-----XEmacs: Fib.thy      (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
Simple arithmetic decision procedure failed.
Now trying full Presburger arithmetic...
```

```
[Isabelle] ### Search depth = 1
```

Running a Proof

- We are about to replay a small proof relating the Fibonacci function, addition and multiplication.
- Processing the `lemma` command displays one subgoal in the proof window.
- The commands `lemma`, `theorem` and `corollary` are essentially equivalent.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State Context Retract Undo Next Use Goto Find Command Stop Restart Info Help
Fib.thy *isabelle*

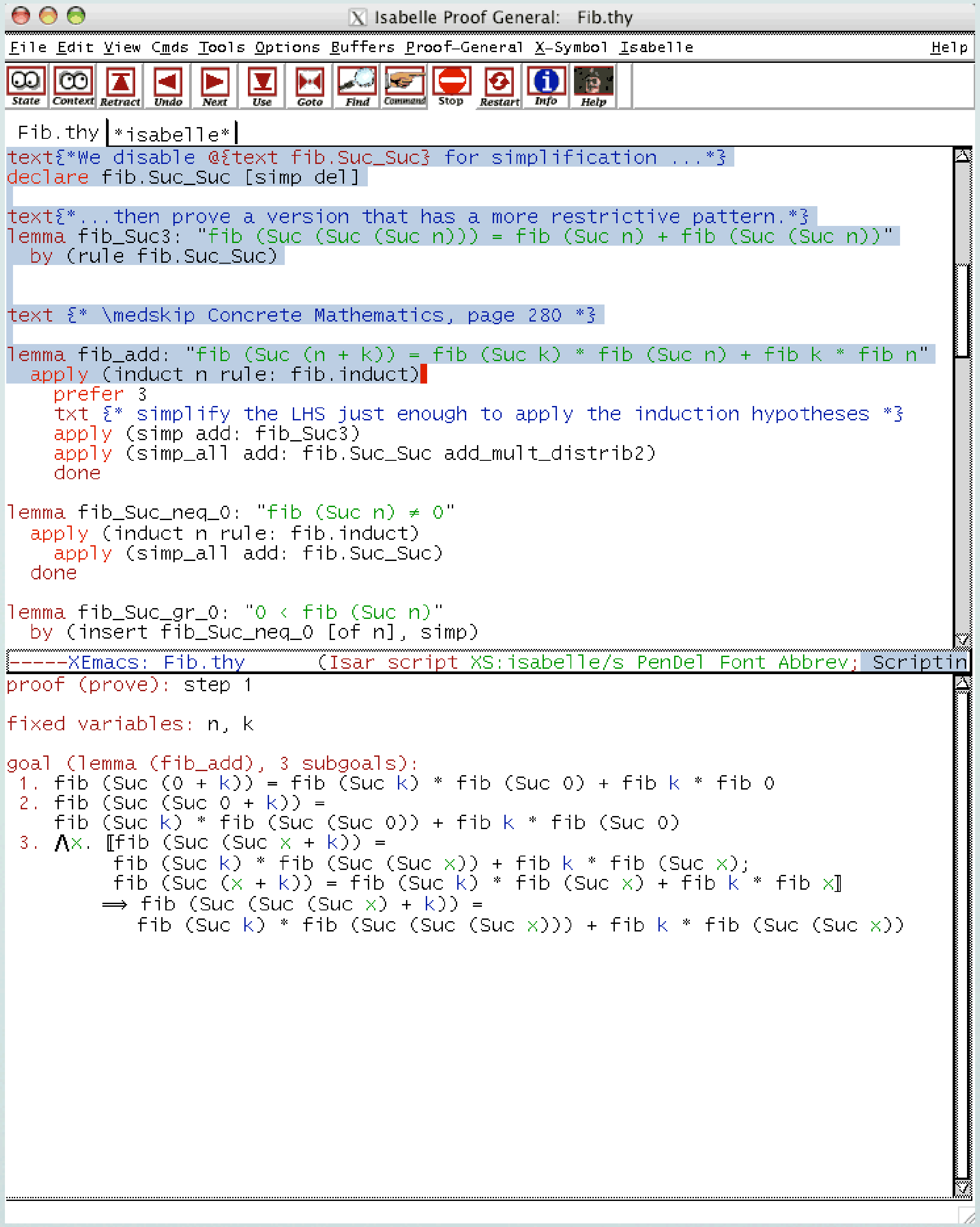
```
text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) ≠ 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done

lemma fib_Suc_gr_0: "0 < fib (Suc n)"
  by (insert fib_Suc_neq_0 [of n], simp)
```

-----XEmacs: Fib.thy   (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin

```
proof (prove): step 0

fixed variables: n, k

goal (lemma (fib_add), 1 subgoal):
 1. fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n
```

Performing Induction

- The first command performs induction on *n* using the rule `fib.induct`.
- Isabelle produced this rule while processing the recursive definition of the Fibonacci function.
- The proof window now displays three subgoals.

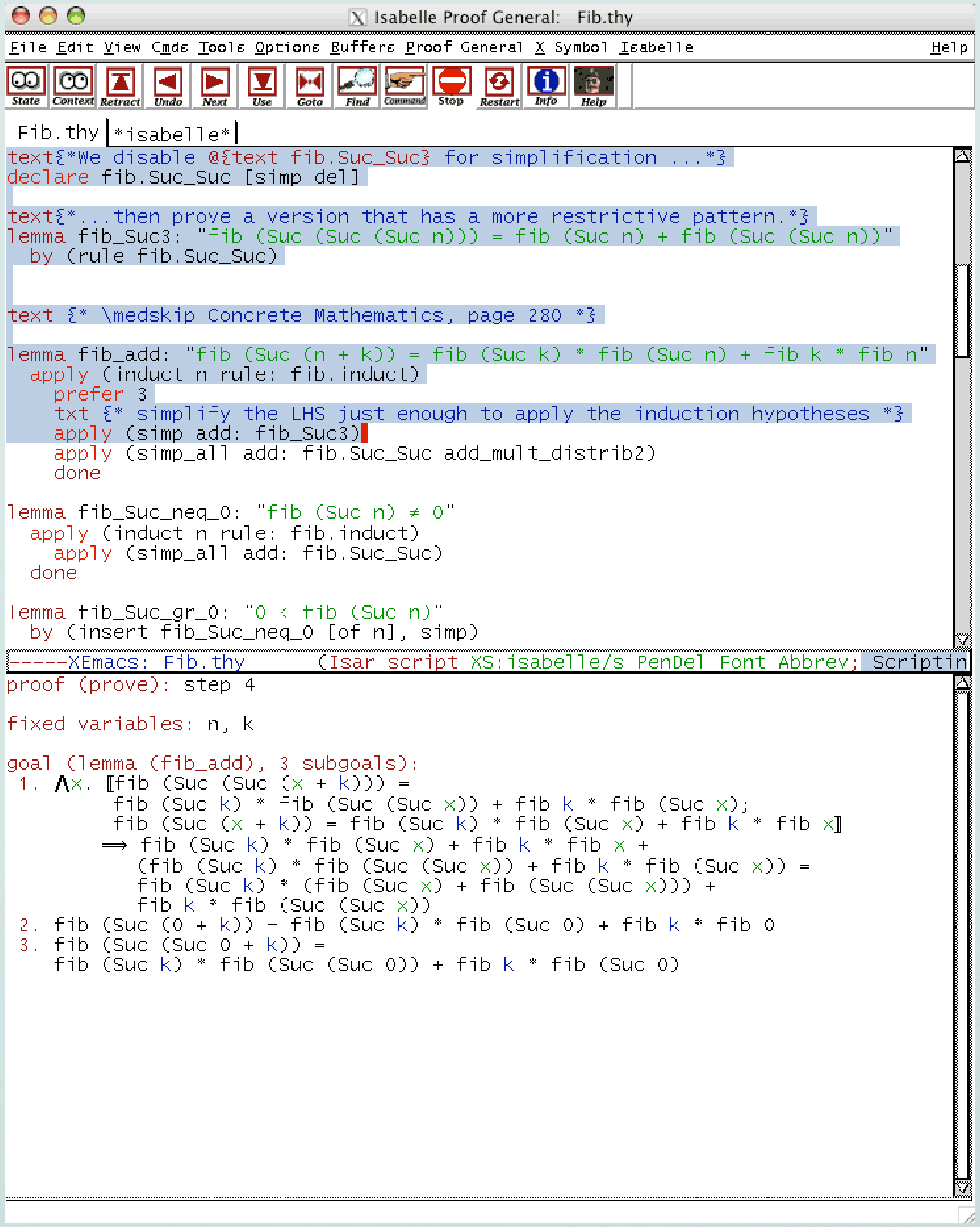

Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle
Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

```
text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) ≠ 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done

lemma fib_Suc_gr_0: "0 < fib (Suc n)"
  by (insert fib_Suc_neq_0 [of n], simp)
-----XEmacs: Fib.thy       (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
proof (prove): step 1

fixed variables: n, k

goal (lemma (fib_add), 3 subgoals):
 1. fib (Suc (0 + k)) = fib (Suc k) * fib (Suc 0) + fib k * fib 0
 2. fib (Suc (Suc 0 + k)) =
    fib (Suc k) * fib (Suc (Suc 0)) + fib k * fib (Suc 0)
 3. ⋀x. ⟦fib (Suc (Suc x + k)) =
         fib (Suc k) * fib (Suc (Suc x)) + fib k * fib (Suc x);
         fib (Suc (x + k)) = fib (Suc k) * fib (Suc x) + fib k * fib x⟧
        ⟹ fib (Suc (Suc (Suc x) + k)) =
           fib (Suc k) * fib (Suc (Suc (Suc x))) + fib k * fib (Suc (Suc x))
```

A Rewriting Step

- The third subgoal is selected: `prefer 3`.
- Then, it is simplified with the help of a rewrite rule called `fib_Suc3`.
- This subgoal is still rather complicated!


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State Context Retract Undo Next Use Goto Find Command Stop Restart Info Help
Fib.thy *isabelle*

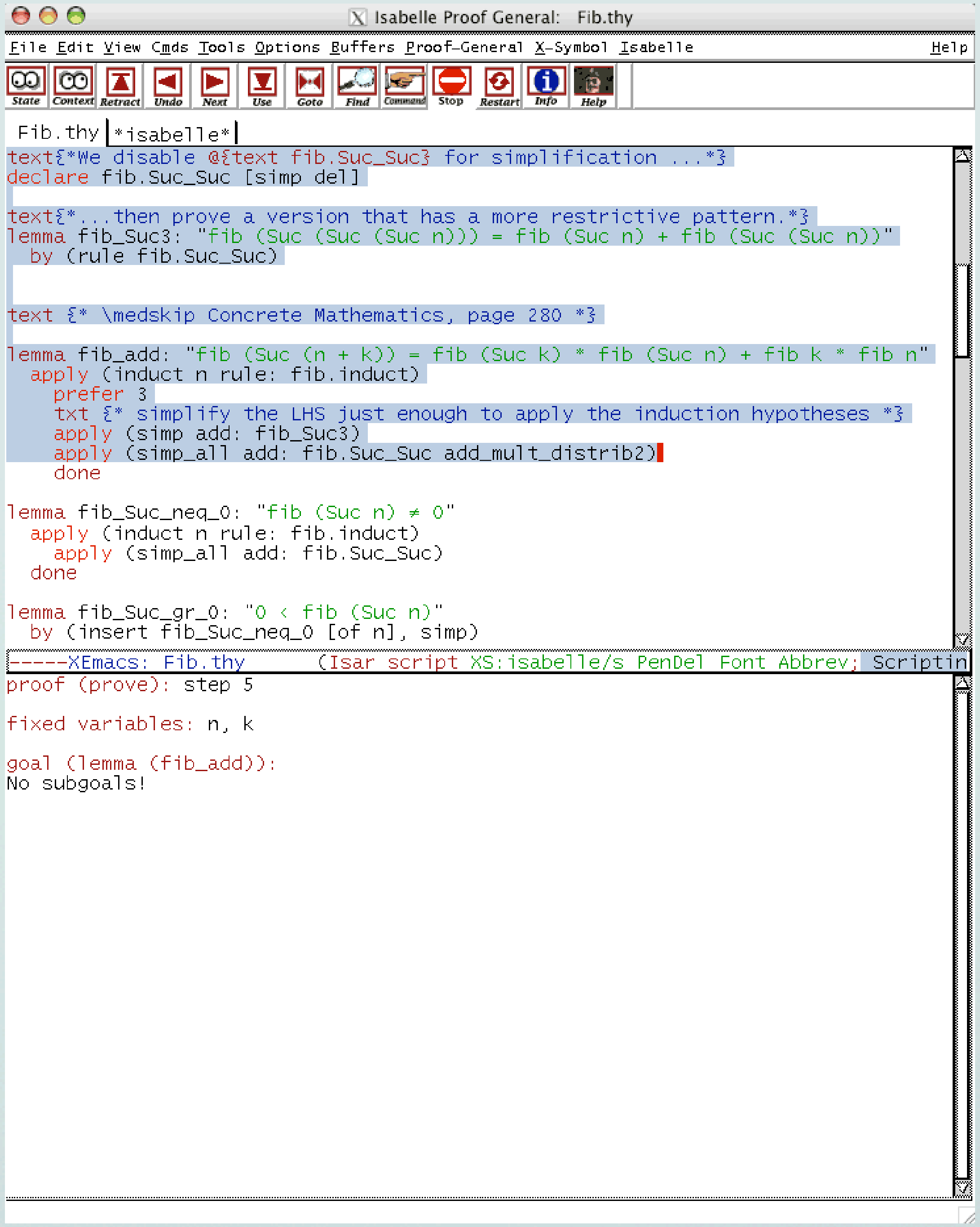
```
text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) ≠ 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done

lemma fib_Suc_gr_0: "0 < fib (Suc n)"
  by (insert fib_Suc_neq_0 [of n], simp)
```

-----XEmacs: Fib.thy   (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin

```
proof (prove): step 4

fixed variables: n, k

goal (lemma (fib_add), 3 subgoals):
 1. ⋀x. ⟦fib (Suc (Suc (x + k))) =
         fib (Suc k) * fib (Suc (Suc x)) + fib k * fib (Suc x);
         fib (Suc (x + k)) = fib (Suc k) * fib (Suc x) + fib k * fib x⟧
        ⟹ fib (Suc k) * fib (Suc x) + fib k * fib x +
           (fib (Suc k) * fib (Suc (Suc x)) + fib k * fib (Suc x)) =
           fib (Suc k) * (fib (Suc x) + fib (Suc (Suc x))) +
           fib k * fib (Suc (Suc x))
 2. fib (Suc (0 + k)) = fib (Suc k) * fib (Suc 0) + fib k * fib 0
 3. fib (Suc (Suc 0 + k)) =
    fib (Suc k) * fib (Suc (Suc 0)) + fib k * fib (Suc 0)
```

Finishing the Proof

- Next, all three subgoals are simplified, with the help of the rewrite rules shown.
- The simplifier automatically includes hundreds of other rewrite rules, as well as various decision procedures.
- This time, no subgoals remain.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State Context Retract Undo Next Use Goto Find Command Stop Restart Info Help
Fib.thy *isabelle*

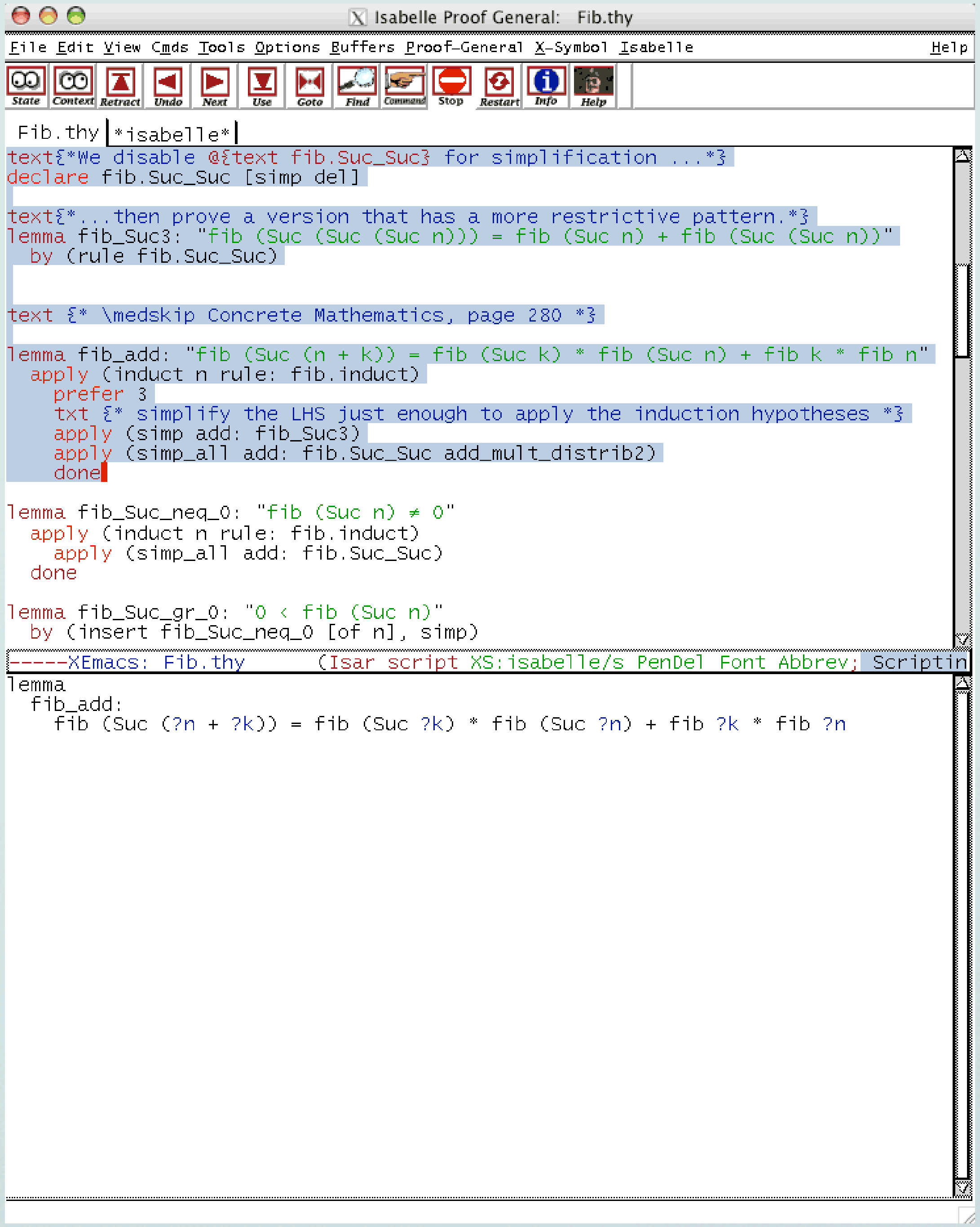
```
text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) ≠ 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done

lemma fib_Suc_gr_0: "0 < fib (Suc n)"
  by (insert fib_Suc_neq_0 [of n], simp)
```

-----XEmacs: Fib.thy   (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin

```
proof (prove): step 5

fixed variables: n, k

goal (lemma (fib_add)):
No subgoals!
```

Storing the Theorem

- The done command causes Isabelle to accept the proof, storing the theorem.
- If you were proving this theorem for the first time, you would try various commands right in the editor buffer. You would use *Undo* frequently!
- Once you have succeeded, the file will hold the final version of your proof.
- Using *Undo* on a done command moves the cursor above its proof. Isabelle "forgets" the theorem.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State Context Retract Undo Next Use Goto Find Command Stop Restart Info Help
Fib.thy *isabelle*

```
text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) ≠ 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done

lemma fib_Suc_gr_0: "0 < fib (Suc n)"
  by (insert fib_Suc_neq_0 [of n], simp)
```

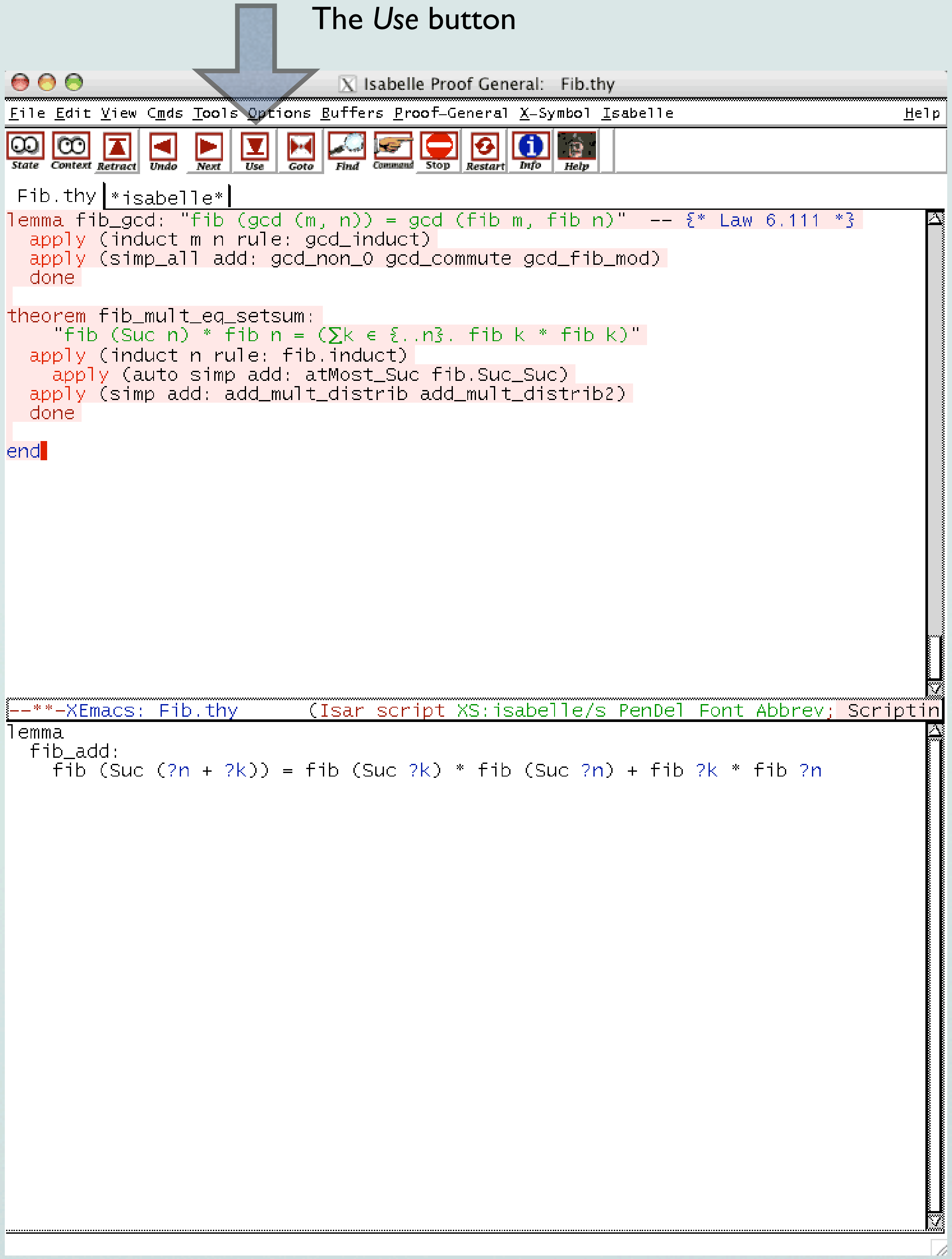
-----XEmacs: Fib.thy   (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin

```
lemma
  fib_add:
    fib (Suc (?n + ?k)) = fib (Suc ?k) * fib (Suc ?n) + fib ?k * fib ?n
```

Processing a Theory

- To run a theory right to the end, click on the *Use* button.
- Now the rest of the theory appears in pink until Isabelle can process it.

The *Use* button

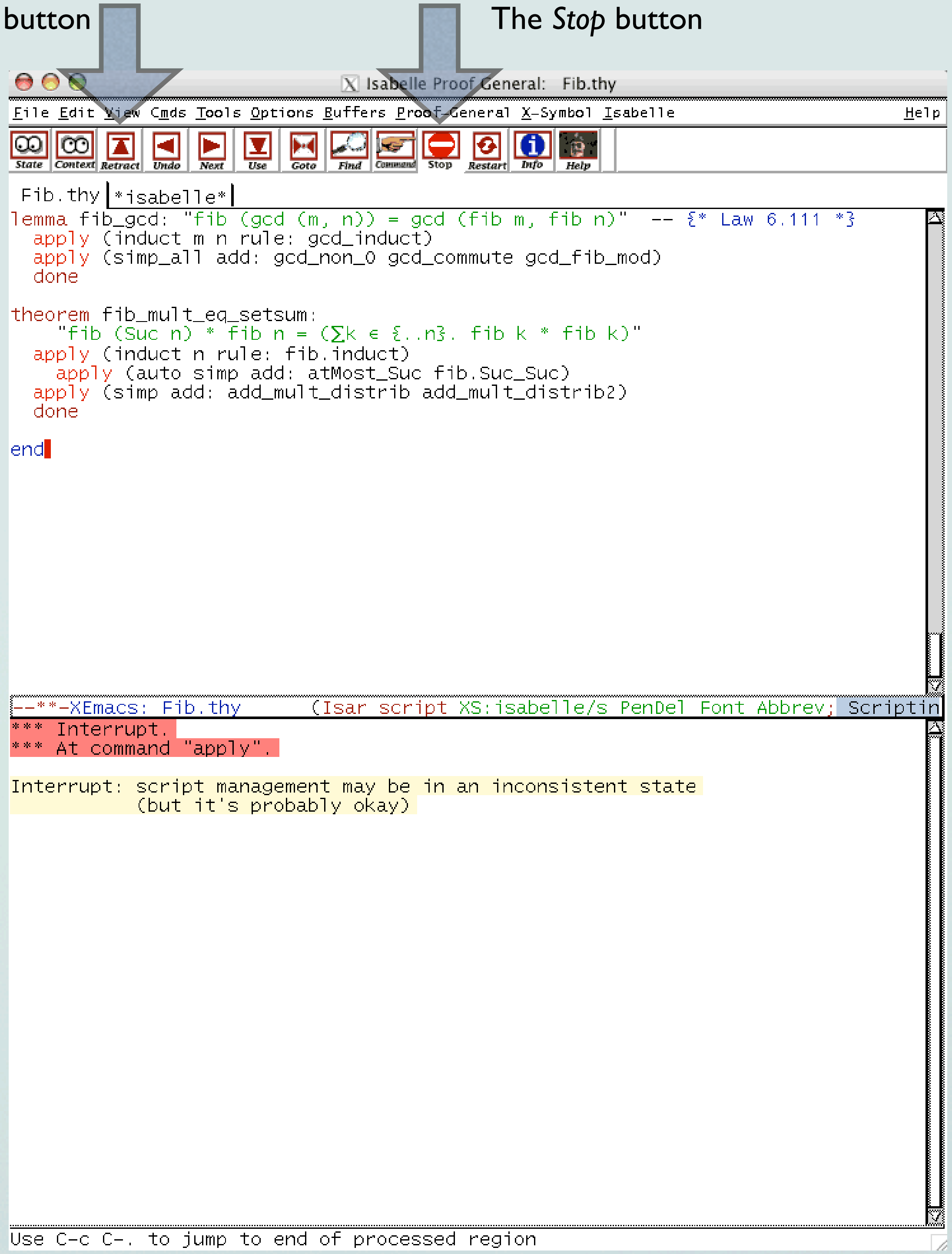

Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle
--**-XEmacs: Fib.thy (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin

```
lemma fib_gcd: "fib (gcd (m, n)) = gcd (fib m, fib n)"  -- {* Law 6.111 *}
  apply (induct m n rule: gcd_induct)
  apply (simp_all add: gcd_non_0 gcd_commute gcd_fib_mod)
  done

theorem fib_mult_eq_setsum:
    "fib (Suc n) * fib n = (∑k ∈ {..n}. fib k * fib k)"
  apply (induct n rule: fib.induct)
    apply (auto simp add: atMost_Suc fib.Suc_Suc)
  apply (simp add: add_mult_distrib add_mult_distrib2)
  done

end
```

```
lemma
  fib_add:
    fib (Suc (?n + ?k)) = fib (Suc ?k) * fib (Suc ?n) + fib ?k * fib ?n
```

Stop!

- Proof taking too long? Simplifier's looping? Clicked the wrong button? Just click on *Stop*.
- If things behave weirdly after this, perhaps Proof General has got out of sync with Isabelle.
- To get back into sync, use *Goto* to go back to the start of the current proof.
- You can use *Revert* to go back to the top of the theory file.



The *Revert* button

The *Stop* button

Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

```
lemma fib_gcd: "fib (gcd (m, n)) = gcd (fib m, fib n)"  -- {* Law 6.111 *}
  apply (induct m n rule: gcd_induct)
  apply (simp_all add: gcd_non_0 gcd_commute gcd_fib_mod)
  done

theorem fib_mult_eq_setsum:
    "fib (Suc n) * fib n = (∑k ∈ {..n}. fib k * fib k)"
  apply (induct n rule: fib.induct)
    apply (auto simp add: atMost_Suc fib.Suc_Suc)
  apply (simp add: add_mult_distrib add_mult_distrib2)
  done

end
```

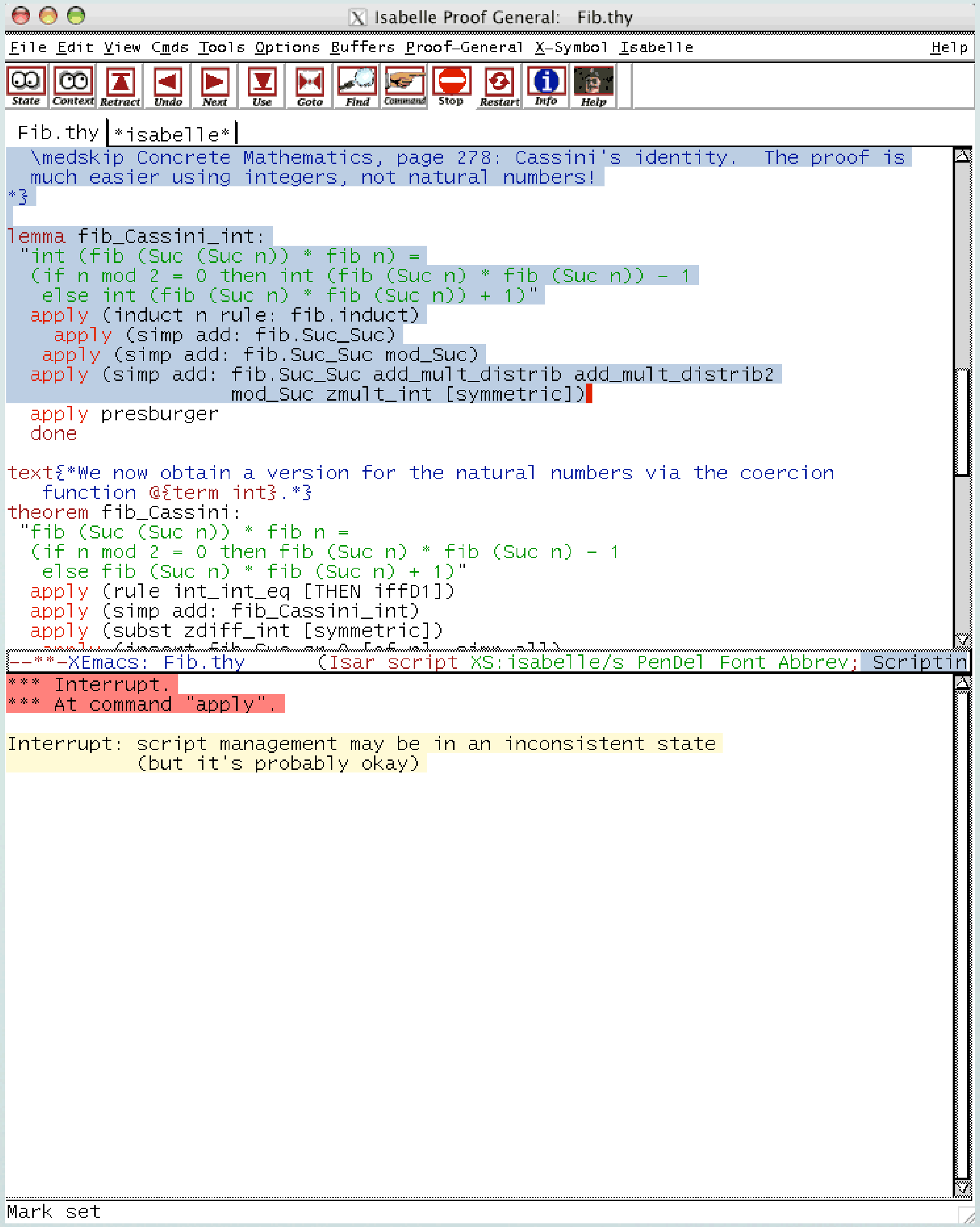
```
--**-XEmacs: Fib.thy      (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
*** Interrupt.
*** At command "apply".

Interrupt: script management may be in an inconsistent state
           (but it's probably okay)
```

```
Use C-c C-. to jump to end of processed region
```

Where Am I?

- If a proof fails—or is interrupted—in a long theory file, how do we locate the critical spot?
- You could simply scroll through the file until you find the end of the blue region.
- To jump right there, use the menu item `Proof General > Goto Locked End`. The key combination `CTRL/C-.` does the same thing.
- The proof was interrupted during a call to `presburger`, an arithmetic decision procedure.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

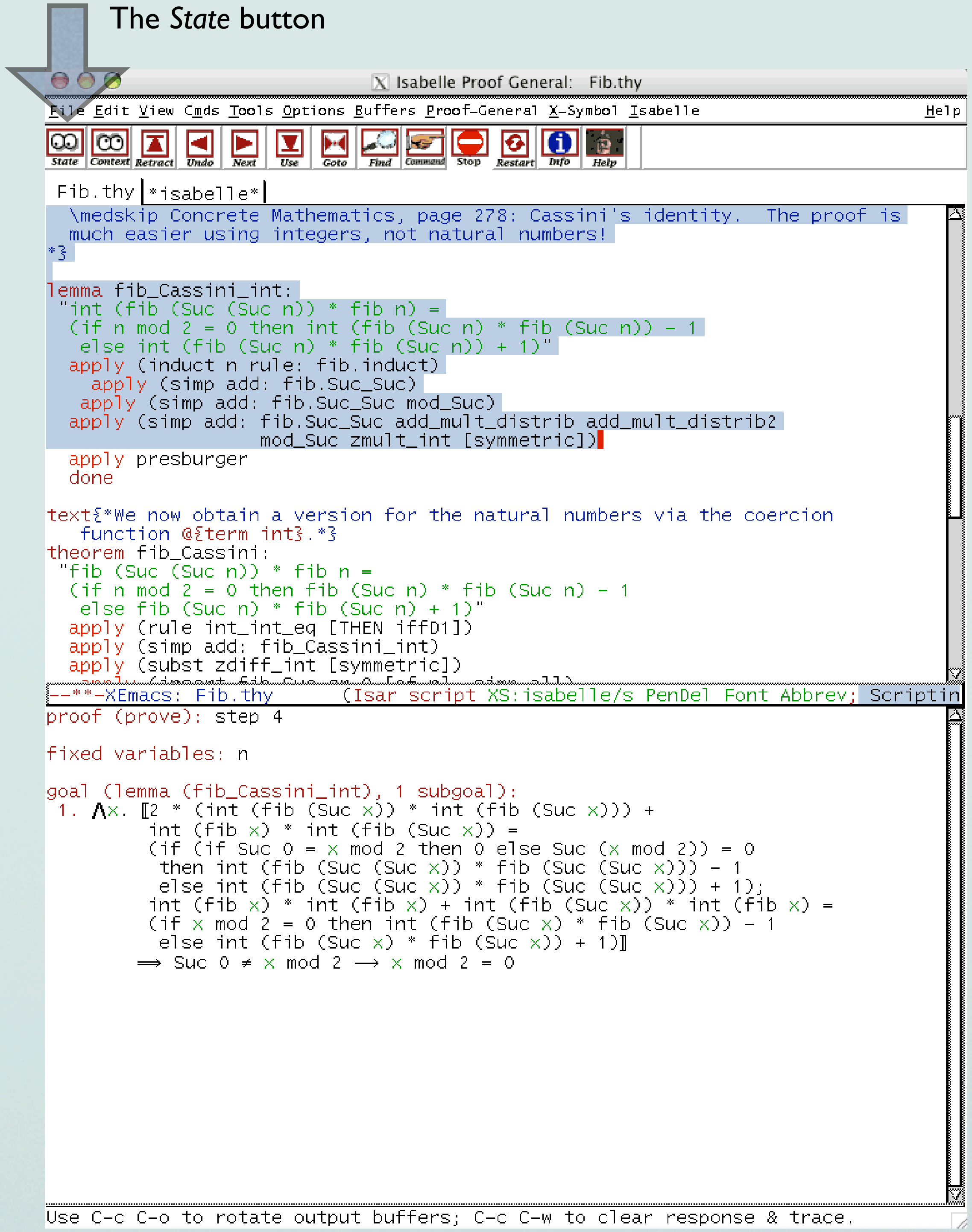
```
  \medskip Concrete Mathematics, page 278: Cassini's identity.  The proof is
  much easier using integers, not natural numbers!
*}

lemma fib_Cassini_int:
 "int (fib (Suc (Suc n)) * fib n) =
  (if n mod 2 = 0 then int (fib (Suc n) * fib (Suc n)) - 1
   else int (fib (Suc n) * fib (Suc n)) + 1)"
  apply (induct n rule: fib.induct)
    apply (simp add: fib.Suc_Suc)
   apply (simp add: fib.Suc_Suc mod_Suc)
  apply (simp add: fib.Suc_Suc add_mult_distrib add_mult_distrib2
                   mod_Suc zmult_int [symmetric])
  apply presburger
  done

text{*We now obtain a version for the natural numbers via the coercion
   function @{term int}.*}
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib n =
  (if n mod 2 = 0 then fib (Suc n) * fib (Suc n) - 1
   else fib (Suc n) * fib (Suc n) + 1)"
  apply (rule int_int_eq [THEN iffD1])
  apply (simp add: fib_Cassini_int)
  apply (subst zdiff_int [symmetric])
```

--**-XEmacs: Fib.thy   (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin

```
*** Interrupt.
*** At command "apply".

Interrupt: script management may be in an inconsistent state
           (but it's probably okay)
```

The Proof State

The *State* button


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State Context Retract Undo Next Use Goto Find Command Stop Restart Info Help
Fib.thy
isabelle

```
  \medskip Concrete Mathematics, page 278: Cassini's identity.  The proof is
  much easier using integers, not natural numbers!
*}

lemma fib_Cassini_int:
 "int (fib (Suc (Suc n)) * fib n) =
  (if n mod 2 = 0 then int (fib (Suc n) * fib (Suc n)) - 1
   else int (fib (Suc n) * fib (Suc n)) + 1)"
  apply (induct n rule: fib.induct)
    apply (simp add: fib.Suc_Suc)
   apply (simp add: fib.Suc_Suc mod_Suc)
  apply (simp add: fib.Suc_Suc add_mult_distrib add_mult_distrib2
                   mod_Suc zmult_int [symmetric])
  apply presburger
  done

text{*We now obtain a version for the natural numbers via the coercion
   function @{term int}.*}
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib n =
  (if n mod 2 = 0 then fib (Suc n) * fib (Suc n) - 1
   else fib (Suc n) * fib (Suc n) + 1)"
  apply (rule int_int_eq [THEN iffD1])
  apply (simp add: fib_Cassini_int)
  apply (subst zdiff_int [symmetric])
```

```
--**-XEmacs: Fib.thy        (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
proof (prove): step 4

fixed variables: n

goal (lemma (fib_Cassini_int), 1 subgoal):
 1. ⋀x. ⟦2 * (int (fib (Suc x)) * int (fib (Suc x))) +
         int (fib x) * int (fib (Suc x)) =
         (if (if Suc 0 = x mod 2 then 0 else Suc (x mod 2)) = 0
          then int (fib (Suc (Suc x)) * fib (Suc (Suc x))) - 1
          else int (fib (Suc (Suc x)) * fib (Suc (Suc x))) + 1);
         int (fib x) * int (fib x) + int (fib (Suc x)) * int (fib x) =
         (if x mod 2 = 0 then int (fib (Suc x) * fib (Suc x)) - 1
          else int (fib (Suc x) * fib (Suc x)) + 1)⟧
        ⟹ Suc 0 ≠ x mod 2 ⟶ x mod 2 = 0
```

```
Use C-c C-o to rotate output buffers; C-c C-w to clear response & trace.
```

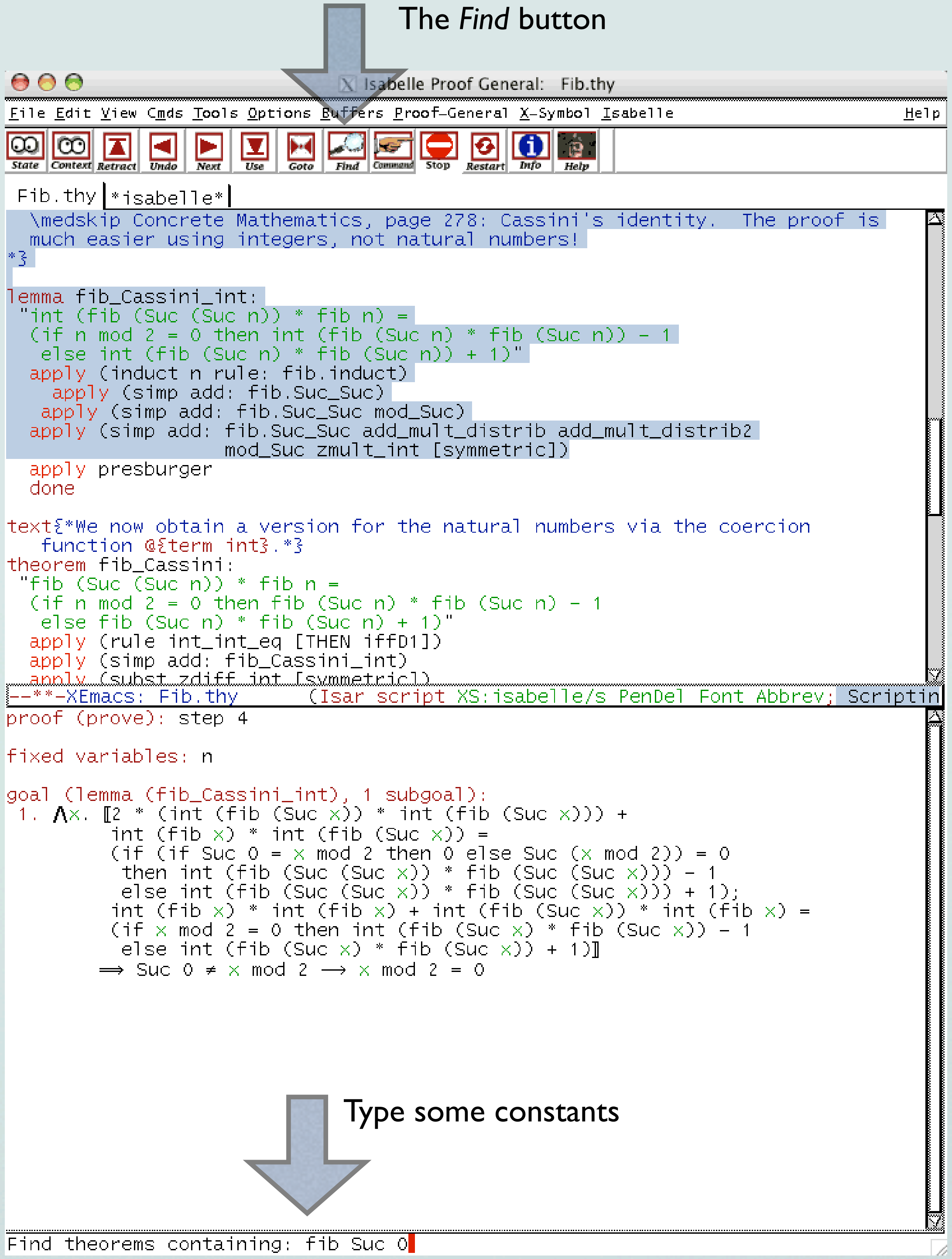
- Clicking on the *State* button reveals the proof state at the given point.

- Here, there was one subgoal left when the proof was interrupted.

Finding Theorems

- Isabelle provides thousands of lemmas. How do you find the ones you need? One way is to click the *Find* button.
- Then, type some constants—or entire terms—into the XEmacs minibuffer.
- Type the term "(_+_)*_ = _", including the quotation marks, to see all theorems containing an instance of this pattern.
- The pattern "_+_" matches all terms containing the infix operator + because _ matches any term.

The *Find* button



Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle
Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

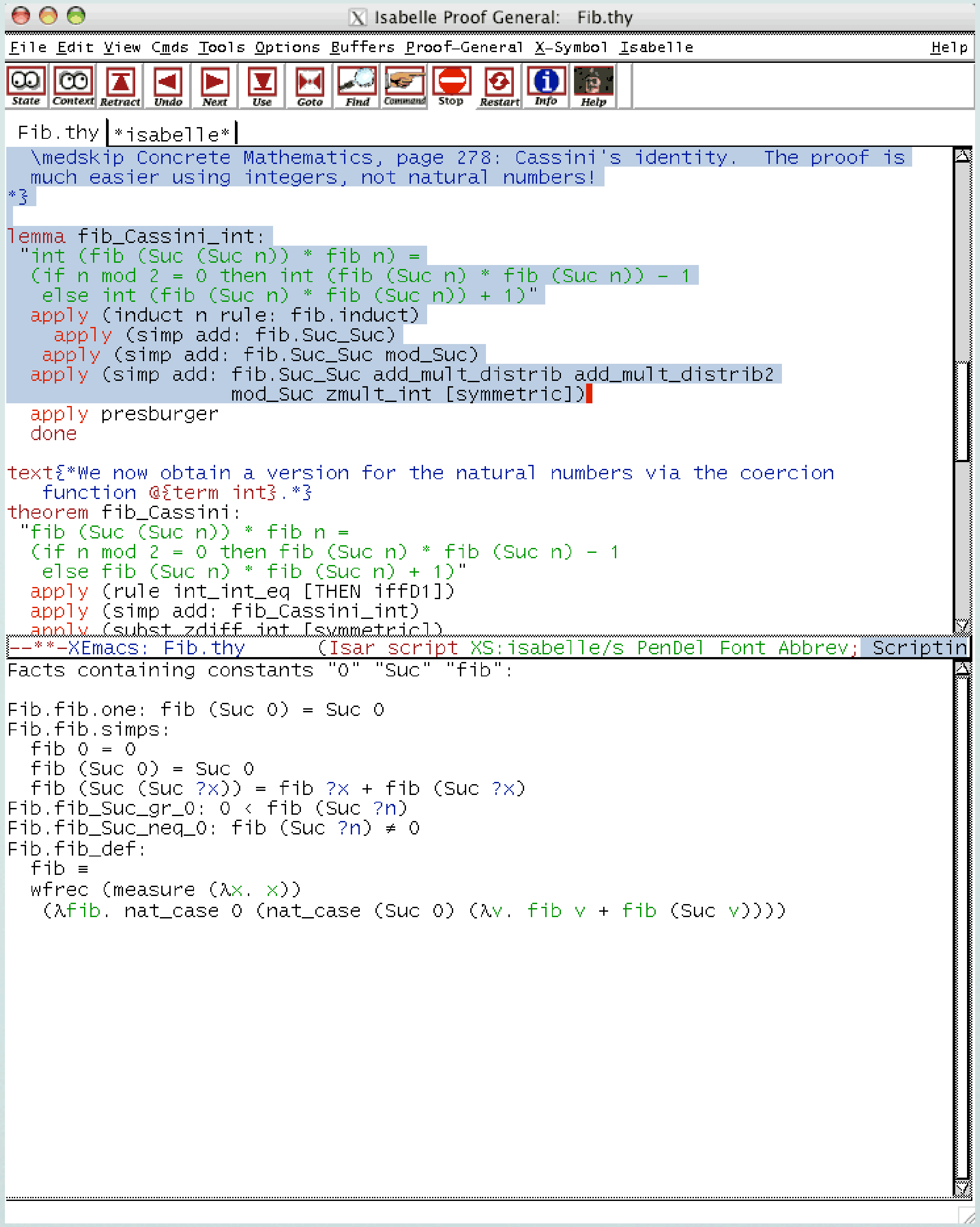
```
  \medskip Concrete Mathematics, page 278: Cassini's identity.  The proof is
  much easier using integers, not natural numbers!
*}

lemma fib_Cassini_int:
 "int (fib (Suc (Suc n)) * fib n) =
  (if n mod 2 = 0 then int (fib (Suc n) * fib (Suc n)) - 1
   else int (fib (Suc n) * fib (Suc n)) + 1)"
  apply (induct n rule: fib.induct)
    apply (simp add: fib.Suc_Suc)
   apply (simp add: fib.Suc_Suc mod_Suc)
  apply (simp add: fib.Suc_Suc add_mult_distrib add_mult_distrib2
                   mod_Suc zmult_int [symmetric])
  apply presburger
  done

text{*We now obtain a version for the natural numbers via the coercion
   function @{term int}.*}
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib n =
  (if n mod 2 = 0 then fib (Suc n) * fib (Suc n) - 1
   else fib (Suc n) * fib (Suc n) + 1)"
  apply (rule int_int_eq [THEN iffD1])
  apply (simp add: fib_Cassini_int)
  apply (subst zdiff_int [symmetric])
--**-XEmacs: Fib.thy       (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
proof (prove): step 4

fixed variables: n

goal (lemma (fib_Cassini_int), 1 subgoal):
 1. ⋀x. ⟦2 * (int (fib (Suc x)) * int (fib (Suc x))) +
        int (fib x) * int (fib (Suc x)) =
        (if (if Suc 0 = x mod 2 then 0 else Suc (x mod 2)) = 0
         then int (fib (Suc (Suc x)) * fib (Suc (Suc x))) - 1
         else int (fib (Suc (Suc x)) * fib (Suc (Suc x))) + 1);
        int (fib x) * int (fib x) + int (fib (Suc x)) * int (fib x) =
        (if x mod 2 = 0 then int (fib (Suc x) * fib (Suc x)) - 1
         else int (fib (Suc x) * fib (Suc x)) + 1)⟧
       ⟹ Suc 0 ≠ x mod 2 ⟶ x mod 2 = 0
```

Type some constants

```
Find theorems containing: fib Suc 0
```

Theorems Found

- The response buffer lists the theorems containing *all* of the listed constants.
- If you are lucky, there will be just a few rather than hundreds!
- The more patterns you type, the fewer theorems will be displayed.
- During the search, variables mentioned in the current goal are viewed as constants.


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle
Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

```
  \medskip Concrete Mathematics, page 278: Cassini's identity.  The proof is
  much easier using integers, not natural numbers!
*}

lemma fib_Cassini_int:
 "int (fib (Suc (Suc n)) * fib n) =
  (if n mod 2 = 0 then int (fib (Suc n) * fib (Suc n)) - 1
   else int (fib (Suc n) * fib (Suc n)) + 1)"
  apply (induct n rule: fib.induct)
    apply (simp add: fib.Suc_Suc)
   apply (simp add: fib.Suc_Suc mod_Suc)
  apply (simp add: fib.Suc_Suc add_mult_distrib add_mult_distrib2
                   mod_Suc zmult_int [symmetric])
  apply presburger
  done

text{*We now obtain a version for the natural numbers via the coercion
   function @{term int}.*}
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib n =
  (if n mod 2 = 0 then fib (Suc n) * fib (Suc n) - 1
   else fib (Suc n) * fib (Suc n) + 1)"
  apply (rule int_int_eq [THEN iffD1])
  apply (simp add: fib_Cassini_int)
  apply (subst zdiff_int [symmetric])
```

```
--**-XEmacs: Fib.thy       (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

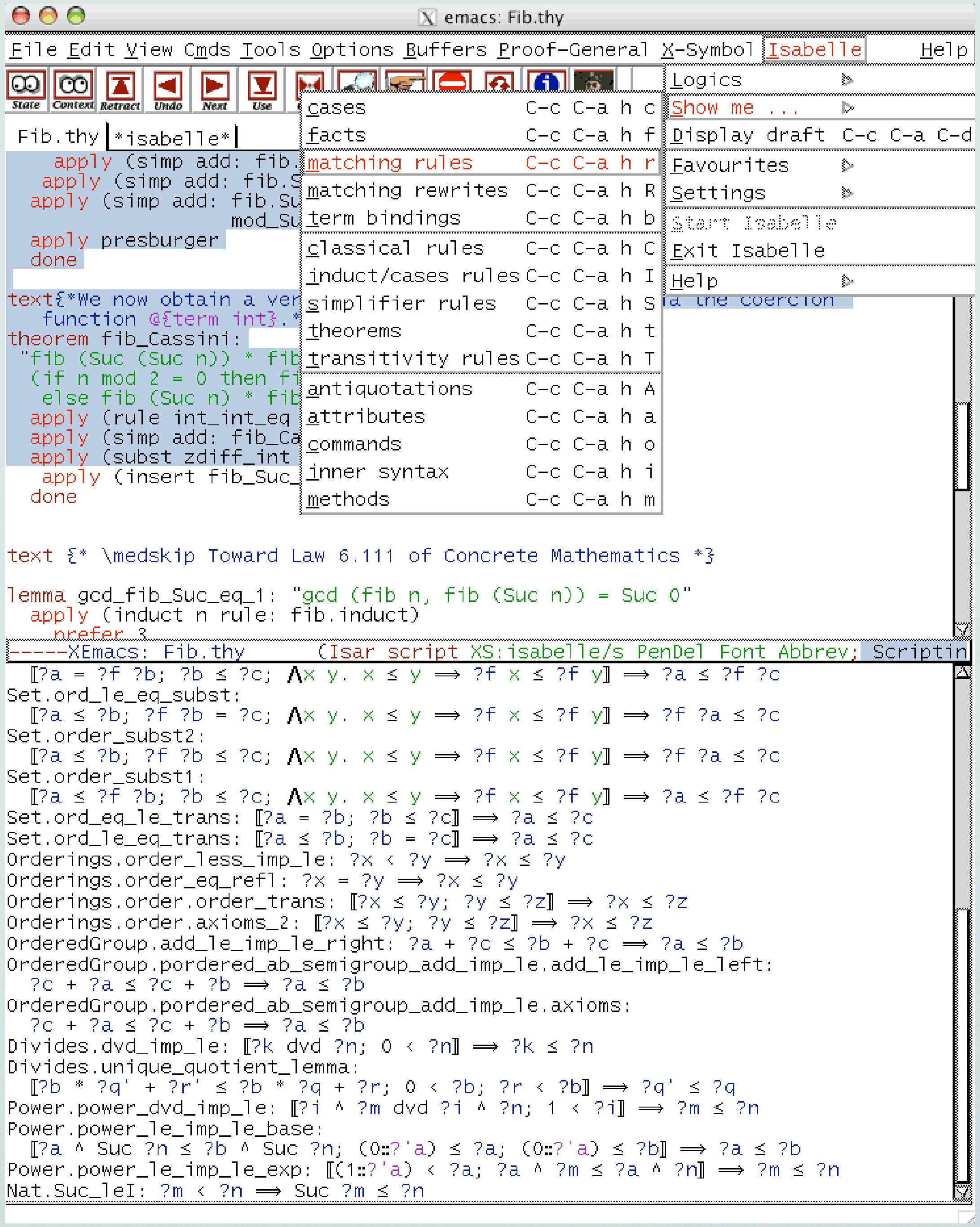
```
Facts containing constants "0" "Suc" "fib":

Fib.fib.one: fib (Suc 0) = Suc 0
Fib.fib.simps:
  fib 0 = 0
  fib (Suc 0) = Suc 0
  fib (Suc (Suc ?x)) = fib ?x + fib (Suc ?x)
Fib.fib_Suc_gr_0: 0 < fib (Suc ?n)
Fib.fib_Suc_neq_0: fib (Suc ?n) ≠ 0
Fib.fib_def:
  fib ≡
  wfrec (measure (λx. x))
   (λfib. nat_case 0 (nat_case (Suc 0) (λv. fib v + fib (Suc v))))
```

The Isabelle Menu

- The `Isabelle` menu gives access to Isabelle commands and information.

- `Isabelle > Show me...` provides other ways of finding theorems: `matching rules` and `matching rewrites`.

- In the example, the current subgoal has the form *x* ≤ *y*, and `matching rules` displays all known theorems that can prove a conclusion of that form.


emacs: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
State
Context
Retract
Undo
Next
Use
Fib.thy
isabelle
Logics ▷
Show me ... ▷
Display draft C-c C-a C-d
Favourites ▷
Settings ▷
Start Isabelle
Exit Isabelle
Help ▷
cases C-c C-a h c
facts C-c C-a h f
matching rules C-c C-a h r
matching rewrites C-c C-a h R
term bindings C-c C-a h b
classical rules C-c C-a h C
induct/cases rules C-c C-a h I
simplifier rules C-c C-a h S
theorems C-c C-a h t
transitivity rules C-c C-a h T
antiquotations C-c C-a h A
attributes C-c C-a h a
commands C-c C-a h o
inner syntax C-c C-a h i
methods C-c C-a h m

```
    apply (simp add: fib.
   apply (simp add: fib.S
  apply (simp add: fib.Su
                   mod_Su
  apply presburger
  done

text{*We now obtain a ver                        a the coercion
   function @{term int}.*
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib
  (if n mod 2 = 0 then fi
   else fib (Suc n) * fib
  apply (rule int_int_eq
  apply (simp add: fib_Ca
  apply (subst zdiff_int
   apply (insert fib_Suc_
  done


text {* \medskip Toward Law 6.111 of Concrete Mathematics *}

lemma gcd_fib_Suc_eq_1: "gcd (fib n, fib (Suc n)) = Suc 0"
  apply (induct n rule: fib.induct)
   prefer 3
-----XEmacs: Fib.thy      (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
  ⟦?a = ?f ?b; ?b ≤ ?c; ⋀x y. x ≤ y ⟹ ?f x ≤ ?f y⟧ ⟹ ?a ≤ ?f ?c
Set.ord_le_eq_subst:
  ⟦?a ≤ ?b; ?f ?b = ?c; ⋀x y. x ≤ y ⟹ ?f x ≤ ?f y⟧ ⟹ ?f ?a ≤ ?c
Set.order_subst2:
  ⟦?a ≤ ?b; ?f ?b ≤ ?c; ⋀x y. x ≤ y ⟹ ?f x ≤ ?f y⟧ ⟹ ?f ?a ≤ ?c
Set.order_subst1:
  ⟦?a ≤ ?f ?b; ?b ≤ ?c; ⋀x y. x ≤ y ⟹ ?f x ≤ ?f y⟧ ⟹ ?a ≤ ?f ?c
Set.ord_eq_le_trans: ⟦?a = ?b; ?b ≤ ?c⟧ ⟹ ?a ≤ ?c
Set.ord_le_eq_trans: ⟦?a ≤ ?b; ?b = ?c⟧ ⟹ ?a ≤ ?c
Orderings.order_less_imp_le: ?x < ?y ⟹ ?x ≤ ?y
Orderings.order_eq_refl: ?x = ?y ⟹ ?x ≤ ?y
Orderings.order.order_trans: ⟦?x ≤ ?y; ?y ≤ ?z⟧ ⟹ ?x ≤ ?z
Orderings.order.axioms_2: ⟦?x ≤ ?y; ?y ≤ ?z⟧ ⟹ ?x ≤ ?z
OrderedGroup.add_le_imp_le_right: ?a + ?c ≤ ?b + ?c ⟹ ?a ≤ ?b
OrderedGroup.pordered_ab_semigroup_add_imp_le.add_le_imp_le_left:
  ?c + ?a ≤ ?c + ?b ⟹ ?a ≤ ?b
OrderedGroup.pordered_ab_semigroup_add_imp_le.axioms:
  ?c + ?a ≤ ?c + ?b ⟹ ?a ≤ ?b
Divides.dvd_imp_le: ⟦?k dvd ?n; 0 < ?n⟧ ⟹ ?k ≤ ?n
Divides.unique_quotient_lemma:
  ⟦?b * ?q' + ?r' ≤ ?b * ?q + ?r; 0 < ?b; ?r < ?b⟧ ⟹ ?q' ≤ ?q
Power.power_dvd_imp_le: ⟦?i ^ ?m dvd ?i ^ ?n; 1 < ?i⟧ ⟹ ?m ≤ ?n
Power.power_le_imp_le_base:
  ⟦?a ^ Suc ?n ≤ ?b ^ Suc ?n; (0::?'a) ≤ ?a; (0::?'a) ≤ ?b⟧ ⟹ ?a ≤ ?b
Power.power_le_imp_le_exp: ⟦(1::?'a) < ?a; ?a ^ ?m ≤ ?a ^ ?n⟧ ⟹ ?m ≤ ?n
Nat.Suc_leI: ?m < ?n ⟹ Suc ?m ≤ ?n
```

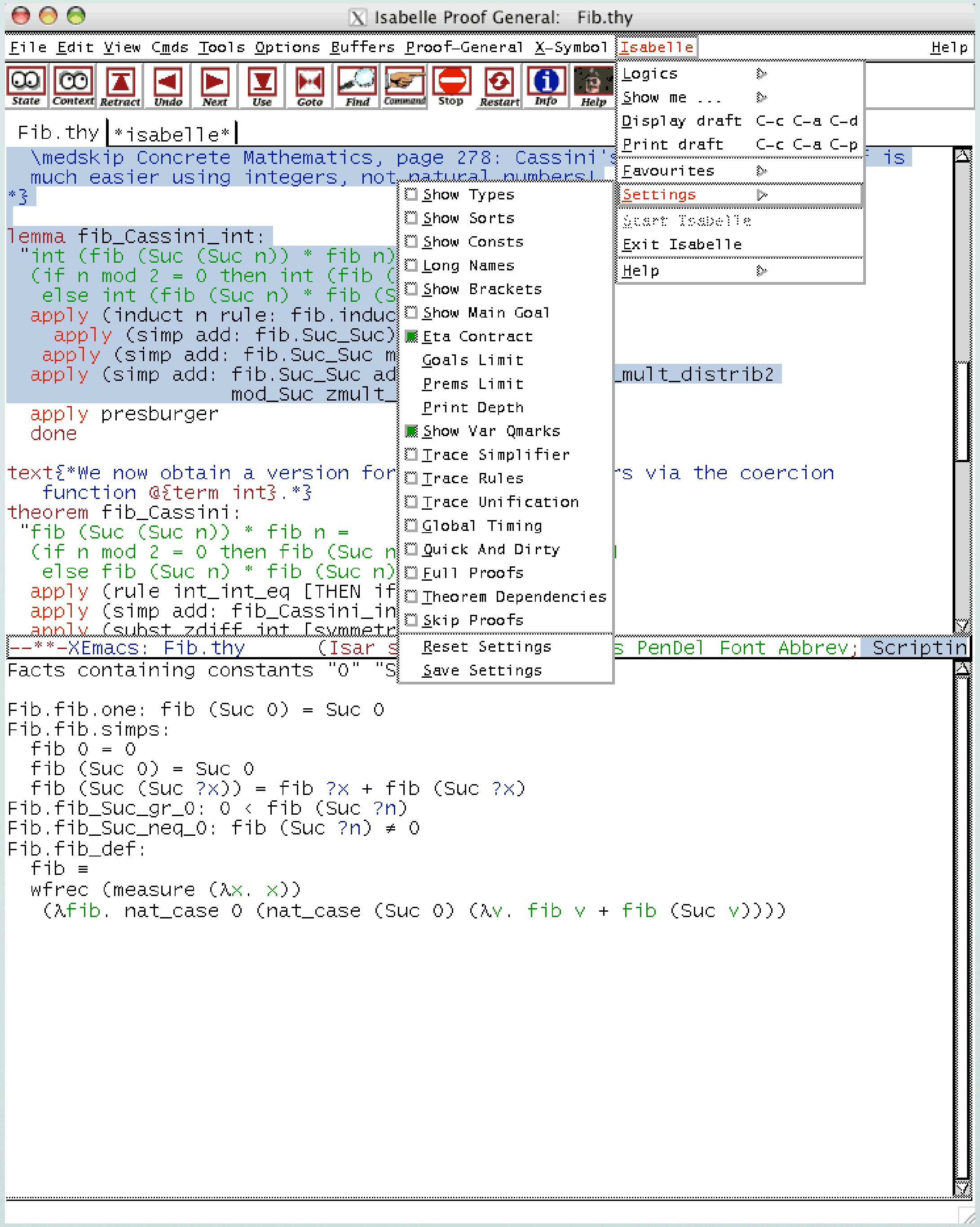
Settings

- The menu `Isabelle > Settings` can request the display of types, execution times, and various traces.

- There are printing options to suit special situations, such as enormous subgoals.

- Use `Show Types` and `Show Sorts` to cause more type information to be displayed.

- The various `Show` options make the output more verbose, but more explicit, and are helpful for diagnosing problems.



```
  \medskip Concrete Mathematics, page 278: Cassini's                       f is
  much easier using integers, not natural numbers!
*}

lemma fib_Cassini_int:
 "int (fib (Suc (Suc n)) * fib n)
  (if n mod 2 = 0 then int (fib (
   else int (fib (Suc n) * fib (S
  apply (induct n rule: fib.induc
    apply (simp add: fib.Suc_Suc)
   apply (simp add: fib.Suc_Suc m
  apply (simp add: fib.Suc_Suc ad                             _mult_distrib2
                    mod_Suc zmult_
  apply presburger
  done

text{*We now obtain a version for                          rs via the coercion
   function @{term int}.*}
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib n =
  (if n mod 2 = 0 then fib (Suc n
   else fib (Suc n) * fib (Suc n)
  apply (rule int_int_eq [THEN if
  apply (simp add: fib_Cassini_in
  apply (subst zdiff_int [symmetr
--**-XEmacs: Fib.thy      (Isar s                        s PenDel Font Abbrev; Scriptin
```

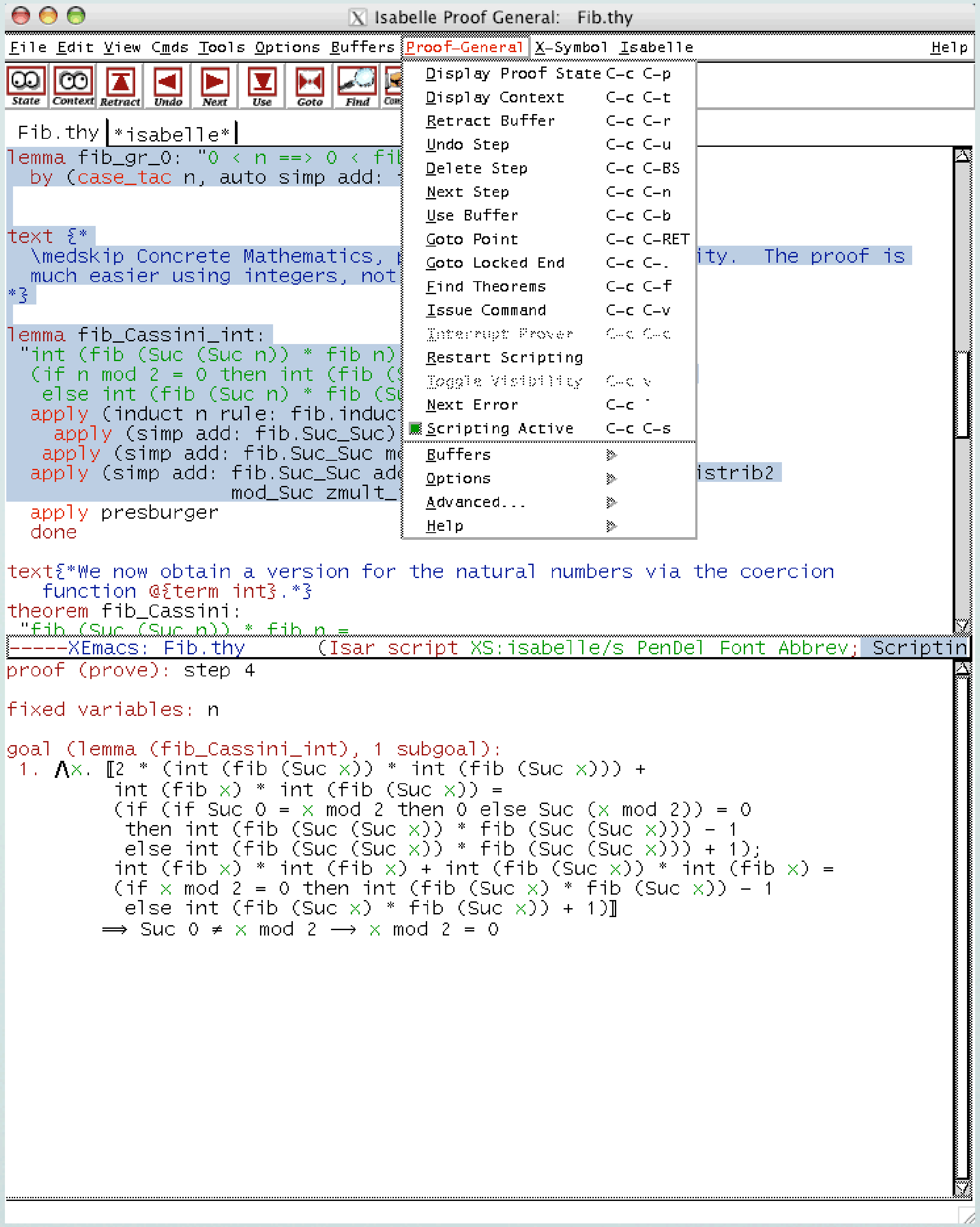
```
Facts containing constants "0" "S

Fib.fib.one: fib (Suc 0) = Suc 0
Fib.fib.simps:
  fib 0 = 0
  fib (Suc 0) = Suc 0
  fib (Suc (Suc ?x)) = fib ?x + fib (Suc ?x)
Fib.fib_Suc_gr_0: 0 < fib (Suc ?n)
Fib.fib_Suc_neq_0: fib (Suc ?n) ≠ 0
Fib.fib_def:
  fib ≡
  wfrec (measure (λx. x))
   (λfib. nat_case 0 (nat_case (Suc 0) (λv. fib v + fib (Suc v))))
```

The PG Menu

- The `Proof General` menu gives access to many commands.

- The main commands are available from the toolbar. A notable exception is `Goto Locked End`.

- Choose `Proof General > Buffers > Trace` to see tracing output.

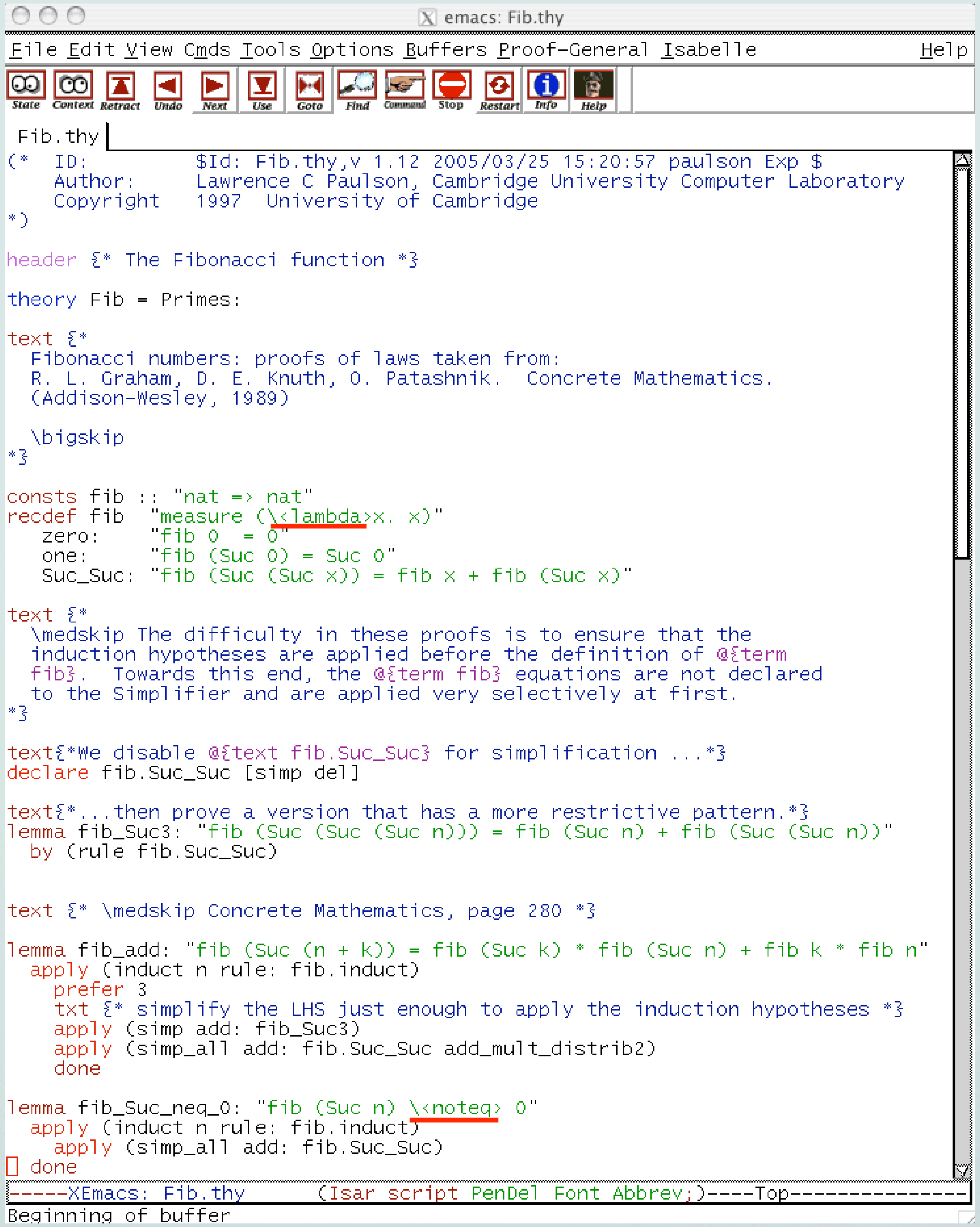


```
lemma fib_gr_0: "0 < n ==> 0 < fi
  by (case_tac n, auto simp add: 


text {*
  \medskip Concrete Mathematics, p                                  ity.  The proof is
  much easier using integers, not
*}

lemma fib_Cassini_int:
 "int (fib (Suc (Suc n)) * fib n)
  (if n mod 2 = 0 then int (fib (S
   else int (fib (Suc n) * fib (Su
  apply (induct n rule: fib.induc
    apply (simp add: fib.Suc_Suc)
   apply (simp add: fib.Suc_Suc mo
  apply (simp add: fib.Suc_Suc add                                  istrib2
                    mod_Suc zmult_
  apply presburger
  done

text{*We now obtain a version for the natural numbers via the coercion
   function @{term int}.*}
theorem fib_Cassini:
 "fib (Suc (Suc n)) * fib n =
-----XEmacs: Fib.thy       (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```

```
proof (prove): step 4

fixed variables: n

goal (lemma (fib_Cassini_int), 1 subgoal):
 1. ⋀x. ⟦2 * (int (fib (Suc x)) * int (fib (Suc x))) +
         int (fib x) * int (fib (Suc x)) =
         (if (if Suc 0 = x mod 2 then 0 else Suc (x mod 2)) = 0
          then int (fib (Suc (Suc x)) * fib (Suc (Suc x))) - 1
          else int (fib (Suc (Suc x)) * fib (Suc (Suc x))) + 1);
         int (fib x) * int (fib x) + int (fib (Suc x)) * int (fib x) =
         (if x mod 2 = 0 then int (fib (Suc x) * fib (Suc x)) - 1
          else int (fib (Suc x) * fib (Suc x)) + 1)⟧
        ⟹ Suc 0 ≠ x mod 2 ⟶ x mod 2 = 0
```

Mathematical Symbols

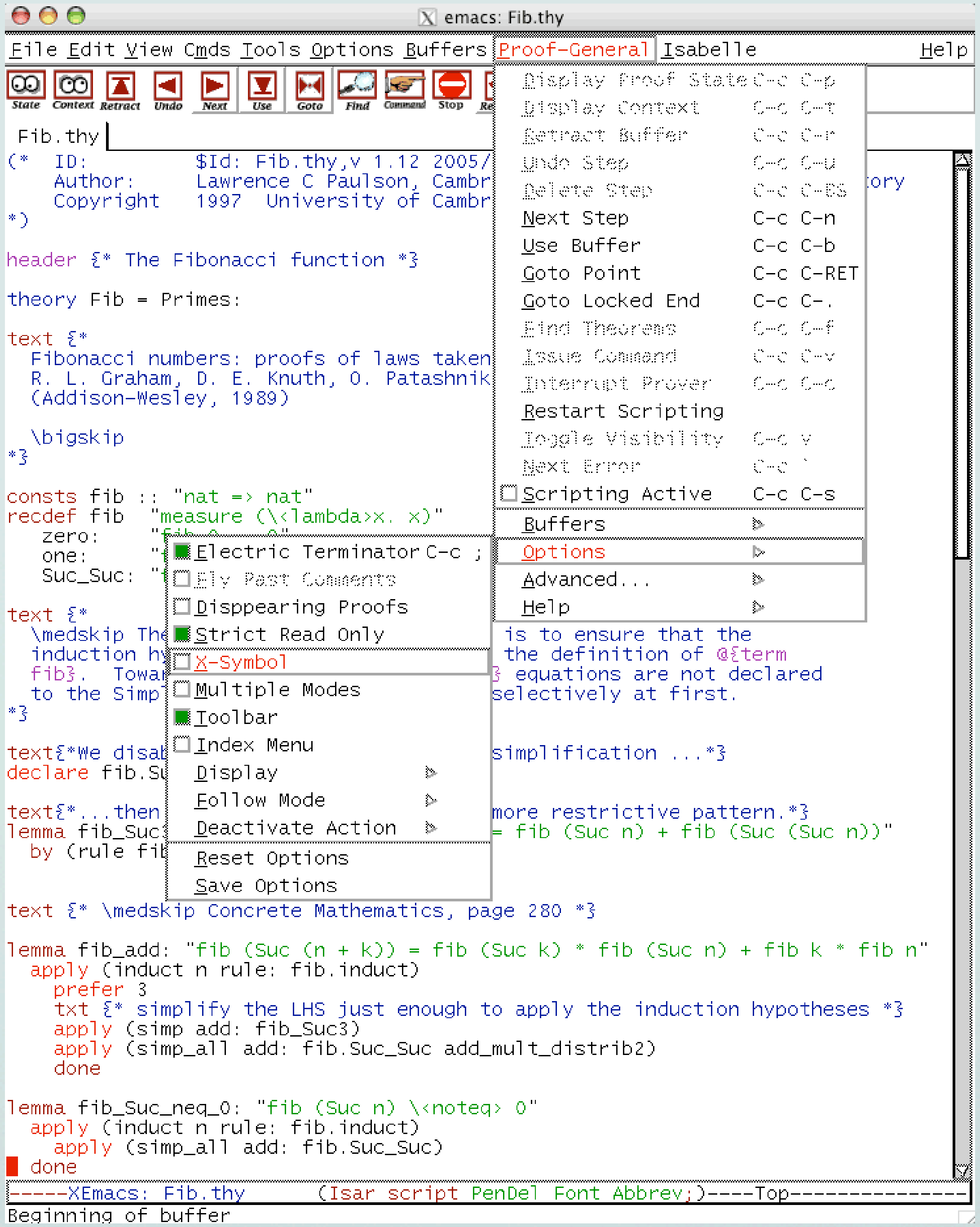
- Proof General uses the X-Symbol package to display mathematical symbols such as λ ≤ ≠ ∈ ∉ ∪ and ∩
- The package is included with Proof General, but may need to be switched on.
- If X-Symbol mode is off, Proof General will display ASCII escape sequences, as shown on the right.



```
(*   ID:         $Id: Fib.thy,v 1.12 2005/03/25 15:20:57 paulson Exp $
     Author:     Lawrence C Paulson, Cambridge University Computer Laboratory
     Copyright   1997  University of Cambridge
*)

header {* The Fibonacci function *}

theory Fib = Primes:

text {*
  Fibonacci numbers: proofs of laws taken from:
  R. L. Graham, D. E. Knuth, O. Patashnik.  Concrete Mathematics.
  (Addison-Wesley, 1989)

  \bigskip
*}

consts fib :: "nat => nat"
recdef fib  "measure (\<lambda>x. x)"
   zero:    "fib 0  = 0"
   one:     "fib (Suc 0) = Suc 0"
   Suc_Suc: "fib (Suc (Suc x)) = fib x + fib (Suc x)"

text {*
  \medskip The difficulty in these proofs is to ensure that the
  induction hypotheses are applied before the definition of @{term
  fib}.  Towards this end, the @{term fib} equations are not declared
  to the Simplifier and are applied very selectively at first.
*}

text{*We disable @{text fib.Suc_Suc} for simplification ...*}
declare fib.Suc_Suc [simp del]

text{*...then prove a version that has a more restrictive pattern.*}
lemma fib_Suc3: "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib.Suc_Suc)


text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) \<noteq> 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done
-----XEmacs: Fib.thy       (Isar script PenDel Font Abbrev;)----Top----------------
Beginning of buffer
```

Enabling Symbols

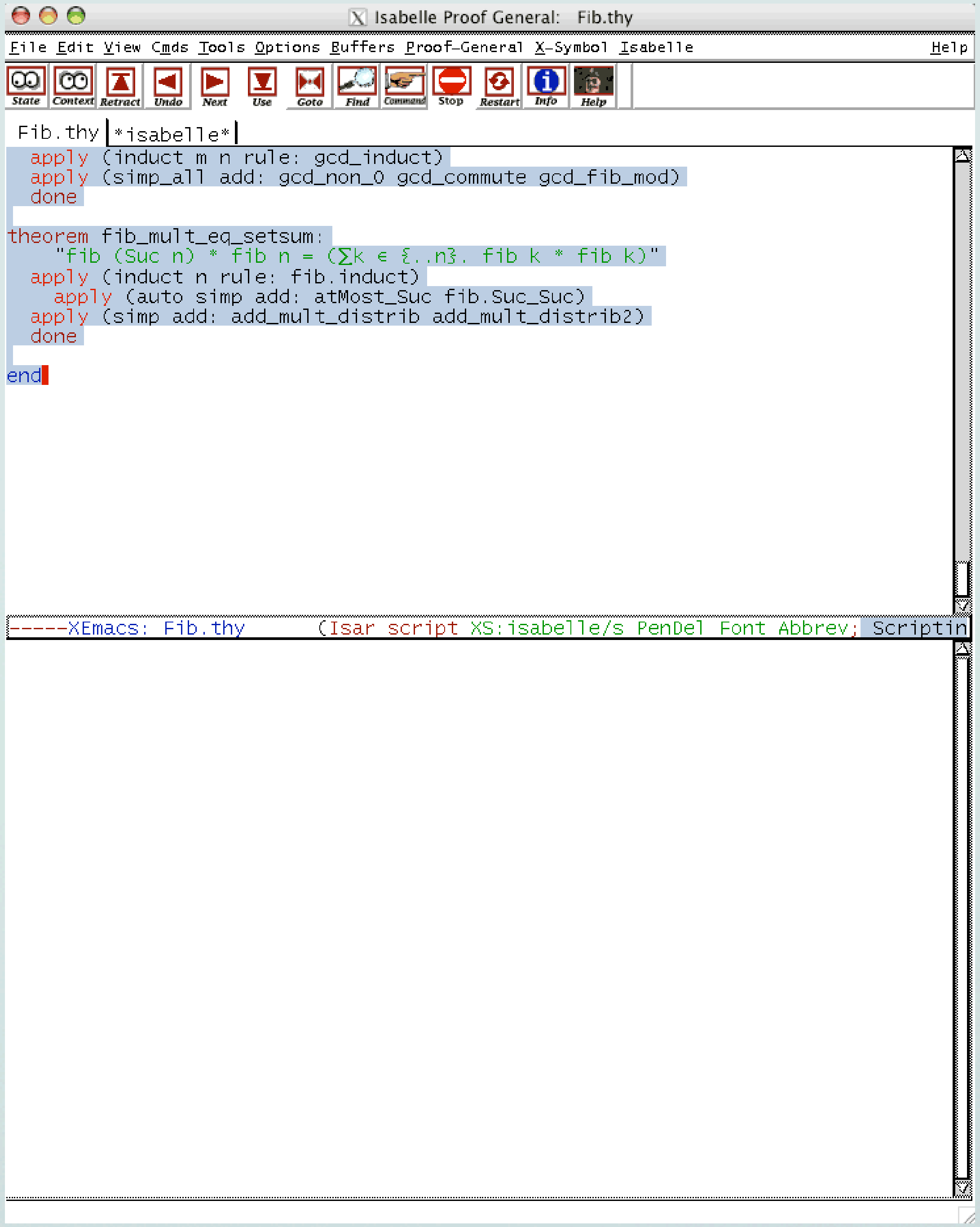
- To enable X-Symbol mode, select the menu item `Proof General > Options > X-Symbol`.
- Then, make this setting permanent using `Proof General > Options > Save Options`.
- Take the time to explore the many other options and settings on offer.



```
Fib.thy
(*  ID:         $Id: Fib.thy,v 1.12 2005/
    Author:     Lawrence C Paulson, Cambr                                     ory
    Copyright   1997  University of Cambr
*)

header {* The Fibonacci function *}

theory Fib = Primes:

text {*
  Fibonacci numbers: proofs of laws taken
  R. L. Graham, D. E. Knuth, O. Patashnik
  (Addison-Wesley, 1989)

  \bigskip
*}

consts fib :: "nat => nat"
recdef fib  "measure (\<lambda>x. x)"
    zero:    "f
    one:     "f
    Suc_Suc: "f

text {*
  \medskip The                                is to ensure that the
  induction hy                                the definition of @{term
  fib}.  Towar                               } equations are not declared
  to the Simp                                selectively at first.
*}

text{*We disab                               simplification ...*}
declare fib.Su

text{*...then                                more restrictive pattern.*}
lemma fib_Suc                                = fib (Suc n) + fib (Suc (Suc n))"
  by (rule fib

text {* \medskip Concrete Mathematics, page 280 *}

lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib n"
  apply (induct n rule: fib.induct)
    prefer 3
    txt {* simplify the LHS just enough to apply the induction hypotheses *}
    apply (simp add: fib_Suc3)
    apply (simp_all add: fib.Suc_Suc add_mult_distrib2)
    done

lemma fib_Suc_neq_0: "fib (Suc n) \<noteq> 0"
  apply (induct n rule: fib.induct)
    apply (simp_all add: fib.Suc_Suc)
  done
-----XEmacs: Fib.thy      (Isar script PenDel Font Abbrev;)----Top---------------
Beginning of buffer
```

Go Forth and Prove!

- Try out this theory yourself: you will find it in `src/HOL/NumberTheory/Fib.thy`.
- For more information on Isabelle, read the documentation.
- For more information on Proof General, see its user manual.
- Have fun!


Isabelle Proof General: Fib.thy
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle
Help
State
Context
Retract
Undo
Next
Use
Goto
Find
Command
Stop
Restart
Info
Help
Fib.thy
isabelle

```
  apply (induct m n rule: gcd_induct)
  apply (simp_all add: gcd_non_0 gcd_commute gcd_fib_mod)
  done

theorem fib_mult_eq_setsum:
    "fib (Suc n) * fib n = (∑k ∈ {..n}. fib k * fib k)"
  apply (induct n rule: fib.induct)
    apply (auto simp add: atMost_Suc fib.Suc_Suc)
  apply (simp add: add_mult_distrib add_mult_distrib2)
  done

end
```

```
-----XEmacs: Fib.thy       (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin
```