



# Isabelle and Proof General: Preview

---

Lawrence C. Paulson  
University of Cambridge



# Getting Started

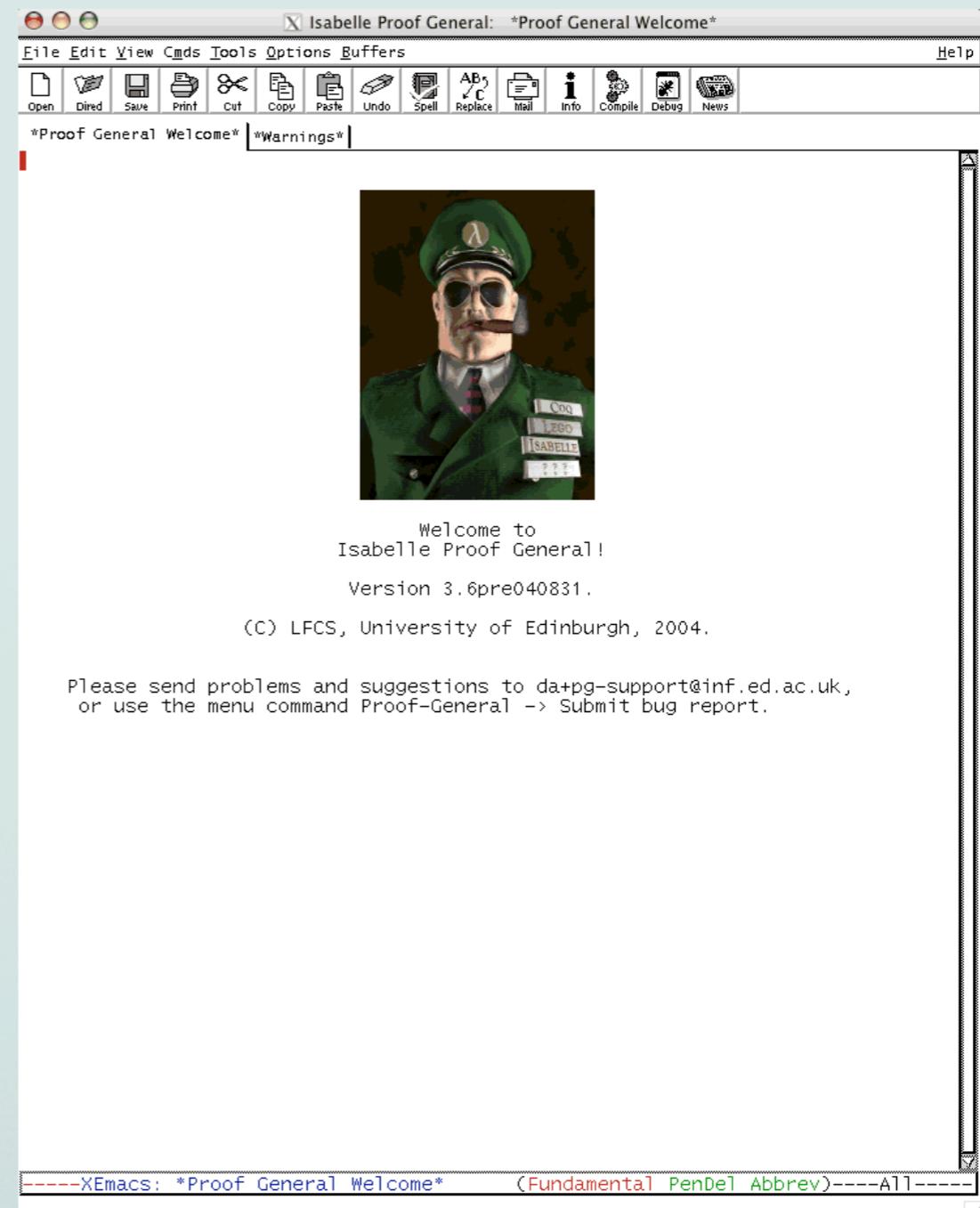
- Install Isabelle, following instructions on the [download page](#).
- Install [Proof General](#).
- Proof General requires the editor [XEmacs](#) to be installed.
- If you have not used XEmacs before, practice on plain text files before attempting proofs!
- Launch Isabelle from the command line.
-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





# Proof General

- 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.

The screenshot shows the Isabelle Proof General interface with the title bar "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, Isabelle, X-Symbol, and Help. The toolbar below has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Comment, Stop, Restart, Info, and Help. The main window displays the content of the Fib.thy theory file. The code defines the Fibonacci function and proves various theorems about it, including properties like additivity and non-negativity. Syntax highlighting is used to distinguish between different parts of the code.

```
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.

The Next button

```
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"
redef 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

The screenshot shows the Isabelle Proof General interface with the title bar "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. The toolbar below has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main window displays the file "Fib.thy" with the following content:

```
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"
redef 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
```

The text "Fibonacci numbers: proofs of laws taken from: R. L. Graham, D. E. Knuth, O. Patashnik. Concrete Mathematics. (Addison-Wesley, 1989)" is highlighted in pink, indicating it is currently being processed by the proof general system.



# 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.

The screenshot shows the Isabelle Proof General interface with the title bar "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. The toolbar below has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main window displays a proof script in the file Fib.thy. The script includes declarations, text comments, and several lemmas. One lemma, fib\_add, is shown with its proof steps involving induction and simplification. The status bar at the bottom indicates the file is XEmacs: Fib.thy, the mode is Isar script, and the buffer is s\_PenDel\_Font\_Abbrev; Scriptin.

```
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.

The screenshot shows the Isabelle Proof General interface with the title bar "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. The toolbar below has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main text area contains the following Isar script:

```
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. Ax. [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!

The screenshot shows the Isabelle Proof General interface with the file Fib.thy open. The proof script includes several text blocks and Isar commands. The text blocks provide context and references to 'Concrete Mathematics, page 280'. The Isar commands include declarations, lemmas, and proof steps involving the fib function and various simplification and induction rules. A specific subgoal is highlighted with a red border, indicating it is the current target for rewriting.

```
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 X$:/isabelle/s_PenDel Font Abbrev; Scriptin
proof (prove): step 4
fixed variables: n, k

goal (lemma (fib_add), 3 subgoals):
1. Ax. [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.

The screenshot shows the Isabelle Proof General interface with the title bar "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. The toolbar below has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main window displays a proof script in the file Fib.thy. The script starts with a note about disabling a specific text command for simplification, followed by a lemma definition for fib\_Suc3. It then includes a text block from Concrete Mathematics, page 280, and a lemma definition for fib\_add. The proof for fib\_add uses induction on n, applies fib.induct, prefers the third hypothesis, and uses fib\_Suc3 and fib\_SucSuc add\_mult\_distrib2. The proof for fib\_Suc\_neq\_0 also uses induction on n and fib.induct, applying simp\_all add: fib.SucSuc. The proof for fib\_Suc\_gr\_0 inserts fib\_Suc\_neq\_0 [of n] and uses simp. The proof concludes with a fixed variables declaration for n, k, and a goal statement for lemma (fib\_add). A status bar at the bottom indicates "----XEmacs: Fib.thy" and "proof (prove): step 5".

```
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.SucSuc add_mult_distrib2)
done

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

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

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.

The screenshot shows the Isabelle Proof General interface. The title bar reads "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. Below the menu is a toolbar with icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main window displays a buffer titled "Fib.thy [\*isabelle\*]". The buffer contains the following Isar script:

```
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)
```

At the bottom of the buffer, there is a status line with "----XEmacs: Fib.thy (Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin".



# 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

```
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
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

The screenshot shows the Isabelle Proof General interface. At the top, there is a menu bar with File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. Below the menu is a toolbar with various icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. A large text area displays a theory file named Fib.thy. The text in the file includes a lemma about gcd and fib, followed by a theorem about fib\_mult\_eq\_setsum, and ends with an end command. At the bottom of the text area, there is a status message: "Interrupt: script management may be in an inconsistent state (but it's probably okay)". A note at the very bottom says "Use C-c C-. to jump to end of processed region". Two blue arrows point downwards from the text "The Revert button" and "The Stop button" to the "Revert" and "Stop" buttons in the toolbar respectively.

```
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)



# 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.

```
Fib.thy [*isabelle*]
\medskip Concrete Mathematics, page 278: Cassini's identity. The proof is
much easier using integers, not natural numbers!
*3

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 zdifff_int [symmetric])
done
-- Emacs: 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)

Mark set
```



# The Proof State

- Clicking on the *State* button reveals the proof state at the given point.
- Here, there was one subgoal left when the proof was interrupted.

The State button

```
Fib.thy [*isabelle*]
\medskip Concrete Mathematics, page 278: Cassini's identity. The proof is
much easier using integers, not natural numbers!
*3

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 zdifff_int [symmetric])
--***-XEmacs: Fib.thy      (Isar script X$ isabelle/s PenDel Font Abbrev; Scriptin
proof (prove): step 4
fixed variables: n
goal (lemma (fib_Cassini_int), 1 subgoal):
  1.  $\lambda x. \llbracket 2 * (\text{int } (\text{fib } (\text{Suc } x)) * \text{int } (\text{fib } (\text{Suc } x))) +$ 
 $\text{int } (\text{fib } x) * \text{int } (\text{fib } (\text{Suc } x)) =$ 
 $(\text{if } (\text{if } \text{Suc } 0 = x \text{ mod } 2 \text{ then } 0 \text{ else } \text{Suc } (x \text{ mod } 2)) = 0$ 
 $\text{then } \text{int } (\text{fib } (\text{Suc } (\text{Suc } x)) * \text{fib } (\text{Suc } (\text{Suc } x))) - 1$ 
 $\text{else } \text{int } (\text{fib } (\text{Suc } (\text{Suc } x)) * \text{fib } (\text{Suc } (\text{Suc } x)) + 1);$ 
 $\text{int } (\text{fib } x) * \text{int } (\text{fib } x) + \text{int } (\text{fib } (\text{Suc } x)) * \text{int } (\text{fib } x) =$ 
 $(\text{if } x \text{ mod } 2 = 0 \text{ then } \text{int } (\text{fib } (\text{Suc } x) * \text{fib } (\text{Suc } x)) - 1$ 
 $\text{else } \text{int } (\text{fib } (\text{Suc } x) * \text{fib } (\text{Suc } x)) + 1)$ 
 $\Rightarrow \text{Suc } 0 \neq x \text{ mod } 2 \rightarrow x \text{ mod } 2 = 0$ 
```

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

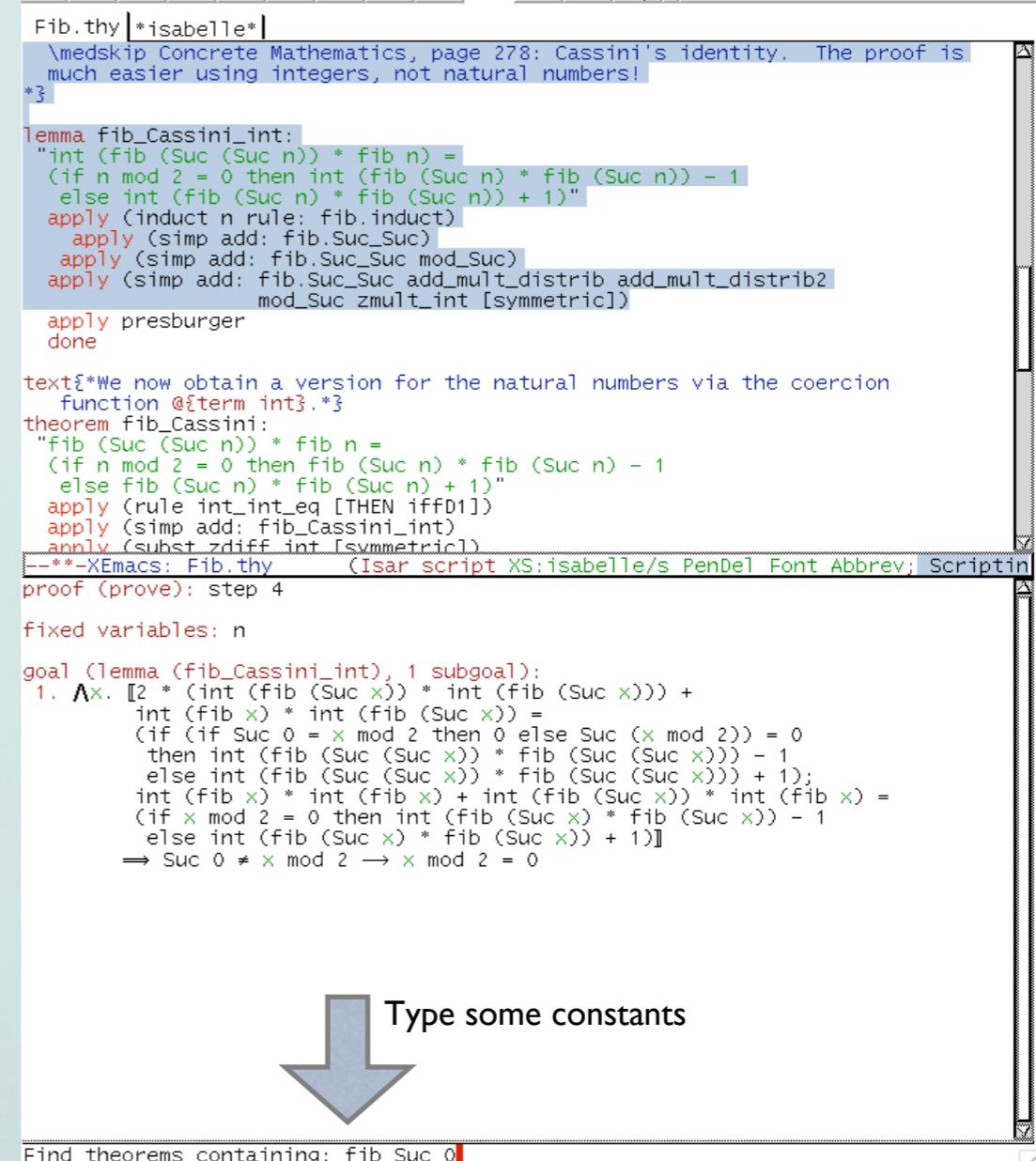


# Finding Theorems

- 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 " $x+y*z = u/v$ ", with the quotation marks. Isabelle finds all theorems involving the infix operators +, \*, / and =.
- A term stands for the set of constants it contains: it is not used as a pattern.



The Find button is highlighted with a blue arrow pointing to it.



Type some constants

Isabelle Proof General: Fib.thy

Fib.thy [\*isabelle\*]

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

\*3

```
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 zdifff_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.  $\lambda x. \exists 2 * (\text{int } (\text{fib } (\text{Suc } x)) * \text{int } (\text{fib } (\text{Suc } x))) +$ 
 $\text{int } (\text{fib } x) * \text{int } (\text{fib } (\text{Suc } x)) =$ 
 $(\text{if } (\text{if } \text{Suc } 0 = x \text{ mod } 2 \text{ then } 0 \text{ else } \text{Suc } (x \text{ mod } 2)) = 0$ 
 $\text{then } \text{int } (\text{fib } (\text{Suc } x)) * \text{fib } (\text{Suc } (\text{Suc } x)) - 1$ 
 $\text{else } \text{int } (\text{fib } (\text{Suc } x)) * \text{fib } (\text{Suc } (\text{Suc } x)) + 1;$ 
 $\text{int } (\text{fib } x) * \text{int } (\text{fib } x) + \text{int } (\text{fib } (\text{Suc } x)) * \text{int } (\text{fib } x) =$ 
 $(\text{if } x \text{ mod } 2 = 0 \text{ then } \text{int } (\text{fib } (\text{Suc } x) * \text{fib } (\text{Suc } x)) - 1$ 
 $\text{else } \text{int } (\text{fib } (\text{Suc } x) * \text{fib } (\text{Suc } x)) + 1)$ 
 $\Rightarrow \text{Suc } 0 \neq x \text{ mod } 2 \rightarrow x \text{ mod } 2 = 0$ 
```

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 constants you type, the fewer theorems will be displayed.
- Variables mentioned in the current goal are viewed as constants for this purpose.

The screenshot shows the Isabelle Proof General interface. The top menu bar includes 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, and Help. The main window displays a buffer titled "Fib.thy [\*isabelle\*]" with the following content:

```
\medskip Concrete Mathematics, page 278: Cassini's identity. The proof is
much easier using integers, not natural numbers!
*3

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 zdifff_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 (?x, ?x))  
    (?fib, nat_case 0 (nat_case (Suc 0) (?v, 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  $x \leq y$ , and matching rules displays all known theorems that can prove a conclusion of that form.

The screenshot shows the Isabelle Emacs interface. The menu bar at the top includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle (which is currently selected), and Help. A toolbar below the menu bar contains icons for State, Context, Retract, Undo, Next, Use, Cases, Facts, Matching Rules, Matching Rewrites, Term Bindings, Classical Rules, Induct/Cases Rules, Simplifier Rules, Theorems, Transitivity Rules, Antiquotations, Attributes, Commands, Inner Syntax, and Methods. The main window displays a file named Fib.thy with Isabelle code. A context menu is open over a specific theorem, showing options like "matching rules" and "matching rewrites". The code in Fib.thy includes definitions for fib\_Cassini and gcd\_fib\_Suc\_eq\_1, and imports from various Isabelle theories like Set, Order, and Divides.

```
File Edit View Cmds Tools Options Buffers Proof-General X-Symbol Isabelle Help
emacs: Fib.thy
Isabelle
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

text {* We now obtain a very efficient function @{term int3}.*}
theorem fib_Cassini:
  "fib (Suc (Suc n)) * fib (Suc (Suc n)) = fib (Suc (Suc (Suc n))) * fib (Suc n)"
  (if n mod 2 = 0 then fib (Suc (Suc n)) * fib (Suc (Suc (Suc n)))
  else fib (Suc n)) * fib (Suc (Suc (Suc (Suc n))))"
apply (rule int_int_eq)
apply (simp add: fib_Cassini)
apply (subst zdifff_int)
apply (insert fib_Suc)
done

text {* \medskip Toward Law 6.111 of Concrete Mathematics *3}
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
Set.ord_le_eq_subst: [|?a = ?b; ?b ≤ ?c; ∀x. x ≤ y ==> ?f x ≤ ?f y|] ==> ?a ≤ ?f ?c
Set.order_subst2: [|?a ≤ ?b; ?f ?b = ?c; ∀x. x ≤ y ==> ?f x ≤ ?f y|] ==> ?f ?a ≤ ?c
Set.order_subst1: [|?a ≤ ?f ?b; ?b ≤ ?c; ∀x. 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.

The screenshot shows the Isabelle Proof General interface with the file `Fib.thy` open. A context menu is open over the text, with the `Isabelle` option selected, revealing the `Settings` submenu. The submenu contains various options for controlling the display and behavior of the proof assistant, such as `Show Types`, `Show Sorts`, `Eta Contract`, and `Trace Simplifier`. The `Show Types` option is checked. The main text area displays the code for the `fib_Cassini_int` lemma and the `fib_Cassini` theorem, along with some trace information and facts about the `fib` function.

```
Fib.thy [*isabelle*]
\medskip Concrete Mathematics, page 278: Cassini's
much easier using integers, not natural numbers!
*3

lemma fib_Cassini_int:
  "int (fib (Suc (Suc n)) * fib n) =
   (if n mod 2 = 0 then int (fib (Suc n)) * fib (Suc n)
    else int (fib (Suc n)) * fib n)"
  apply (induct n rule: fib.induct)
  apply (simp add: fib.Suc_Suc)
  apply (simp add: fib.Suc_Suc)
  apply (simp add: fib.Suc_Suc)
  apply presburger
done

text{*We now obtain a version for
function @{term int}.*}
theorem fib_Cassini:
  "fib (Suc (Suc n)) * fib n =
   (if n mod 2 = 0 then fib (Suc n) * 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 zdifff_int [symmetric])
  --***-XEmacs: Fib.thy (Isar s)
Facts containing constants "0" "Suc"
Reset Settings
Save Settings

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 (?x, ?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.

The screenshot shows the Isabelle Proof General interface for a file named Fib.thy. The Proof-General menu is open, displaying various commands and their keyboard shortcuts. The main window shows some Isar script code related to the Fibonacci sequence. The code includes definitions for fib\_gr\_0 and fib\_Cassini\_int, and a proof for fib\_Cassini\_int involving presburger logic. The Proof-General menu includes options like Display Proof State, Display Context, Retract Buffer, Undo Step, Delete Step, Next Step, Use Buffer, Goto Point, Goto Locked End, Find Theorems, Issue Command, Interrupt Prover, Restart Scripting, Toggle Visibility, Next Error, Scripting Active, and Help. The toolbar at the top has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, and Com.

```
Fib.thy [*isabelle*]
Lemma fib_gr_0: "0 < n ==> 0 < fib n"
by (case_tac n, auto simp add:)

text {*
\medskip Concrete Mathematics, p. 152. It is much easier using integers, not rationals.
*}

Lemma fib_Cassini_int:
"int (fib (Suc (Suc n)) * fib n) = int (fib (Suc (Suc n)) * fib (Suc n)) + int (fib (Suc (Suc n)) * fib (Suc (Suc n)))"
apply (induct n rule: fib.induct)
apply (simp add: fib.Suc_Suc_mod_Suc)
apply (simp add: fib.Suc_Suc_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 = int (fib (Suc (Suc n)) * fib n) + int (fib (Suc (Suc n)) * fib (Suc (Suc n)))"
proof (prove): step 4
fixed variables: n
goal (lemma (fib_Cassini_int), 1 subgoal)
1.  $\forall x. \exists 2 * (\text{int } (\text{fib } (\text{Suc } x)) * \text{int } (\text{fib } (\text{Suc } x))) + \text{int } (\text{fib } x) * \text{int } (\text{fib } (\text{Suc } x)) = (\text{if } (\text{if } \text{Suc } 0 = x \text{ mod } 2 \text{ then } 0 \text{ else } \text{Suc } (x \text{ mod } 2)) = 0 \text{ then } \text{int } (\text{fib } (\text{Suc } (\text{Suc } x)) * \text{fib } (\text{Suc } (\text{Suc } x))) - 1 \text{ else } \text{int } (\text{fib } (\text{Suc } (\text{Suc } x)) * \text{fib } (\text{Suc } (\text{Suc } x))) + 1;$ 
 $\text{int } (\text{fib } x) * \text{int } (\text{fib } x) + \text{int } (\text{fib } (\text{Suc } x)) * \text{int } (\text{fib } x) = (\text{if } x \text{ mod } 2 = 0 \text{ then } \text{int } (\text{fib } (\text{Suc } x) * \text{fib } (\text{Suc } x)) - 1 \text{ else } \text{int } (\text{fib } (\text{Suc } x) * \text{fib } (\text{Suc } x)) + 1)$ 
 $\Rightarrow \text{Suc } 0 \neq x \text{ mod } 2 \rightarrow x \text{ mod } 2 = 0$ 
```



# Mathematical Symbols

- Proof General uses the X-Symbol package to display mathematical symbols such as  $\lambda \leq \neq \in \notin \cup \cap$ .
- 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.

The screenshot shows an Emacs window titled "emacs: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, Isabelle, and Help. The toolbar below the menu has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main buffer contains Isabelle code for the Fibonacci function:

```
Fib.thy
(* 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)*}

\bigrskip
consts fib :: "nat => nat"
redef 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.

The screenshot shows the Isabelle/Isar proof editor running in Emacs. The buffer contains a theory definition for the Fibonacci function. A context menu is open over the code, specifically over the 'X-Symbol' option under 'Tools'. The menu also includes 'Electric Terminator', 'Fly Past Comments', 'Disappearing Proofs', 'Strict Read Only', 'Multiple Modes', 'Toolbar', 'Index Menu', 'Display', 'Follow Mode', 'Deactivate Action', 'Reset Options', and 'Save Options'. The status bar at the bottom indicates the file is 'Fib.thy' and the mode is 'Isar script Pendel Font Abbrev;'. The title bar shows 'emacs: Fib.thy' and the menu bar includes 'File Edit View Cmds Tools Options Buffers Proof-General Isabelle Help'.

```
Fib.thy
(* ID: $Id: Fib.thy,v 1.12 2005/02/10 13:45:00 paulson Exp $
Author: Lawrence C Paulson, Cambridge University
Copyright 1997 University of Cambridge *)
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"
redef fib "measure (\<lambda>x. x)"
zero: "fib 0"
one: "fib 1"
Suc_Suc: "fib (Suc (Suc n))"

text {* \medskip The induction hypothesis for fib. Toward
to the Simplification ... *}

text {* We disable the declaration of fib. Suc_Suc
by (rule fib_Suc_Suc) for now. This is a more
restrictive pattern. *}
= fib (Suc n) + fib (Suc (Suc n))

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
```



# 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!

The screenshot shows the Isabelle Proof General interface. The title bar reads "Isabelle Proof General: Fib.thy". The menu bar includes File, Edit, View, Cmds, Tools, Options, Buffers, Proof-General, X-Symbol, Isabelle, and Help. The toolbar below has icons for State, Context, Retract, Undo, Next, Use, Goto, Find, Command, Stop, Restart, Info, and Help. The main window displays a proof script in Isar. The script starts with a theorem definition:

```
Fib.thy [*isabelle*]
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
```

At the bottom of the window, there is a status bar with the text "XEmacs: Fib.thy" and "Isar script XS:isabelle/s PenDel Font Abbrev; Scriptin".