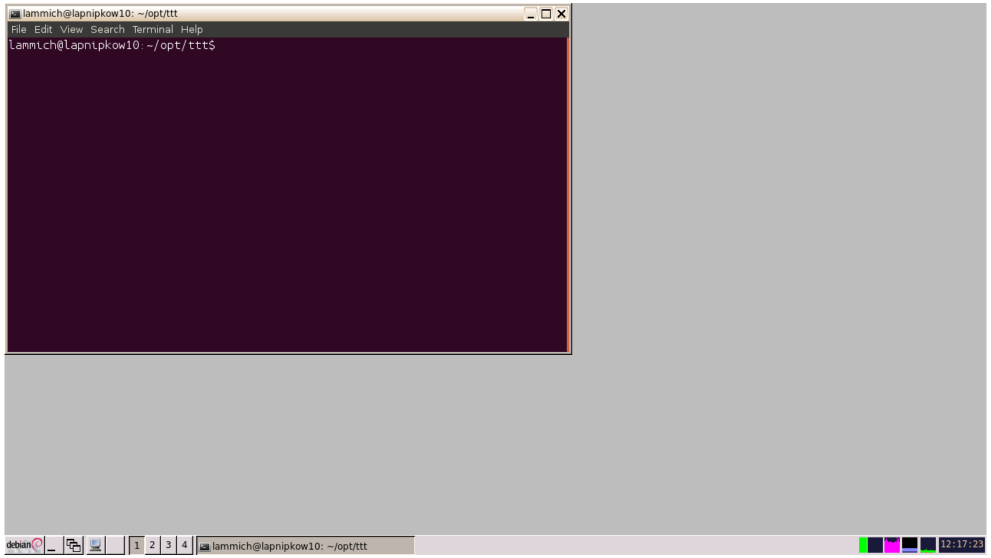
Script **generated by TTT**

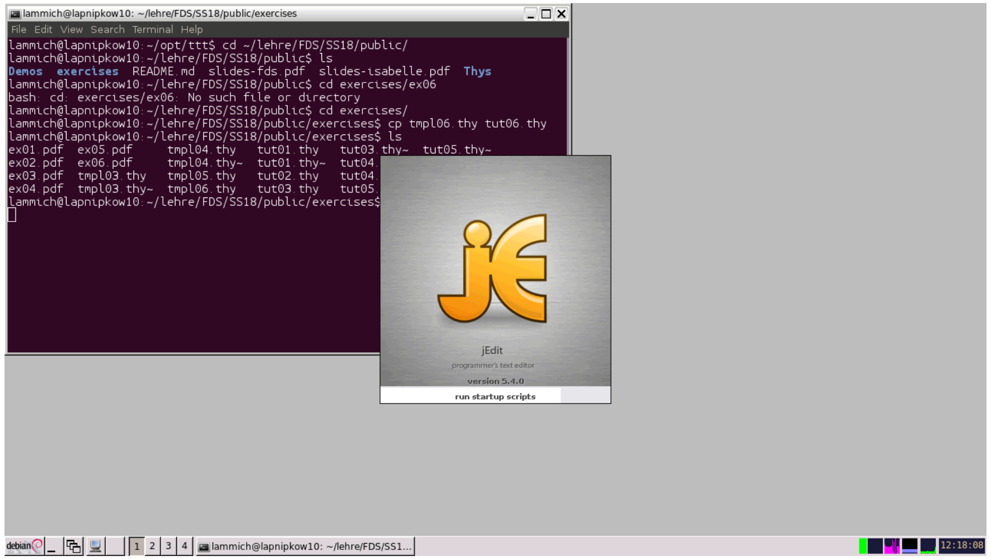
Title: Lammich: FDS Tutorial (18.05.2018)

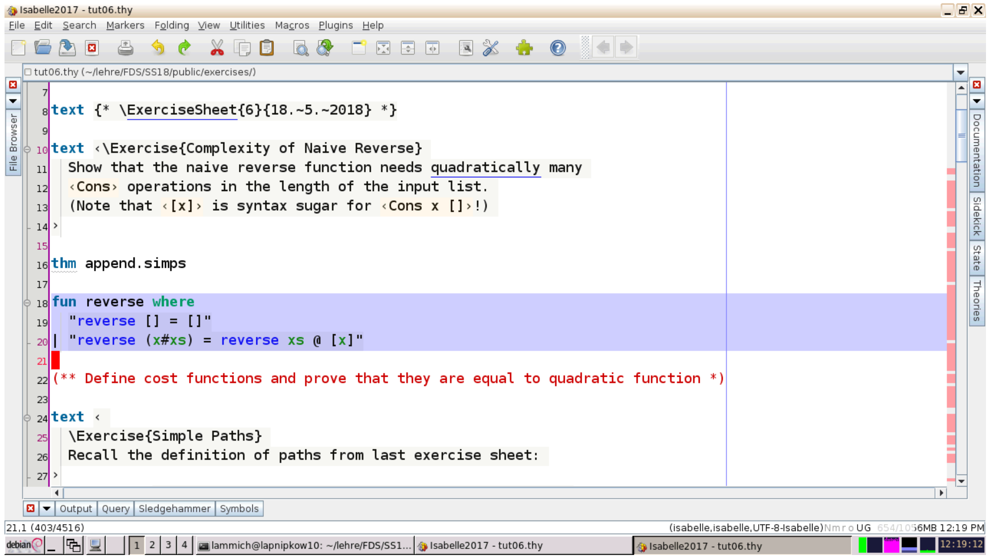
Date: Fri May 18 12:17:23 CEST 2018

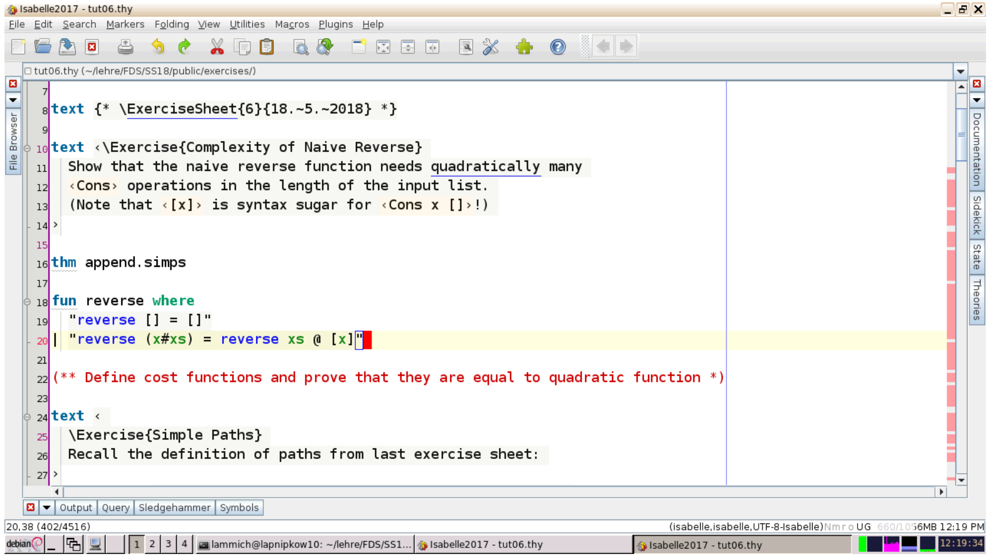
Duration: 76:40 min

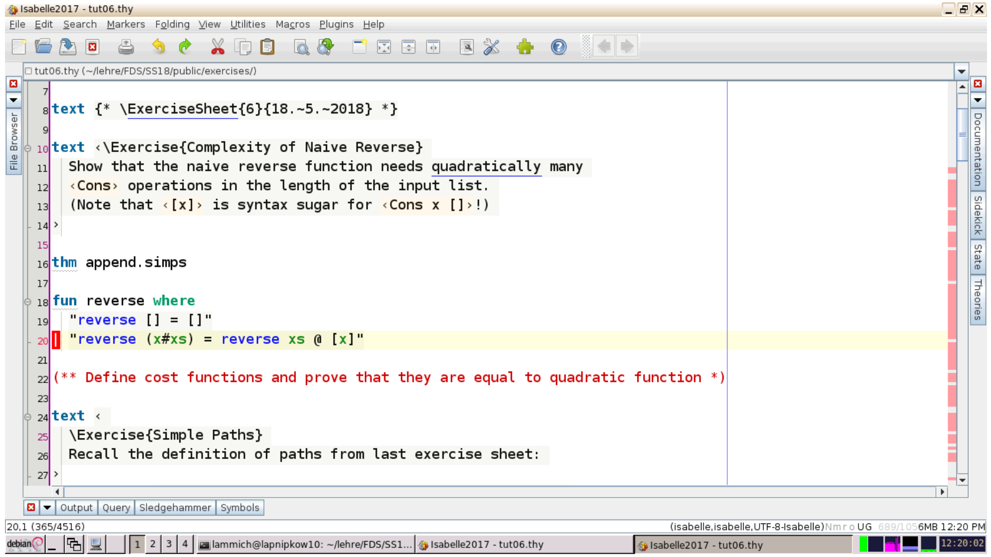
Pages: 87

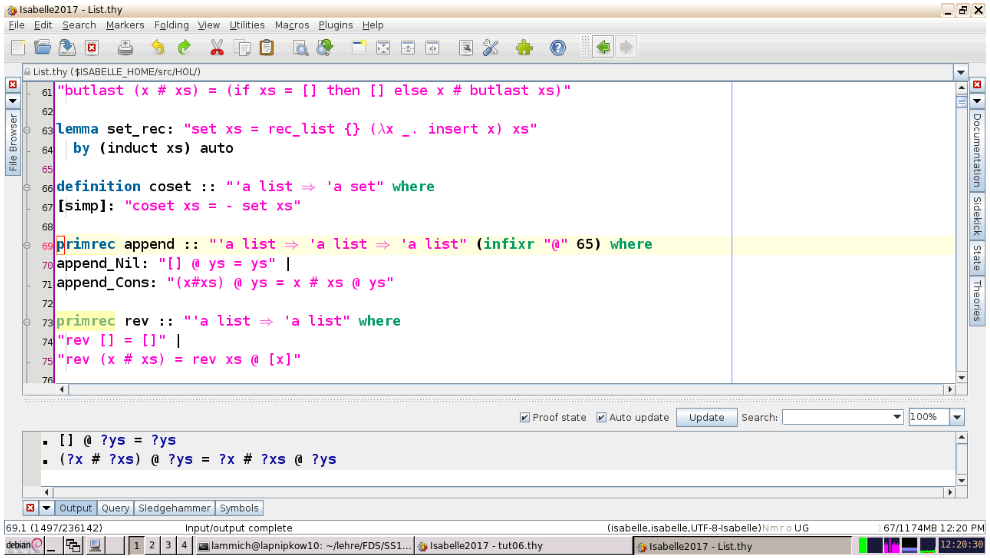


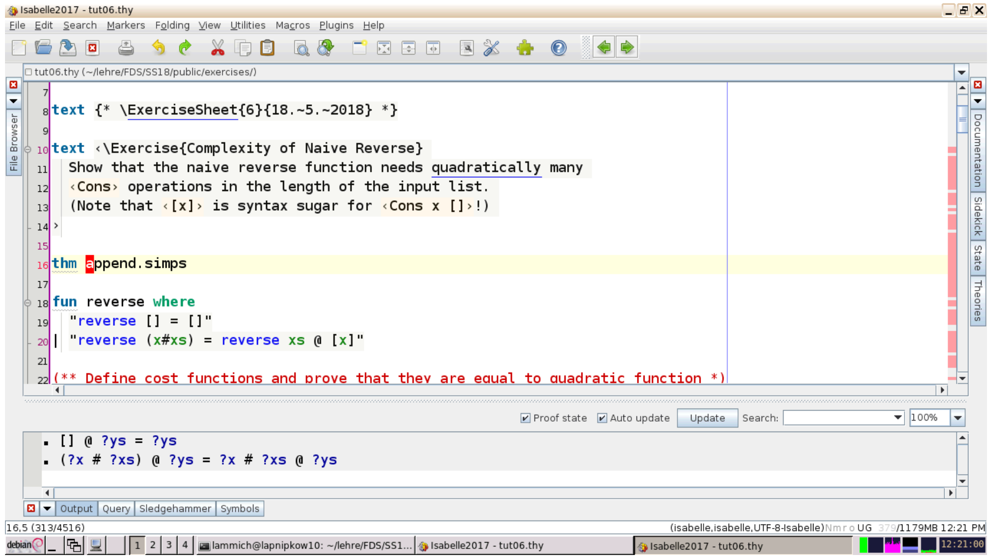


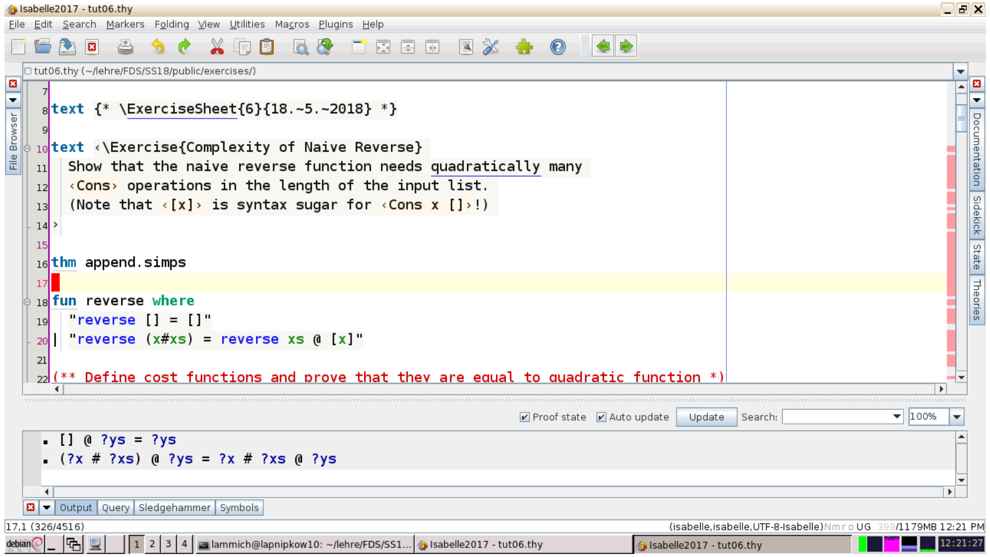


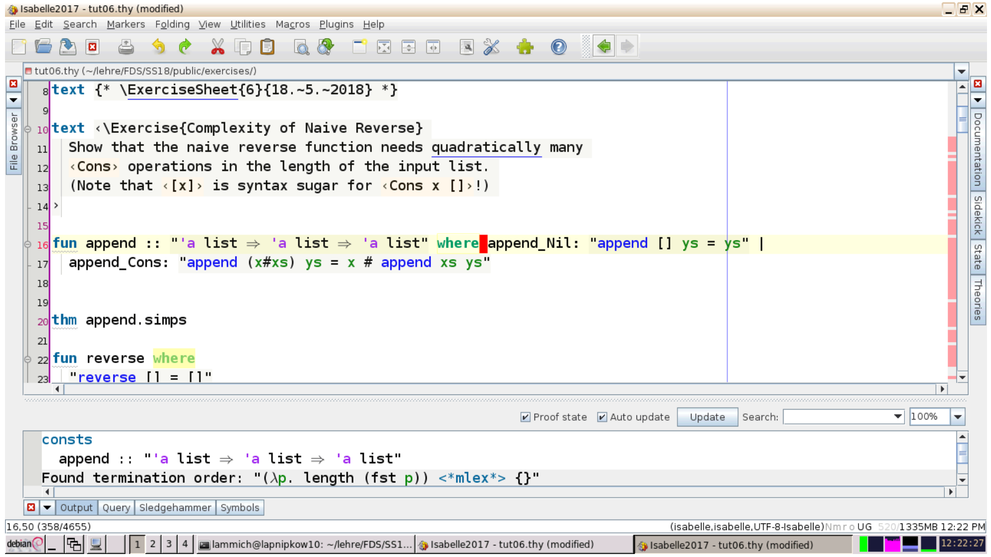

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
text {* \ExerciseSheet{6}{18.~5.~2018} *}
text ‹\Exercise{Complexity of Naive Reverse}
Show that the naive reverse function needs quadratically many
‹Cons› operations in the length of the input list.
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
thm append.simps
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
(** Define cost functions and prove that they are equal to quadratic function *)
text ‹
\Exercise{Simple Paths}
Recall the definition of paths from last exercise sheet:
›

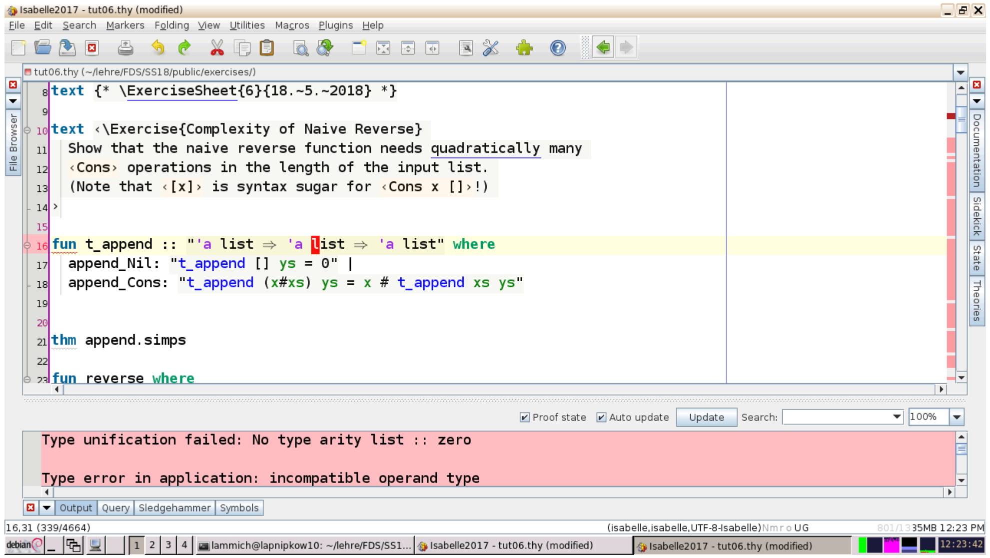

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
text {* \ExerciseSheet{6}{18.~5.~2018} *}
text ‹\Exercise{Complexity of Naive Reverse}
Show that the naive reverse function needs quadratically many
‹Cons› operations in the length of the input list.
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
thm append.simps
fun reverse where
"reverse [] = []"
"reverse (x#xs) = reverse xs @ [x]"
(** Define cost functions and prove that they are equal to quadratic function *)
text ‹
\Exercise{Simple Paths}
Recall the definition of paths from last exercise sheet:
›

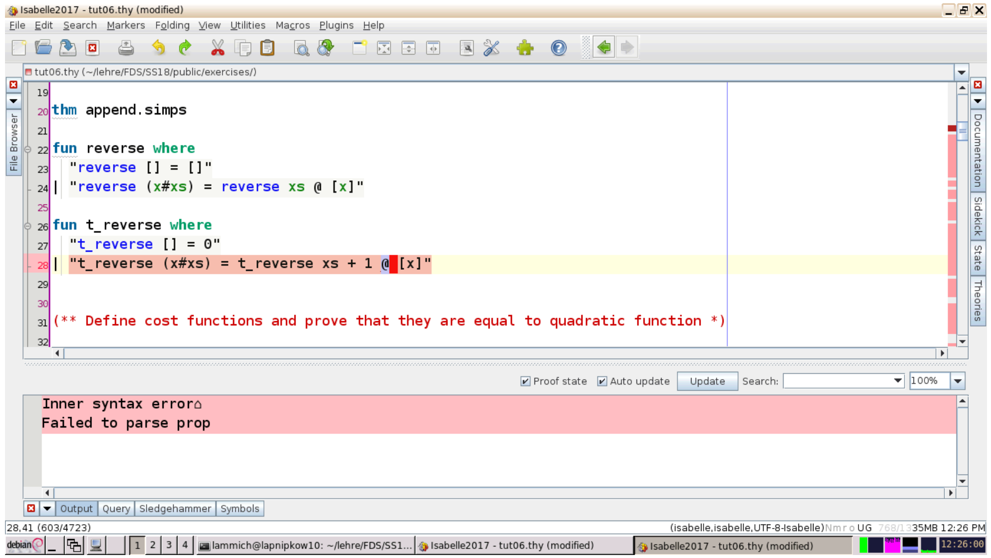

Isabelle2017 - List.thy
List.thy ($ISABELLE_HOME/src/HOL/)
"butlast (x # xs) = (if xs = [] then [] else x # butlast xs)"
lemma set_rec: "set xs = rec_list {} (λx _. insert x) xs"
by (induct xs) auto
definition coset :: "'a list ⇒ 'a set" where
[simp]: "coset xs = - set xs"
primrec append :: "'a list ⇒ 'a list ⇒ 'a list" (infixr "@" 65) where
append_Nil: "[] @ ys = ys" |
append_Cons: "(x#xs) @ ys = x # xs @ ys"
primrec rev :: "'a list ⇒ 'a list" where
"rev [] = []" |
"rev (x # xs) = rev xs @ [x]"
[] @ ?ys = ?ys
(?x # ?xs) @ ?ys = ?x # ?xs @ ?ys

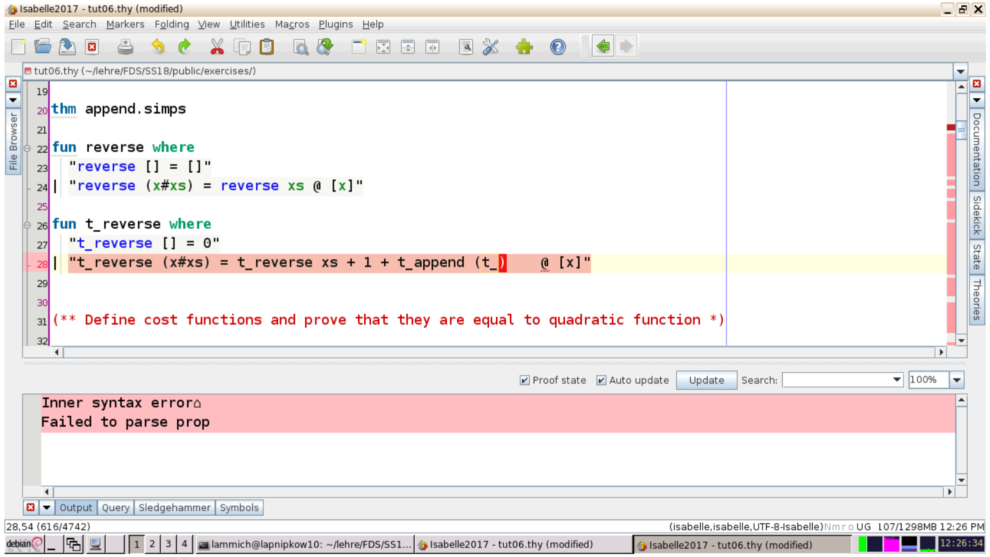

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
text {* \ExerciseSheet{6}{18.~5.~2018} *}
text ‹\Exercise{Complexity of Naive Reverse}
Show that the naive reverse function needs quadratically many
‹Cons› operations in the length of the input list.
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
thm append.simps
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
(** Define cost functions and prove that they are equal to quadratic function *)
[] @ ?ys = ?ys
(?x # ?xs) @ ?ys = ?x # ?xs @ ?ys

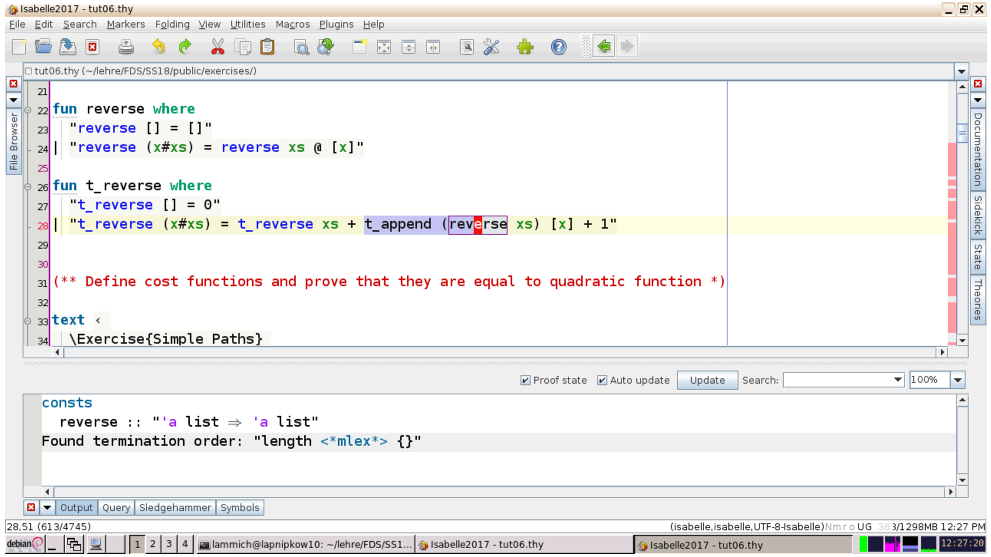

Isabelle2017 - tut06.thy
text {* \ExerciseSheet{6}{18.~5.~2018} *}
text ‹\Exercise{Complexity of Naive Reverse}
Show that the naive reverse function needs quadratically many
‹Cons› operations in the length of the input list.
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
thm append.simps
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
(** Define cost functions and prove that they are equal to quadratic function *)
• [] @ ?ys = ?ys
• (?x # ?xs) @ ?ys = ?x # ?xs @ ?ys

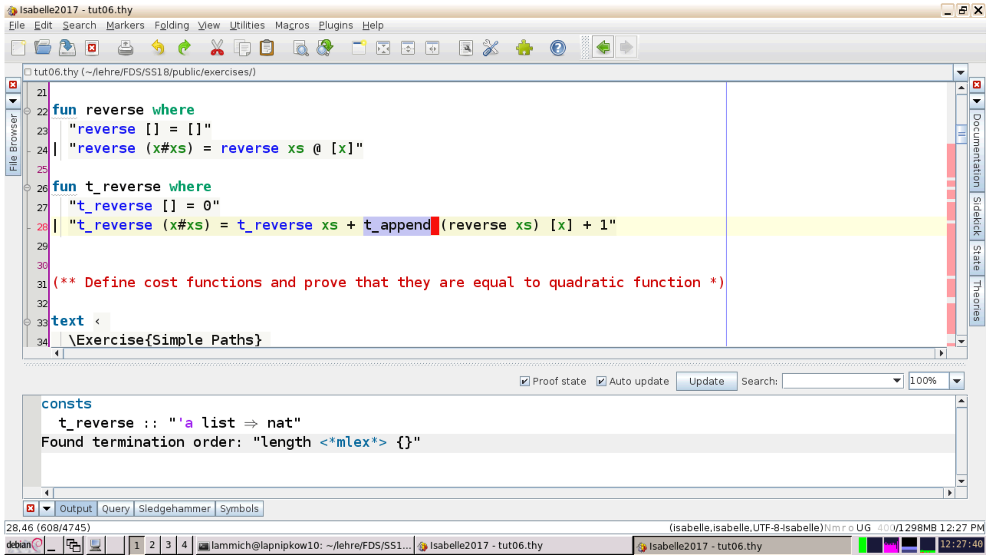

Isabelle2017 - tut06.thy (modified)
text {* \ExerciseSheet{6}{18.~5.~2018} *}
text ‹\Exercise{Complexity of Naive Reverse}
Show that the naive reverse function needs quadratically many
‹Cons› operations in the length of the input list.
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
fun append :: "'a list ⇒ 'a list ⇒ 'a list" where append_Nil: "append [] ys = ys" |
append_Cons: "append (x#xs) ys = x # append xs ys"
thm append.simps
fun reverse where
"reverse [] = []"
consts
append :: "'a list ⇒ 'a list ⇒ 'a list"
Found termination order: "(λp. length (fst p)) <*mlex*> {}"

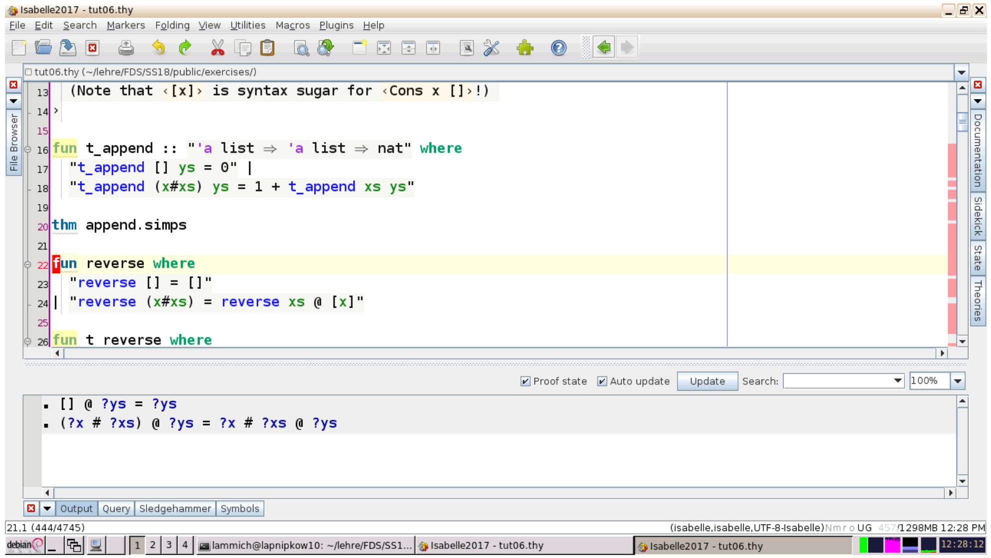

Isabelle2017 - tut06.thy (modified)
text {* \ExerciseSheet{6}{18.~5.~2018} *}
text ‹\Exercise{Complexity of Naive Reverse}
Show that the naive reverse function needs quadratically many
‹Cons› operations in the length of the input list.
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
fun t_append :: "'a list ⇒ 'a list ⇒ 'a list" where
append_Nil: "t_append [] ys = 0" |
append_Cons: "t_append (x#xs) ys = x # t_append xs ys"
thm append.simps
fun reverse where
Type unification failed: No type arity list :: zero
Type error in application: incompatible operand type

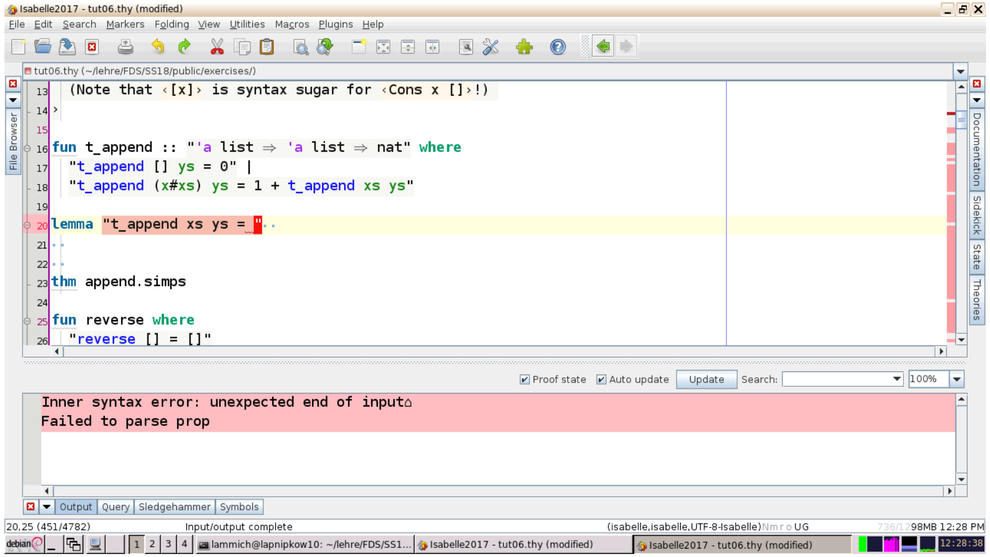

Isabelle2017 - tut06.thy (modified)
thm append.simps
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + 1 @ [x]"
(** Define cost functions and prove that they are equal to quadratic function *)
Inner syntax error
Failed to parse prop

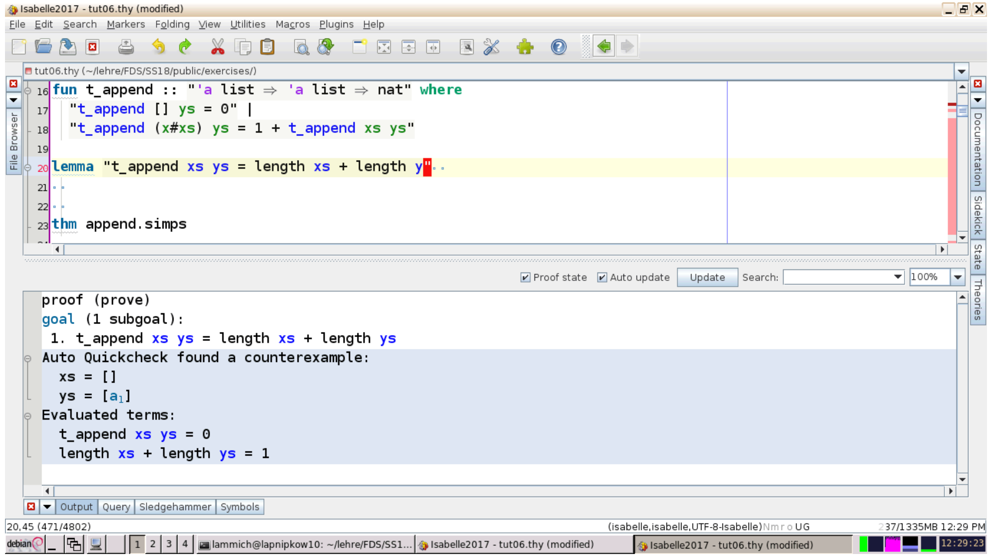

Isabelle2017 - tut06.thy (modified)
thm append.simps
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + 1 + t_append (t_)   @ [x]"
(** Define cost functions and prove that they are equal to quadratic function *)
Inner syntax error
Failed to parse prop

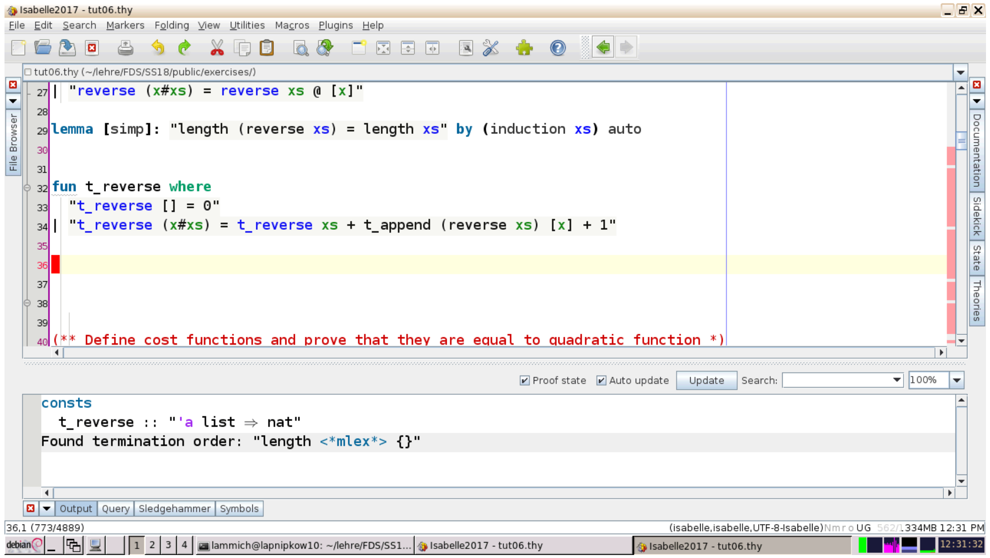

Isabelle2017 - tut06.thy
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(** Define cost functions and prove that they are equal to quadratic function *)
text ‹
\Exercise{Simple Paths}
consts
reverse :: "'a list ⇒ 'a list"
Found termination order: "length <*mlex*> {}"

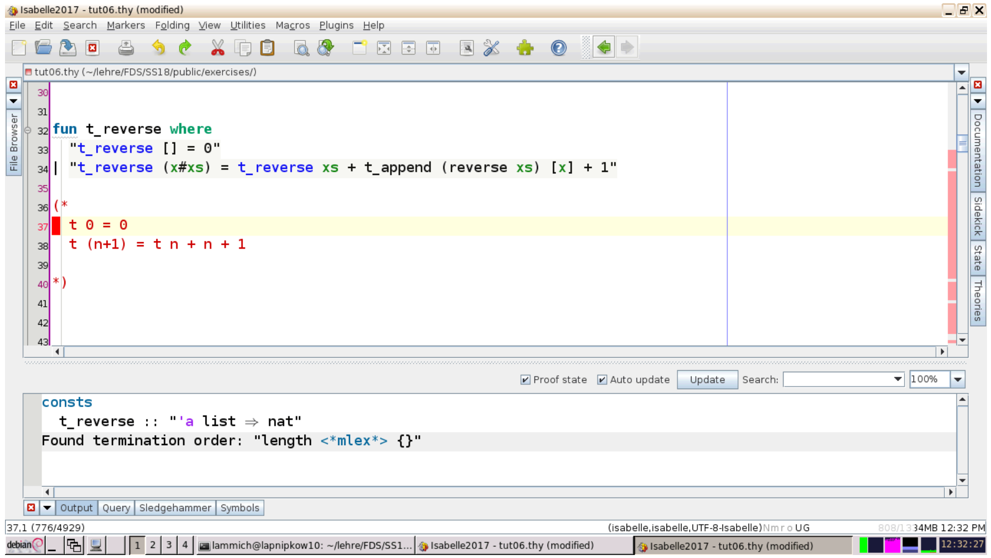

Isabelle2017 - tut06.thy
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(** Define cost functions and prove that they are equal to quadratic function *)
text ‹
\Exercise{Simple Paths}
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

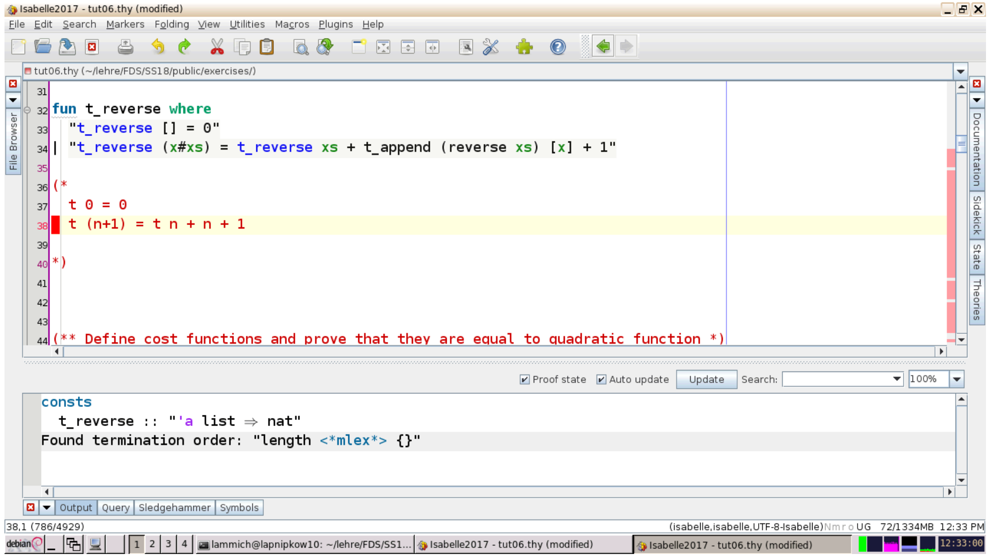

Isabelle2017 - tut06.thy
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
fun t_append :: "'a list ⇒ 'a list ⇒ nat" where
"t_append [] ys = 0" |
"t_append (x#xs) ys = 1 + t_append xs ys"
thm append.simps
fun reverse where
"reverse [] = []"
| "reverse (x#xs) = reverse xs @ [x]"
fun t_reverse where
[] @ ?ys = ?ys
(?x # ?xs) @ ?ys = ?x # ?xs @ ?ys

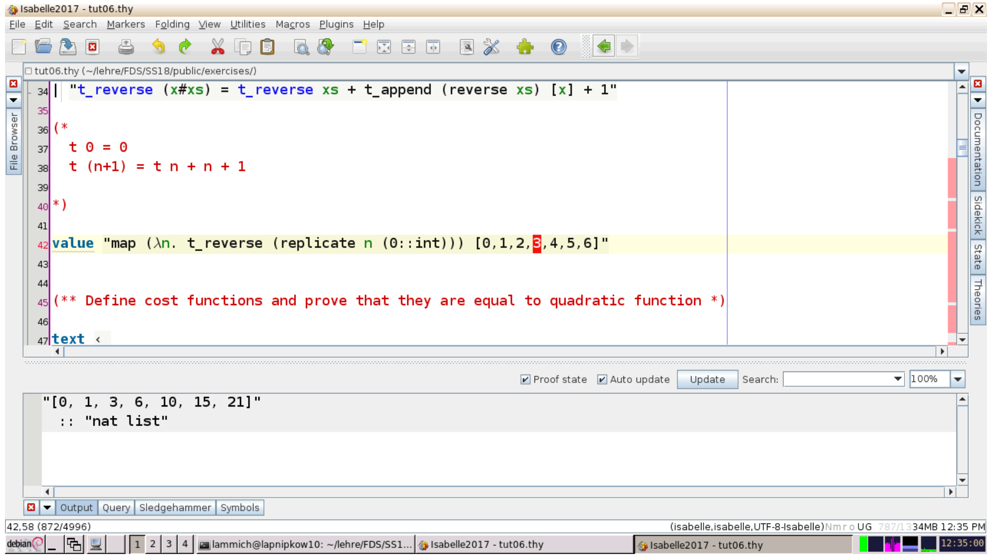

Isabelle2017 - tut06.thy (modified)
(Note that ‹[x]› is syntax sugar for ‹Cons x []›!)
›
fun t_append :: "'a list ⇒ 'a list ⇒ nat" where
"t_append [] ys = 0" |
"t_append (x#xs) ys = 1 + t_append xs ys"
lemma "t_append xs ys = "
thm append.simps
fun reverse where
"reverse [] = []"
Inner syntax error: unexpected end of input
Failed to parse prop

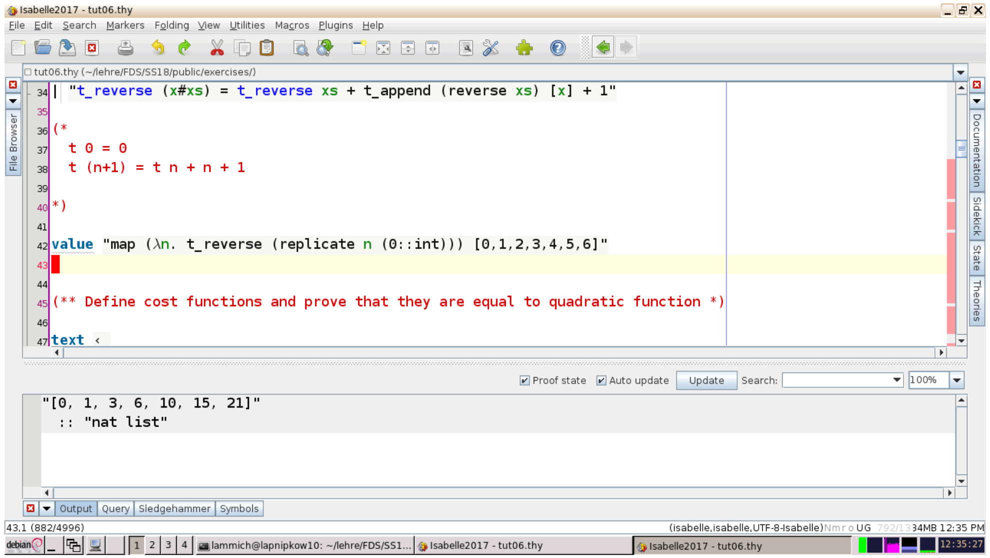

Isabelle2017 - tut06.thy (modified)
fun t_append :: "'a list ⇒ 'a list ⇒ nat" where
"t_append [] ys = 0" |
"t_append (x#xs) ys = 1 + t_append xs ys"
lemma "t_append xs ys = length xs + length ys"
thm append.simps
proof (prove)
goal (1 subgoal):
1. t_append xs ys = length xs + length ys
Auto Quickcheck found a counterexample:
xs = []
ys = [a1]
Evaluated terms:
t_append xs ys = 0
length xs + length ys = 1

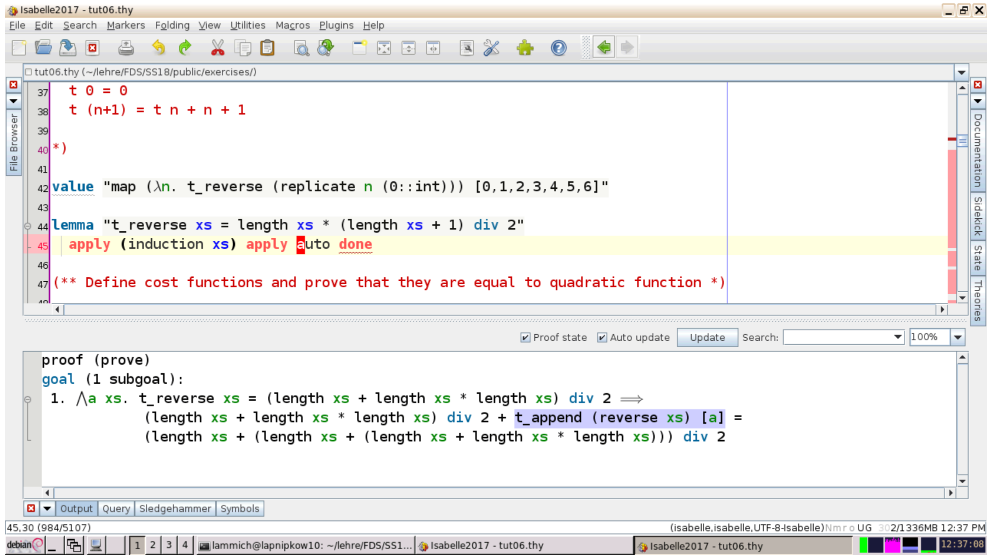

Isabelle2017 - tut06.thy
| "reverse (x#xs) = reverse xs @ [x]"
lemma [simp]: "length (reverse xs) = length xs" by (induction xs) auto
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(** Define cost functions and prove that they are equal to quadratic function *)
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

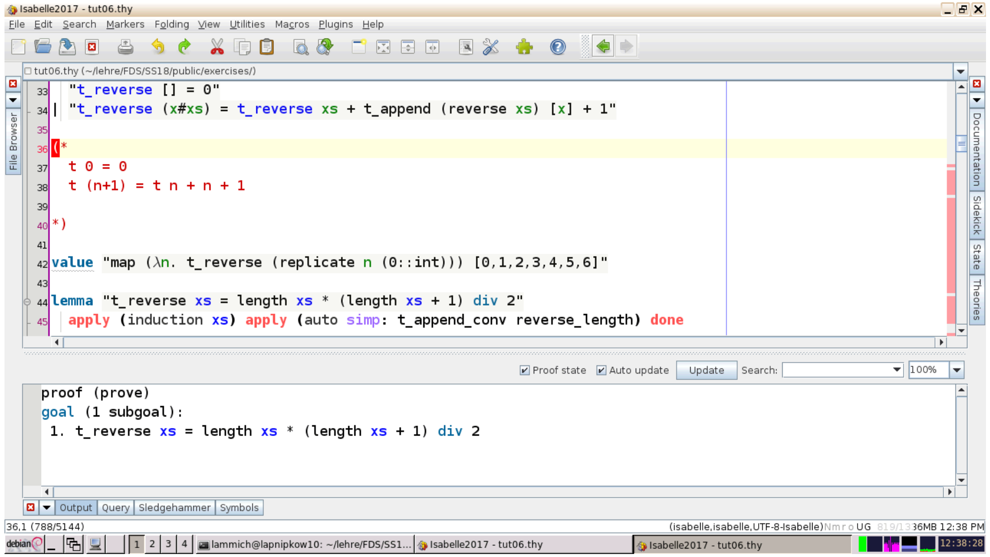

Isabelle2017 - tut06.thy (modified)
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
*)
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

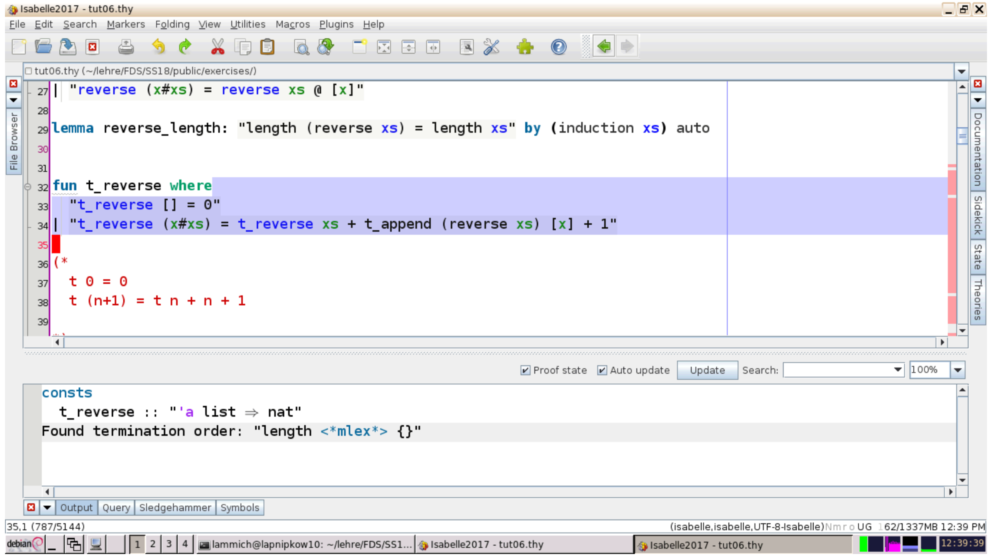

Isabelle2017 - tut06.thy (modified)
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
*)
(** Define cost functions and prove that they are equal to quadratic function *)
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

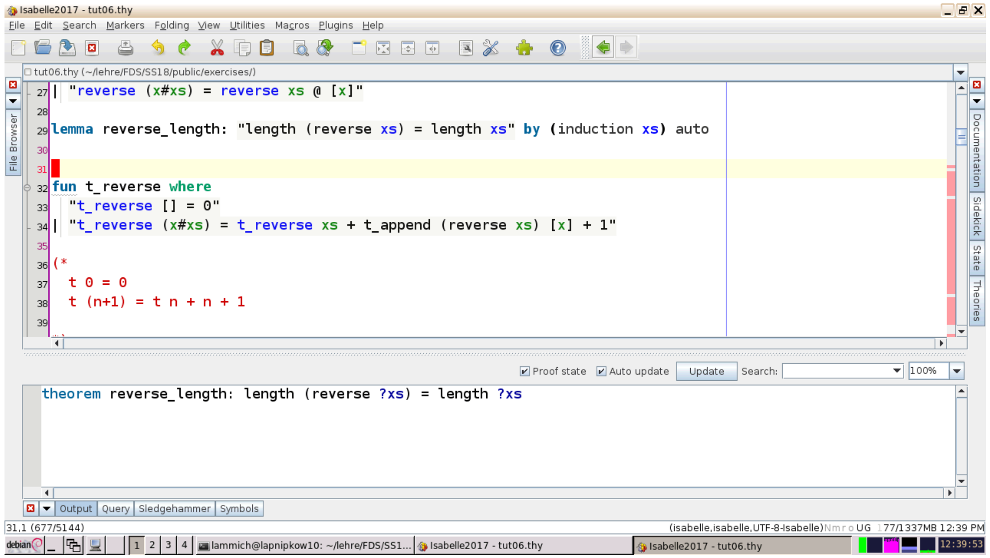

Isabelle2017 - tut06.thy
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
*)
value "map (λn. t_reverse (replicate n (0::int))) [0,1,2,3,4,5,6]"
(** Define cost functions and prove that they are equal to quadratic function *)
text ‹
"[0, 1, 3, 6, 10, 15, 21]"
:: "nat list"

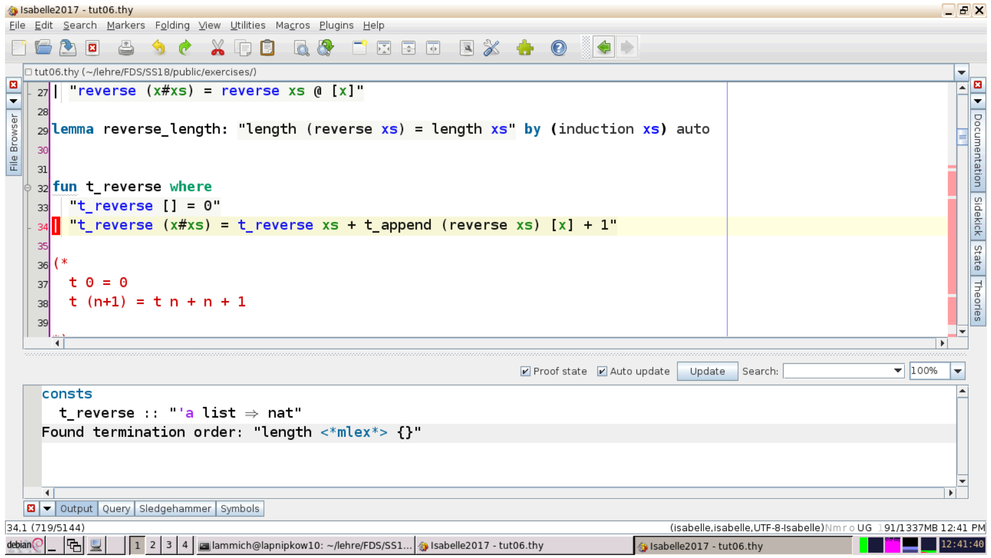

Isabelle2017 - tut06.thy
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
*)
value "map (λn. t_reverse (replicate n (0::int))) [0,1,2,3,4,5,6]"
(** Define cost functions and prove that they are equal to quadratic function *)
text ‹
"[0, 1, 3, 6, 10, 15, 21]"
:: "nat list"

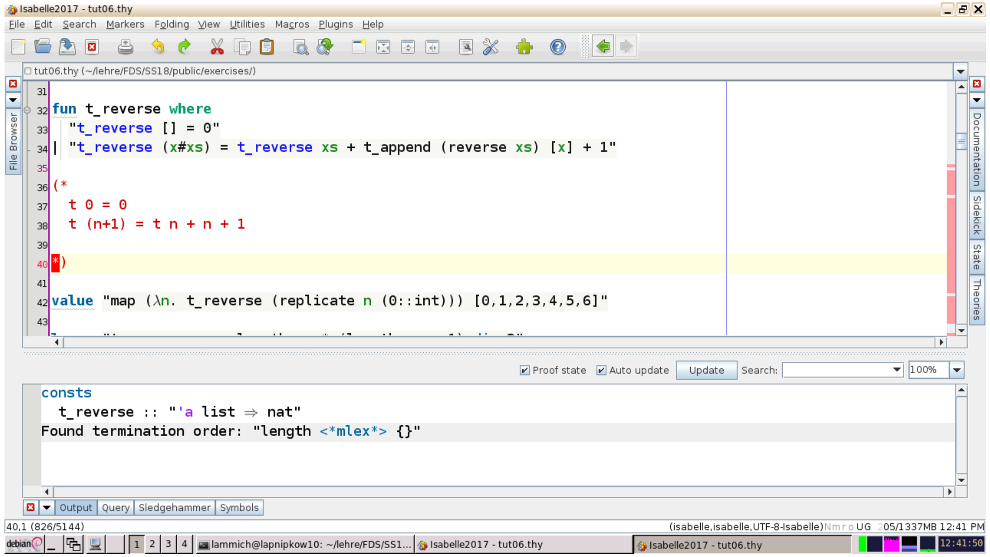

Isabelle2017 - tut06.thy
t 0 = 0
t (n+1) = t n + n + 1
*)
value "map (λn. t_reverse (replicate n (0::int))) [0,1,2,3,4,5,6]"
lemma "t_reverse xs = length xs * (length xs + 1) div 2"
apply (induction xs) apply auto done
(** Define cost functions and prove that they are equal to quadratic function *)
proof (prove)
goal (1 subgoal):
1. ⋀a xs. t_reverse xs = (length xs + length xs * length xs) div 2 ⟹
(length xs + length xs * length xs) div 2 + t_append (reverse xs) [a] =
(length xs + (length xs + (length xs + length xs * length xs))) div 2

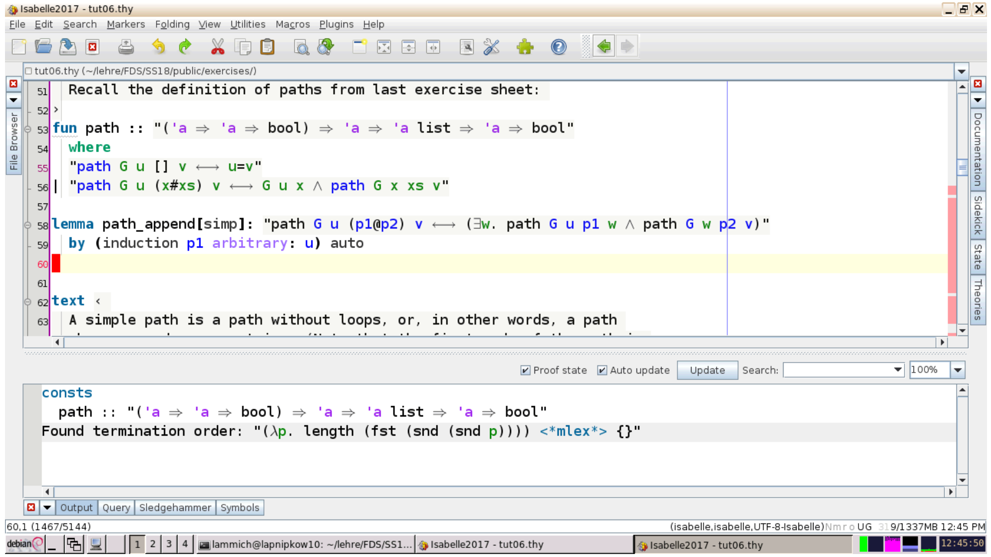

Isabelle2017 - tut06.thy
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
*)
value "map (λn. t_reverse (replicate n (0::int))) [0,1,2,3,4,5,6]"
lemma "t_reverse xs = length xs * (length xs + 1) div 2"
apply (induction xs) apply (auto simp: t_append_conv reverse_length) done
proof (prove)
goal (1 subgoal):
1. t_reverse xs = length xs * (length xs + 1) div 2

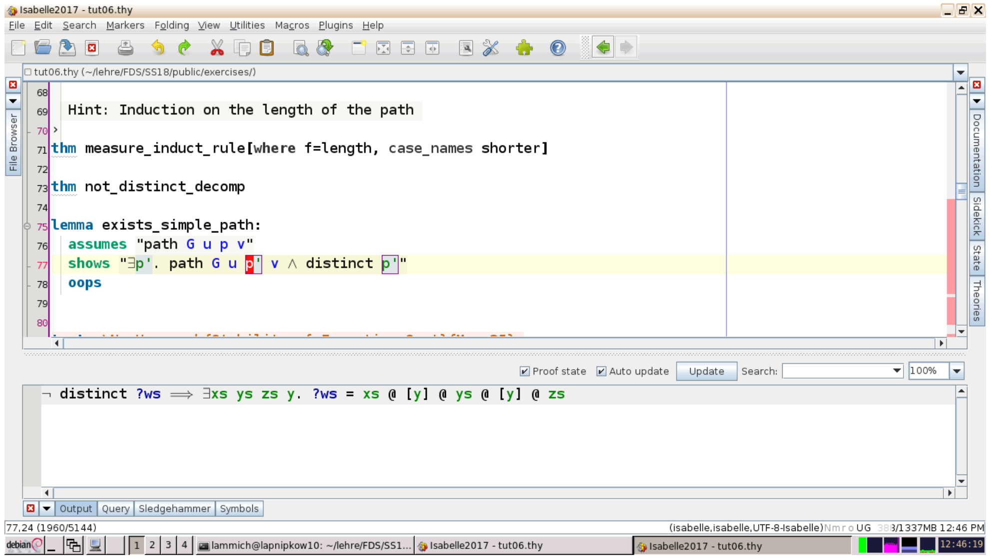

Isabelle2017 - tut06.thy
| "reverse (x#xs) = reverse xs @ [x]"
lemma reverse_length: "length (reverse xs) = length xs" by (induction xs) auto
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

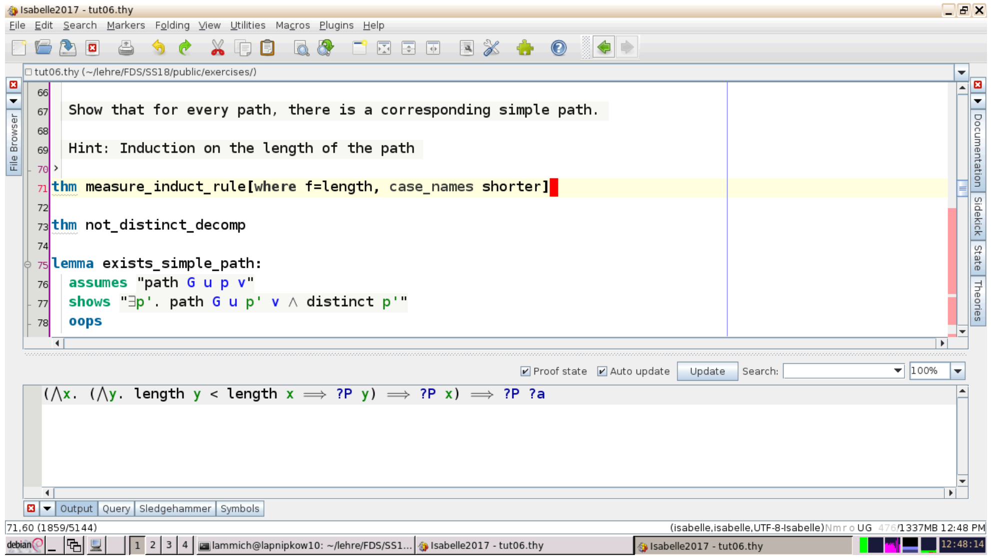

Isabelle2017 - tut06.thy
| "reverse (x#xs) = reverse xs @ [x]"
lemma reverse_length: "length (reverse xs) = length xs" by (induction xs) auto
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
theorem reverse_length: length (reverse ?xs) = length ?xs

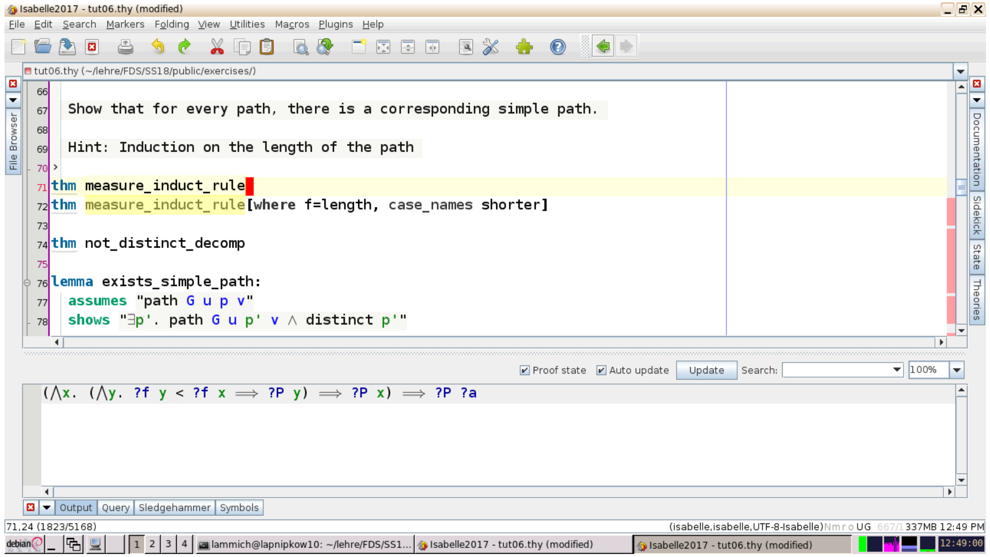

Isabelle2017 - tut06.thy
| "reverse (x#xs) = reverse xs @ [x]"
lemma reverse_length: "length (reverse xs) = length xs" by (induction xs) auto
fun t_reverse where
"t_reverse [] = 0"
| "t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

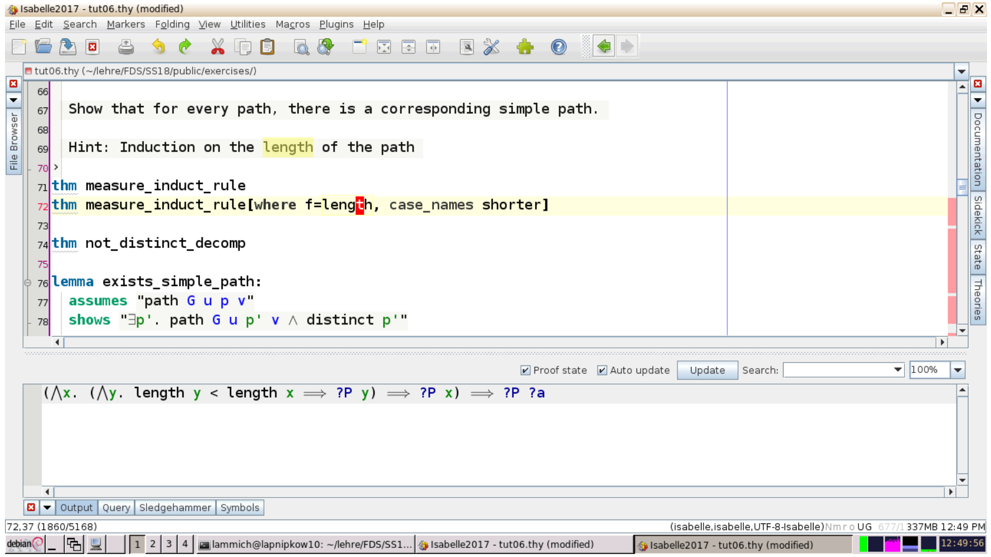

Isabelle2017 - tut06.thy
fun t_reverse where
"t_reverse [] = 0"
"t_reverse (x#xs) = t_reverse xs + t_append (reverse xs) [x] + 1"
(*
t 0 = 0
t (n+1) = t n + n + 1
*)
value "map (λn. t_reverse (replicate n (0::int))) [0,1,2,3,4,5,6]"
consts
t_reverse :: "'a list ⇒ nat"
Found termination order: "length <*mlex*> {}"

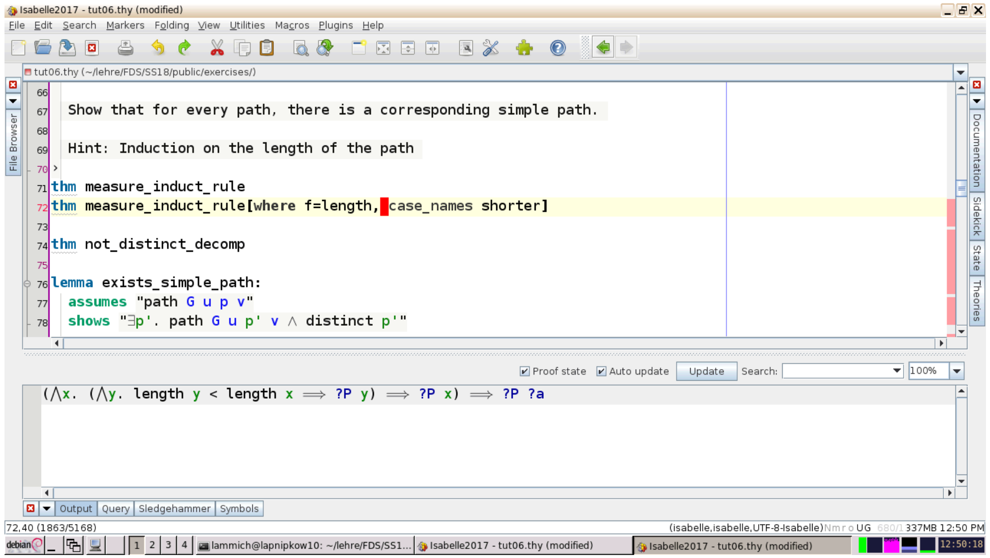

Isabelle2017 - tut06.thy
Recall the definition of paths from last exercise sheet:
fun path :: "('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a list ⇒ 'a ⇒ bool"
where
"path G u [] v ⟷ u=v"
"path G u (x#xs) v ⟷ G u x ∧ path G x xs v"
lemma path_append[simp]: "path G u (p1@p2) v ⟷ (∃w. path G u p1 w ∧ path G w p2 v)"
by (induction p1 arbitrary: u) auto
text ‹
A simple path is a path without loops, or, in other words, a path
consts
path :: "('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a list ⇒ 'a ⇒ bool"
Found termination order: "(λp. length (fst (snd (snd p)))) <*mlex*> {}"

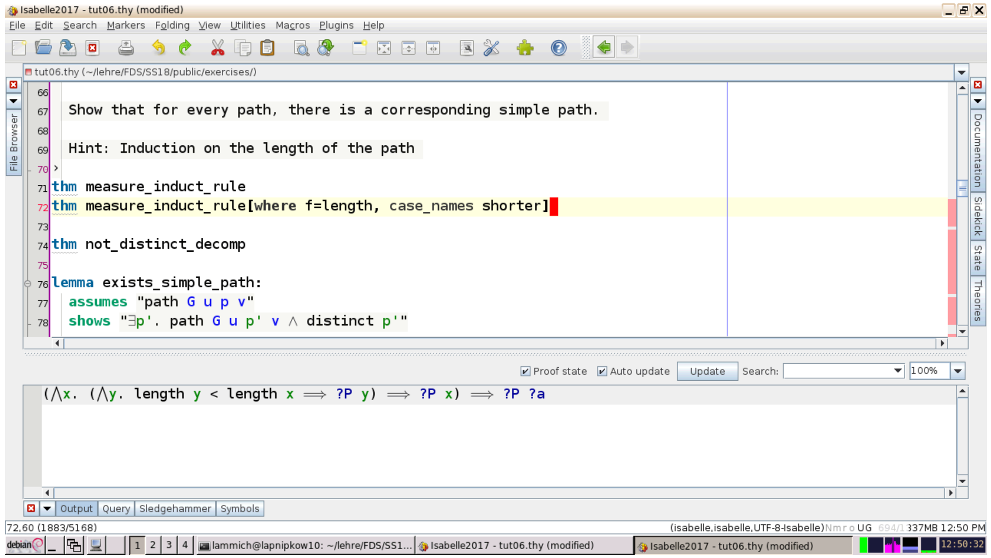

Isabelle2017 - tut06.thy
Hint: Induction on the length of the path
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
oops
¬ distinct ?ws ⟹ ∃xs ys zs y. ?ws = xs @ [y] @ ys @ [y] @ zs

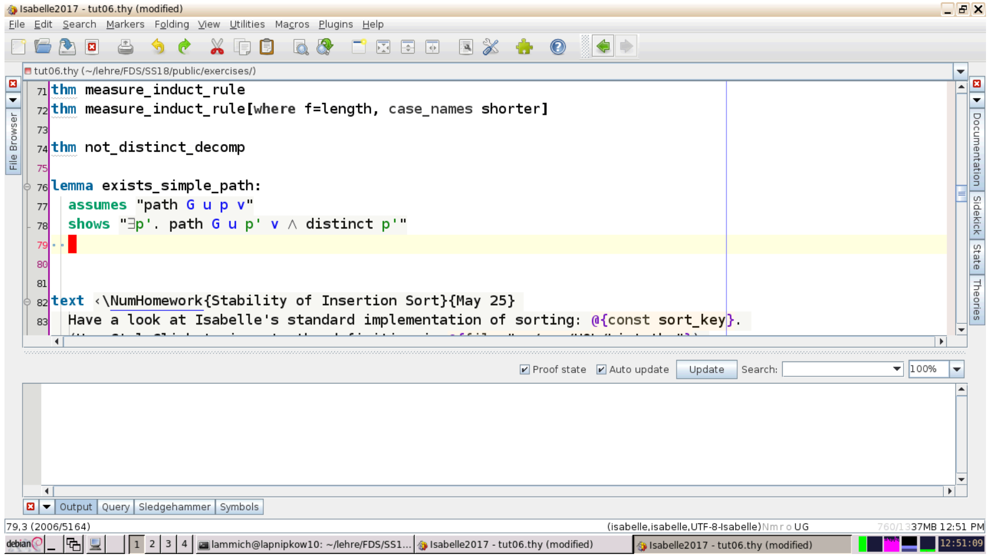

Isabelle2017 - tut06.thy
Show that for every path, there is a corresponding simple path.
Hint: Induction on the length of the path
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
oops
(⋀x. (⋀y. length y < length x ⟹ ?P y) ⟹ ?P x) ⟹ ?P ?a

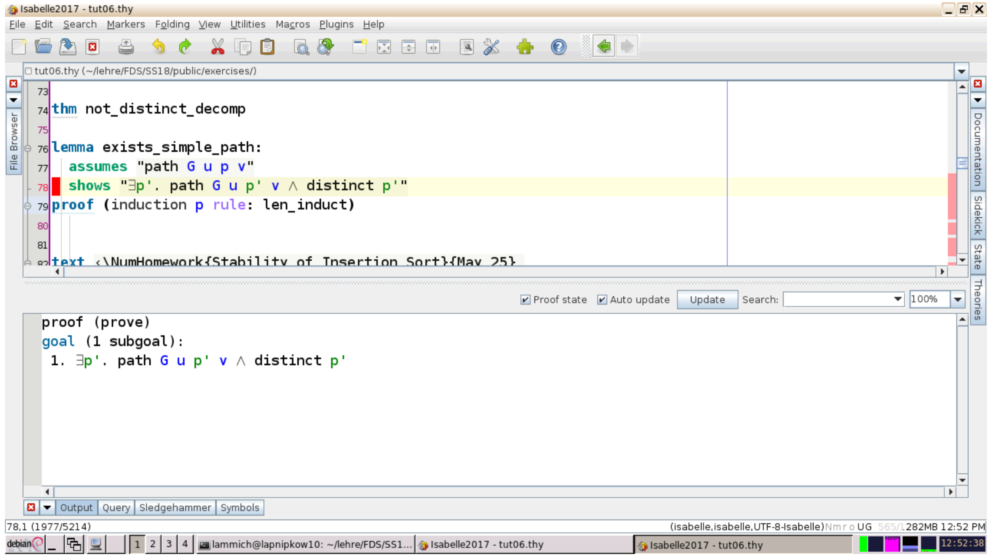

Isabelle2017 - tut06.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
Show that for every path, there is a corresponding simple path.
Hint: Induction on the length of the path
thm measure_induct_rule
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
Proof state Auto update Update Search: 100%
(⋀x. (⋀y. ?f y < ?f x ⟹ ?P y) ⟹ ?P x) ⟹ ?P ?a
Output Query Sledgehammer Symbols
71.24 (1823/5168)
(isabelle,isabelle,UTF-8-Isabelle)Nmro UG 337MB 12:49 PM
12:49:00

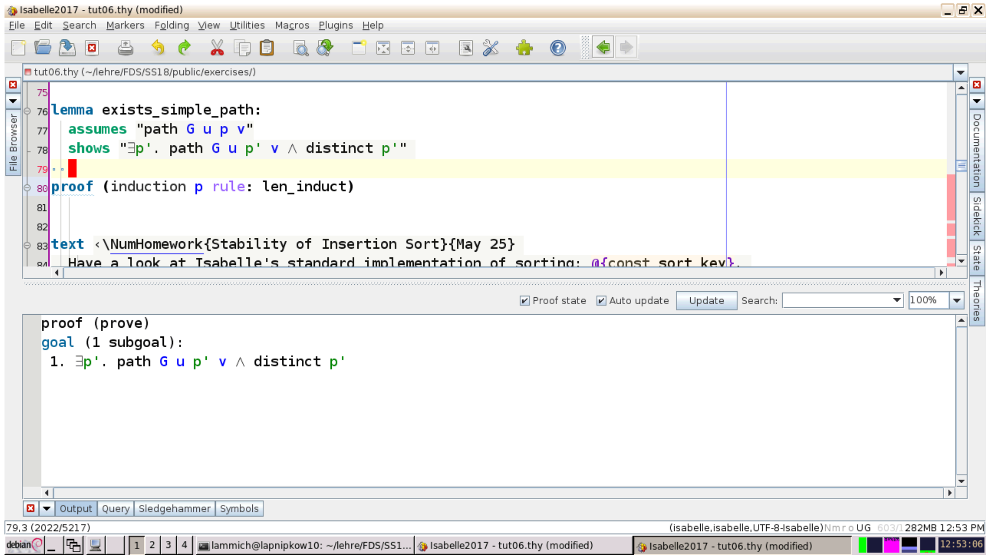

Isabelle2017 - tut06.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
Show that for every path, there is a corresponding simple path.
Hint: Induction on the length of the path
thm measure_induct_rule
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
Proof state Auto update Update Search: 100%
(⋀x. (⋀y. length y < length x ⟹ ?P y) ⟹ ?P x) ⟹ ?P ?a
Output Query Sledgehammer Symbols
72.37 (1860/5168)
(isabelle,isabelle,UTF-8-Isabelle)Nmro UG 337MB 12:49 PM
12:49:56

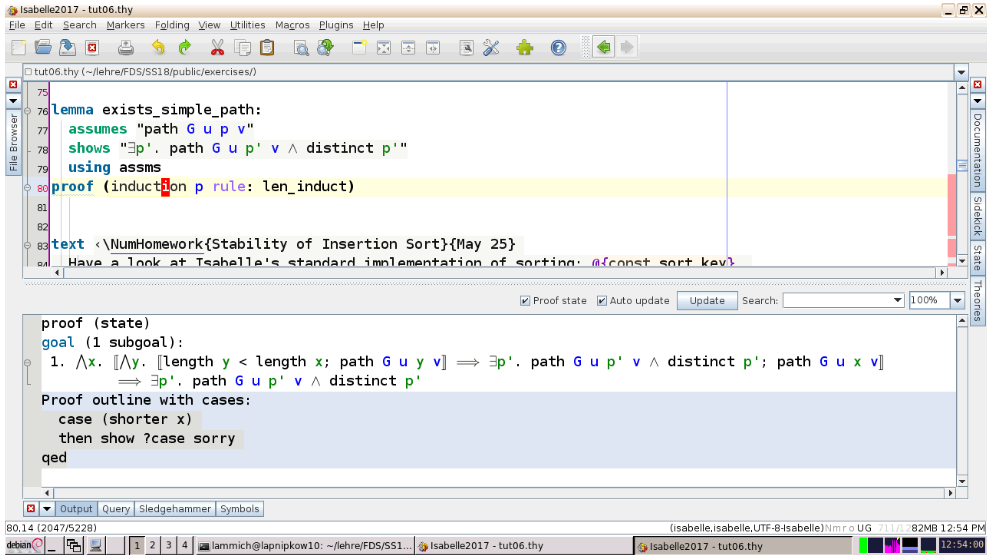

Isabelle2017 - tut06.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
Show that for every path, there is a corresponding simple path.
Hint: Induction on the length of the path
thm measure_induct_rule
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
Proof state Auto update Update Search: 100%
(⋀x. (⋀y. length y < length x ⟹ ?P y) ⟹ ?P x) ⟹ ?P ?a
Output Query Sledgehammer Symbols
72.40 (1863/5168)
(isabelle,isabelle,UTF-8-Isabelle)Nmro UG 337MB 12:50 PM
12:50:18

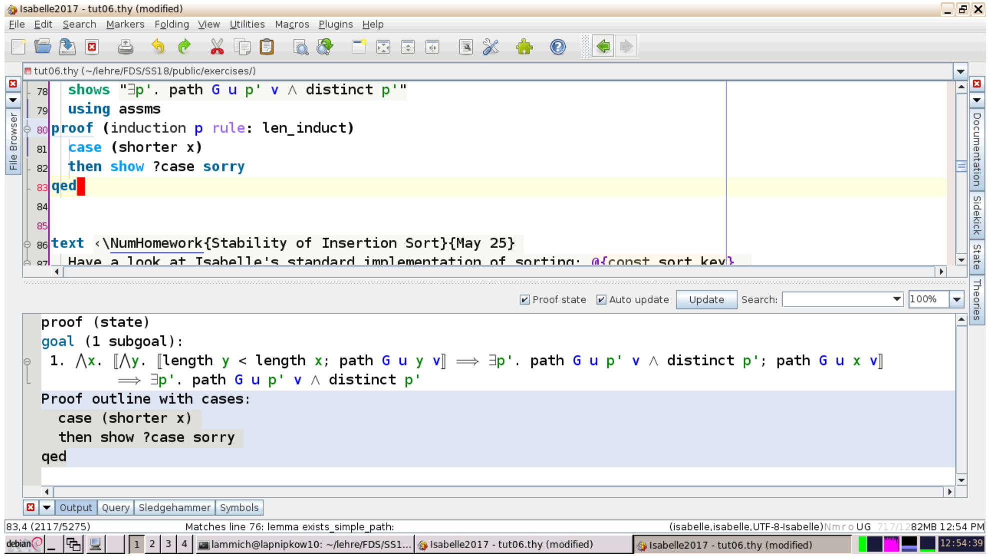

Isabelle2017 - tut06.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
Show that for every path, there is a corresponding simple path.
Hint: Induction on the length of the path
thm measure_induct_rule
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
Proof state Auto update Update Search: 100%
(⋀x. (⋀y. length y < length x ⟹ ?P y) ⟹ ?P x) ⟹ ?P ?a
Output Query Sledgehammer Symbols
72.60 (1883/5168)
(isabelle,isabelle,UTF-8-Isabelle)Nmro UG 337MB 12:50 PM
12:50:32

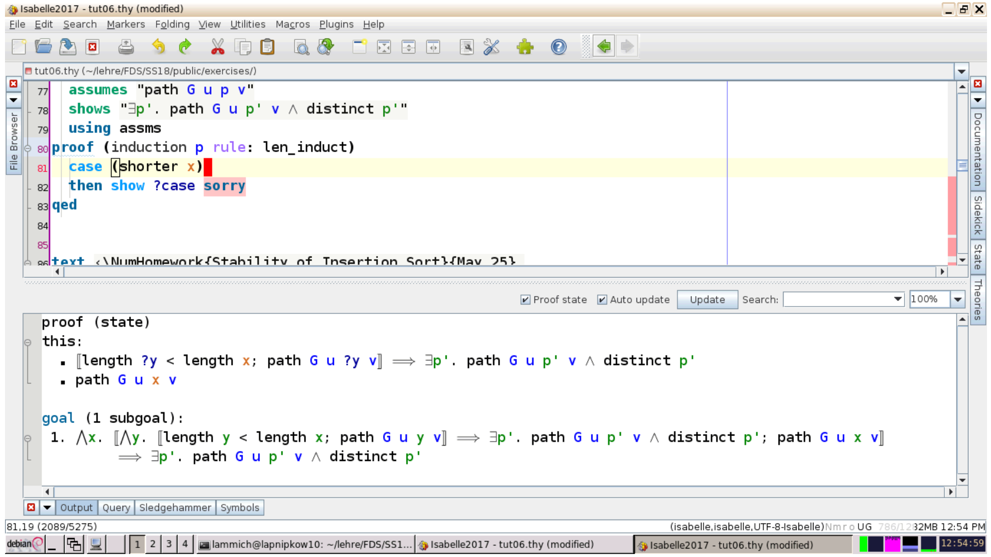

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
thm measure_induct_rule
thm measure_induct_rule[where f=length, case_names shorter]
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
79,3 (2006/5164)
(isabelle,isabelle,UTF-8-Isabelle)
12:51 PM

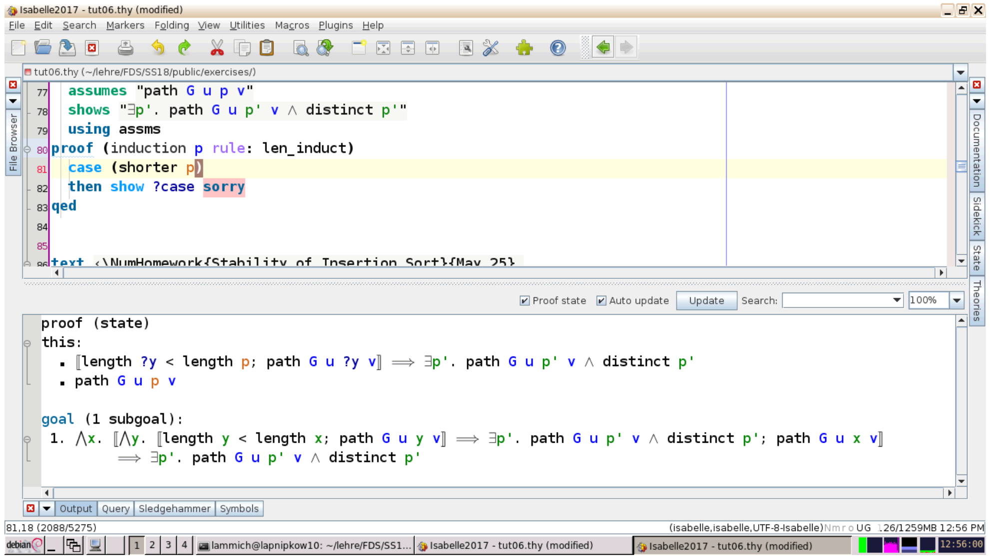

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
thm not_distinct_decomp
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
proof (induction p rule: len_induct)
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. ∃p'. path G u p' v ∧ distinct p'
Output
Query
Sledgehammer
Symbols
78,1 (1977/5214)
(isabelle,isabelle,UTF-8-Isabelle)
12:52 PM

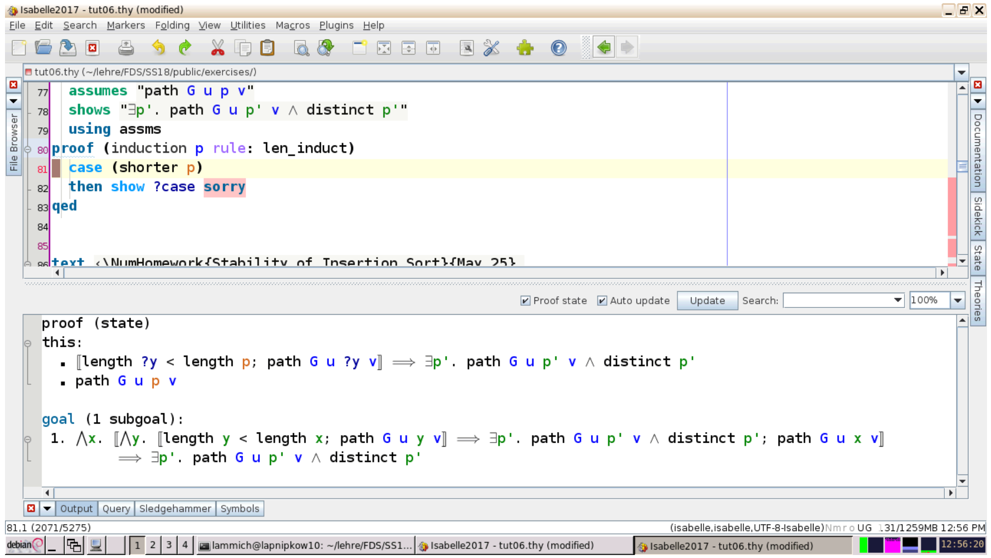

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
proof (induction p rule: len_induct)
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. ∃p'. path G u p' v ∧ distinct p'
Output
Query
Sledgehammer
Symbols
79,3 (2022/5217)
(isabelle,isabelle,UTF-8-Isabelle)
12:53 PM

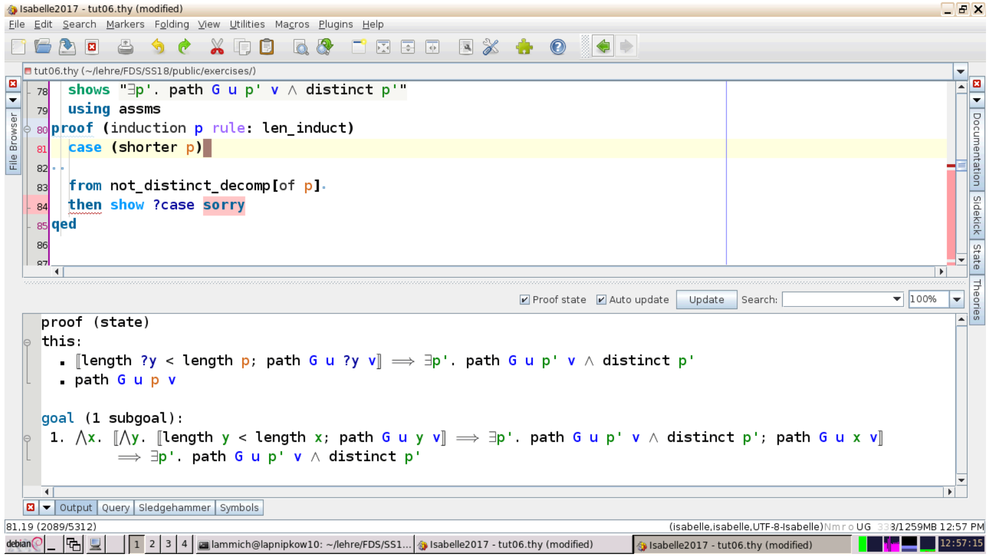

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}
Proof state
Auto update
Update
Search:
100%
proof (state)
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'
Proof outline with cases:
case (shorter x)
then show ?case sorry
qed
Output
Query
Sledgehammer
Symbols
80,14 (2047/5228)
(isabelle,isabelle,UTF-8-Isabelle)
12:54 PM

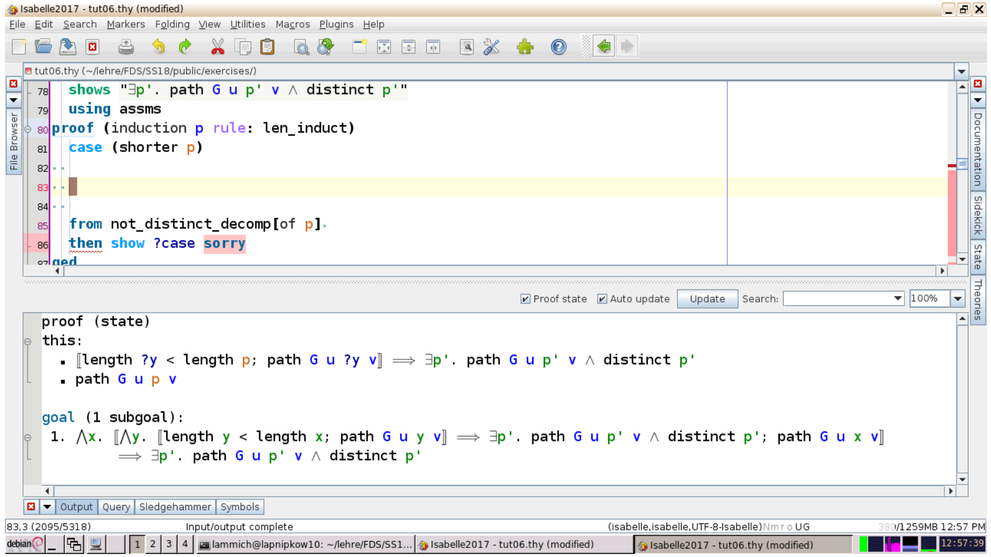

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter x)
then show ?case sorry
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
proof (state)
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'
Proof outline with cases:
case (shorter x)
then show ?case sorry
qed
Matches line 76: lemma exists_simple_path:

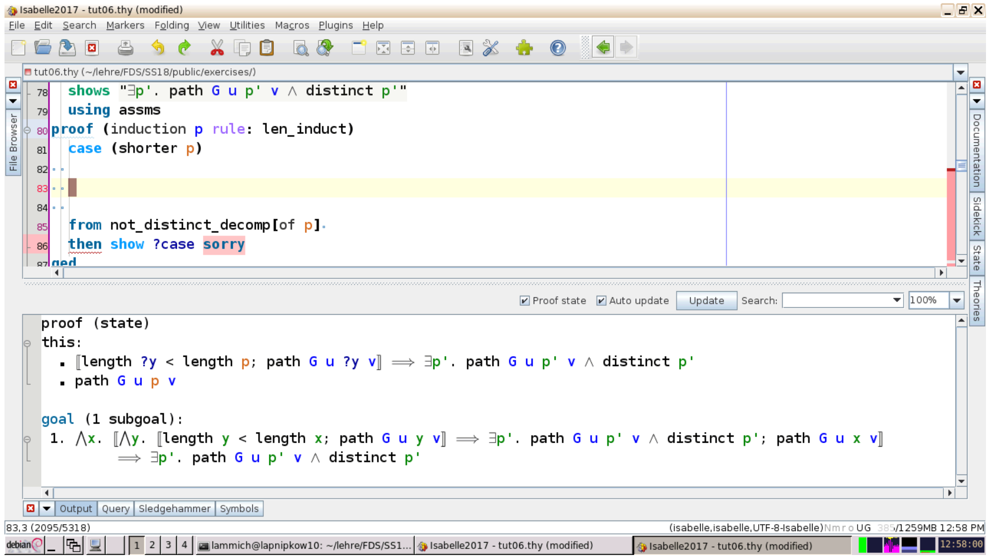

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter x)
then show ?case sorry
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
proof (state)
this:
⟦length ?y < length x; path G u ?y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'
path G u x v
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'

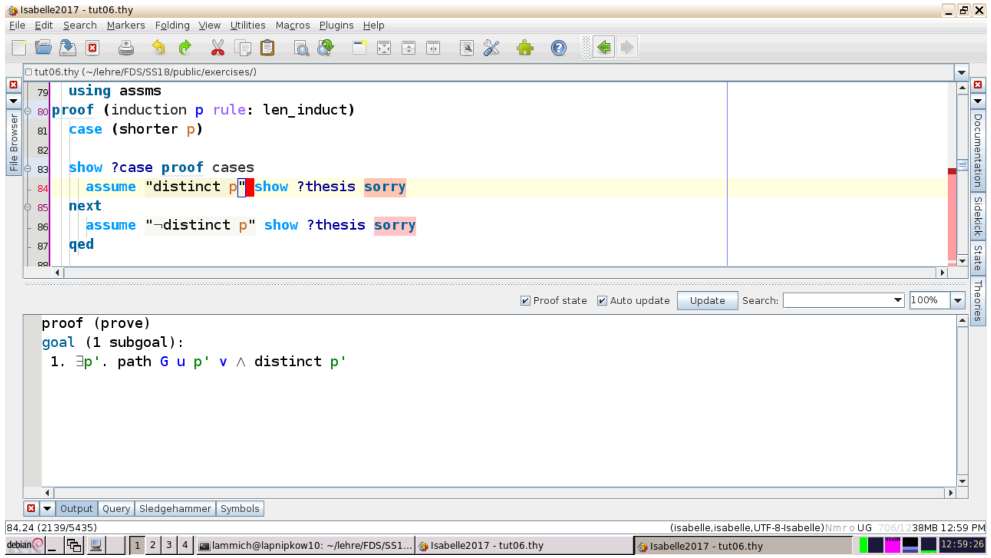

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter p)
then show ?case sorry
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
proof (state)
this:
⟦length ?y < length p; path G u ?y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'
path G u p v
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'

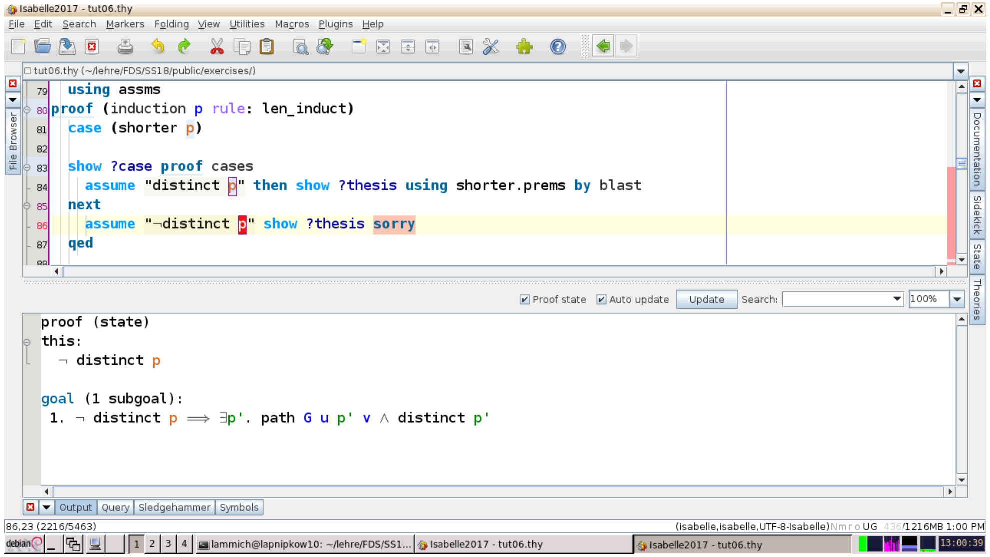

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter p)
then show ?case sorry
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
proof (state)
this:
⟦length ?y < length p; path G u ?y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'
path G u p v
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'

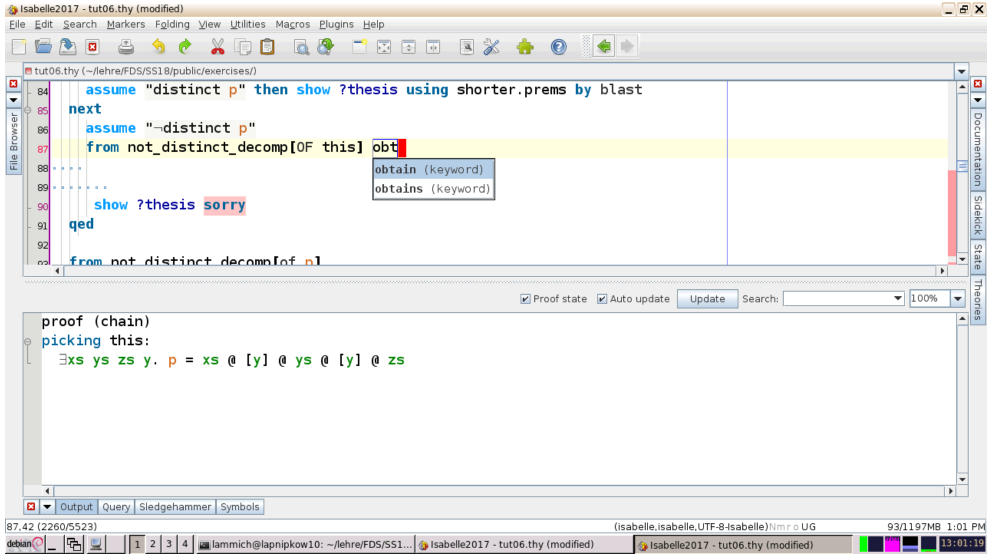

Isabelle2017 - tut06.thy (modified)
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter p)
from not_distinct_decomp[of p]
then show ?case sorry
qed
proof (state)
this:
⟦length ?y < length p; path G u ?y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'
path G u p v
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'

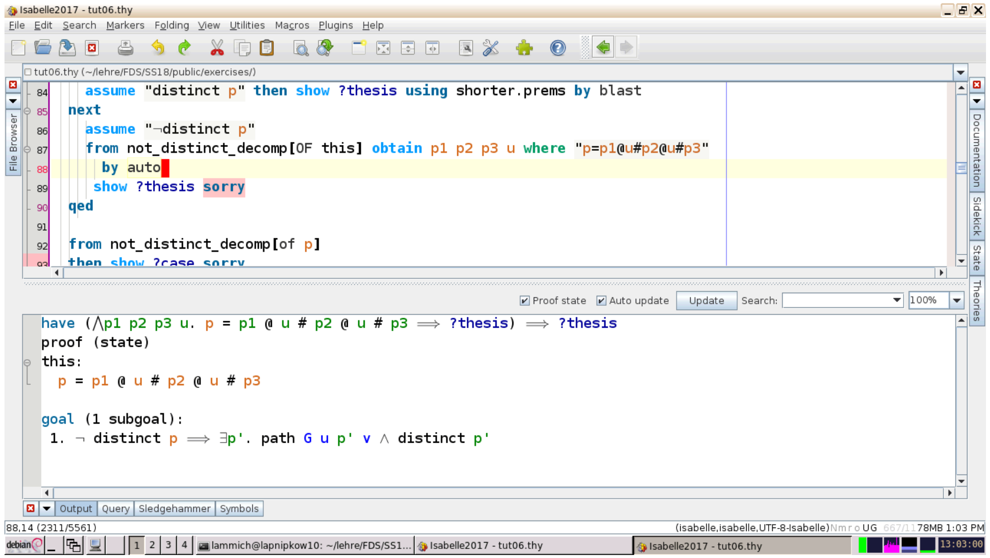

Isabelle2017 - tut06.thy (modified)
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter p)
from not_distinct_decomp[of p]
then show ?case sorry
qed
proof (state)
this:
⟦length ?y < length p; path G u ?y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'
path G u p v
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'

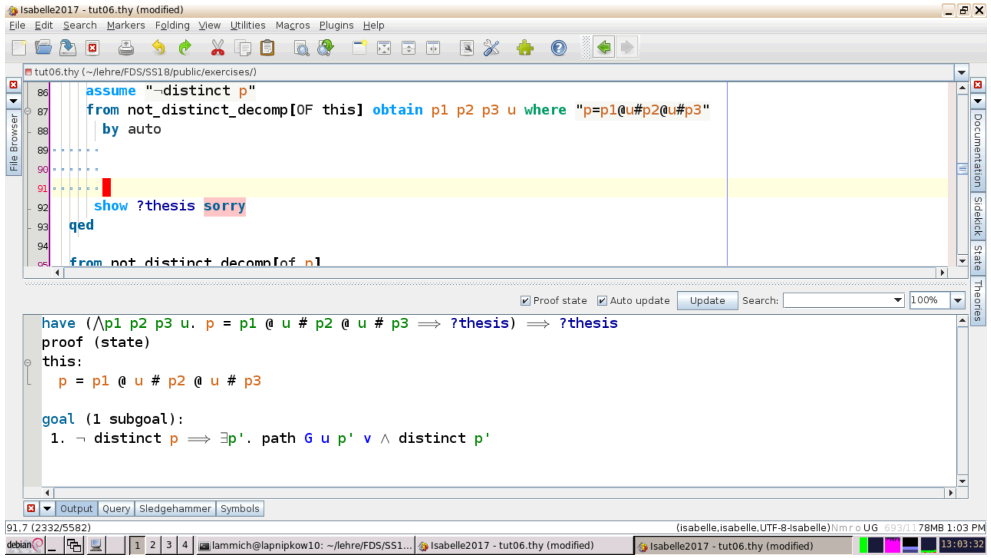

Isabelle2017 - tut06.thy (modified)
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter p)
from not_distinct_decomp[of p]
then show ?case sorry
qed
proof (state)
this:
⟦length ?y < length p; path G u ?y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'
path G u p v
goal (1 subgoal):
1. ⋀x. ⟦⋀y. ⟦length y < length x; path G u y v⟧ ⟹ ∃p'. path G u p' v ∧ distinct p'; path G u x v⟧
⟹ ∃p'. path G u p' v ∧ distinct p'

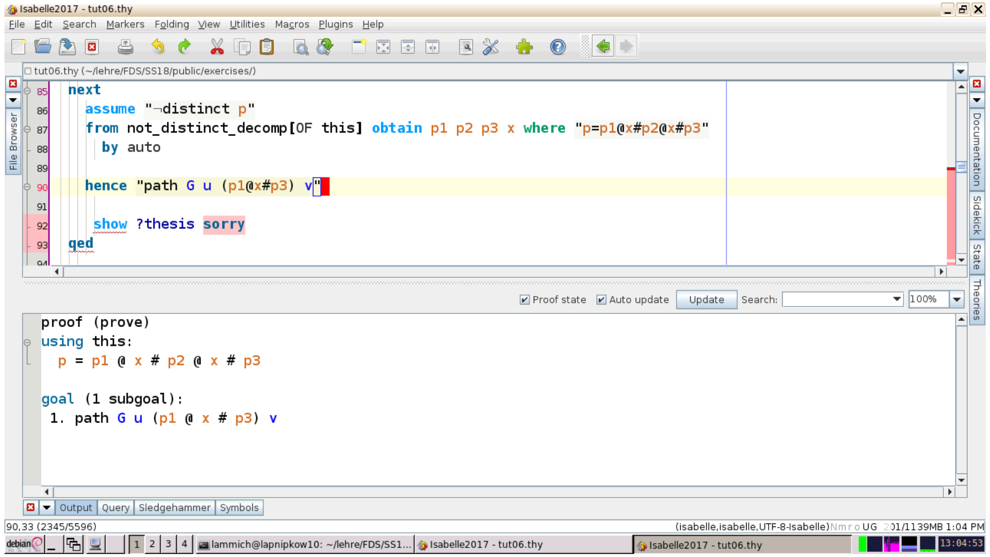
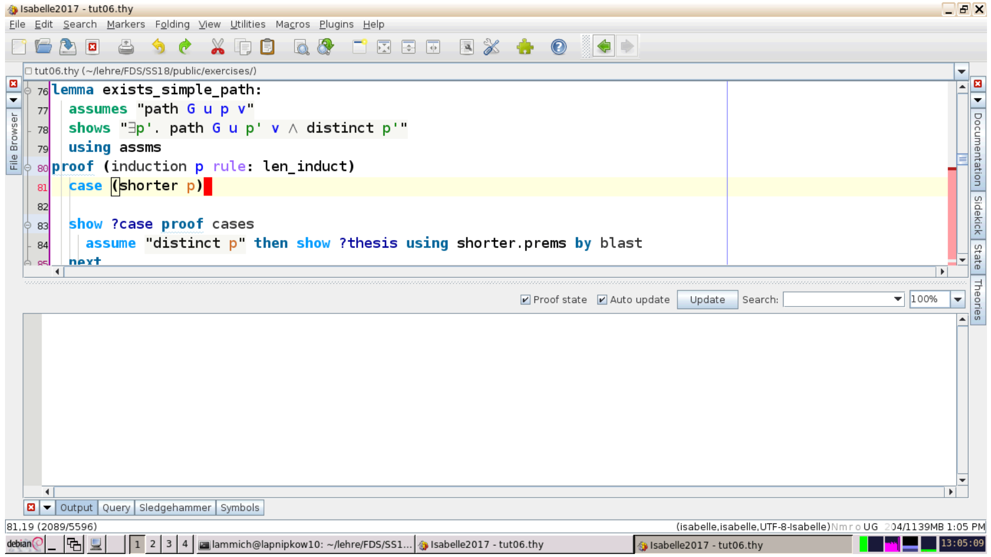
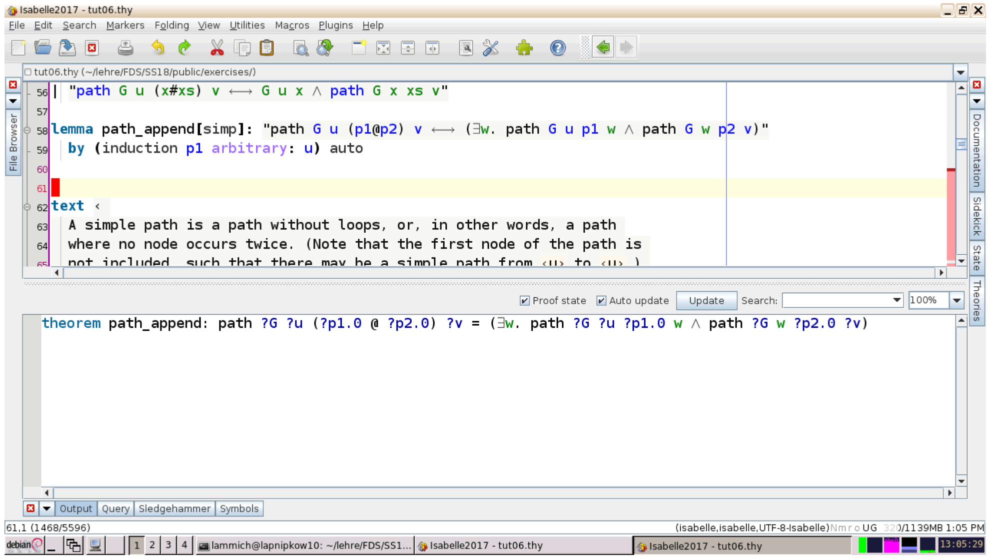
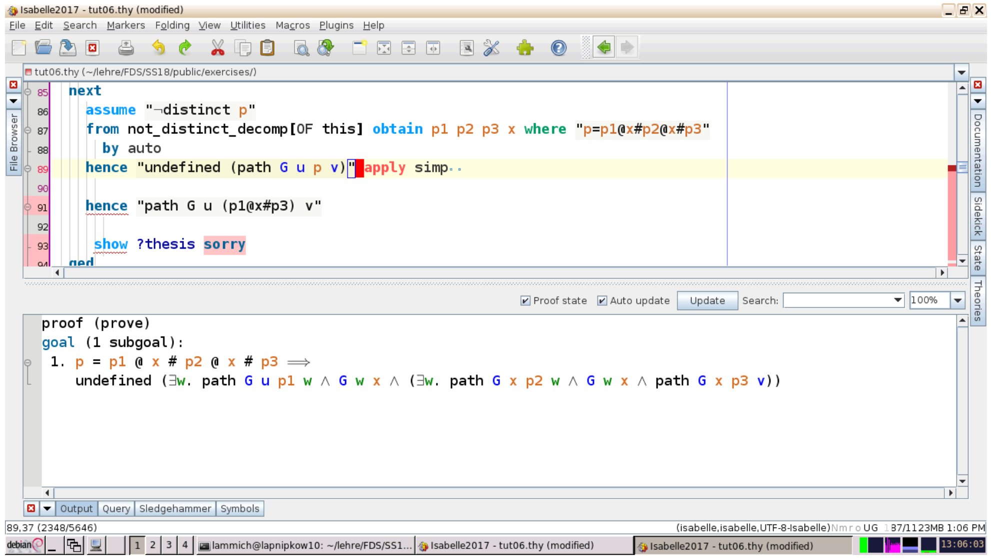

Isabelle2017 - tut06.thy
using assms
proof (induction p rule: len_induct)
case (shorter p)
show ?case proof cases
assume "distinct p" show ?thesis sorry
next
assume "¬distinct p" show ?thesis sorry
qed
proof (prove)
goal (1 subgoal):
1. ∃p'. path G u p' v ∧ distinct p'

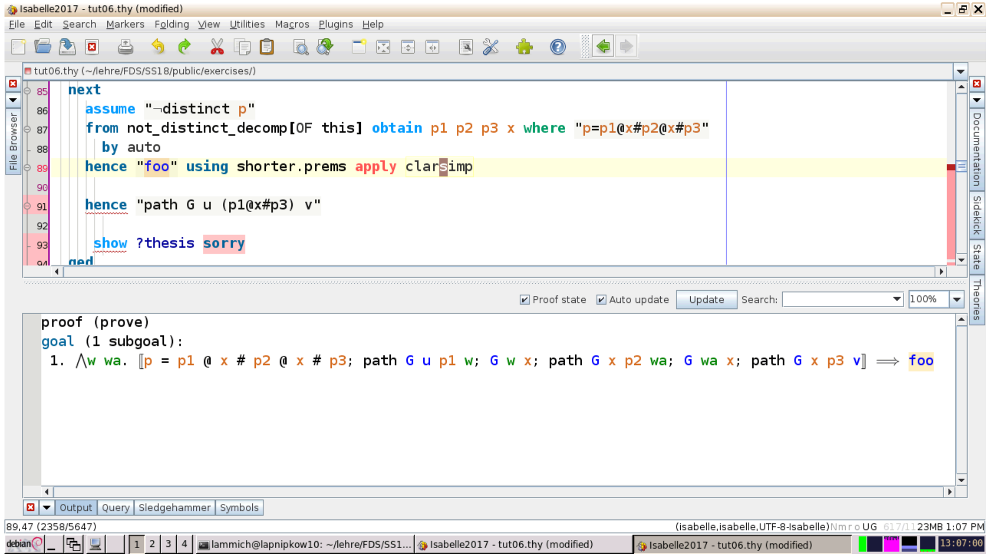

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
using assms
proof (induction p rule: len_induct)
case (shorter p)
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p" show ?thesis sorry
qed
proof (state)
this:
¬ distinct p
goal (1 subgoal):
1. ¬ distinct p ⟹ ∃p'. path G u p' v ∧ distinct p'

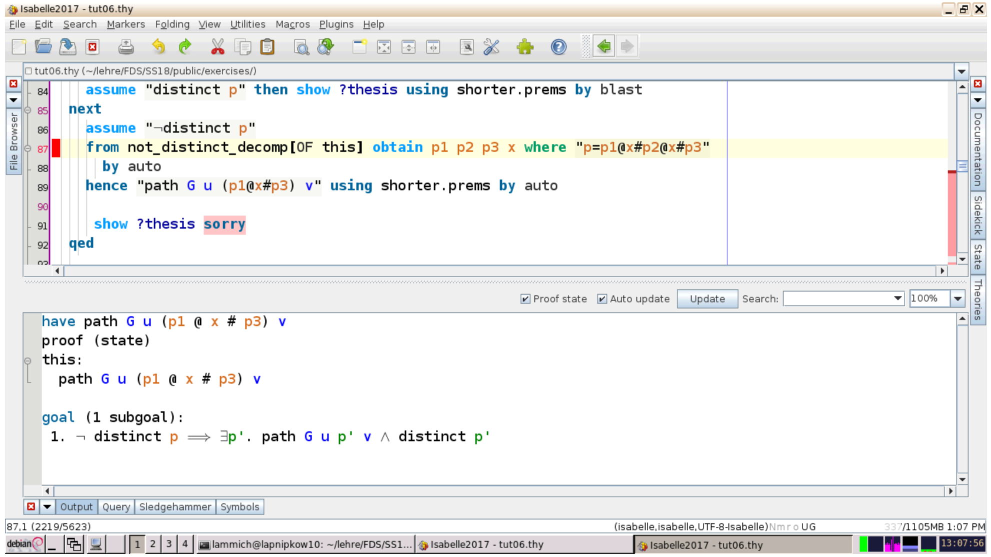

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obt
obtain (keyword)
obtains (keyword)
show ?thesis sorry
qed
from not_distinct_decomp[of p]
proof (chain)
picking this:
∃xs ys zs y. p = xs @ [y] @ ys @ [y] @ zs

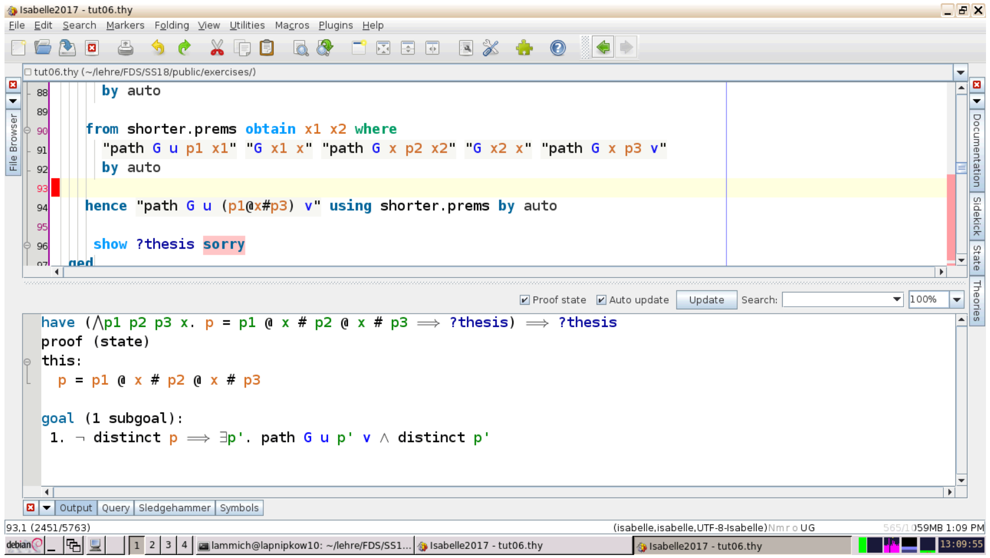

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 u where "p=p1@u#p2@u#p3"
by auto
show ?thesis sorry
qed
from not_distinct_decomp[of p]
then show ?case sorry
have (⋀p1 p2 p3 u. p = p1 @ u # p2 @ u # p3 ⟹ ?thesis) ⟹ ?thesis
proof (state)
this:
p = p1 @ u # p2 @ u # p3
goal (1 subgoal):
1. ¬ distinct p ⟹ ∃p'. path G u p' v ∧ distinct p'

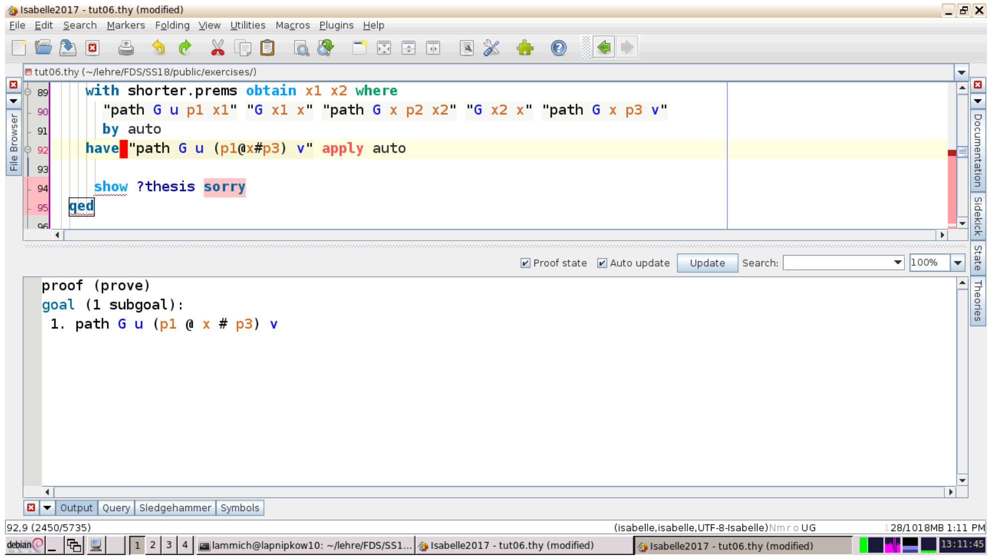

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 u where "p=p1@u#p2@u#p3"
by auto
show ?thesis sorry
qed
from not_distinct_decomp[of p]
have (⋀p1 p2 p3 u. p = p1 @ u # p2 @ u # p3 ⟹ ?thesis) ⟹ ?thesis
proof (state)
this:
p = p1 @ u # p2 @ u # p3
goal (1 subgoal):
1. ¬ distinct p ⟹ ∃p'. path G u p' v ∧ distinct p'

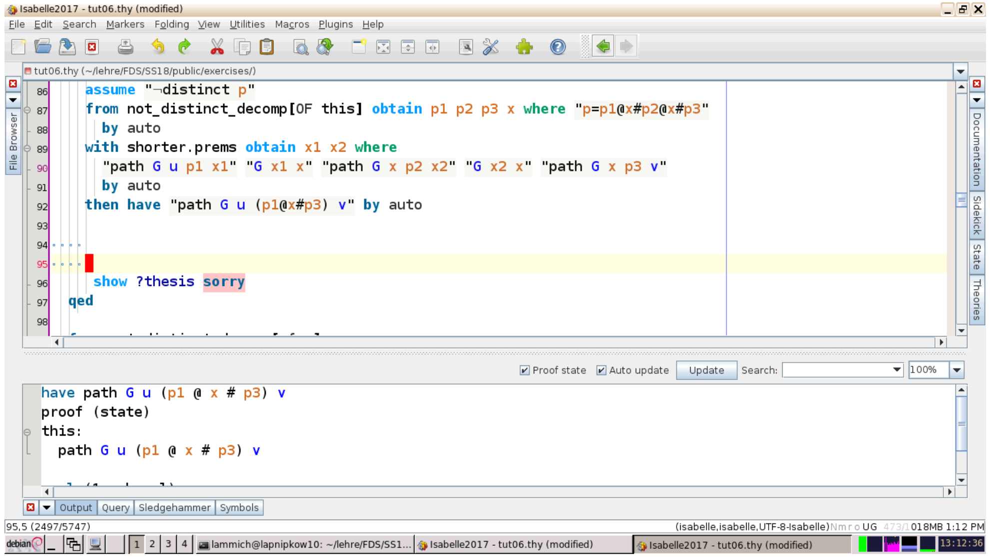

Isabelle2017 - tut06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where "p=p1@x#p2@x#p3"
by auto
hence "path G u (p1@x#p3) v"
show ?thesis sorry
qed
Proof state
Auto update
Update
Search:
100%
proof (prove)
using this:
p = p1 @ x # p2 @ x # p3
goal (1 subgoal):
1. path G u (p1 @ x # p3) v
Output
Query
Sledgehammer
Symbols
90,33 (2345/5596)
(isabelle,isabelle,UTF-8-Isabelle)
1:04 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut06.thy
Isabelle2017 - tut06.thy
13:04:53

Isabelle2017 - tut06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
lemma exists_simple_path:
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter p)
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems by blast
next
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
81,19 (2089/5596)
(isabelle,isabelle,UTF-8-Isabelle)
1:05 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut06.thy
Isabelle2017 - tut06.thy
13:05:09

Isabelle2017 - tut06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
| "path G u (x#xs) v ⟷ G u x ∧ path G x xs v"
lemma path_append[simp]: "path G u (p1@p2) v ⟷ (∃w. path G u p1 w ∧ path G w p2 v)"
by (induction p1 arbitrary: u) auto
text ‹
A simple path is a path without loops, or, in other words, a path
where no node occurs twice. (Note that the first node of the path is
not included, such that there may be a simple path from ‹u› to ‹u›.)
Proof state
Auto update
Update
Search:
100%
theorem path_append: path ?G ?u (?p1.0 @ ?p2.0) ?v = (∃w. path ?G ?u ?p1.0 w ∧ path ?G w ?p2.0 ?v)
Output
Query
Sledgehammer
Symbols
61,1 (1468/5596)
(isabelle,isabelle,UTF-8-Isabelle)
1:05 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut06.thy
Isabelle2017 - tut06.thy
13:05:29

Isabelle2017 - tut06.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where "p=p1@x#p2@x#p3"
by auto
hence "undefined (path G u p v)" apply simp
hence "path G u (p1@x#p3) v"
show ?thesis sorry
qed
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal (1 subgoal):
1. p = p1 @ x # p2 @ x # p3 ⟹
undefined (∃w. path G u p1 w ∧ G w x ∧ (∃w. path G x p2 w ∧ G w x ∧ path G x p3 v))
Output
Query
Sledgehammer
Symbols
89,37 (2348/5646)
(isabelle,isabelle,UTF-8-Isabelle)
1:06 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut06.thy (modified)
Isabelle2017 - tut06.thy (modified)
13:06:03

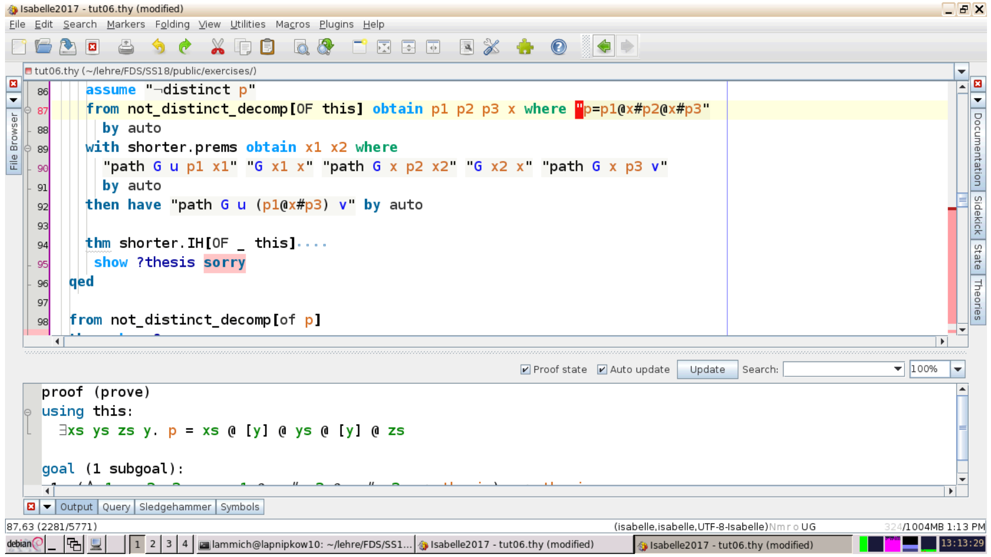

Isabelle2017 - tut06.thy (modified)
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where "p=p1@x#p2@x#p3"
by auto
hence "foo" using shorter.prems apply clarsimp
hence "path G u (p1@x#p3) v"
show ?thesis sorry
qed
proof (prove)
goal (1 subgoal):
1. ⋀w wa. ⟦p = p1 @ x # p2 @ x # p3; path G u p1 w; G w x; path G x p2 wa; G wa x; path G x p3 v⟧ ⟹ foo

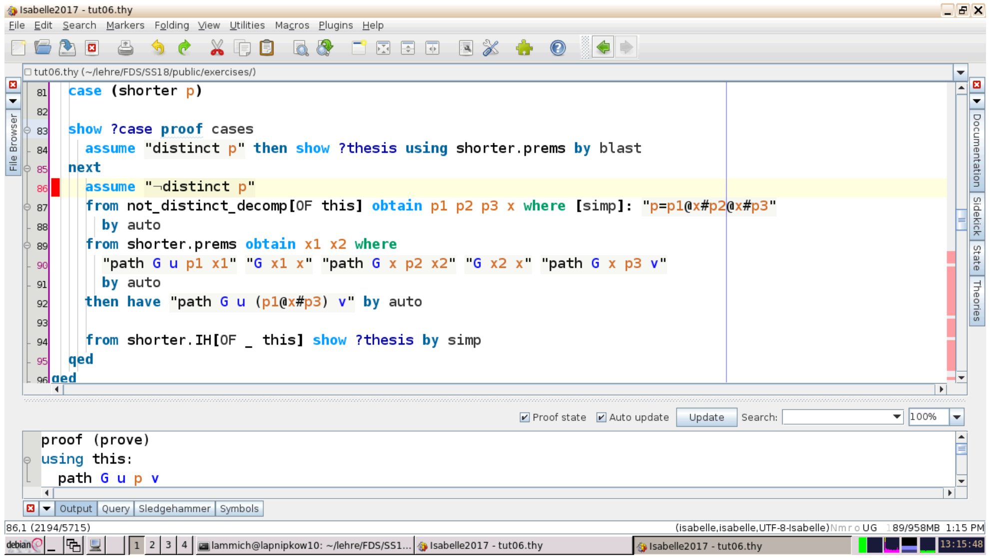

Isabelle2017 - tut06.thy
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where "p=p1@x#p2@x#p3"
by auto
hence "path G u (p1@x#p3) v" using shorter.prems by auto
show ?thesis sorry
qed
have path G u (p1 @ x # p3) v
proof (state)
this:
path G u (p1 @ x # p3) v
goal (1 subgoal):
1. ¬ distinct p ⟹ ∃p'. path G u p' v ∧ distinct p'

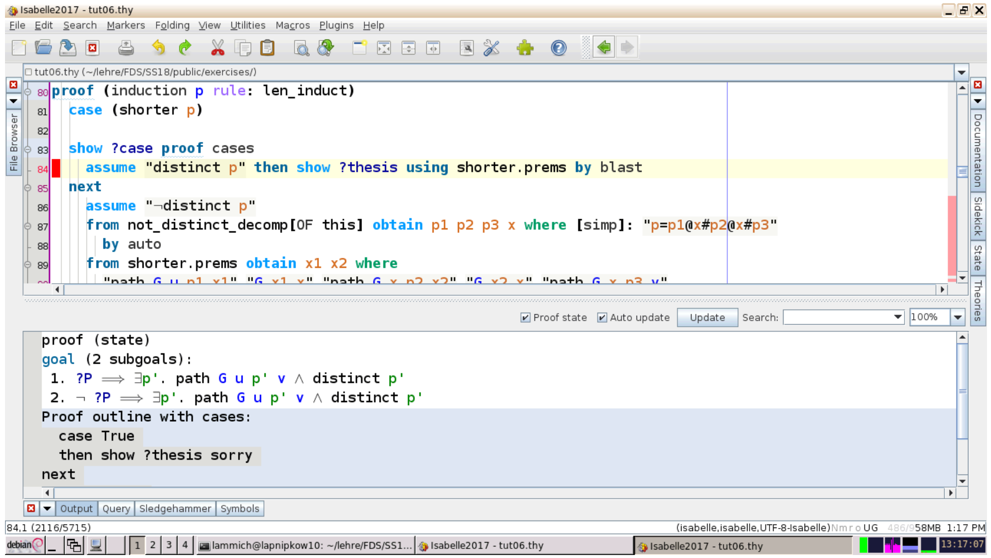

Isabelle2017 - tut06.thy
by auto
from shorter.prems obtain x1 x2 where
"path G u p1 x1" "G x1 x" "path G x p2 x2" "G x2 x" "path G x p3 v"
by auto
hence "path G u (p1@x#p3) v" using shorter.prems by auto
show ?thesis sorry
qed
have (⋀p1 p2 p3 x. p = p1 @ x # p2 @ x # p3 ⟹ ?thesis) ⟹ ?thesis
proof (state)
this:
p = p1 @ x # p2 @ x # p3
goal (1 subgoal):
1. ¬ distinct p ⟹ ∃p'. path G u p' v ∧ distinct p'

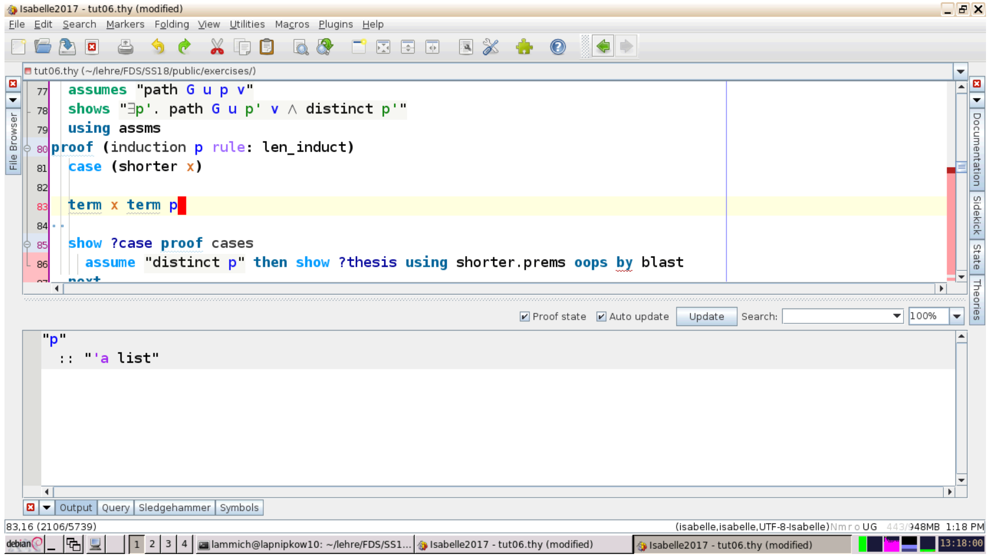

Isabelle2017 - tut06.thy (modified)
with shorter.prems obtain x1 x2 where
"path G u p1 x1" "G x1 x" "path G x p2 x2" "G x2 x" "path G x p3 v"
by auto
have "path G u (p1@x#p3) v" apply auto
show ?thesis sorry
qed
proof (prove)
goal (1 subgoal):
1. path G u (p1 @ x # p3) v

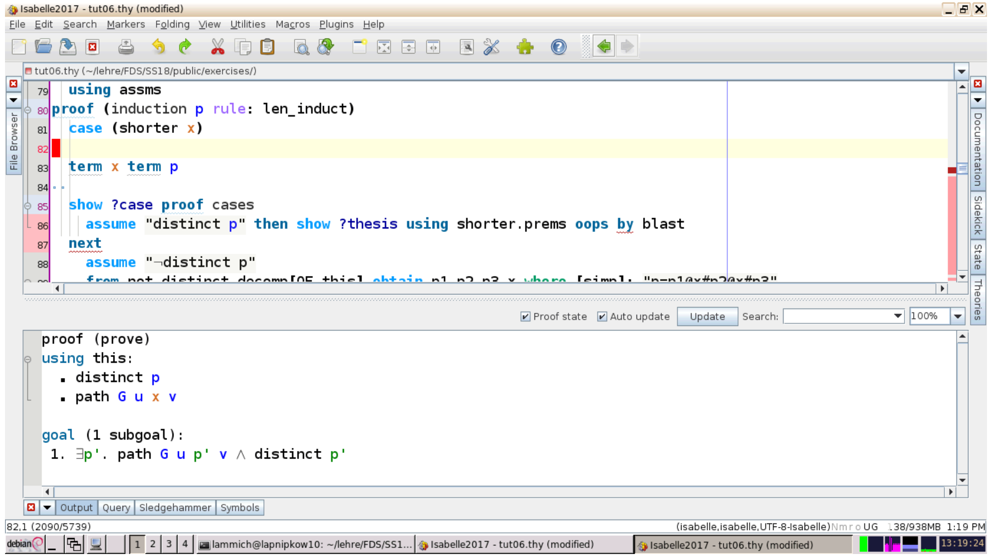

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where "p=p1@x#p2@x#p3"
by auto
with shorter.prems obtain x1 x2 where
"path G u p1 x1" "G x1 x" "path G x p2 x2" "G x2 x" "path G x p3 v"
by auto
then have "path G u (p1@x#p3) v" by auto
show ?thesis sorry
qed
have path G u (p1 @ x # p3) v
proof (state)
this:
path G u (p1 @ x # p3) v
Output
Query
Sledgehammer
Symbols

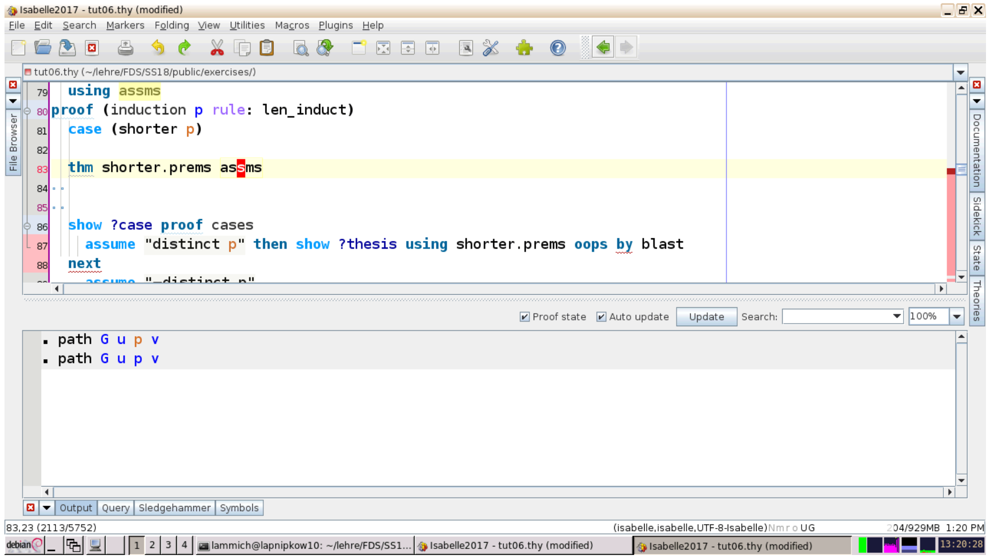

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where "p=p1@x#p2@x#p3"
by auto
with shorter.prems obtain x1 x2 where
"path G u p1 x1" "G x1 x" "path G x p2 x2" "G x2 x" "path G x p3 v"
by auto
then have "path G u (p1@x#p3) v" by auto
thm shorter.IH[OF _ this]
show ?thesis sorry
qed
from not_distinct_decomp[of p]
proof (prove)
using this:
∃xs ys zs y. p = xs @ [y] @ ys @ [y] @ zs
goal (1 subgoal):
Output
Query
Sledgehammer
Symbols

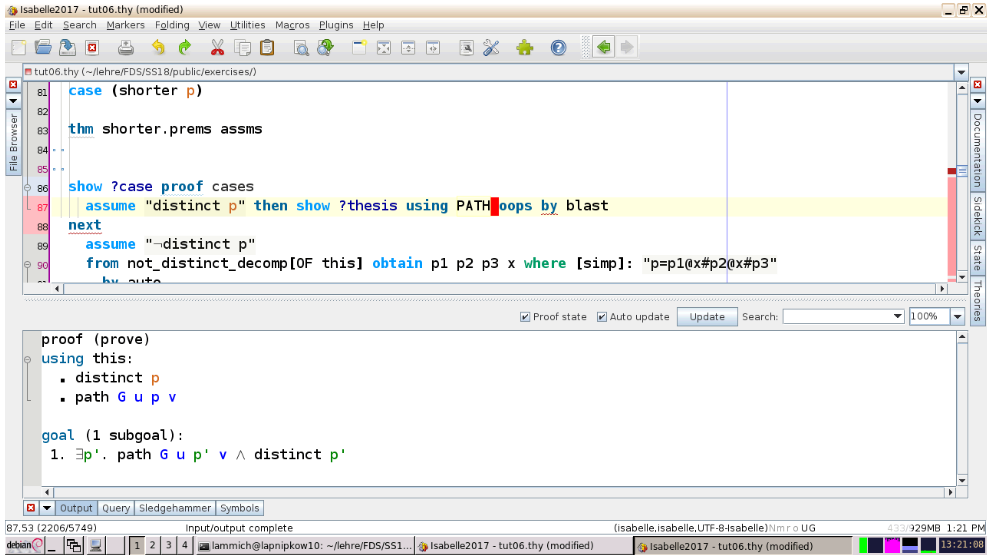

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
case (shorter p)
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where [simp]: "p=p1@x#p2@x#p3"
by auto
from shorter.prems obtain x1 x2 where
"path G u p1 x1" "G x1 x" "path G x p2 x2" "G x2 x" "path G x p3 v"
by auto
then have "path G u (p1@x#p3) v" by auto
from shorter.IH[OF _ this] show ?thesis by simp
qed
qed
proof (prove)
using this:
path G u p v
Output
Query
Sledgehammer
Symbols

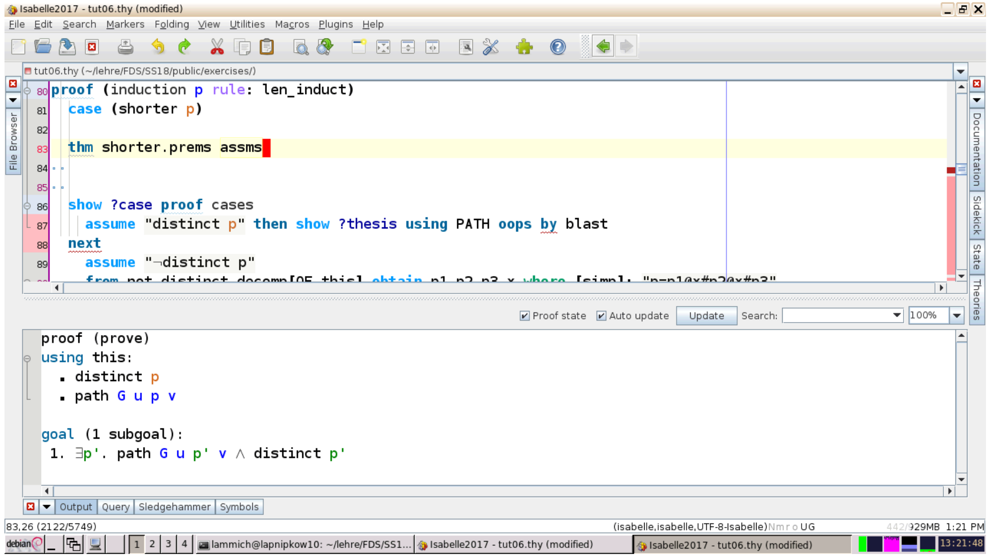

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
proof (induction p rule: len_induct)
case (shorter p)
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where [simp]: "p=p1@x#p2@x#p3"
by auto
from shorter.prems obtain x1 x2 where
proof (state)
goal (2 subgoals):
1. ?P ⟹ ∃p'. path G u p' v ∧ distinct p'
2. ¬ ?P ⟹ ∃p'. path G u p' v ∧ distinct p'
Proof outline with cases:
case True
then show ?thesis sorry
next
Output
Query
Sledgehammer
Symbols

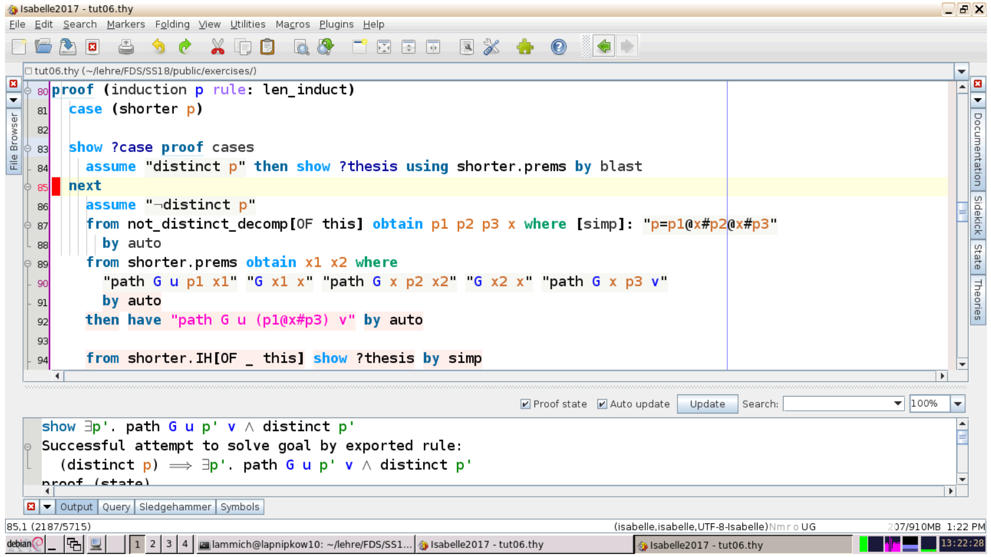

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
assumes "path G u p v"
shows "∃p'. path G u p' v ∧ distinct p'"
using assms
proof (induction p rule: len_induct)
case (shorter x)
term x term p
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems oops by blast
"p"
:: "'a list"
Output Query Sledgehammer Symbols

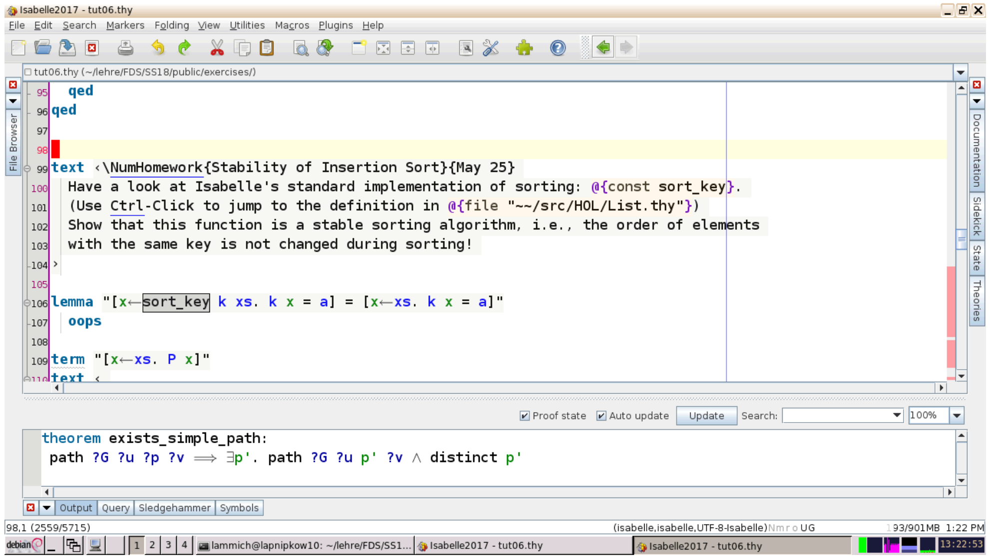

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
using assms
proof (induction p rule: len_induct)
case (shorter x)
term x term p
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems oops by blast
next
assume "¬distinct p"
proof (prove)
using this:
distinct p
path G u x v
goal (1 subgoal):
1. ∃p'. path G u p' v ∧ distinct p'
Output Query Sledgehammer Symbols

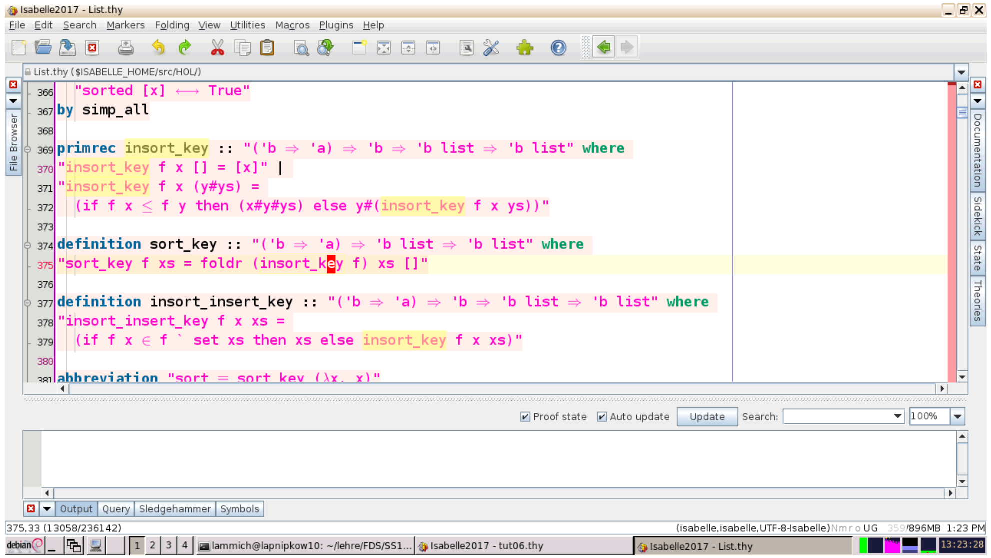

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
using assms
proof (induction p rule: len_induct)
case (shorter p)
thm shorter.prems assms
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems oops by blast
next
path G u p v
path G u p v
Output Query Sledgehammer Symbols

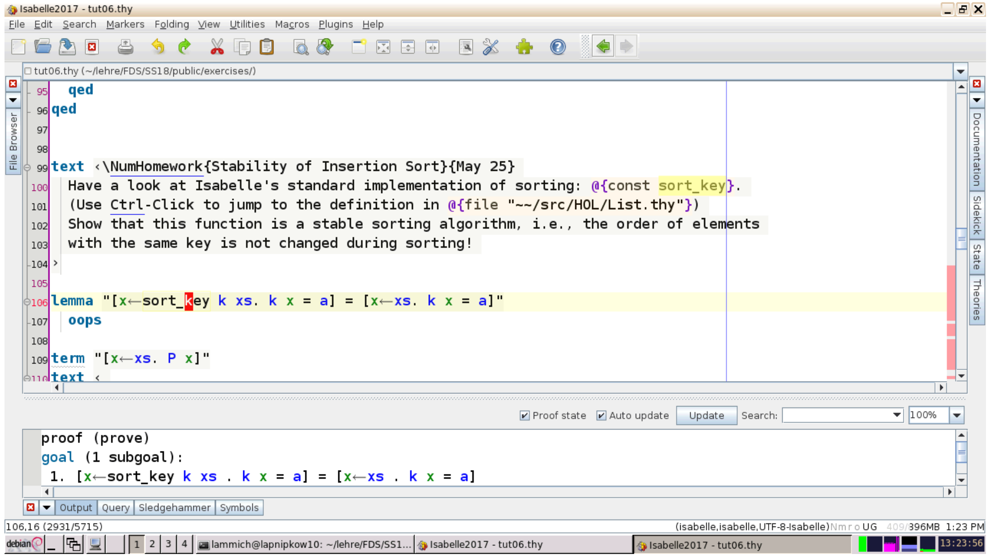
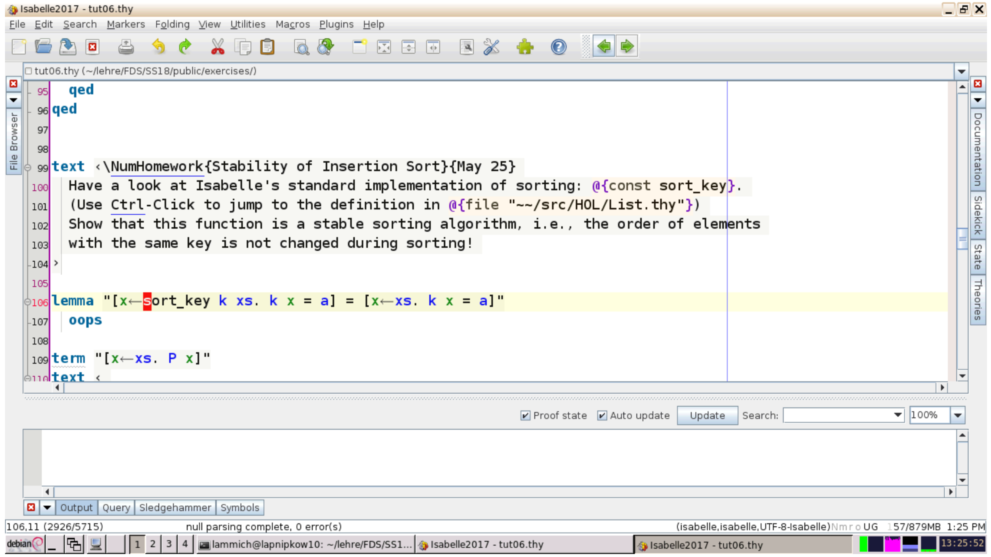
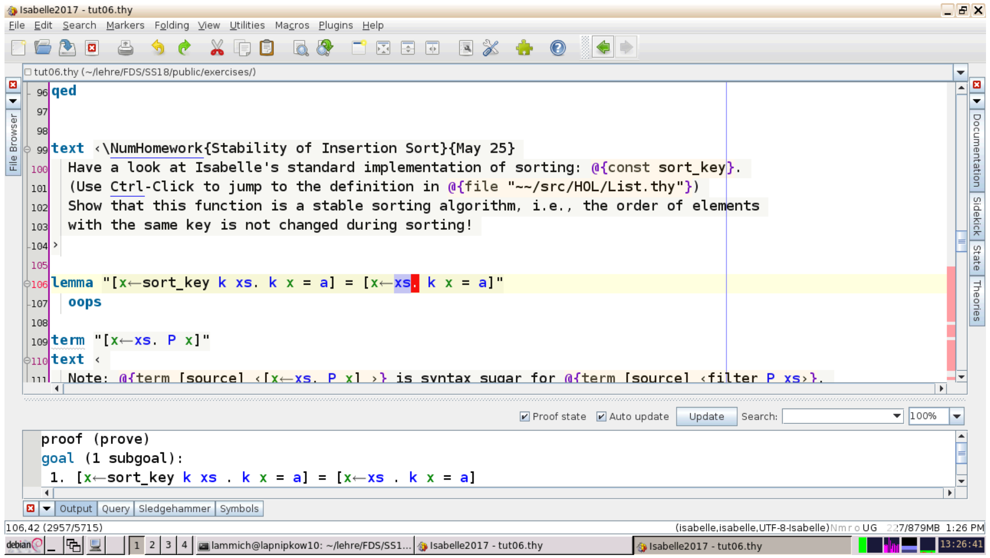
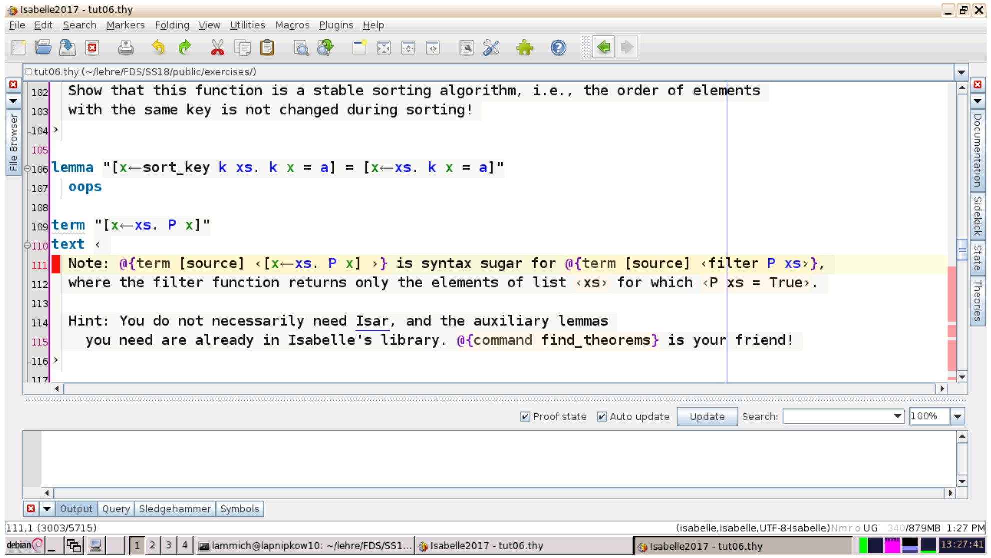

Isabelle2017 - tut06.thy (modified)
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
case (shorter p)
thm shorter.prems assms
show ?case proof cases
assume "distinct p" then show ?thesis using PATH oops by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where [simp]: "p=p1@x#p2@x#p3"
proof (prove)
using this:
distinct p
path G u p v
goal (1 subgoal):
1. ∃p'. path G u p' v ∧ distinct p'
Output Query Sledgehammer Symbols
Input/output complete

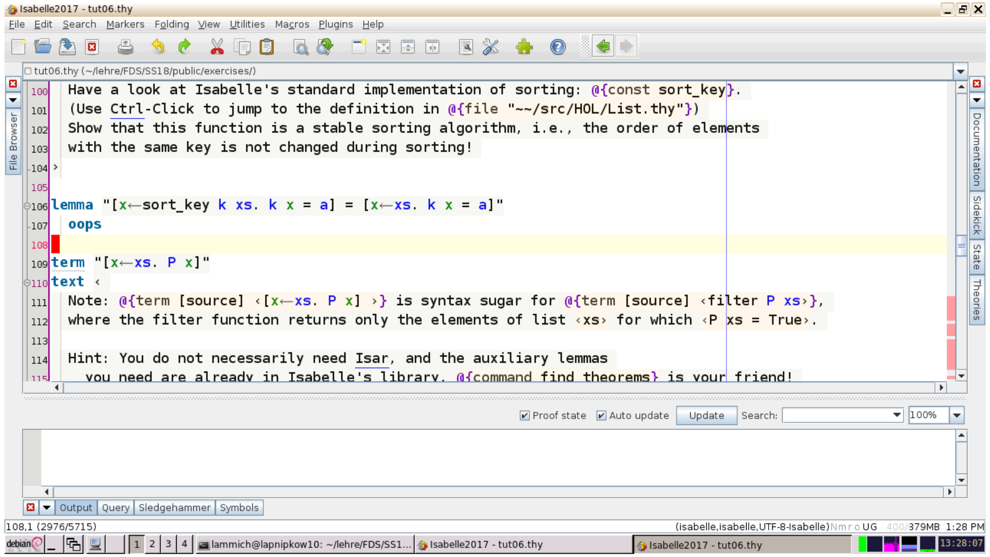

Isabelle2017 - tut06.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
proof (induction p rule: len_induct)
case (shorter p)
thm shorter.prems assms
show ?case proof cases
assume "distinct p" then show ?thesis using PATH oops by blast
next
assume "¬distinct p"
Proof state Auto update Update Search: 100%
proof (prove)
using this:
distinct p
path G u p v
goal (1 subgoal):
1. ∃p'. path G u p' v ∧ distinct p'
Output Query Sledgehammer Symbols
83,26 (2122/5749)
(isabelle,isabelle,UTF-8-Isabelle)

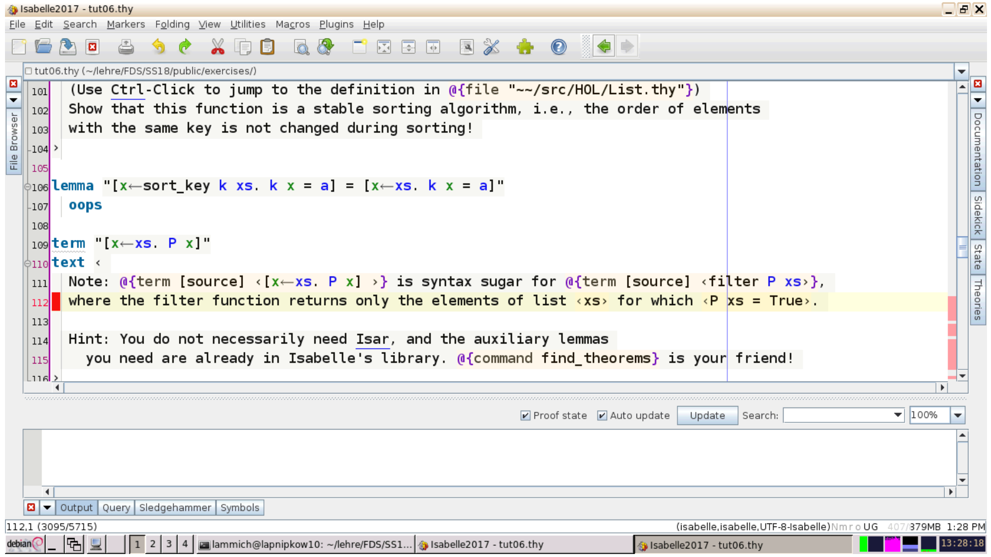

Isabelle2017 - tut06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
proof (induction p rule: len_induct)
case (shorter p)
show ?case proof cases
assume "distinct p" then show ?thesis using shorter.prems by blast
next
assume "¬distinct p"
from not_distinct_decomp[OF this] obtain p1 p2 p3 x where [simp]: "p=p1@x#p2@x#p3"
by auto
from shorter.prems obtain x1 x2 where
"path G u p1 x1" "G x1 x" "path G x p2 x2" "G x2 x" "path G x p3 v"
by auto
then have "path G u (p1@x#p3) v" by auto
from shorter.IH[OF _ this] show ?thesis by simp
Proof state Auto update Update Search: 100%
show ∃p'. path G u p' v ∧ distinct p'
Successful attempt to solve goal by exported rule:
(distinct p) ⟹ ∃p'. path G u p' v ∧ distinct p'
Output Query Sledgehammer Symbols
85,1 (2187/5715)
(isabelle,isabelle,UTF-8-Isabelle)

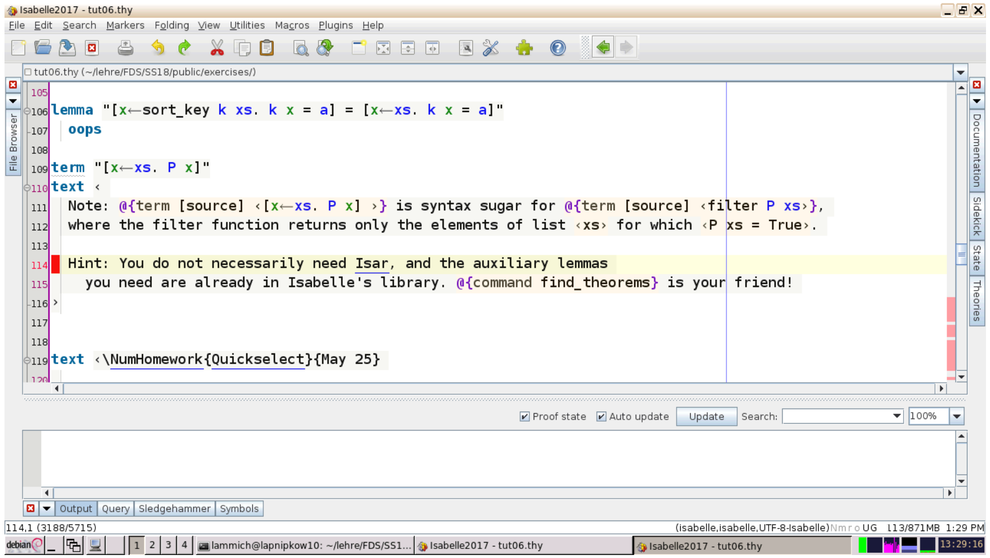

Isabelle2017 - tut06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
qed
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
Proof state Auto update Update Search: 100%
theorem exists_simple_path:
path ?G ?u ?p ?v ⟹ ∃p'. path ?G ?u p' ?v ∧ distinct p'
Output Query Sledgehammer Symbols
98,1 (2559/5715)
(isabelle,isabelle,UTF-8-Isabelle)

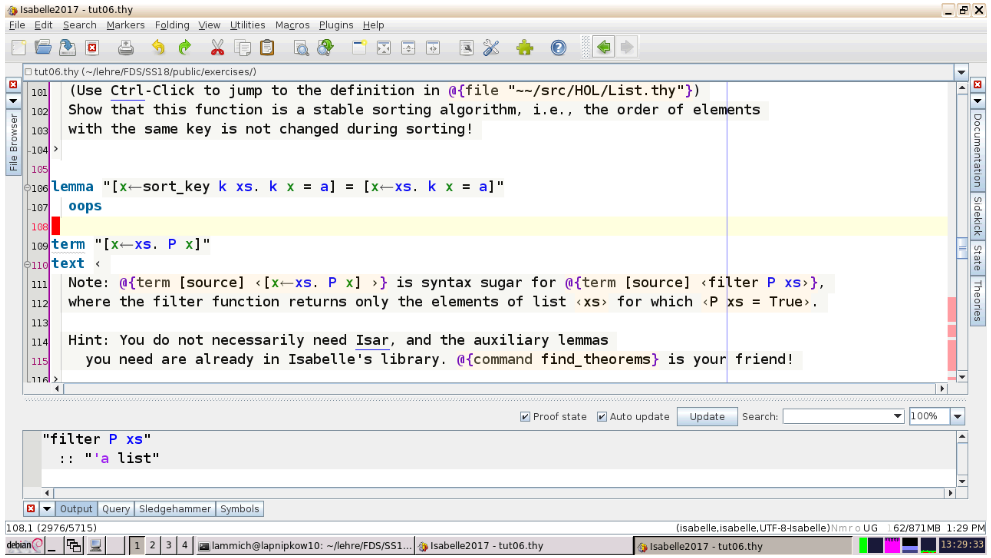

Isabelle2017 - List.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
List.thy ($ISABELLE_HOME/src/HOL/)
"sorted [x] ⟷ True"
by simp_all
primrec insort_key :: "('b ⇒ 'a) ⇒ 'b ⇒ 'b list ⇒ 'b list" where
"insort_key f x [] = [x]" |
"insort_key f x (y#ys) =
(if f x ≤ f y then (x#y#ys) else y#(insort_key f x ys))"
definition sort_key :: "('b ⇒ 'a) ⇒ 'b list ⇒ 'b list" where
"sort_key f xs = foldr (insort_key f) xs []"
definition insort_insert_key :: "('b ⇒ 'a) ⇒ 'b ⇒ 'b list ⇒ 'b list" where
"insort_insert_key f x xs =
(if f x ∈ f ` set xs then xs else insort_key f x xs)"
Proof state Auto update Update Search: 100%
Output Query Sledgehammer Symbols
375,33 (13058/236142)
(isabelle,isabelle,UTF-8-Isabelle)

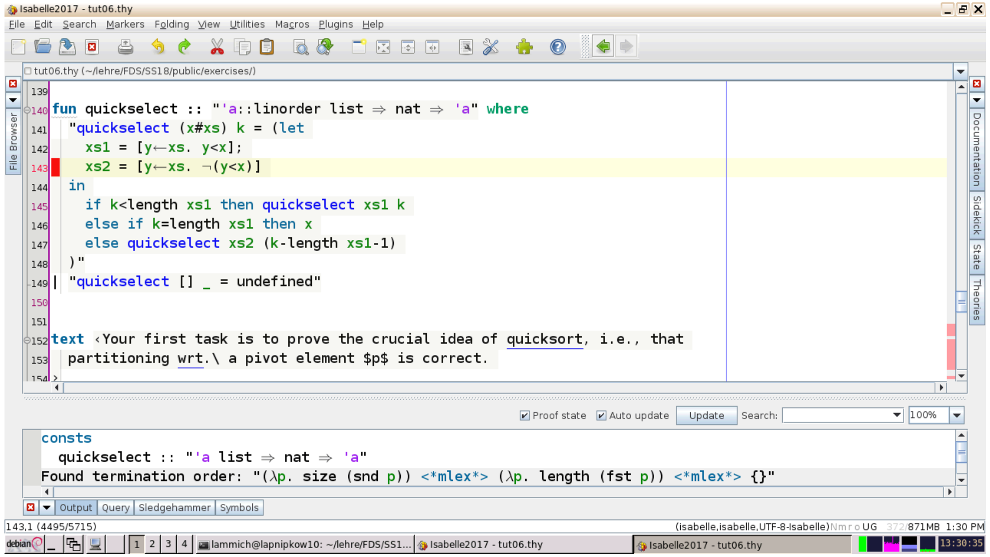
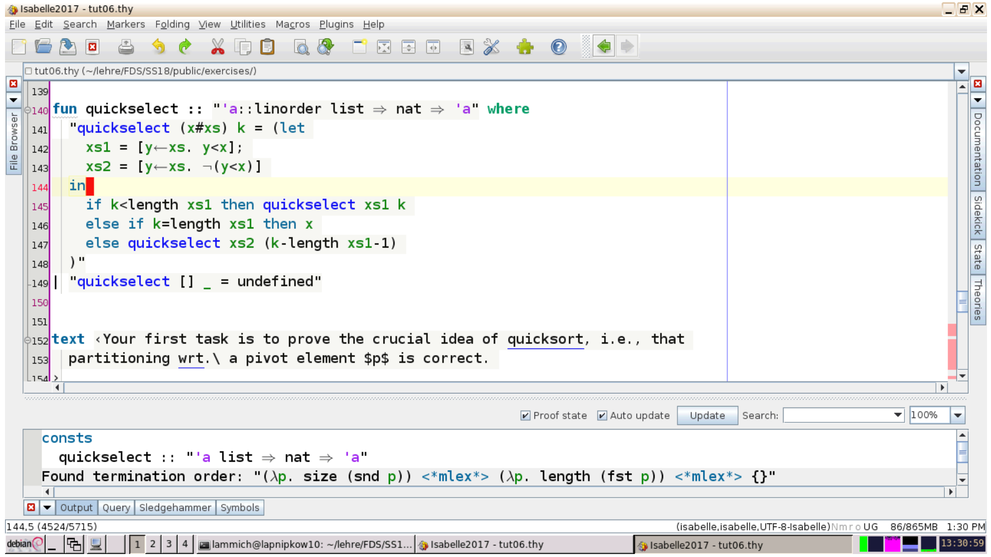
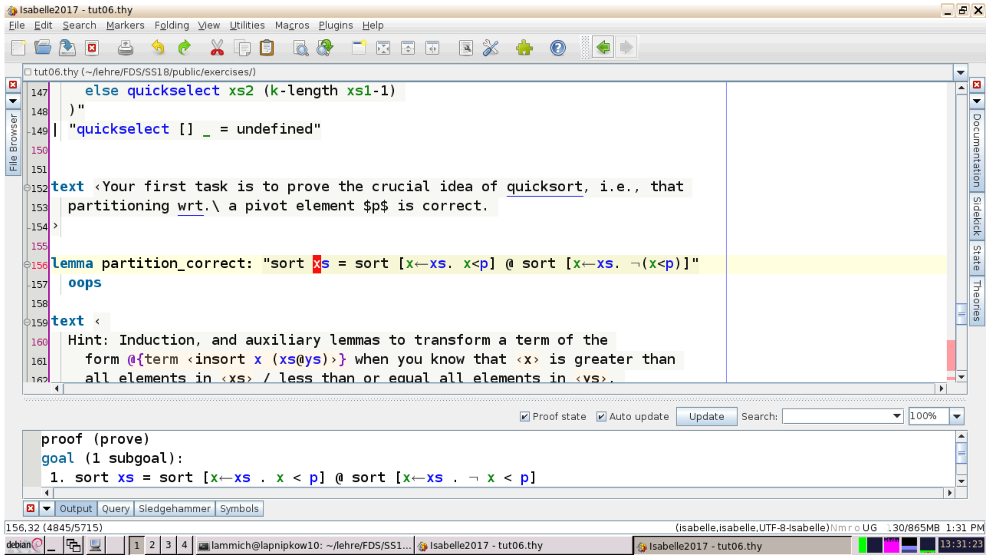
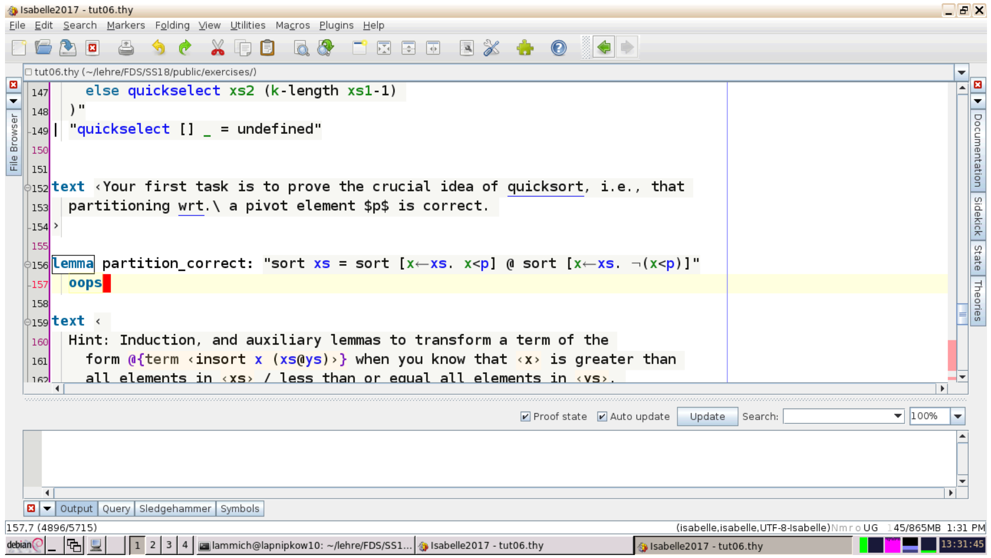

Isabelle2017 - tut06.thy
qed
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
proof (prove)
goal (1 subgoal):
1. [x←sort_key k xs . k x = a] = [x←xs . k x = a]

Isabelle2017 - tut06.thy
qed
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
null parsing complete, 0 error(s)

Isabelle2017 - tut06.thy
qed
text ‹\NumHomework{Stability of Insertion Sort}{May 25}
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
text ‹
proof (prove)
goal (1 subgoal):
1. [x←sort_key k xs . k x = a] = [x←xs . k x = a]

Isabelle2017 - tut06.thy
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
text ‹
Note: @{term [source] ‹[x←xs. P x] ›} is syntax sugar for @{term [source] ‹filter P xs›},
where the filter function returns only the elements of list ‹xs› for which ‹P xs = True›.
Hint: You do not necessarily need Isar, and the auxiliary lemmas
you need are already in Isabelle's library. @{command find_theorems} is your friend!
›

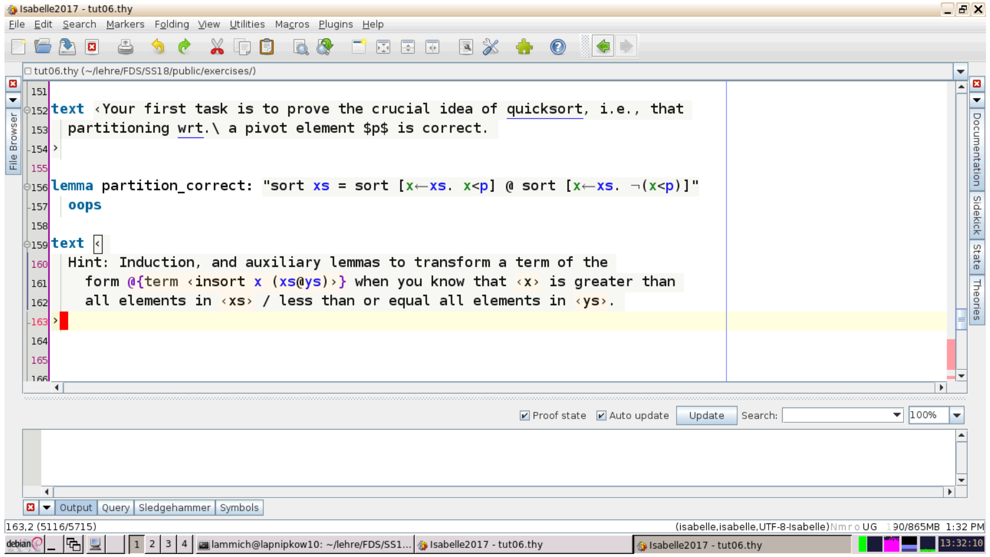

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
Have a look at Isabelle's standard implementation of sorting: @{const sort_key}.
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
text ‹
Note: @{term [source] ‹[x←xs. P x] ›} is syntax sugar for @{term [source] ‹filter P xs›},
where the filter function returns only the elements of list ‹xs› for which ‹P xs = True›.
Hint: You do not necessarily need Isar, and the auxiliary lemmas
you need are already in Isabelle's library. @{command find_theorems} is your friend!
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
108,1 (2976/5715)
(isabelle,isabelle,UTF-8-Isabelle)
1:28 PM

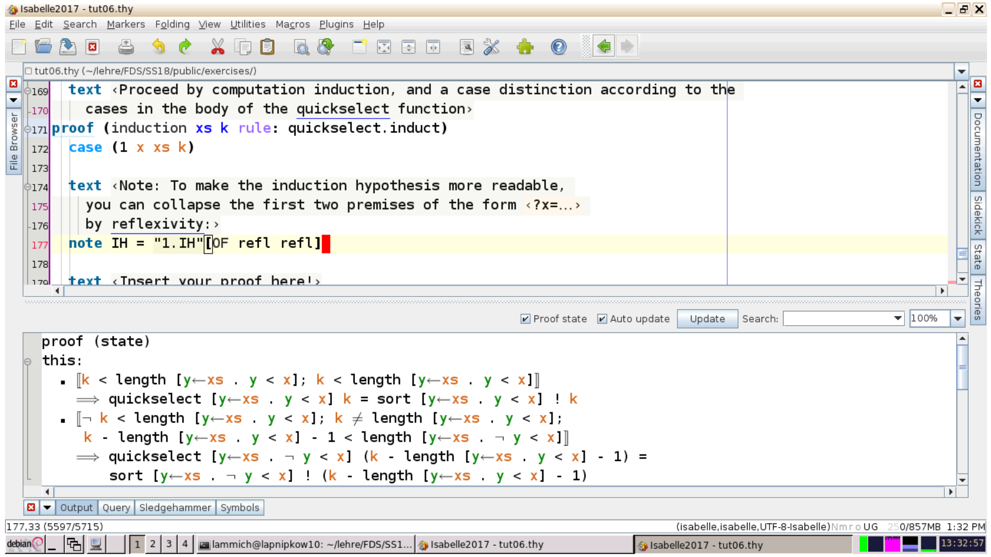

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
text ‹
Note: @{term [source] ‹[x←xs. P x] ›} is syntax sugar for @{term [source] ‹filter P xs›},
where the filter function returns only the elements of list ‹xs› for which ‹P xs = True›.
Hint: You do not necessarily need Isar, and the auxiliary lemmas
you need are already in Isabelle's library. @{command find_theorems} is your friend!
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
112,1 (3095/5715)
(isabelle,isabelle,UTF-8-Isabelle)
1:28 PM

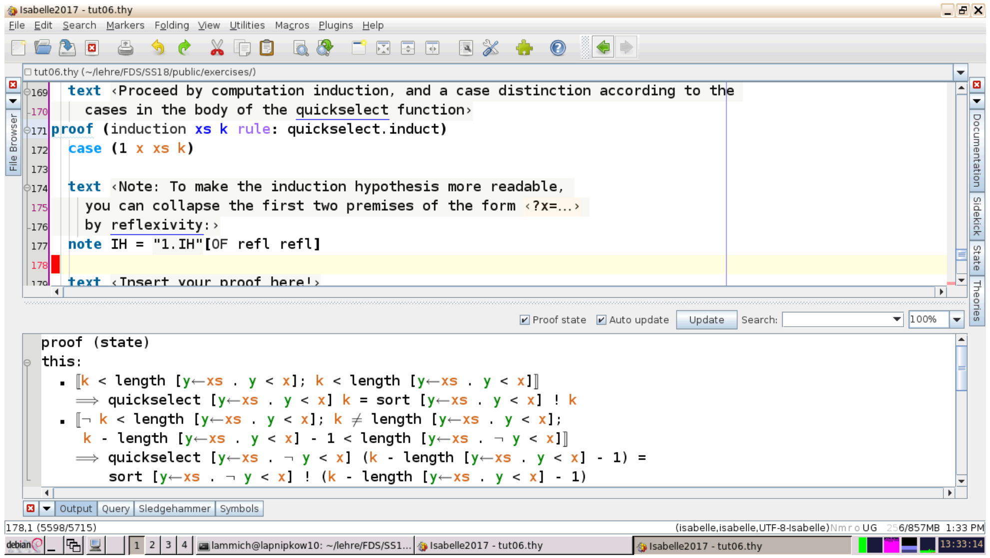

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
text ‹
Note: @{term [source] ‹[x←xs. P x] ›} is syntax sugar for @{term [source] ‹filter P xs›},
where the filter function returns only the elements of list ‹xs› for which ‹P xs = True›.
Hint: You do not necessarily need Isar, and the auxiliary lemmas
you need are already in Isabelle's library. @{command find_theorems} is your friend!
›
text ‹\NumHomework{Quickselect}{May 25}
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
114,1 (3188/5715)
(isabelle,isabelle,UTF-8-Isabelle)
1:29 PM

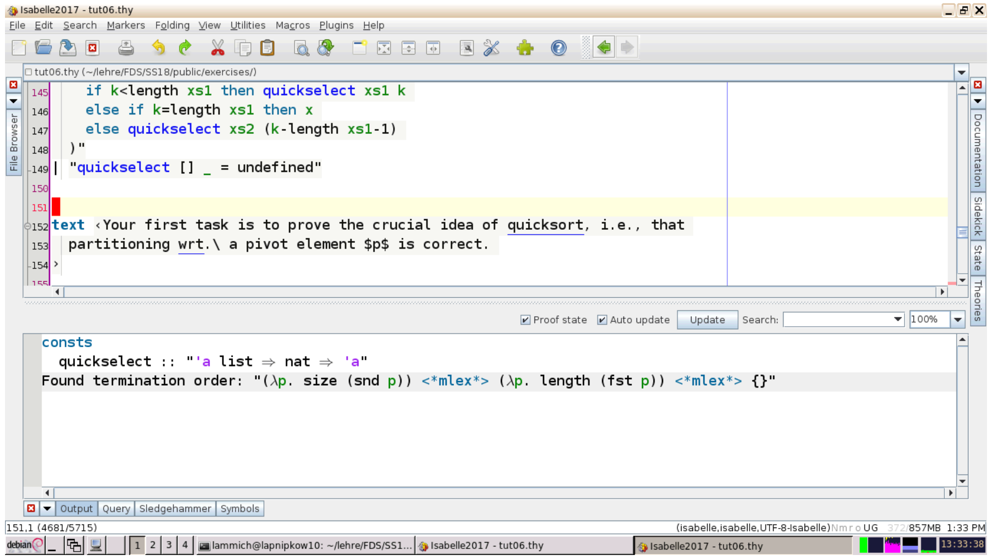

Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
(Use Ctrl-Click to jump to the definition in @{file "~~/src/HOL/List.thy"})
Show that this function is a stable sorting algorithm, i.e., the order of elements
with the same key is not changed during sorting!
›
lemma "[x←sort_key k xs. k x = a] = [x←xs. k x = a]"
oops
term "[x←xs. P x]"
text ‹
Note: @{term [source] ‹[x←xs. P x] ›} is syntax sugar for @{term [source] ‹filter P xs›},
where the filter function returns only the elements of list ‹xs› for which ‹P xs = True›.
Hint: You do not necessarily need Isar, and the auxiliary lemmas
you need are already in Isabelle's library. @{command find_theorems} is your friend!
Proof state
Auto update
Update
Search:
100%
"filter P xs"
:: "'a list"
Output
Query
Sledgehammer
Symbols
108,1 (2976/5715)
(isabelle,isabelle,UTF-8-Isabelle)
1:29 PM


Isabelle2017 - tut06.thy
fun quickselect :: "'a::linorder list ⇒ nat ⇒ 'a" where
"quickselect (x#xs) k = (let
xs1 = [y←xs. y<x];
xs2 = [y←xs. ¬(y<x)]
in
if k<length xs1 then quickselect xs1 k
else if k=length xs1 then x
else quickselect xs2 (k-length xs1-1)
)"
| "quickselect [] _ = undefined"
text ‹Your first task is to prove the crucial idea of quicksort, i.e., that
partitioning wrt.\ a pivot element p is correct.
consts
quickselect :: "'a list ⇒ nat ⇒ 'a"
Found termination order: "(λp. size (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"

Isabelle2017 - tut06.thy
fun quickselect :: "'a::linorder list ⇒ nat ⇒ 'a" where
"quickselect (x#xs) k = (let
xs1 = [y←xs. y<x];
xs2 = [y←xs. ¬(y<x)]
in
if k<length xs1 then quickselect xs1 k
else if k=length xs1 then x
else quickselect xs2 (k-length xs1-1)
)"
| "quickselect [] _ = undefined"
text ‹Your first task is to prove the crucial idea of quicksort, i.e., that
partitioning wrt.\ a pivot element p is correct.
consts
quickselect :: "'a list ⇒ nat ⇒ 'a"
Found termination order: "(λp. size (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"

Isabelle2017 - tut06.thy
else quickselect xs2 (k-length xs1-1)
)"
| "quickselect [] _ = undefined"
text ‹Your first task is to prove the crucial idea of quicksort, i.e., that
partitioning wrt.\ a pivot element p is correct.
›
lemma partition_correct: "sort xs = sort [x←xs. x<p] @ sort [x←xs. ¬(x<p)]"
oops
text ‹
Hint: Induction, and auxiliary lemmas to transform a term of the
form @{term ‹insort x (xs@ys)›} when you know that ‹x› is greater than
all elements in ‹xs› / less than or equal all elements in ‹ys›.
proof (prove)
goal (1 subgoal):
1. sort xs = sort [x←xs . x < p] @ sort [x←xs . ¬ x < p]

Isabelle2017 - tut06.thy
else quickselect xs2 (k-length xs1-1)
)"
| "quickselect [] _ = undefined"
text ‹Your first task is to prove the crucial idea of quicksort, i.e., that
partitioning wrt.\ a pivot element p is correct.
›
lemma partition_correct: "sort xs = sort [x←xs. x<p] @ sort [x←xs. ¬(x<p)]"
oops
text ‹
Hint: Induction, and auxiliary lemmas to transform a term of the
form @{term ‹insort x (xs@ys)›} when you know that ‹x› is greater than
all elements in ‹xs› / less than or equal all elements in ‹ys›.


Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Your first task is to prove the crucial idea of quicksort, i.e., that
partitioning wrt.\ a pivot element p is correct.
›
lemma partition_correct: "sort xs = sort [x←xs. x<p] @ sort [x←xs. ¬(x<p)]"
oops
text ‹
Hint: Induction, and auxiliary lemmas to transform a term of the
form @{term ‹insort x (xs@ys)›} when you know that ‹x› is greater than
all elements in ‹xs› / less than or equal all elements in ‹ys›.
›
Proof state Auto update Update Search: 100%
Output Query Sledgehammer Symbols
163.2 (5116/5715) (isabelle,isabelle,UTF-8-Isabelle) 1:32 PM


Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Proceed by computation induction, and a case distinction according to the
cases in the body of the quickselect function›
proof (induction xs k rule: quickselect.induct)
case (1 x xs k)
text ‹Note: To make the induction hypothesis more readable,
you can collapse the first two premises of the form ‹?x=...›
by reflexivity:›
note IH = "1.IH"[OF refl refl]
text ‹Insert your proof here!›
Proof state Auto update Update Search: 100%
proof (state)
this:
⟦k < length [y←xs . y < x]; k < length [y←xs . y < x]⟧
⟹ quickselect [y←xs . y < x] k = sort [y←xs . y < x] ! k
⟦¬ k < length [y←xs . y < x]; k ≠ length [y←xs . y < x];
k - length [y←xs . y < x] - 1 < length [y←xs . ¬ y < x]⟧
⟹ quickselect [y←xs . ¬ y < x] (k - length [y←xs . y < x] - 1) =
sort [y←xs . ¬ y < x] ! (k - length [y←xs . y < x] - 1)
Output Query Sledgehammer Symbols
177.33 (5597/5715) (isabelle,isabelle,UTF-8-Isabelle) 1:32 PM


Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Proceed by computation induction, and a case distinction according to the
cases in the body of the quickselect function›
proof (induction xs k rule: quickselect.induct)
case (1 x xs k)
text ‹Note: To make the induction hypothesis more readable,
you can collapse the first two premises of the form ‹?x=...›
by reflexivity:›
note IH = "1.IH"[OF refl refl]
text ‹Insert your proof here!›
Proof state Auto update Update Search: 100%
proof (state)
this:
⟦k < length [y←xs . y < x]; k < length [y←xs . y < x]⟧
⟹ quickselect [y←xs . y < x] k = sort [y←xs . y < x] ! k
⟦¬ k < length [y←xs . y < x]; k ≠ length [y←xs . y < x];
k - length [y←xs . y < x] - 1 < length [y←xs . ¬ y < x]⟧
⟹ quickselect [y←xs . ¬ y < x] (k - length [y←xs . y < x] - 1) =
sort [y←xs . ¬ y < x] ! (k - length [y←xs . y < x] - 1)
Output Query Sledgehammer Symbols
178.1 (5598/5715) (isabelle,isabelle,UTF-8-Isabelle) 1:33 PM


Isabelle2017 - tut06.thy
tut06.thy (~/lehre/FDS/SS18/public/exercises/)
if k<length xs1 then quickselect xs1 k
else if k=length xs1 then x
else quickselect xs2 (k-length xs1-1)
)"
| "quickselect [] _ = undefined"
text ‹Your first task is to prove the crucial idea of quicksort, i.e., that
partitioning wrt.\ a pivot element p is correct.
›
Proof state Auto update Update Search: 100%
consts
quickselect :: "'a list ⇒ nat ⇒ 'a"
Found termination order: "(λp. size (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"
Output Query Sledgehammer Symbols
151.1 (4681/5715) (isabelle,isabelle,UTF-8-Isabelle) 1:33 PM