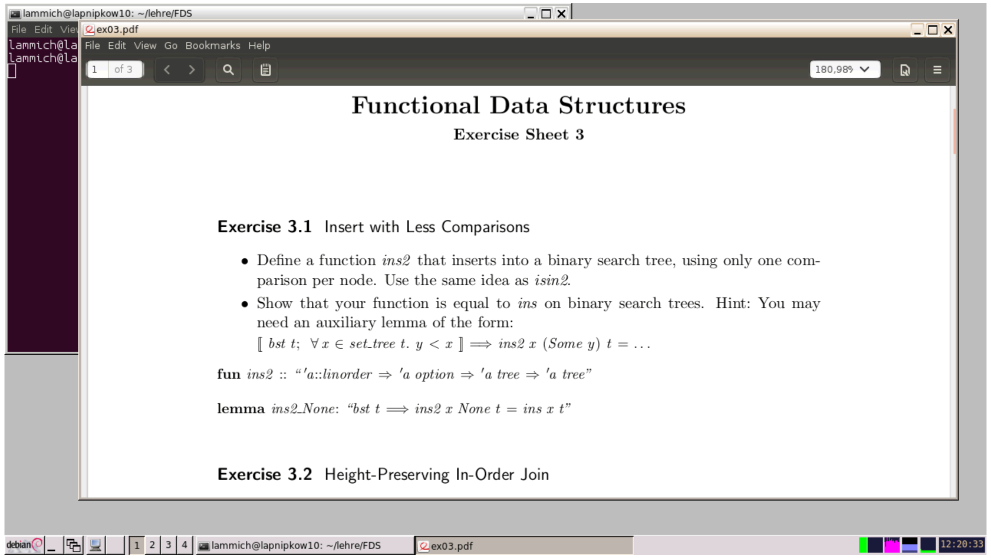
Script **generated by TTT**

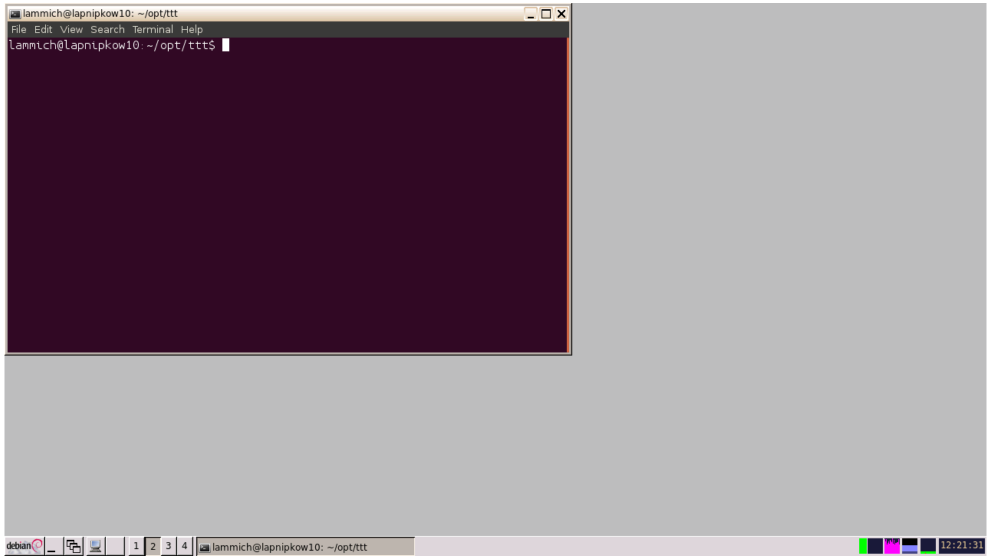
Title: Lammich: FDS Tutorial (12.05.2017)

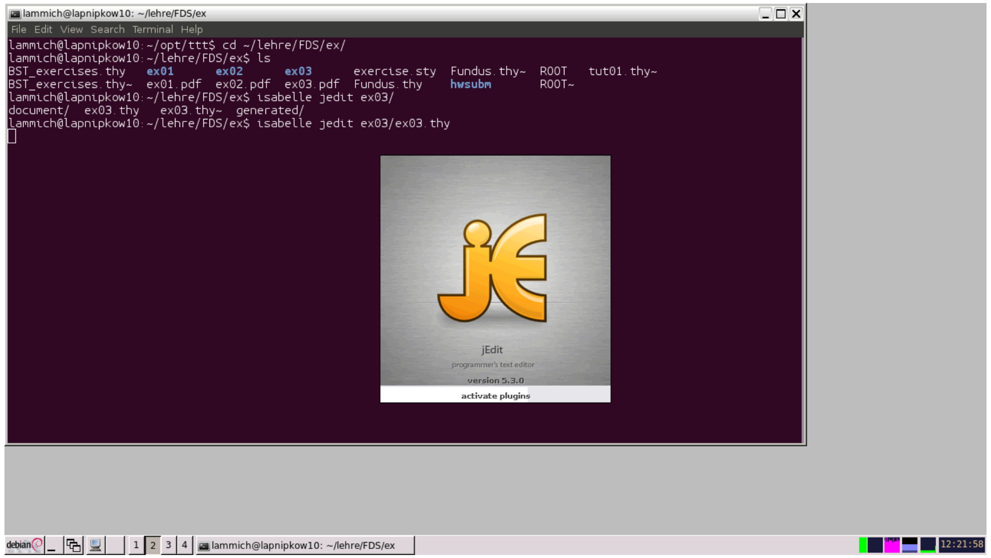
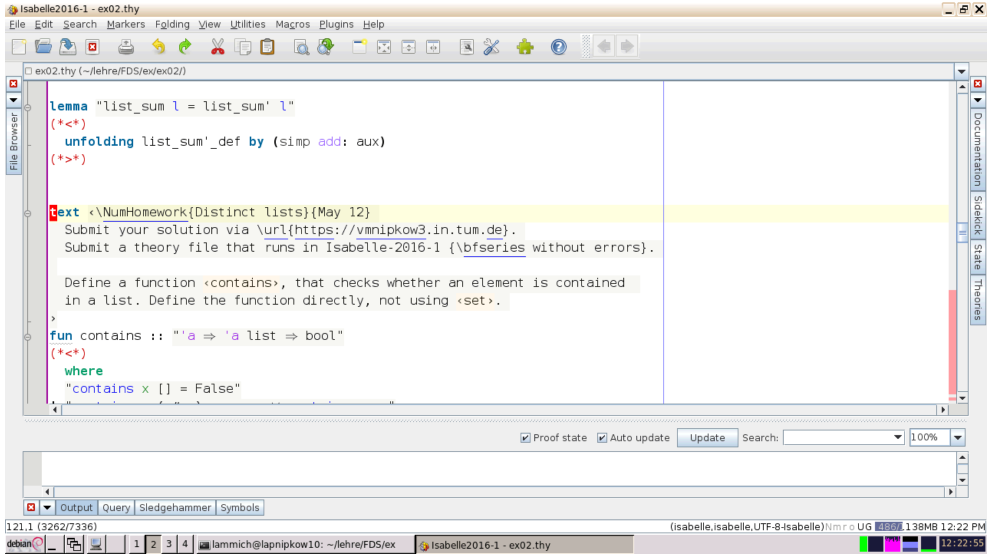
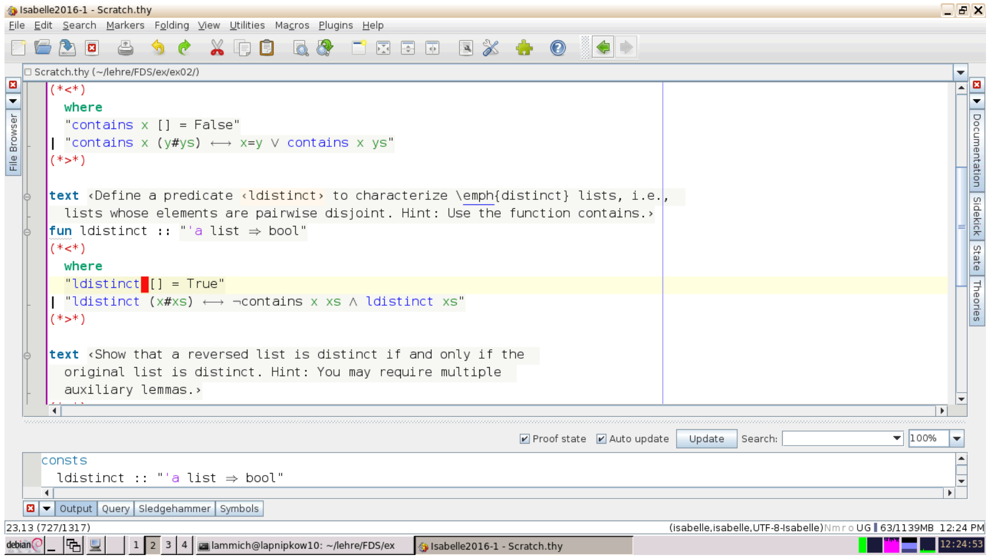
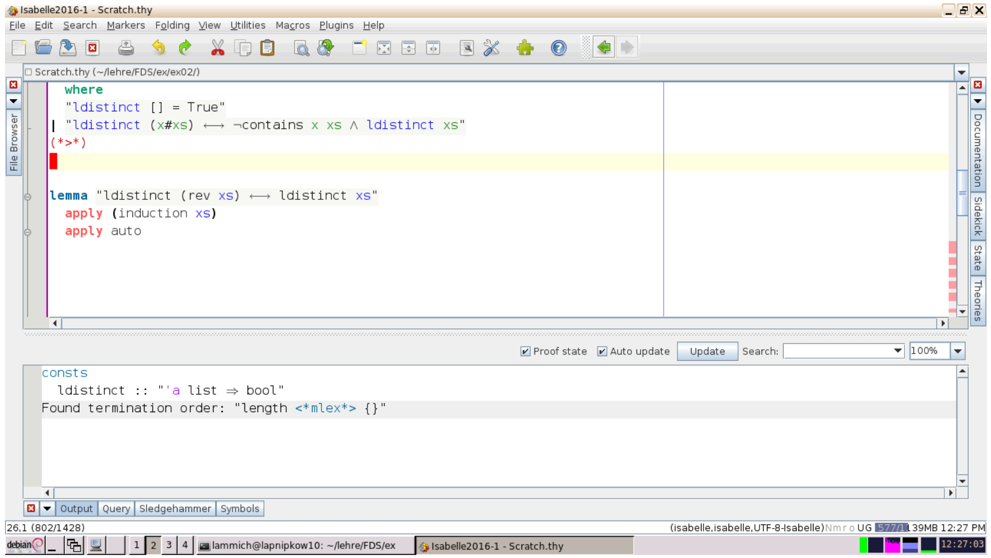
Date: Fri May 12 12:20:33 CEST 2017

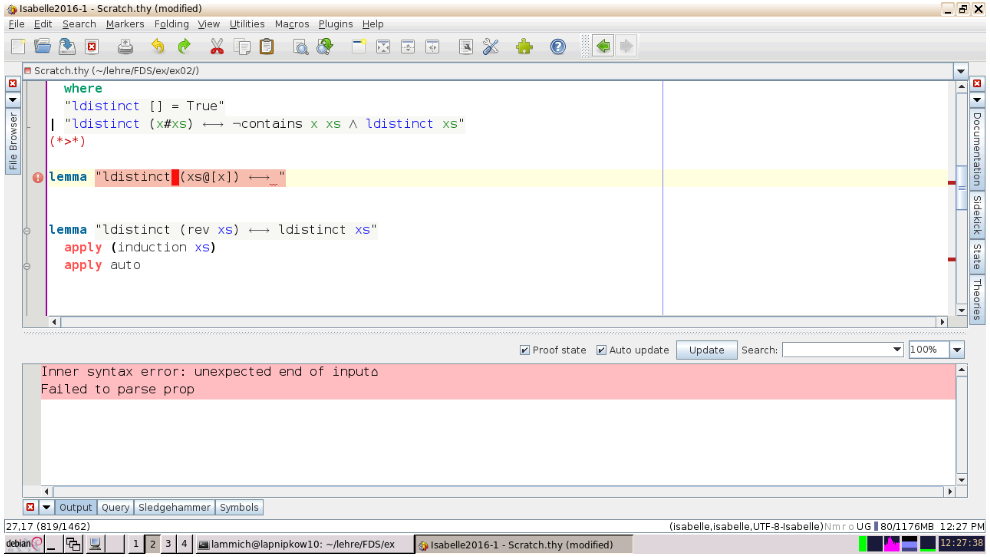
Duration: 97:27 min

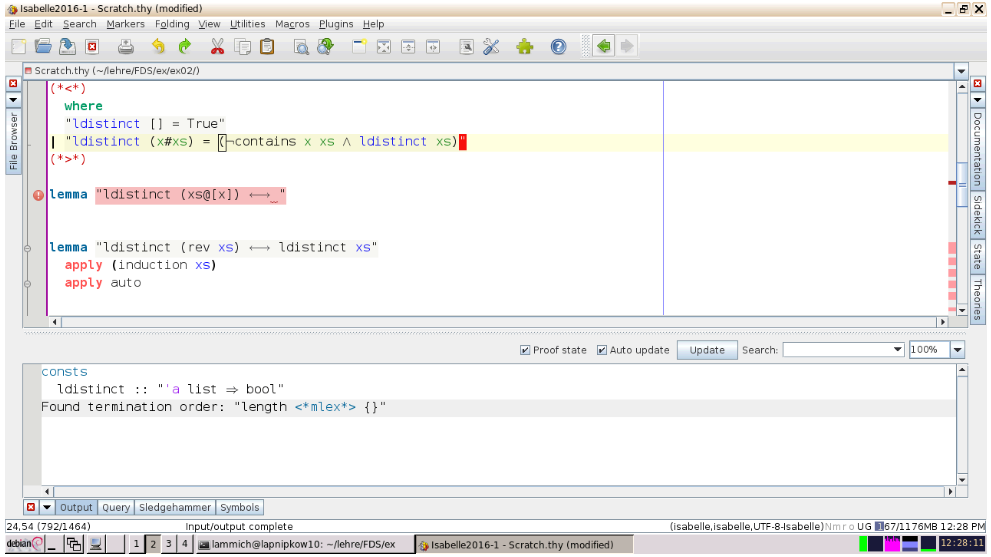
Pages: 93

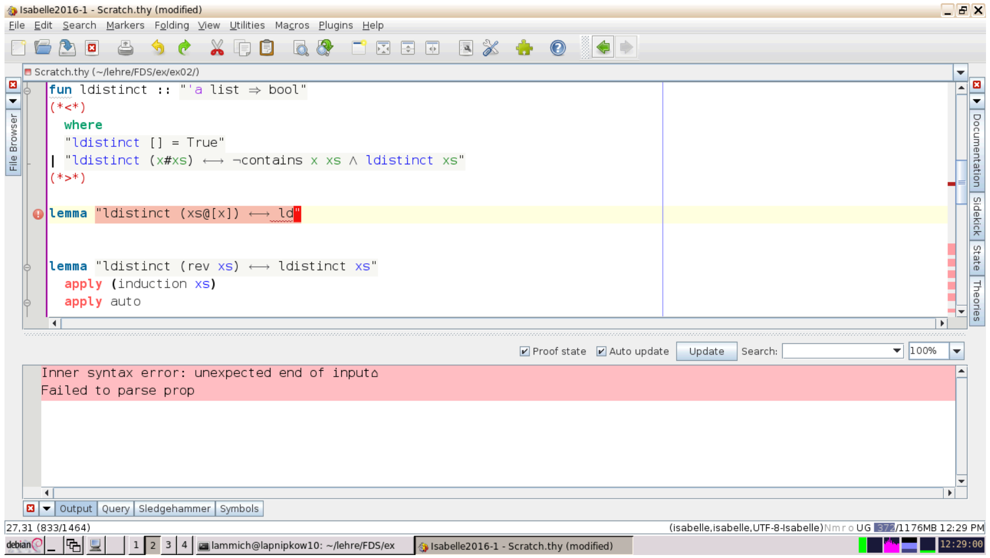


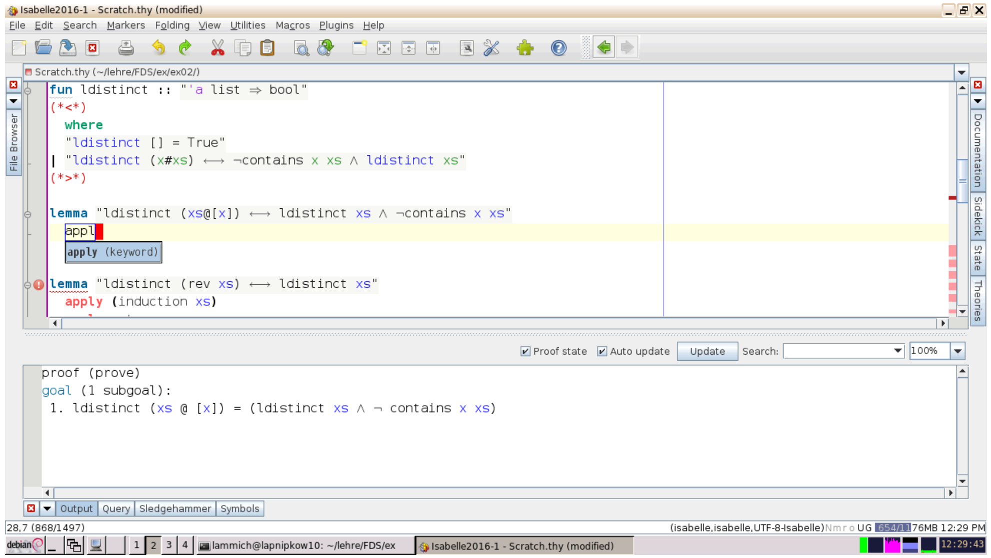


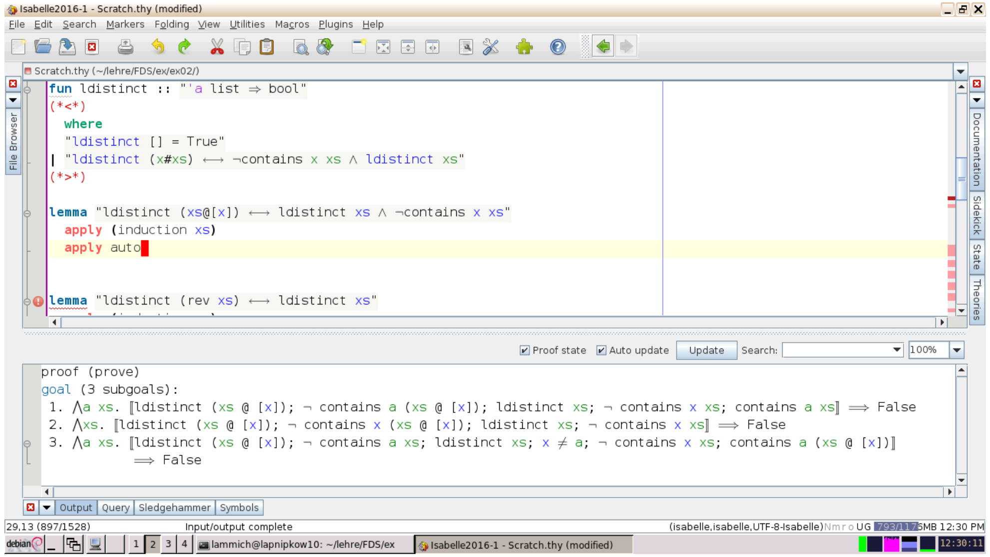

lammich@lapnipkow10: ~/lehre/FDS/ex
File Edit View Search Terminal Help
lammich@lapnipkow10:~/opt/ttt$ cd ~/lehre/FDS/ex/
lammich@lapnipkow10:~/lehre/FDS/ex$ ls
BST_exercises.thy ex01 ex02 ex03 exercise.sty Fundus.thy~ ROOT tut01.thy~
BST_exercises.thy~ ex01.pdf ex02.pdf ex03.pdf Fundus.thy hwsubm ROOT~
lammich@lapnipkow10:~/lehre/FDS/ex$ isabelle jedit ex03/
document/ ex03.thy ex03.thy~ generated/
lammich@lapnipkow10:~/lehre/FDS/ex$ isabelle jedit ex03/ex03.thy
jEdit
version 5.3.0
activate plugins
12:21:58

Isabelle2016-1 - ex02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
ex02.thy (~/lehre/FDS/ex/ex02/)
lemma "list_sum l = list_sum' l"
(*<*)
unfolding list_sum'_def by (simp add: aux)
(*>*)
text ‹\NumHomework{Distinct lists}{May 12}
Submit your solution via \url{https://vmnipkow3.in.tum.de}.
Submit a theory file that runs in Isabelle-2016-1 {\bfseries without errors}.
Define a function ‹contains›, that checks whether an element is contained
in a list. Define the function directly, not using ‹set›.
›
fun contains :: "'a ⇒ 'a list ⇒ bool"
(*<*)
where
"contains x [] = False"
Proof state Auto update Update Search: 100%
Output Query Sledgehammer Symbols
121,1 (3262/7336)
12:22:55

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex/ex02/)
(*<*)
where
"contains x [] = False"
| "contains x (y#ys) ⟷ x=y ∨ contains x ys"
(*>*)
text ‹Define a predicate ‹ldistinct› to characterize \emph{distinct} lists, i.e.,
lists whose elements are pairwise disjoint. Hint: Use the function contains.›
fun ldistinct :: "'a list ⇒ bool"
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
text ‹Show that a reversed list is distinct if and only if the
original list is distinct. Hint: You may require multiple
auxiliary lemmas.›
consts
ldistinct :: "'a list ⇒ bool"
Output Query Sledgehammer Symbols
23,13 (727/1317)
12:24:53

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex/ex02/)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
consts
ldistinct :: "'a list ⇒ bool"
Found termination order: "length <*mlex*> {}"
Output Query Sledgehammer Symbols
26,1 (802/1428)
12:27:03

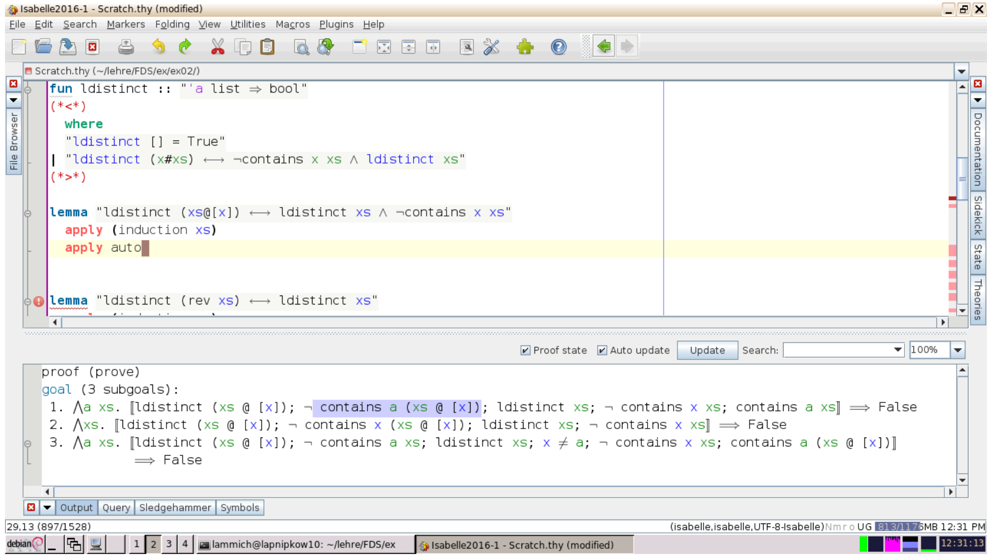

Isabelle2016-1 - Scratch.thy (modified)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma "ldistinct (xs@[x]) ⟷ _"
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
Inner syntax error: unexpected end of input
Failed to parse prop

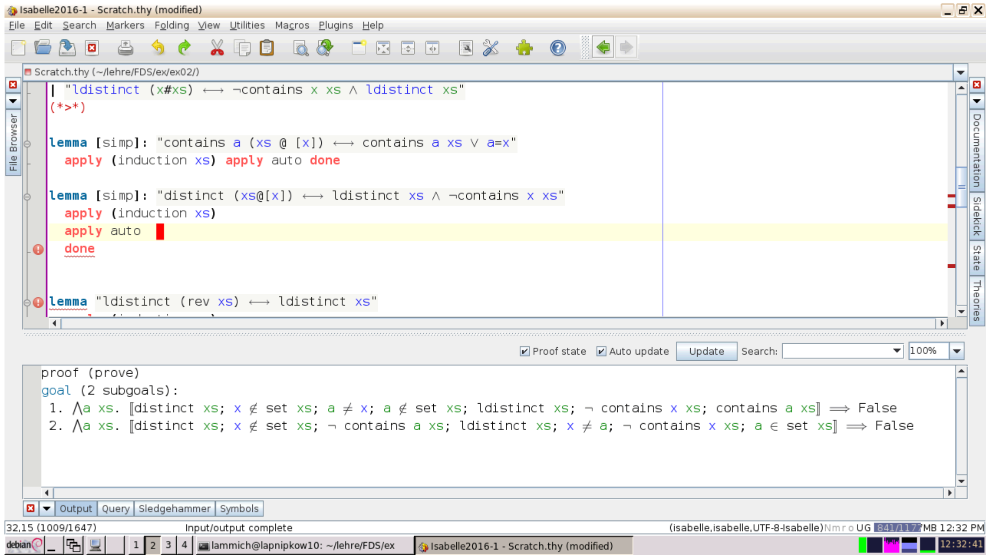

Isabelle2016-1 - Scratch.thy (modified)
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) = (¬contains x xs ∧ ldistinct xs)"
(*>*)
lemma "ldistinct (xs@[x]) ⟷ _"
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
consts
ldistinct :: "'a list ⇒ bool"
Found termination order: "length <*mlex*> {}"

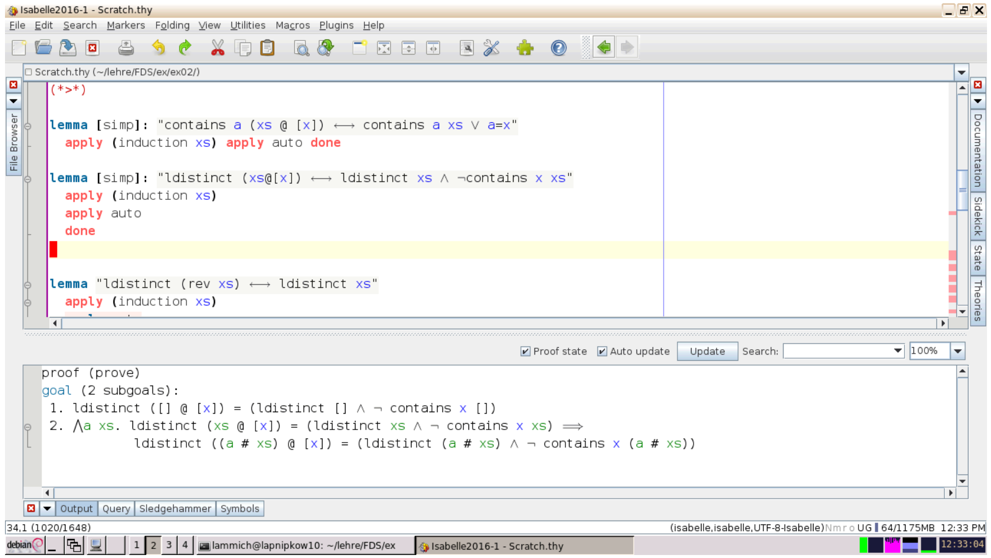

Isabelle2016-1 - Scratch.thy (modified)
fun ldistinct :: "'a list ⇒ bool"
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma "ldistinct (xs@[x]) ⟷ ld"
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
Inner syntax error: unexpected end of input
Failed to parse prop

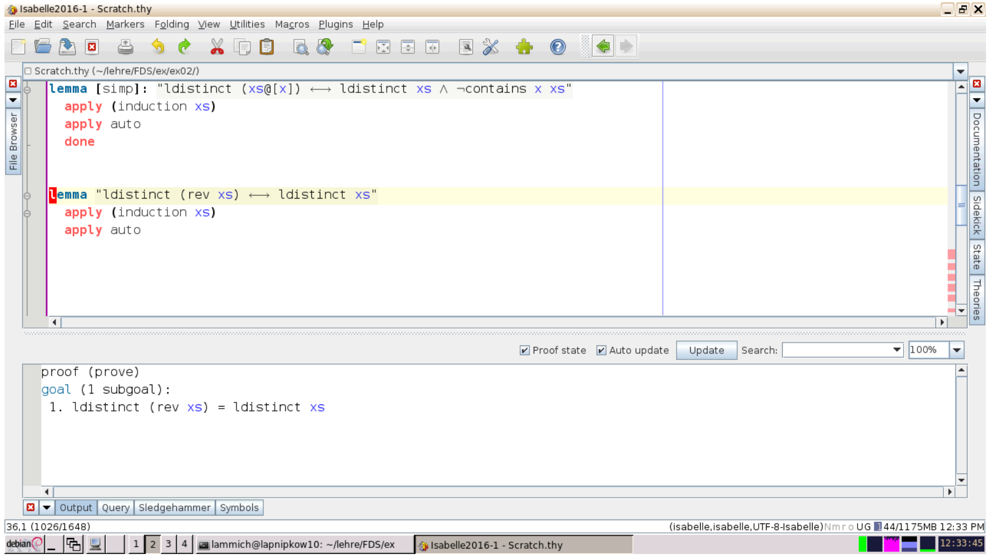

Isabelle2016-1 - Scratch.thy (modified)
fun ldistinct :: "'a list ⇒ bool"
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
appl
apply (keyword)
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
proof (prove)
goal (1 subgoal):
1. ldistinct (xs @ [x]) = (ldistinct xs ∧ ¬ contains x xs)

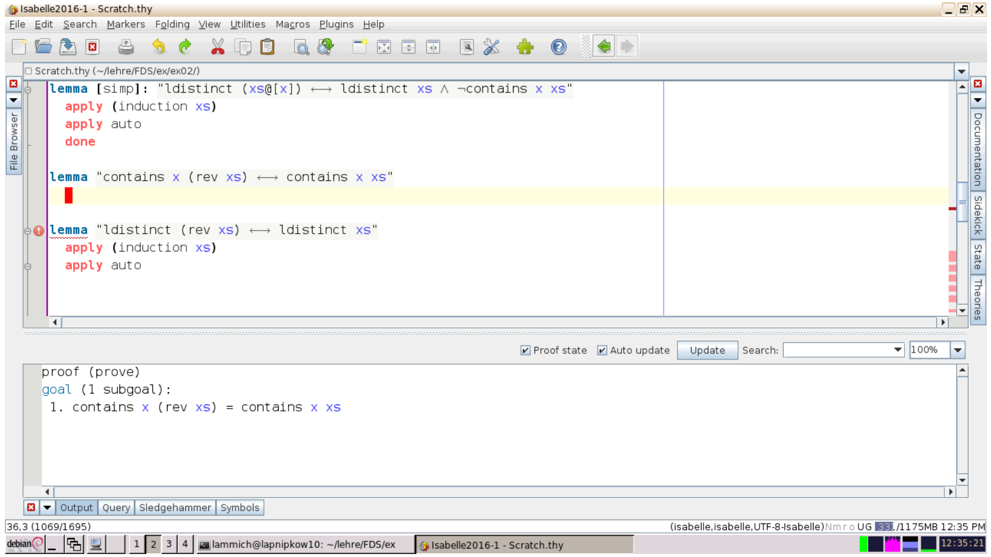

Isabelle2016-1 - Scratch.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
Scratch.thy (~/lehre/FDS/ex/ex02/)
fun ldistinct :: "'a list ⇒ bool"
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
Proof state Auto update Update Search: 100%
proof (prove)
goal (3 subgoals):
1. ⋀a xs. ⟦ldistinct (xs @ [x]); ¬ contains a (xs @ [x]); ldistinct xs; ¬ contains x xs; contains a xs⟧ ⟹ False
2. ⋀xs. ⟦ldistinct (xs @ [x]); ¬ contains x (xs @ [x]); ldistinct xs; ¬ contains x xs⟧ ⟹ False
3. ⋀a xs. ⟦ldistinct (xs @ [x]); ¬ contains a xs; ldistinct xs; x ≠ a; ¬ contains x xs; contains a (xs @ [x])⟧ ⟹ False
Output Query Sledgehammer Symbols
29,13 (897/1528) Input/output complete (isabelle,isabelle,UTF-8-Isabelle) 12:30 PM

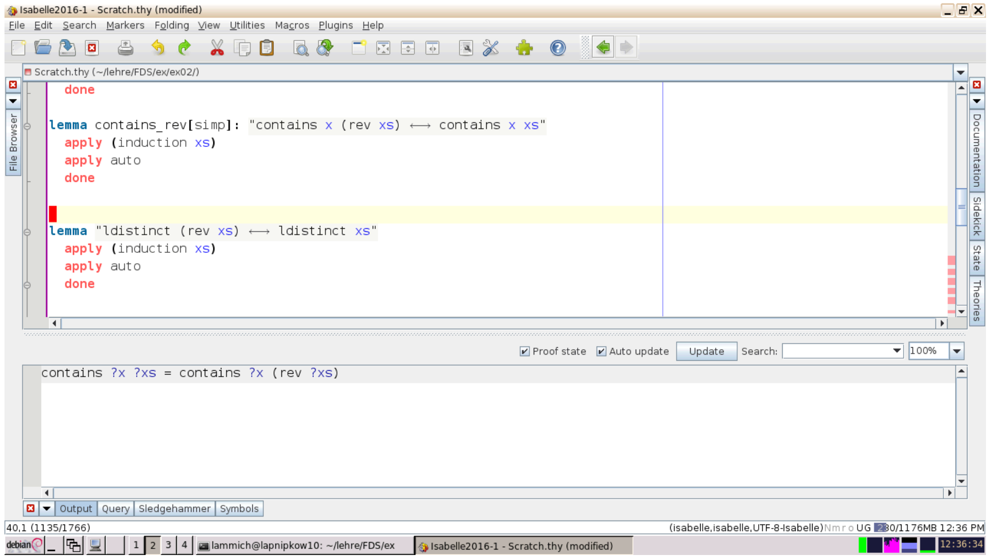

Isabelle2016-1 - Scratch.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
Scratch.thy (~/lehre/FDS/ex/ex02/)
fun ldistinct :: "'a list ⇒ bool"
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
Proof state Auto update Update Search: 100%
proof (prove)
goal (3 subgoals):
1. ⋀a xs. ⟦ldistinct (xs @ [x]); ¬ contains a (xs @ [x]); ldistinct xs; ¬ contains x xs; contains a xs⟧ ⟹ False
2. ⋀xs. ⟦ldistinct (xs @ [x]); ¬ contains x (xs @ [x]); ldistinct xs; ¬ contains x xs⟧ ⟹ False
3. ⋀a xs. ⟦ldistinct (xs @ [x]); ¬ contains a xs; ldistinct xs; x ≠ a; ¬ contains x xs; contains a (xs @ [x])⟧ ⟹ False
Output Query Sledgehammer Symbols
29,13 (897/1528) (isabelle,isabelle,UTF-8-Isabelle) 12:31 PM

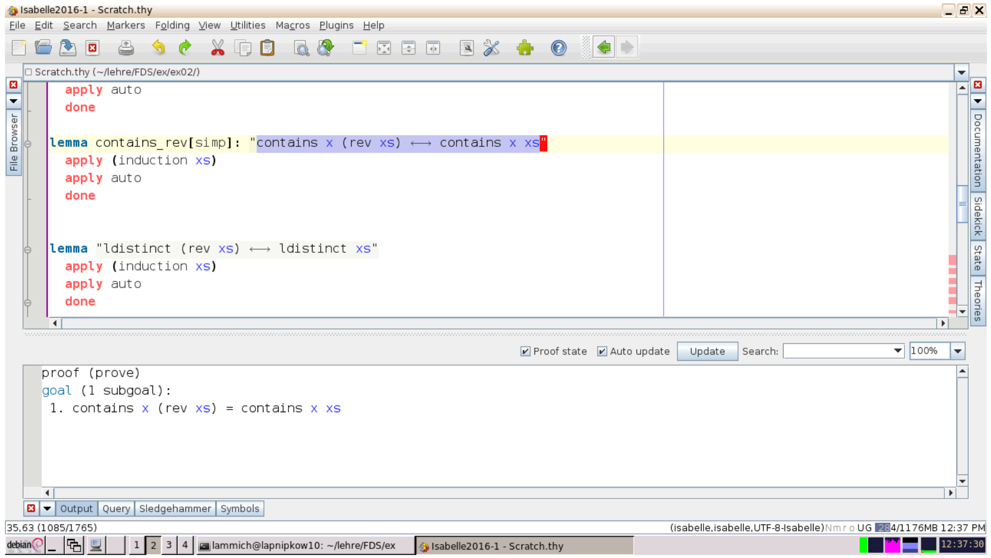

Isabelle2016-1 - Scratch.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
Scratch.thy (~/lehre/FDS/ex/ex02/)
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma [simp]: "contains a (xs @ [x]) ⟷ contains a xs ∨ a=x"
apply (induction xs) apply auto done
lemma [simp]: "distinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
Proof state Auto update Update Search: 100%
proof (prove)
goal (2 subgoals):
1. ⋀a xs. ⟦distinct xs; x ∉ set xs; a ≠ x; a ∉ set xs; ldistinct xs; ¬ contains x xs; contains a xs⟧ ⟹ False
2. ⋀a xs. ⟦distinct xs; x ∉ set xs; ¬ contains a xs; ldistinct xs; x ≠ a; ¬ contains x xs; a ∈ set xs⟧ ⟹ False
Output Query Sledgehammer Symbols
32,15 (1009/1647) Input/output complete (isabelle,isabelle,UTF-8-Isabelle) 12:32 PM

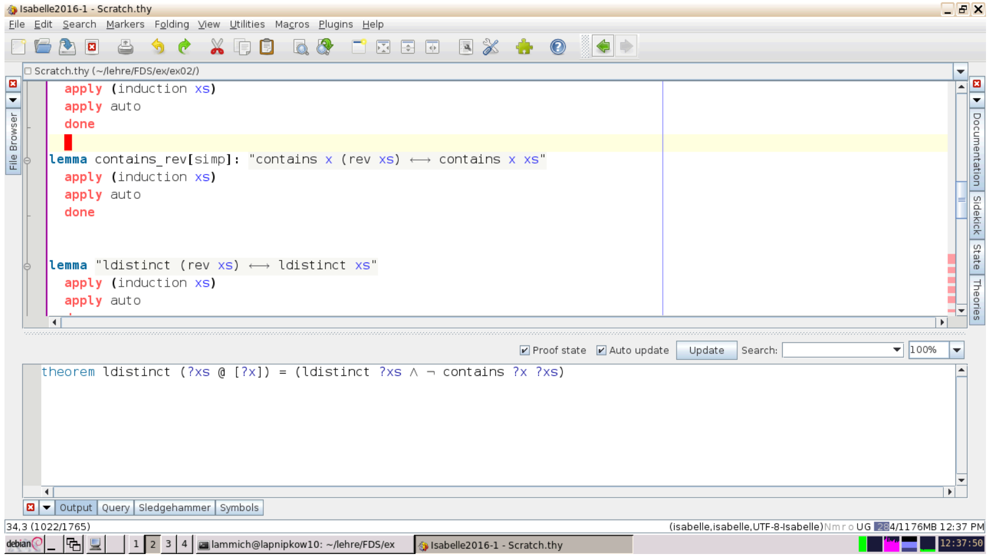

Isabelle2016-1 - Scratch.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Scratch.thy (~/lehre/FDS/ex/ex02/)
(*>*)
lemma [simp]: "contains a (xs @ [x]) ⟷ contains a xs ∨ a=x"
apply (induction xs) apply auto done
lemma [simp]: "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
Proof state Auto update Update Search: 100%
proof (prove)
goal (2 subgoals):
1. ldistinct ([] @ [x]) = (ldistinct [] ∧ ¬ contains x [])
2. ⋀a xs. ldistinct (xs @ [x]) = (ldistinct xs ∧ ¬ contains x xs) ⟹
ldistinct ((a # xs) @ [x]) = (ldistinct (a # xs) ∧ ¬ contains x (a # xs))
Output Query Sledgehammer Symbols
34,1 (1020/1648) (isabelle,isabelle,UTF-8-Isabelle) 64/1175MB 12:33 PM

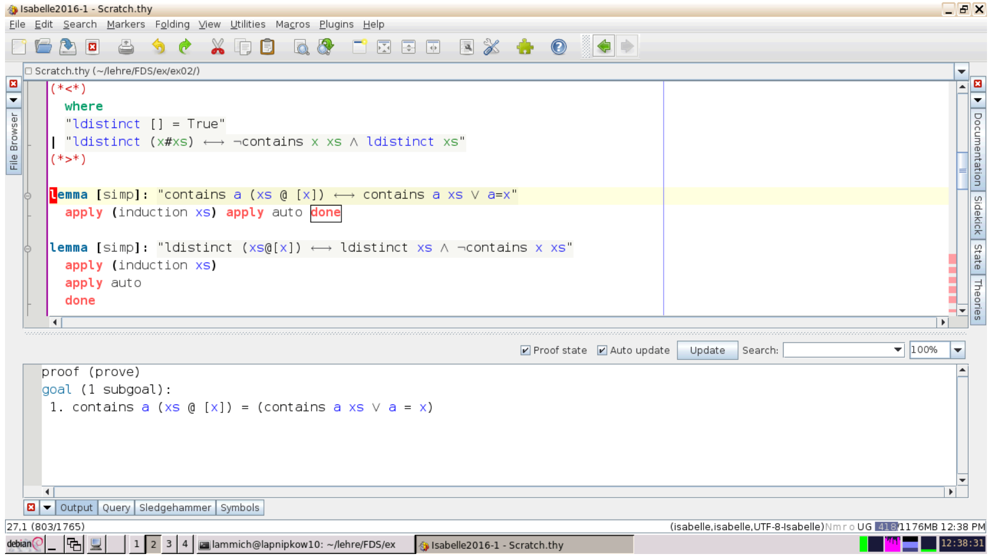

Isabelle2016-1 - Scratch.thy
lemma [simp]: "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs

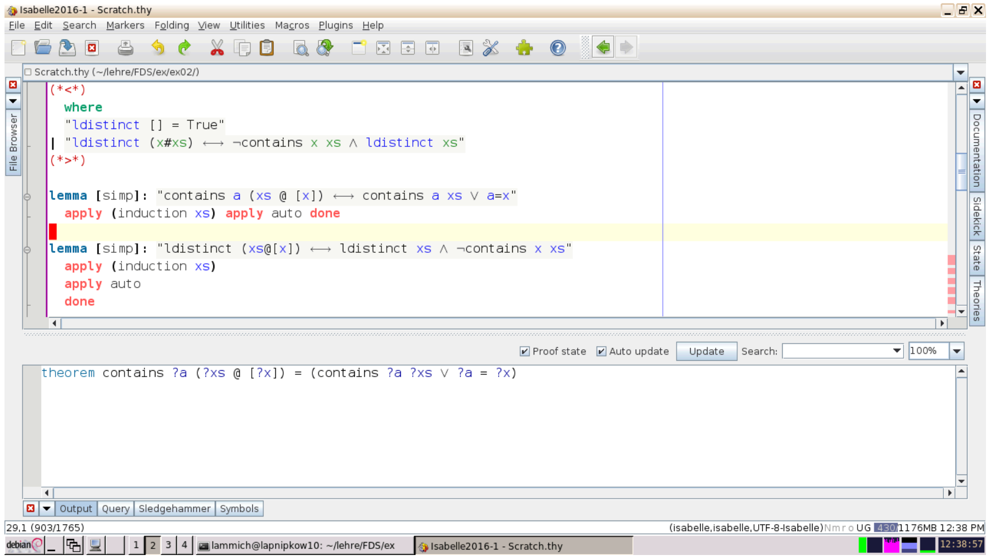

Isabelle2016-1 - Scratch.thy
lemma [simp]: "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
done
lemma "contains x (rev xs) ⟷ contains x xs"
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
proof (prove)
goal (1 subgoal):
1. contains x (rev xs) = contains x xs

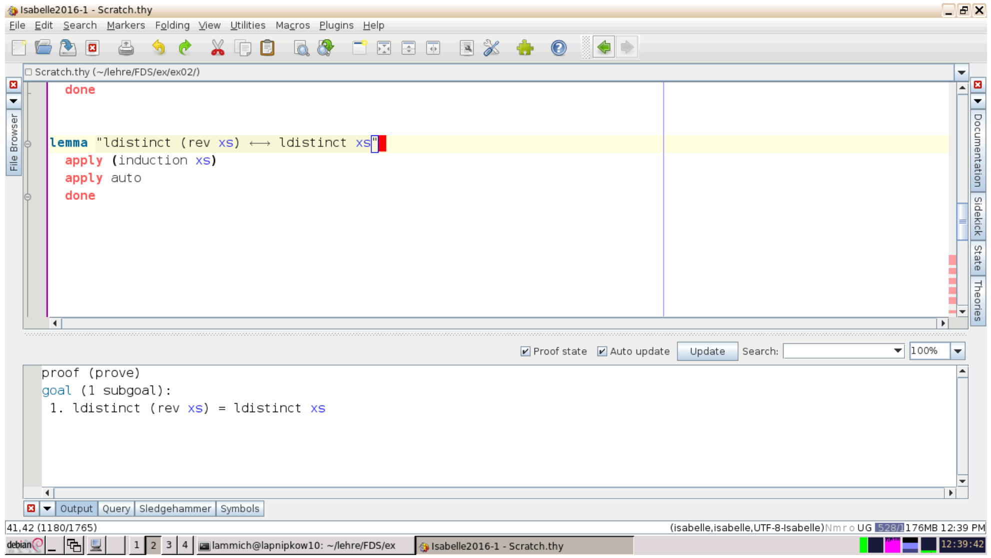

Isabelle2016-1 - Scratch.thy (modified)
done
lemma contains_rev[simp]: "contains x (rev xs) ⟷ contains x xs"
apply (induction xs)
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
done
contains ?x ?xs = contains ?x (rev ?xs)

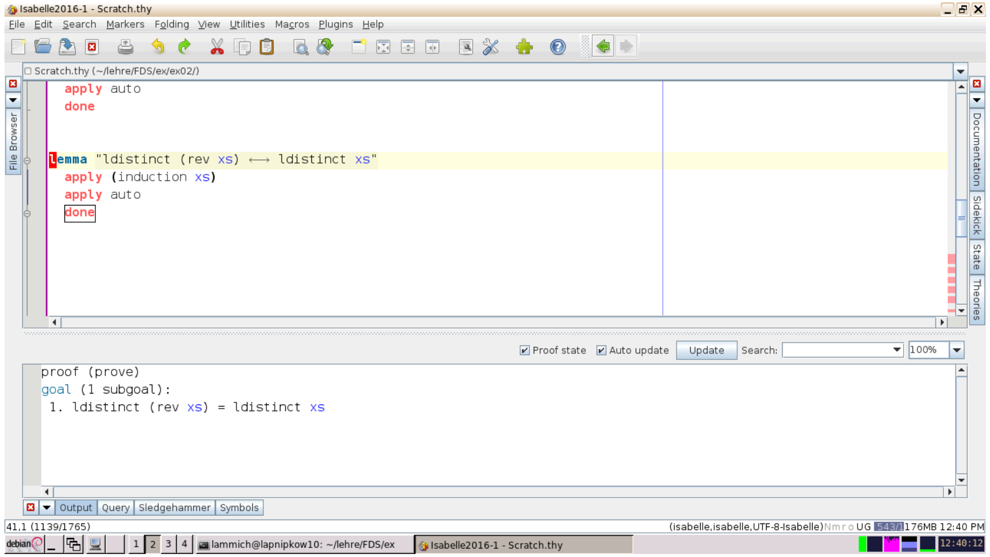
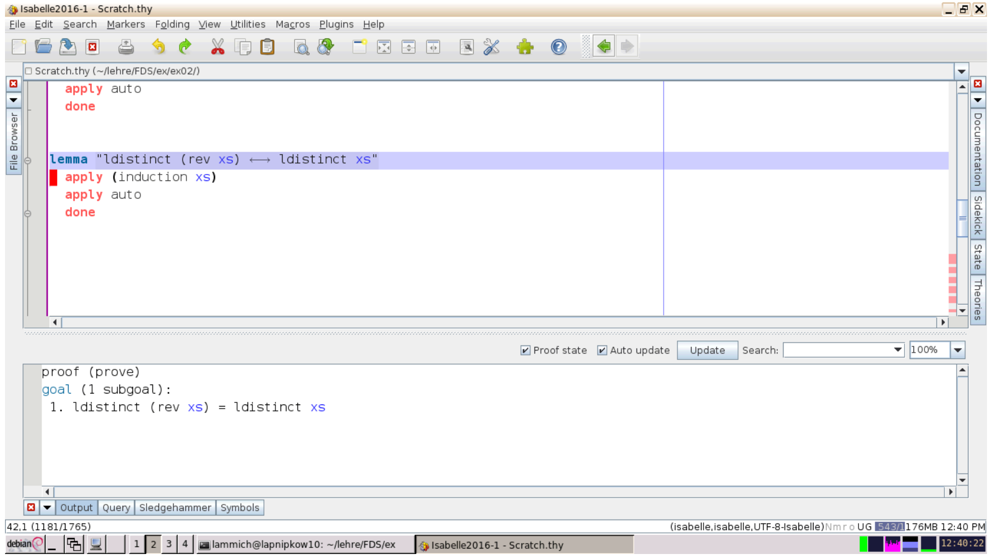
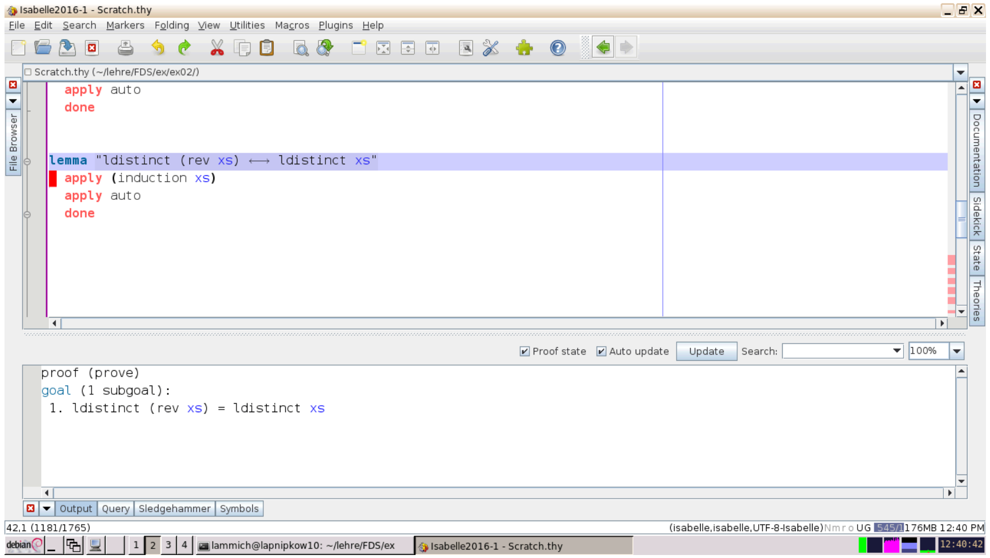
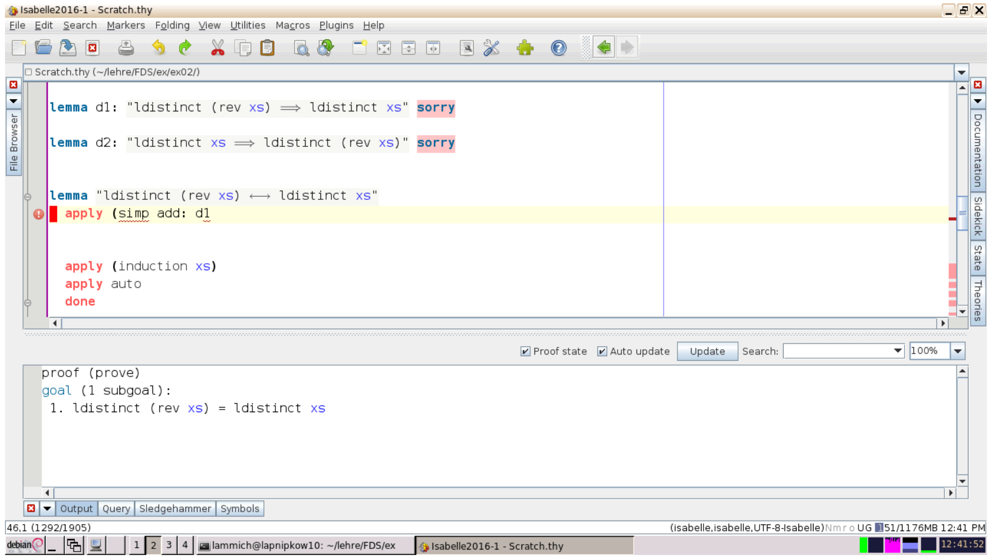

Isabelle2016-1 - Scratch.thy
apply auto
done
lemma contains_rev[simp]: "contains x (rev xs) ⟷ contains x xs"
apply (induction xs)
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. contains x (rev xs) = contains x xs

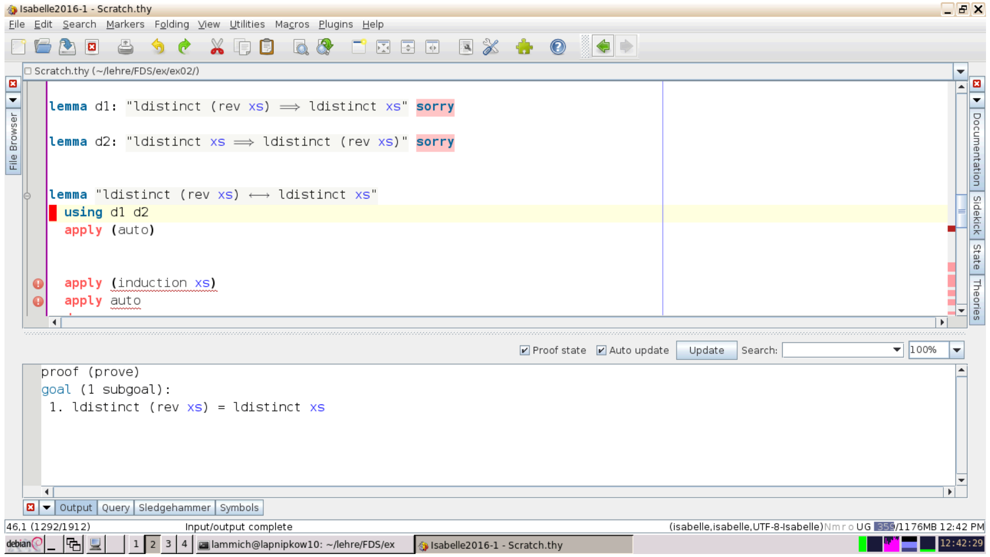

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex02/)
apply (induction xs)
apply auto
done
lemma contains_rev[simp]: "contains x (rev xs) ⟷ contains x xs"
apply (induction xs)
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
theorem ldistinct (?xs @ [?x]) = (ldistinct ?xs ∧ ¬ contains ?x ?xs)
Output Query Sledgehammer Symbols

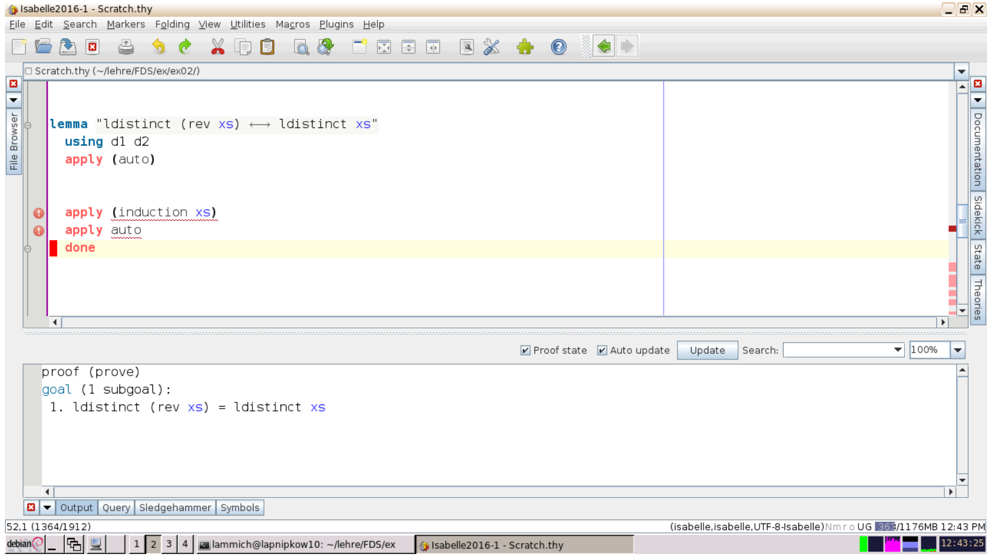

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex02/)
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma [simp]: "contains a (xs @ [x]) ⟷ contains a xs ∨ a=x"
apply (induction xs) apply auto done
lemma [simp]: "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. contains a (xs @ [x]) = (contains a xs ∨ a = x)
Output Query Sledgehammer Symbols

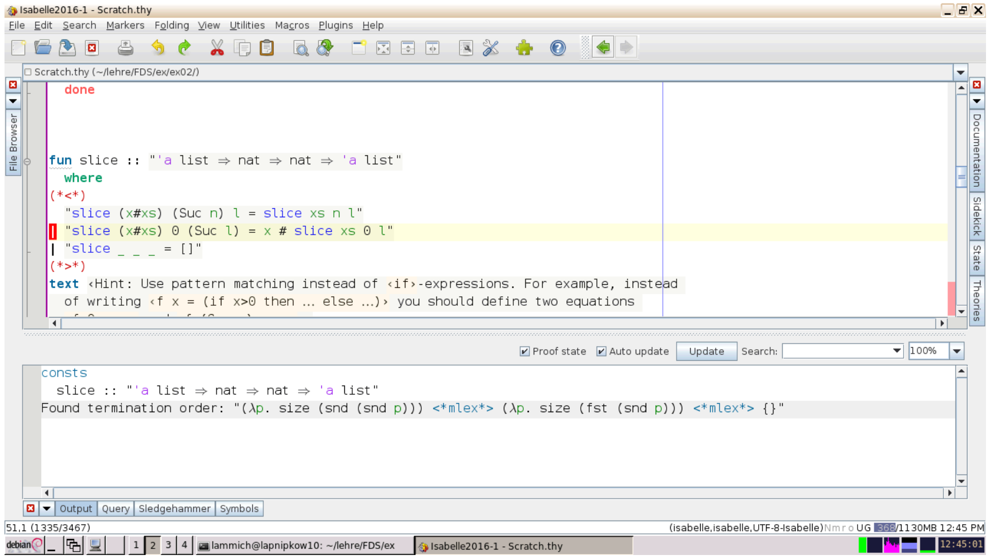

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex02/)
(*<*)
where
"ldistinct [] = True"
| "ldistinct (x#xs) ⟷ ¬contains x xs ∧ ldistinct xs"
(*>*)
lemma [simp]: "contains a (xs @ [x]) ⟷ contains a xs ∨ a=x"
apply (induction xs) apply auto done
lemma [simp]: "ldistinct (xs@[x]) ⟷ ldistinct xs ∧ ¬contains x xs"
apply (induction xs)
apply auto
done
theorem contains ?a (?xs @ [?x]) = (contains ?a ?xs ∨ ?a = ?x)
Output Query Sledgehammer Symbols

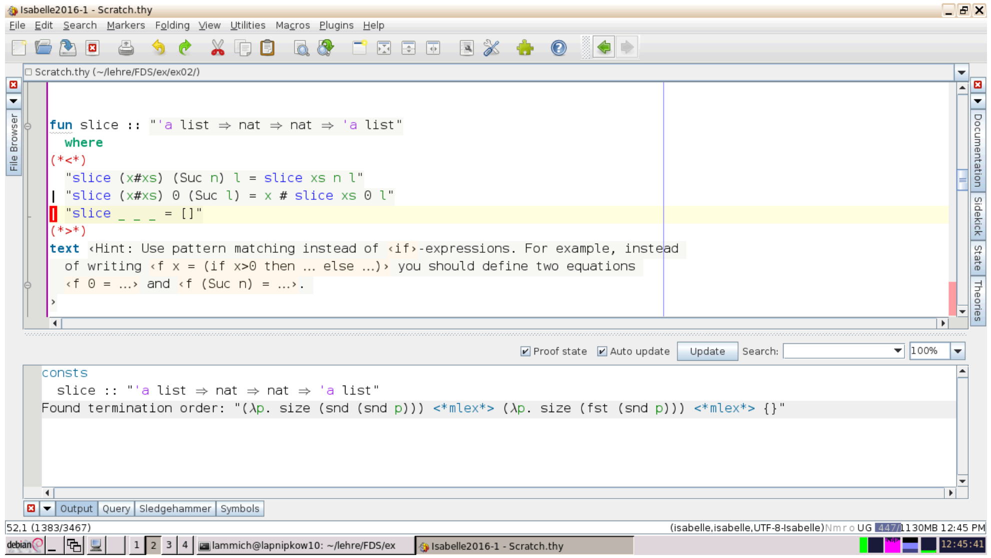

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex02/)
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs
Output Query Sledgehammer Symbols

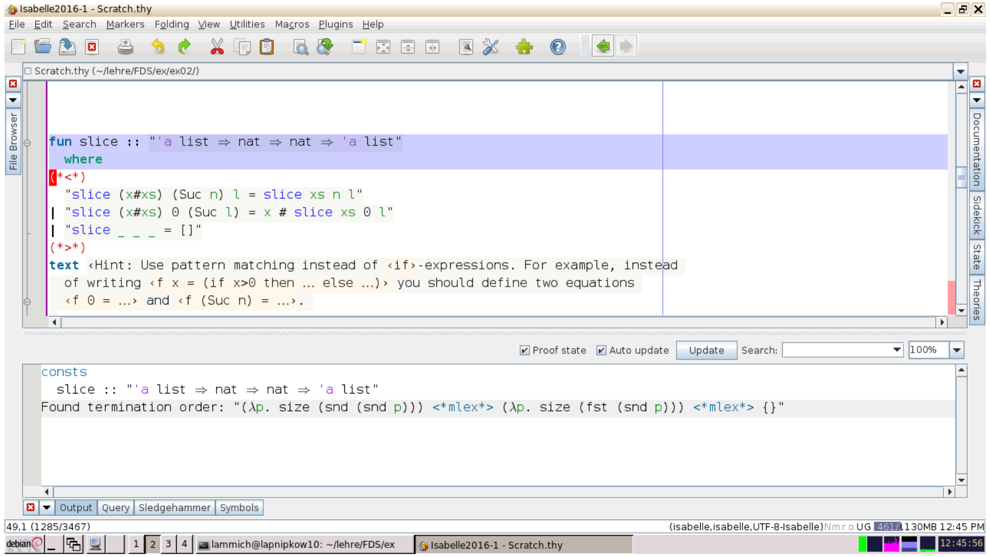

Isabelle2016-1 - Scratch.thy
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs

Isabelle2016-1 - Scratch.thy
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs

Isabelle2016-1 - Scratch.thy
apply auto
done
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs

Isabelle2016-1 - Scratch.thy
lemma d1: "ldistinct (rev xs) ⟹ ldistinct xs" sorry
lemma d2: "ldistinct xs ⟹ ldistinct (rev xs)" sorry
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
apply (simp add: d1
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs

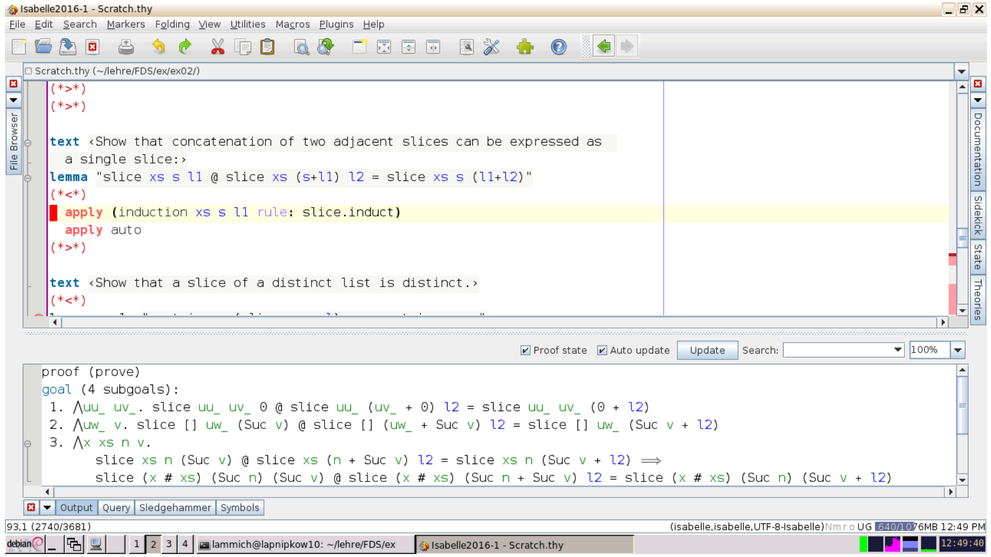

Isabelle2016-1 - Scratch.thy
lemma d1: "ldistinct (rev xs) ⟹ ldistinct xs" sorry
lemma d2: "ldistinct xs ⟹ ldistinct (rev xs)" sorry
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
using d1 d2
apply (auto)
apply (induction xs)
apply auto
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs
Input/output complete

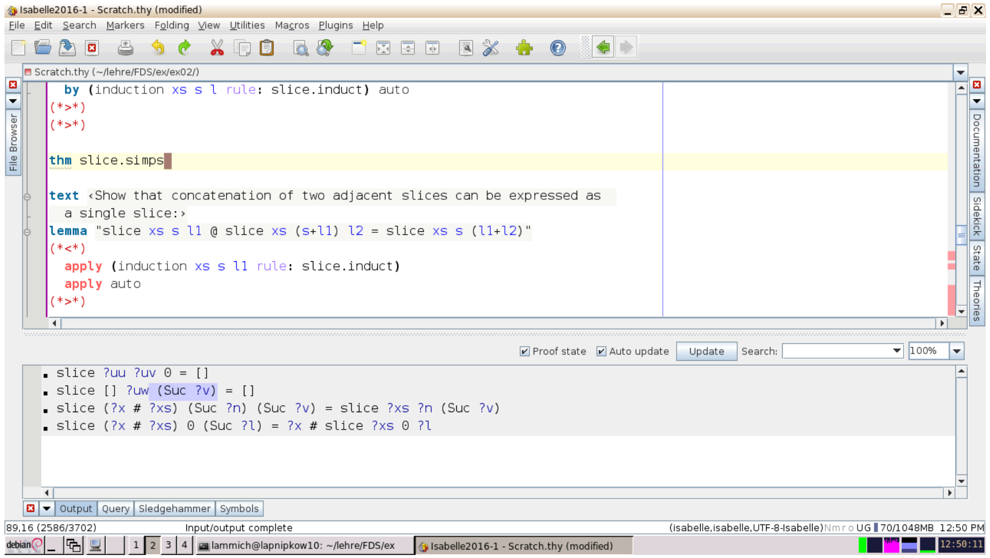

Isabelle2016-1 - Scratch.thy
lemma "ldistinct (rev xs) ⟷ ldistinct xs"
using d1 d2
apply (auto)
apply (induction xs)
apply auto
done
proof (prove)
goal (1 subgoal):
1. ldistinct (rev xs) = ldistinct xs

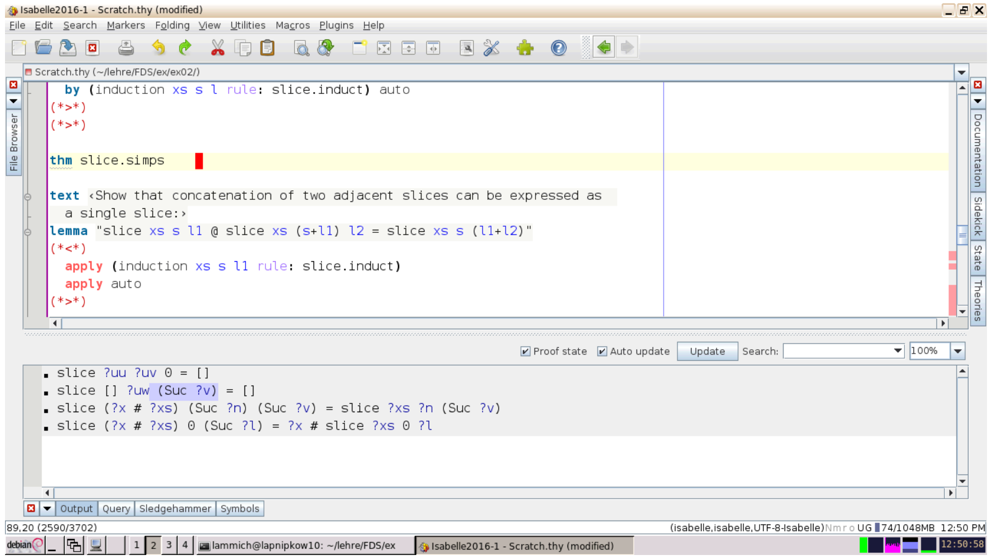

Isabelle2016-1 - Scratch.thy
done
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
(*<*)
"slice (x#xs) (Suc n) l = slice xs n l"
"slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
| "slice _ _ _ = []"
(*>*)
text ‹Hint: Use pattern matching instead of ‹if›-expressions. For example, instead
of writing ‹f x = (if x>0 then … else …)› you should define two equations
consts
slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> (λp. size (fst (snd p))) <*mlex*> {}"

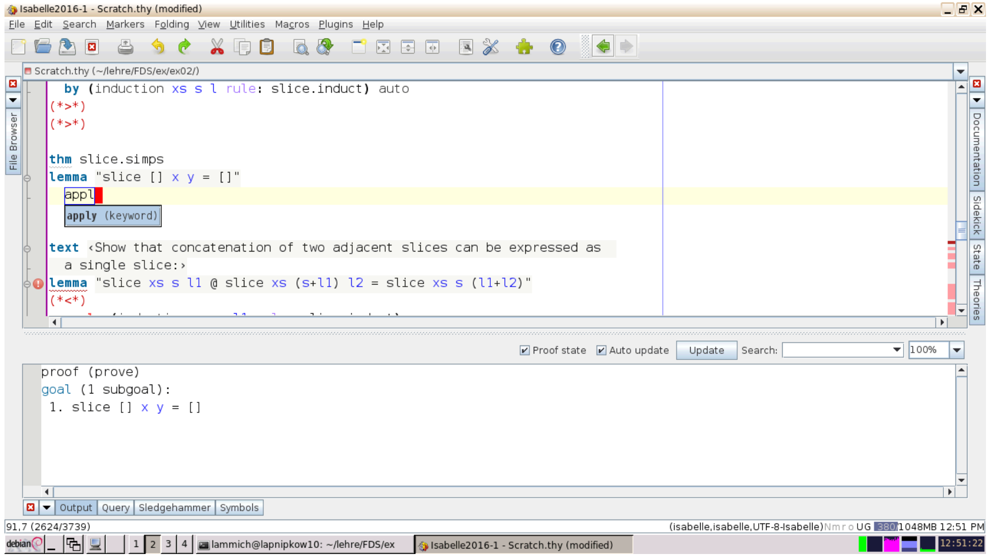

Isabelle2016-1 - Scratch.thy
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
(*<*)
"slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
| "slice _ _ _ = []"
(*>*)
text ‹Hint: Use pattern matching instead of ‹if›-expressions. For example, instead
of writing ‹f x = (if x>0 then … else …)› you should define two equations
‹f 0 = …› and ‹f (Suc n) = …›.
›
consts
slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> (λp. size (fst (snd p))) <*mlex*> {}"

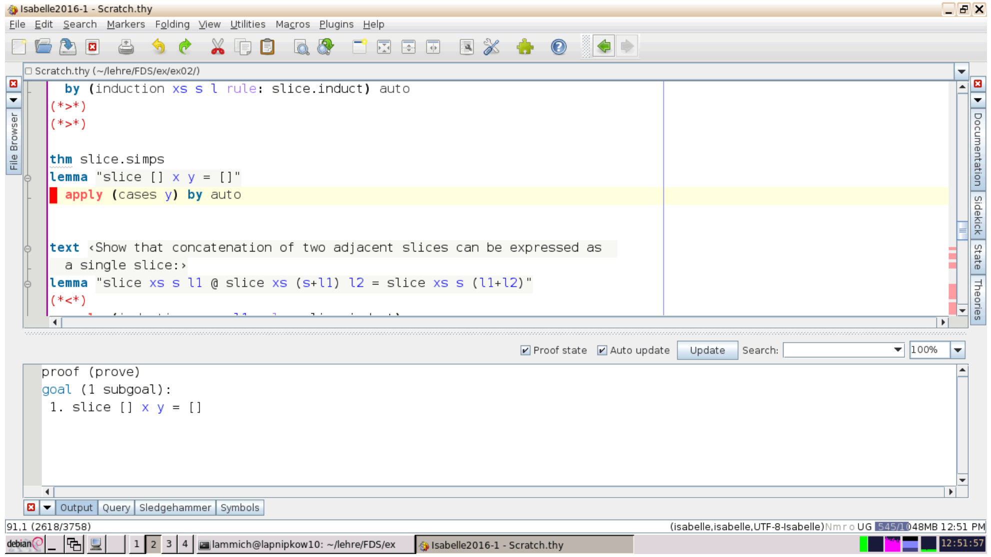

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex/ex02/)
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
(*<*)
"slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
| "slice _ _ _ = []"
(*>*)
text ‹Hint: Use pattern matching instead of ‹if›-expressions. For example, instead
of writing ‹f x = (if x>0 then … else …)› you should define two equations
‹f 0 = …› and ‹f (Suc n) = …›.
consts
slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> (λp. size (fst (snd p))) <*mlex*> {}"

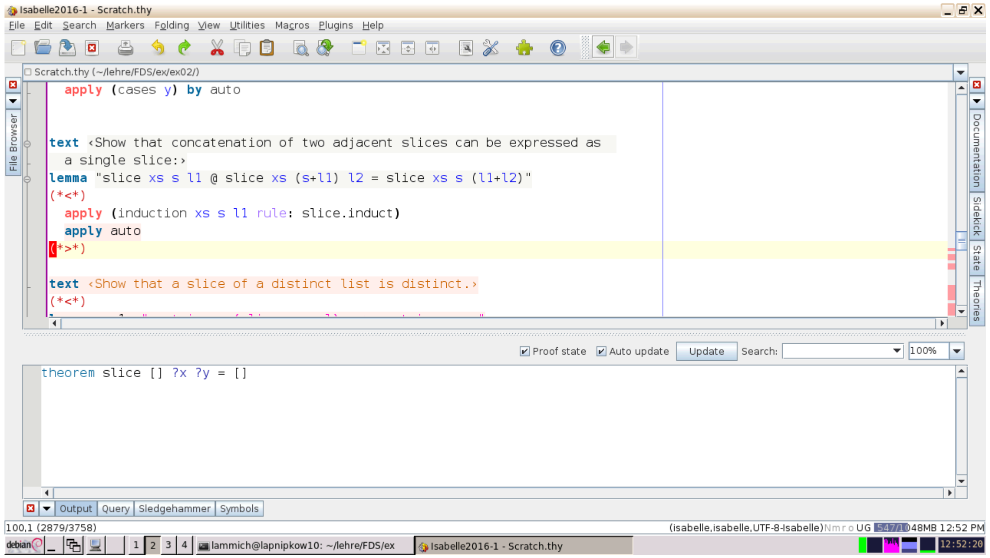

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex/ex02/)
(*>*)
(*>*)
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
apply (induction xs s l1 rule: slice.induct)
apply auto
(*>*)
text ‹Show that a slice of a distinct list is distinct.›
(*<*)
proof (prove)
goal (4 subgoals):
1. ⋀uu_ uv_. slice uu_ uv_ 0 @ slice uu_ (uv_ + 0) l2 = slice uu_ uv_ (0 + l2)
2. ⋀uw_ v. slice [] uw_ (Suc v) @ slice [] (uw_ + Suc v) l2 = slice [] uw_ (Suc v + l2)
3. ⋀x xs n v.
slice xs n (Suc v) @ slice xs (n + Suc v) l2 = slice xs n (Suc v + l2) ⟹
slice (x # xs) (Suc n) (Suc v) @ slice (x # xs) (Suc n + Suc v) l2 = slice (x # xs) (Suc n) (Suc v + l2)

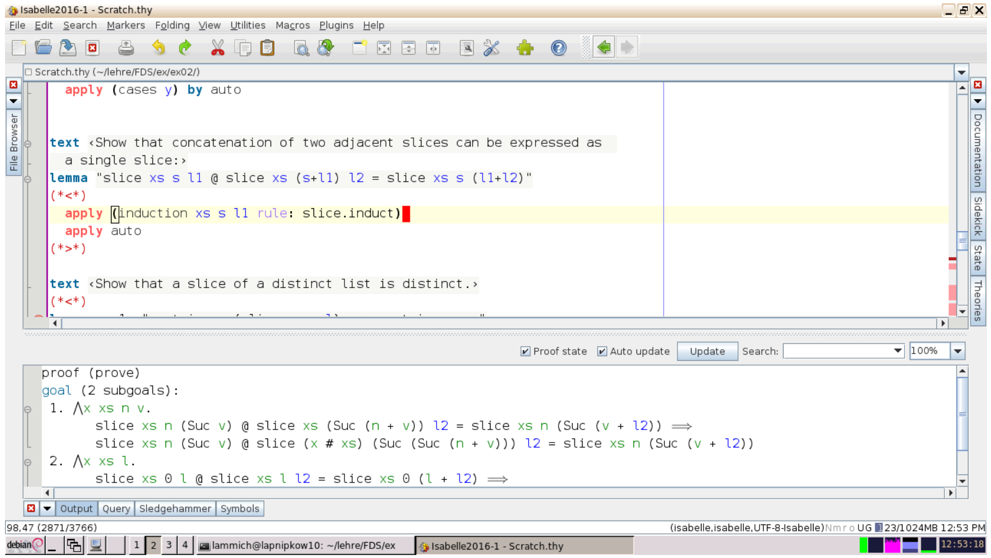

Isabelle2016-1 - Scratch.thy (modified)
Scratch.thy (~/lehre/FDS/ex/ex02/)
by (induction xs s l rule: slice.induct) auto
(*>*)
(*>*)
thm slice.simps
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
apply (induction xs s l1 rule: slice.induct)
apply auto
(*>*)
slice ?uu ?uv 0 = []
slice [] ?uw (Suc ?v) = []
slice (?x # ?xs) (Suc ?n) (Suc ?v) = slice ?xs ?n (Suc ?v)
slice (?x # ?xs) 0 (Suc ?l) = ?x # slice ?xs 0 ?l

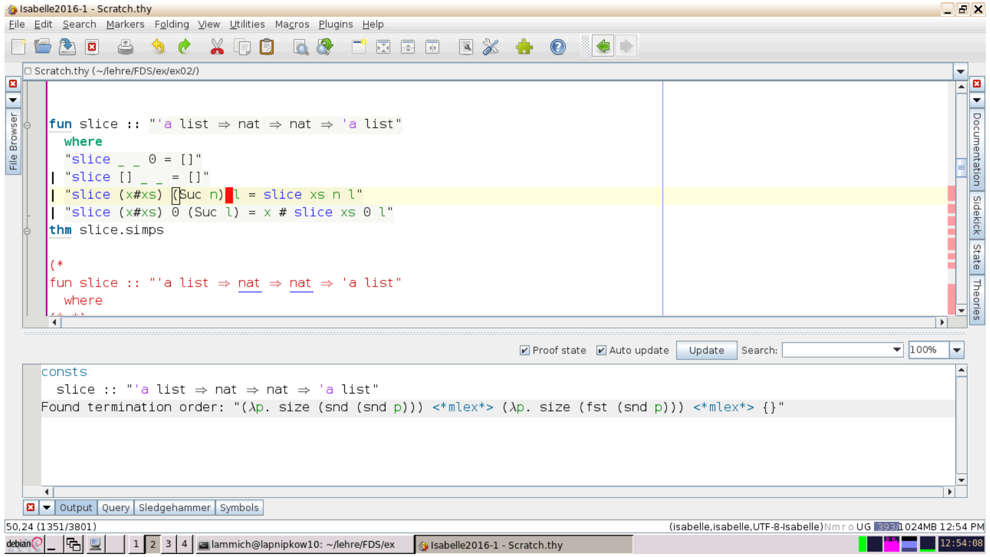

Isabelle2016-1 - Scratch.thy (modified)
Scratch.thy (~/lehre/FDS/ex/ex02/)
by (induction xs s l rule: slice.induct) auto
(*>*)
(*>*)
thm slice.simps
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
apply (induction xs s l1 rule: slice.induct)
apply auto
(*>*)
slice ?uu ?uv 0 = []
slice [] ?uw (Suc ?v) = []
slice (?x # ?xs) (Suc ?n) (Suc ?v) = slice ?xs ?n (Suc ?v)
slice (?x # ?xs) 0 (Suc ?l) = ?x # slice ?xs 0 ?l

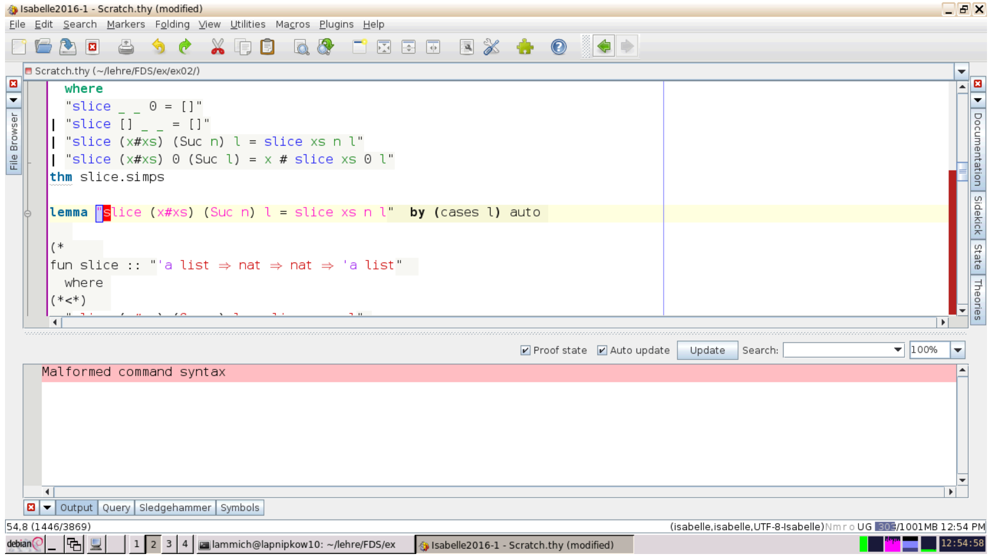

Isabelle2016-1 - Scratch.thy (modified)
by (induction xs s l rule: slice.induct) auto
(*>*)
(*>*)
thm slice.simps
lemma "slice [] x y = []"
appl
apply (keyword)
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
proof (prove)
goal (1 subgoal):
1. slice [] x y = []

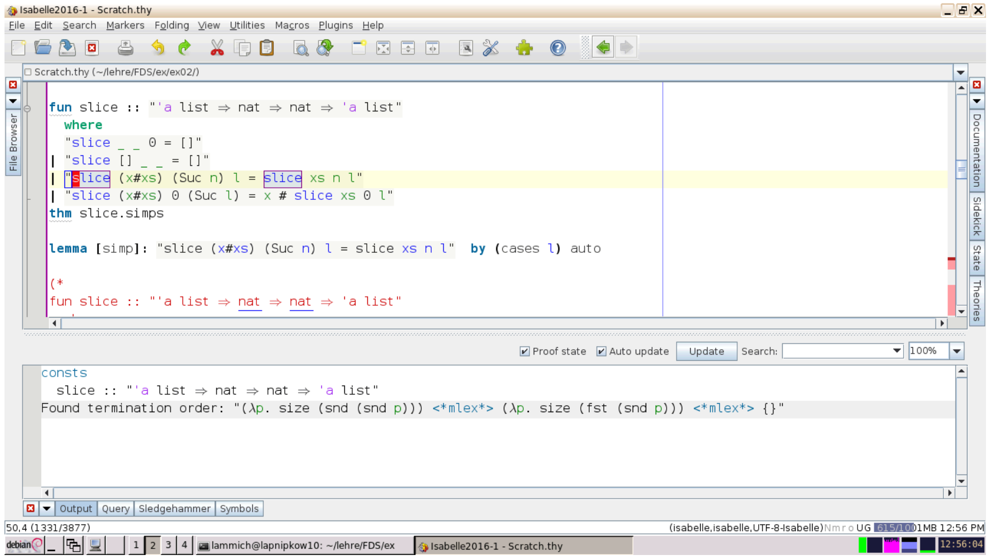

Isabelle2016-1 - Scratch.thy
by (induction xs s l rule: slice.induct) auto
(*>*)
(*>*)
thm slice.simps
lemma "slice [] x y = []"
apply (cases y) by auto
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
proof (prove)
goal (1 subgoal):
1. slice [] x y = []

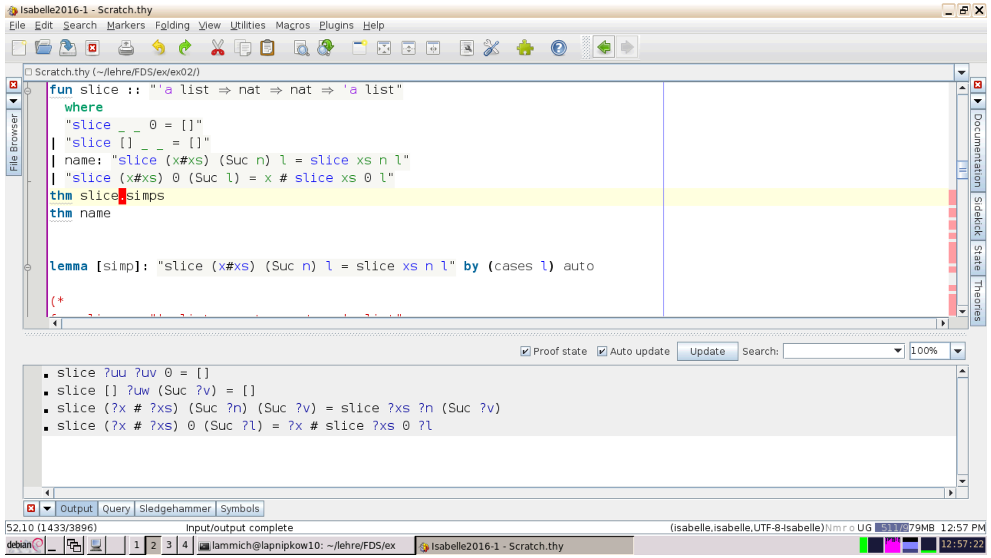

Isabelle2016-1 - Scratch.thy
apply (cases y) by auto
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
apply (induction xs s l1 rule: slice.induct)
apply auto
(*>*)
text ‹Show that a slice of a distinct list is distinct.›
(*<*)
theorem slice [] ?x ?y = []

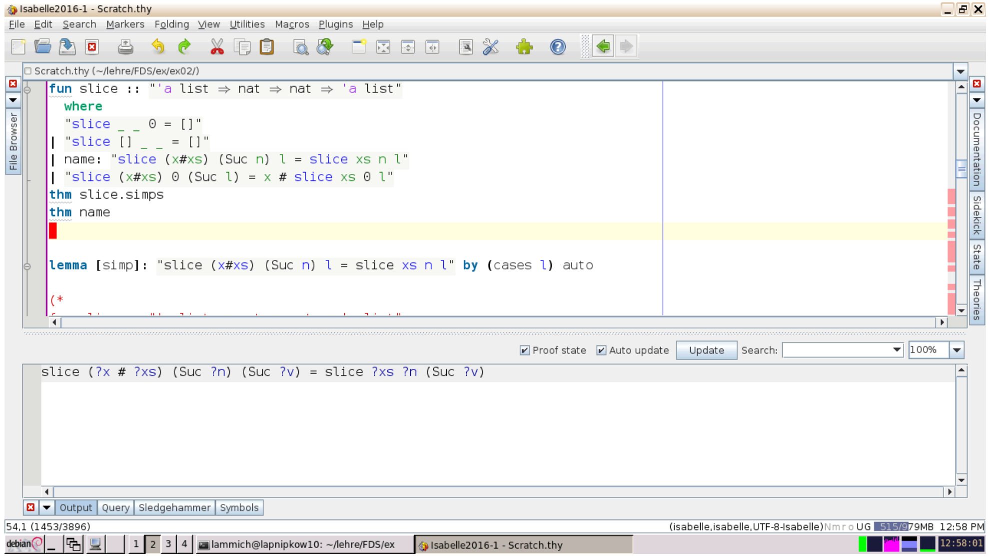

Isabelle2016-1 - Scratch.thy
apply (cases y) by auto
text ‹Show that concatenation of two adjacent slices can be expressed as
a single slice:›
lemma "slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"
(*<*)
apply (induction xs s l1 rule: slice.induct)
apply auto
(*>*)
text ‹Show that a slice of a distinct list is distinct.›
(*<*)
proof (prove)
goal (2 subgoals):
1. ⋀x xs n v.
slice xs n (Suc v) @ slice xs (Suc (n + v)) l2 = slice xs n (Suc (v + l2)) ⟹
slice xs n (Suc v) @ slice (x # xs) (Suc (Suc (n + v))) l2 = slice xs n (Suc (v + l2))
2. ⋀x xs l.
slice xs 0 l @ slice xs l l2 = slice xs 0 (l + l2) ⟹

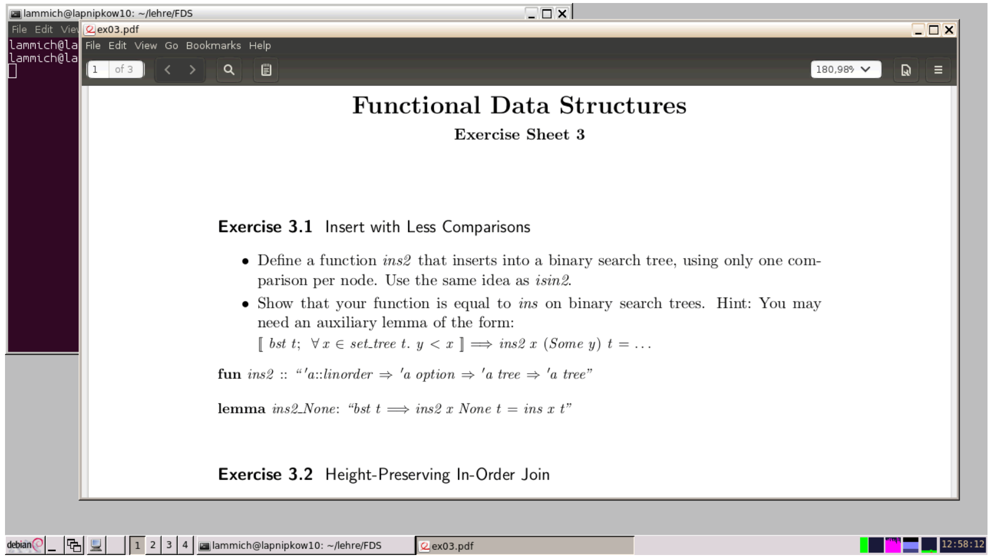

Isabelle2016-1 - Scratch.thy
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
"slice _ _ 0 = []"
| "slice [] _ _ = []"
| "slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
thm slice.simps
(*
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
consts
slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> (λp. size (fst (snd p))) <*mlex*> {}"

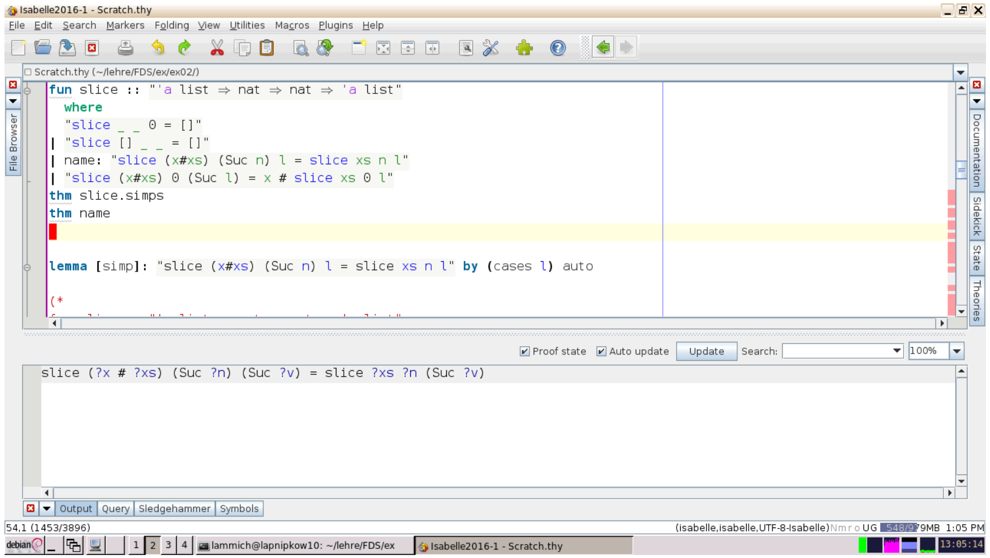

Isabelle2016-1 - Scratch.thy (modified)
where
"slice _ _ 0 = []"
| "slice [] _ _ = []"
| "slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
thm slice.simps
lemma "slice (x#xs) (Suc n) l = slice xs n l" by (cases l) auto
(*
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
(*<*)
Malformed command syntax

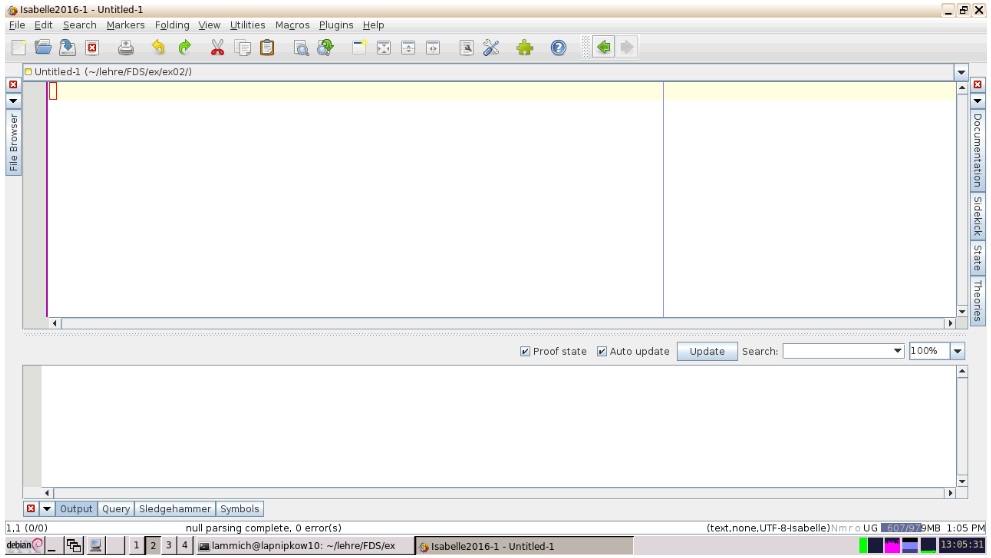

Isabelle2016-1 - Scratch.thy
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
"slice _ _ 0 = []"
| "slice [] _ _ = []"
| "slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
thm slice.simps
lemma [simp]: "slice (x#xs) (Suc n) l = slice xs n l" by (cases l) auto
(*
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
consts
slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> (λp. size (fst (snd p))) <*mlex*> {}"

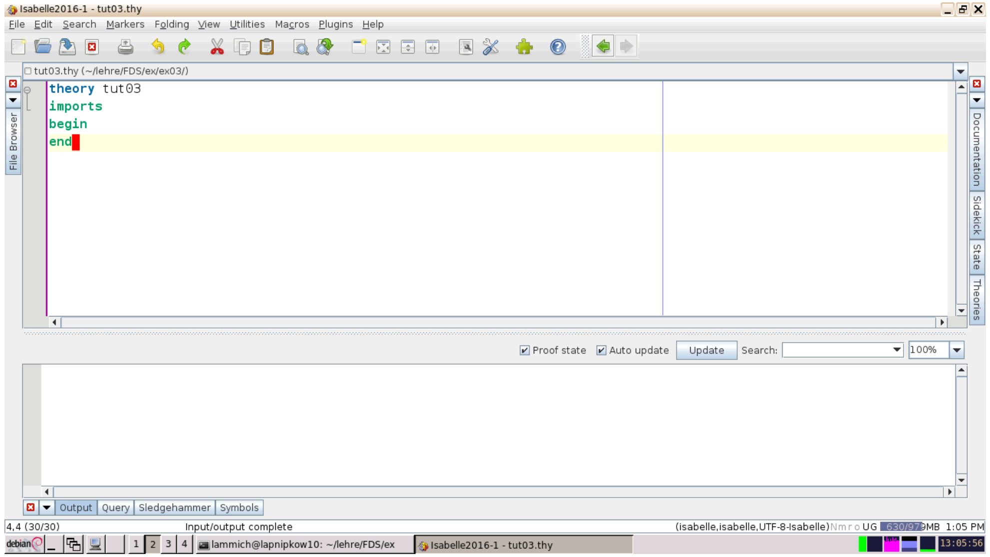

Isabelle2016-1 - Scratch.thy
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
"slice _ _ 0 = []"
| "slice [] _ _ = []"
| name: "slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
thm slice.simps
thm name
lemma [simp]: "slice (x#xs) (Suc n) l = slice xs n l" by (cases l) auto
(*
slice ?uu ?uv 0 = []
slice [] ?uw (Suc ?v) = []
slice (?x # ?xs) (Suc ?n) (Suc ?v) = slice ?xs ?n (Suc ?v)
slice (?x # ?xs) 0 (Suc ?l) = ?x # slice ?xs 0 ?l
Input/output complete

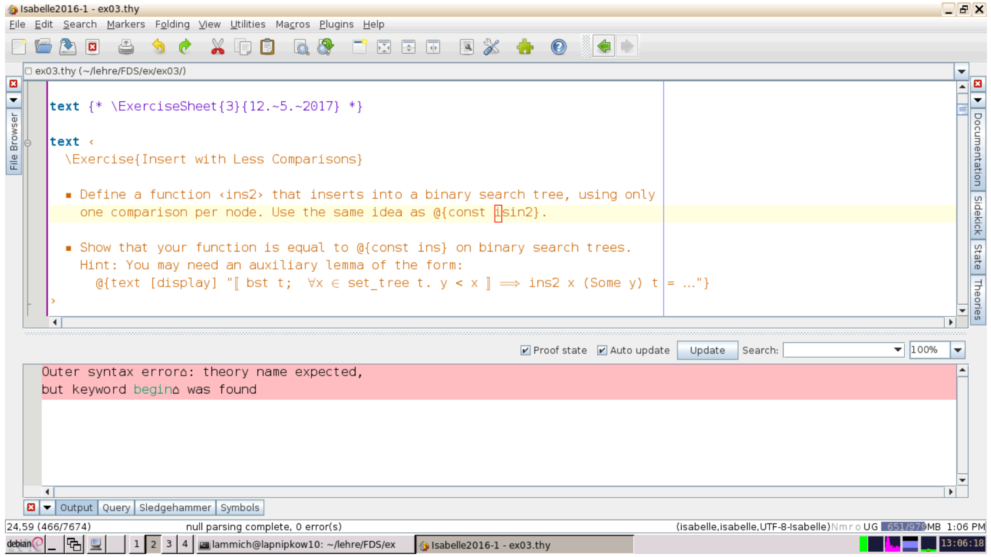

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex/ex02/)
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
"slice _ _ 0 = []"
| "slice [] _ _ = []"
| name: "slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
thm slice.simps
thm name
lemma [simp]: "slice (x#xs) (Suc n) l = slice xs n l" by (cases l) auto
Proof state
Auto update
Update
Search:
100%
slice (?x # ?xs) (Suc ?n) (Suc ?v) = slice ?xs ?n (Suc ?v)
Output
Query
Sledgehammer
Symbols
54,1 (1453/3896)
12:58 PM

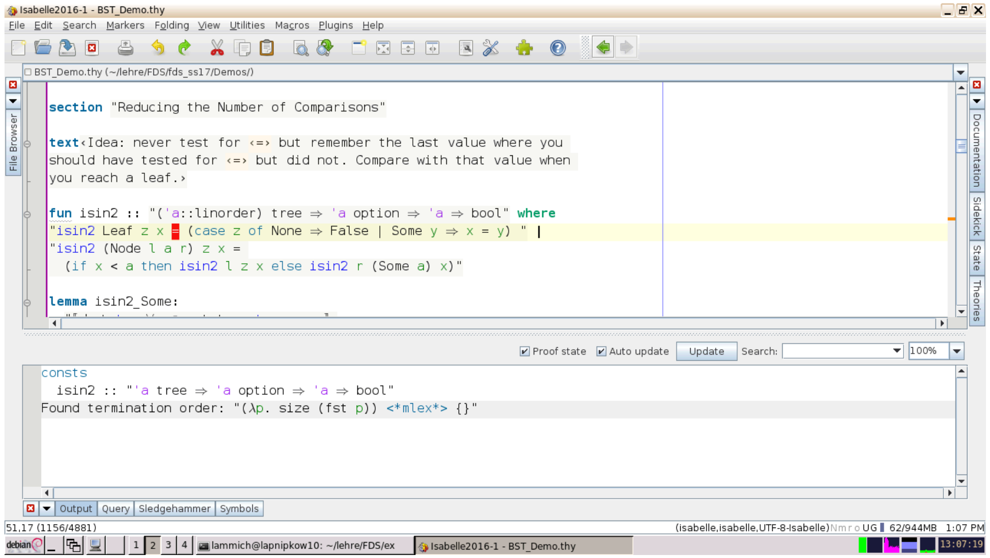

lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
Functional Data Structures
Exercise Sheet 3
Exercise 3.1 Insert with Less Comparisons
• Define a function ins2 that inserts into a binary search tree, using only one comparison per node. Use the same idea as isin2.
• Show that your function is equal to ins on binary search trees. Hint: You may need an auxiliary lemma of the form:
[bst t; ∀x ∈ set_tree t. y < x] ⟹ ins2 x (Some y) t = ...
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
lemma ins2_None: "bst t ⟹ ins2 x None t = ins x t"
Exercise 3.2 Height-Preserving In-Order Join
12:58:12

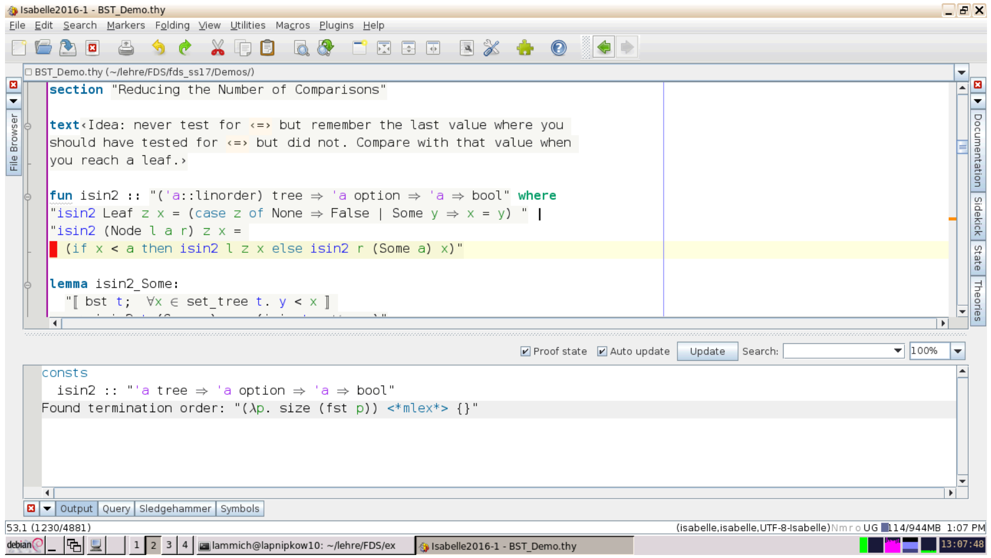

Isabelle2016-1 - Scratch.thy
Scratch.thy (~/lehre/FDS/ex/ex02/)
fun slice :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where
"slice _ _ 0 = []"
| "slice [] _ _ = []"
| name: "slice (x#xs) (Suc n) l = slice xs n l"
| "slice (x#xs) 0 (Suc l) = x # slice xs 0 l"
thm slice.simps
thm name
lemma [simp]: "slice (x#xs) (Suc n) l = slice xs n l" by (cases l) auto
slice (?x # ?xs) (Suc ?n) (Suc ?v) = slice ?xs ?n (Suc ?v)
54,1 (1453/3896)
1:05 PM

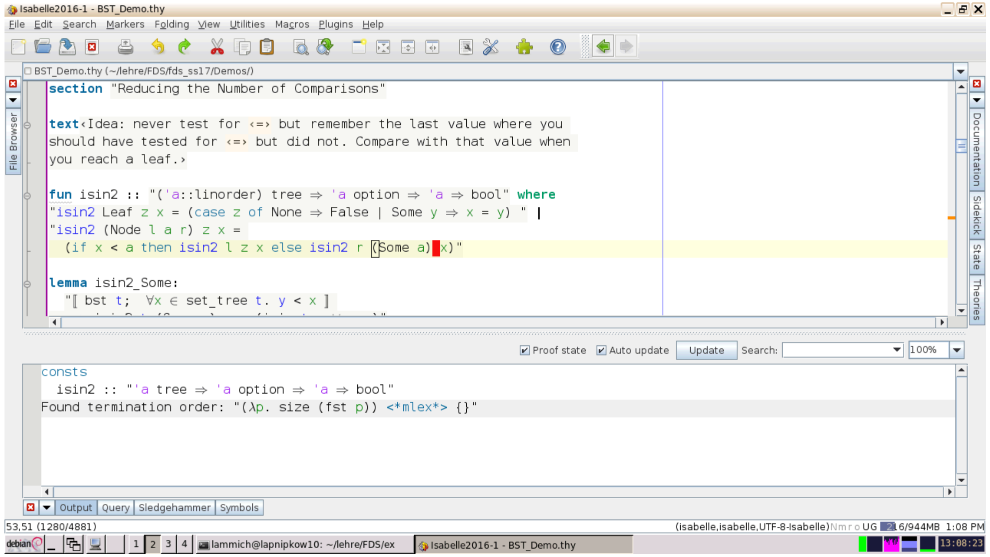

Isabelle2016-1 - Untitled-1
Untitled-1 (~/lehre/FDS/ex/ex02/)
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
1,1 (0/0)
null parsing complete, 0 error(s)
1:05 PM


Isabelle2016-1 - tut03.thy
tut03.thy (~/lehre/FDS/ex/ex03/)
theory tut03
imports
begin
end
Proof state Auto update Update Search: 100%
Output Query Sledgehammer Symbols
4,4 (30/30) Input/output complete

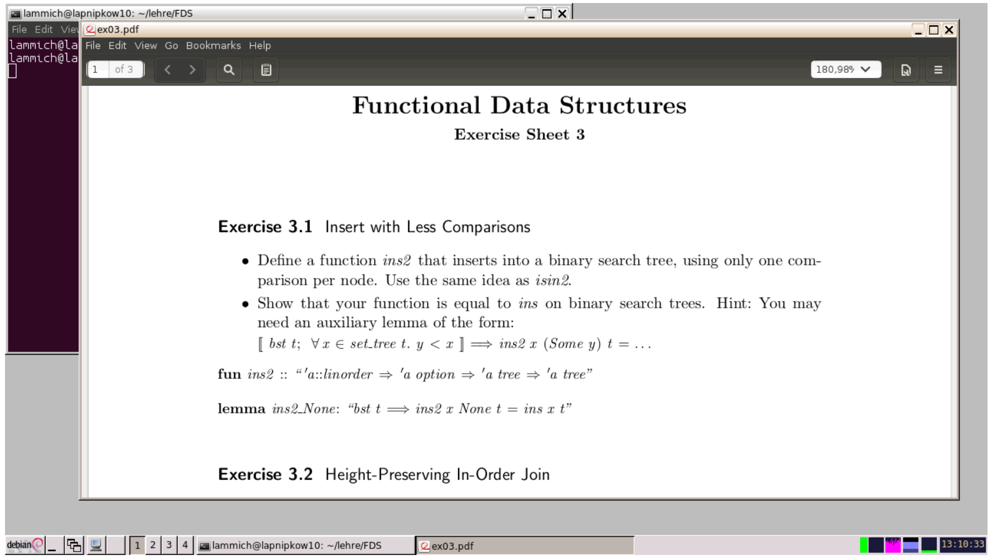

Isabelle2016-1 - ex03.thy
ex03.thy (~/lehre/FDS/ex/ex03/)
text {* \ExerciseSheet{3}{12.~5.~2017} *}
text ‹
\Exercise{Insert with Less Comparisons}
▪ Define a function ‹ins2› that inserts into a binary search tree, using only one comparison per node. Use the same idea as @{const isin2}.
▪ Show that your function is equal to @{const ins} on binary search trees.
Hint: You may need an auxiliary lemma of the form:
@{text [display] "⟦ bst t; ∀x ∈ set_tree t. y < x ⟧ ⟹ ins2 x (Some y) t = …"}
›
Outer syntax error: theory name expected,
but keyword begin was found
Output Query Sledgehammer Symbols
24,59 (466/7674) null parsing complete, 0 error(s)


Isabelle2016-1 - BST_Demo.thy
BST_Demo.thy (~/lehre/FDS/fds_ss17/Demos/)
section "Reducing the Number of Comparisons"
text‹Idea: never test for ‹=› but remember the last value where you should have tested for ‹=› but did not. Compare with that value when you reach a leaf.›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool" where
"isin2 Leaf z x = (case z of None ⇒ False | Some y ⇒ x = y)" |
"isin2 (Node l a r) z x =
(if x < a then isin2 l z x else isin2 r (Some a) x)"
lemma isin2_Some:
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
Output Query Sledgehammer Symbols
51,17 (1156/4881)

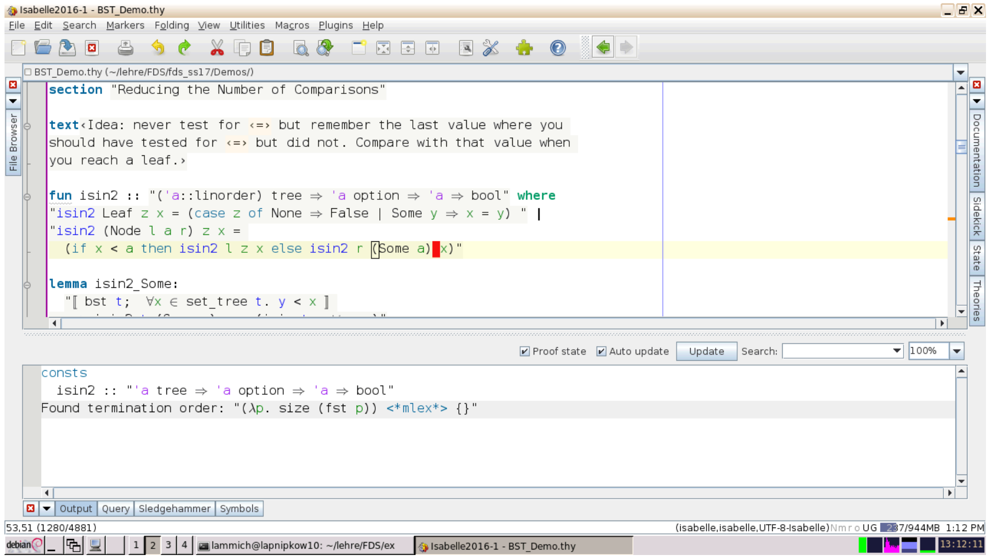

Isabelle2016-1 - BST_Demo.thy
BST_Demo.thy (~/lehre/FDS/fds_ss17/Demos/)
section "Reducing the Number of Comparisons"
text‹Idea: never test for ‹=› but remember the last value where you should have tested for ‹=› but did not. Compare with that value when you reach a leaf.›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool" where
"isin2 Leaf z x = (case z of None ⇒ False | Some y ⇒ x = y)" |
"isin2 (Node l a r) z x =
(if x < a then isin2 l z x else isin2 r (Some a) x)"
lemma isin2_Some:
"⟦ bst t; ∀x ∈ set_tree t. y < x ⟧
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
Output Query Sledgehammer Symbols
53,1 (1230/4881)



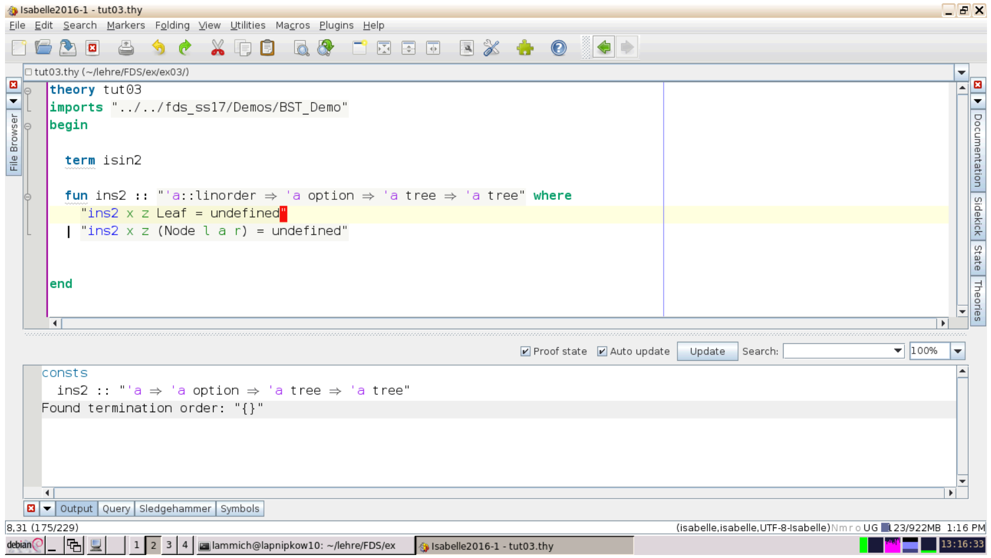
Isabelle2016-1 - BST_Demo.thy
BST_Demo.thy (~/lehre/FDS/fds_ss17/Demos/)
section "Reducing the Number of Comparisons"
text‹Idea: never test for ‹=› but remember the last value where you should have tested for ‹=› but did not. Compare with that value when you reach a leaf.›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool" where
"isin2 Leaf z x = (case z of None ⇒ False | Some y ⇒ x = y)" |
"isin2 (Node l a r) z x =
(if x < a then isin2 l z x else isin2 r (Some a) x)"
lemma isin2_Some:
"⟦ bst t; ∀x ∈ set_tree t. y < x ⟧
Proof state
Auto update
Update
Search:
100%
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
Output
Query
Sledgehammer
Symbols
53,51 (1280/4881)
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - BST_Demo.thy
13:08:23



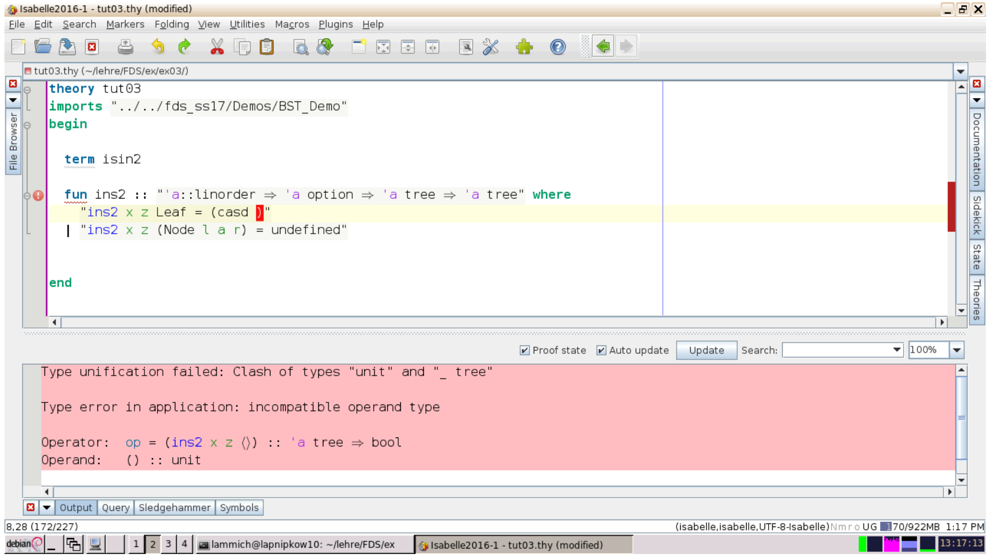
lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
1 of 3
180,98%
Functional Data Structures
Exercise Sheet 3
Exercise 3.1 Insert with Less Comparisons
• Define a function ins2 that inserts into a binary search tree, using only one comparison per node. Use the same idea as isin2.
• Show that your function is equal to ins on binary search trees. Hint: You may need an auxiliary lemma of the form:
⟦ bst t; ∀x ∈ set_tree t. y < x ⟧ ⟹ ins2 x (Some y) t = ...
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
lemma ins2_None: "bst t ⟹ ins2 x None t = ins x t"
Exercise 3.2 Height-Preserving In-Order Join
lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
13:10:33

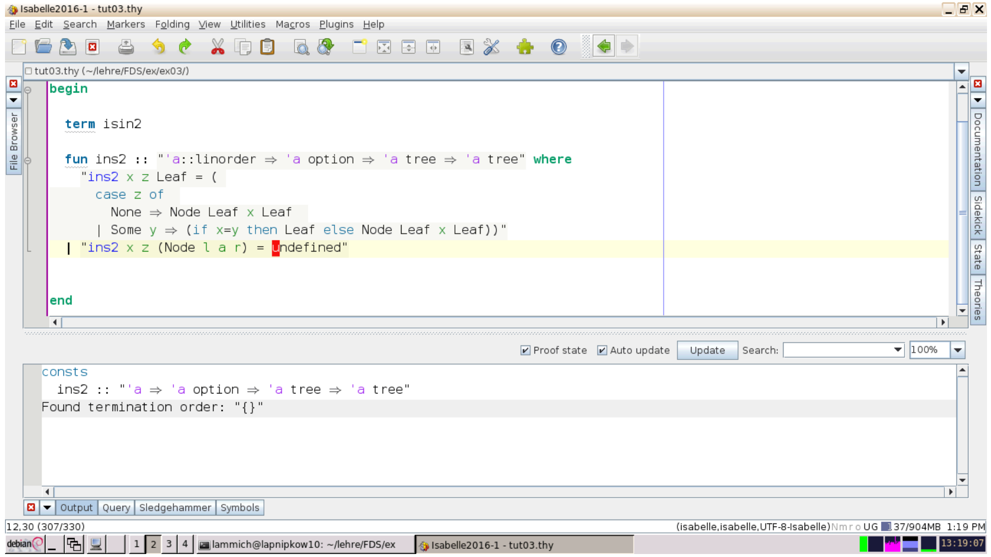


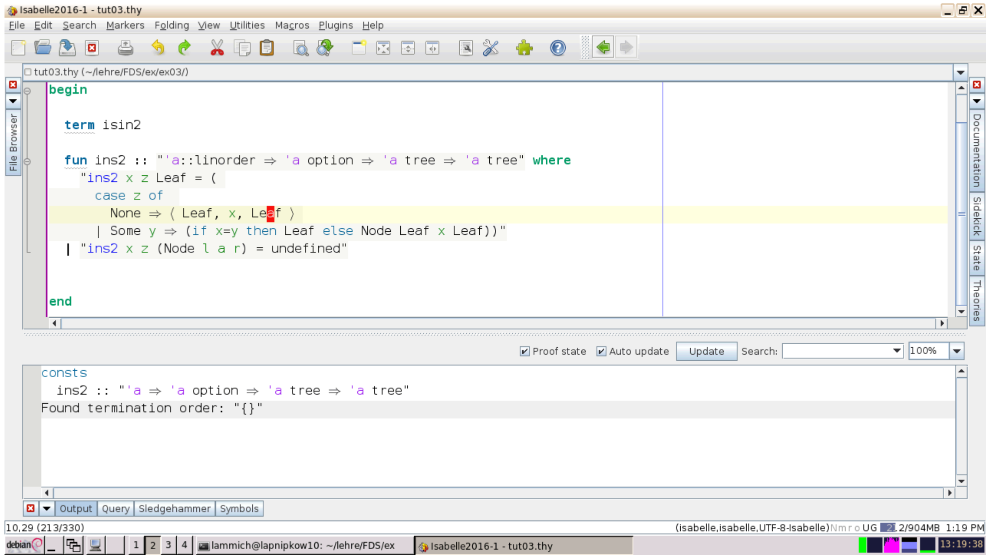
Isabelle2016-1 - BST_Demo.thy
BST_Demo.thy (~/lehre/FDS/fds_ss17/Demos/)
section "Reducing the Number of Comparisons"
text‹Idea: never test for ‹=› but remember the last value where you should have tested for ‹=› but did not. Compare with that value when you reach a leaf.›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool" where
"isin2 Leaf z x = (case z of None ⇒ False | Some y ⇒ x = y)" |
"isin2 (Node l a r) z x =
(if x < a then isin2 l z x else isin2 r (Some a) x)"
lemma isin2_Some:
"⟦ bst t; ∀x ∈ set_tree t. y < x ⟧
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
53,51 (1280/4881)
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - BST_Demo.thy
13:12:11

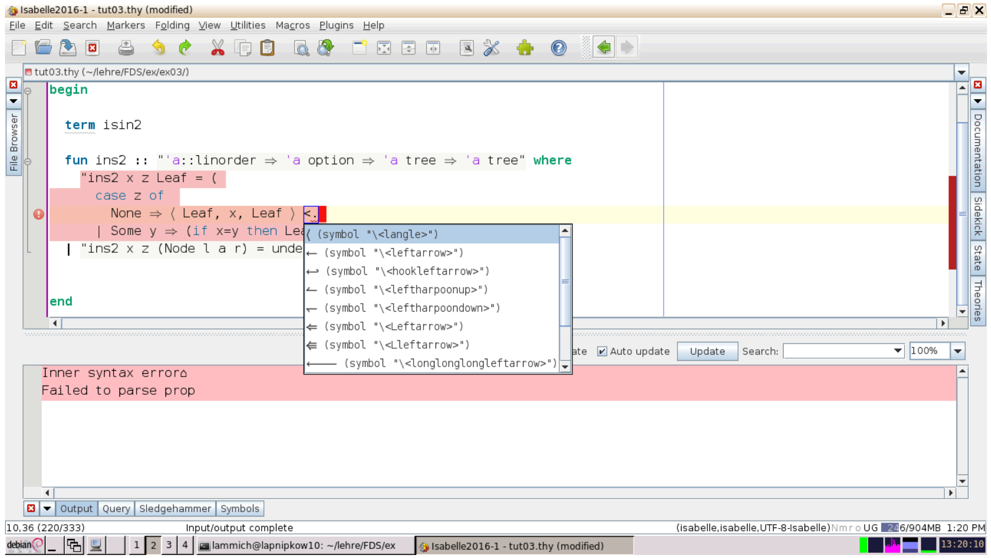


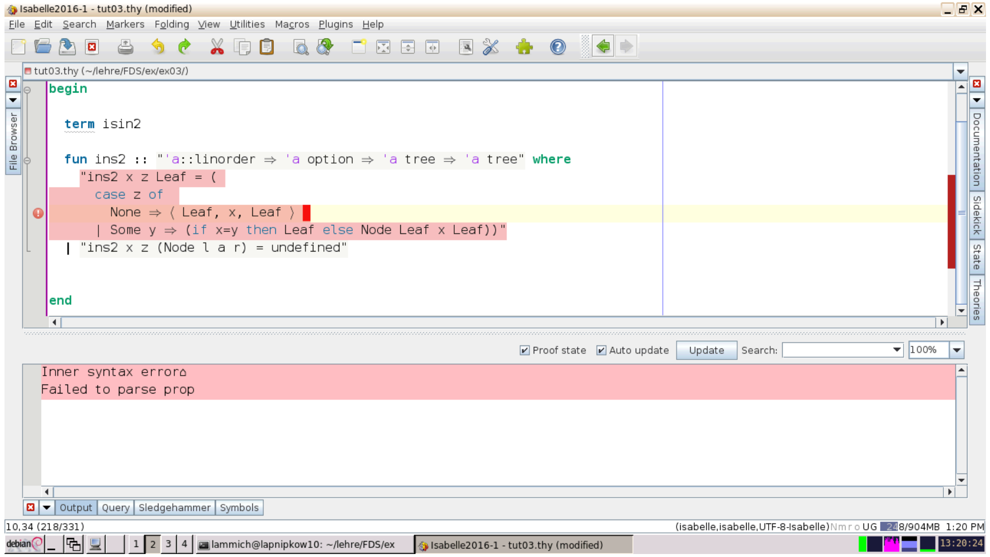
Isabelle2016-1 - tut03.thy
tut03.thy (~/lehre/FDS/ex/ex03/)
theory tut03
imports "../../fds_ss17/Demos/BST_Demo"
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = undefined"
| "ins2 x z (Node l a r) = undefined"
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "{}"
8,31 (175/229)
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut03.thy
13:16:33

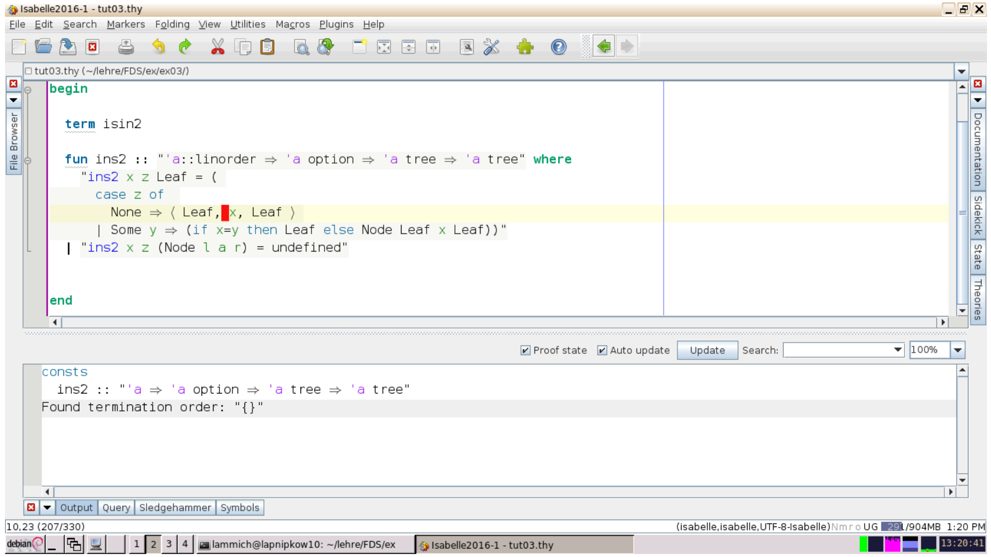

Isabelle2016-1 - tut03.thy (modified)
theory tut03
imports "../../fds_ss17/Demos/BST_Demo"
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (casd)"
| "ins2 x z (Node l a r) = undefined"
end
Type unification failed: Clash of types "unit" and "_ tree"
Type error in application: incompatible operand type
Operator: op = (ins2 x z ()) :: 'a tree ⇒ bool
Operand: () :: unit

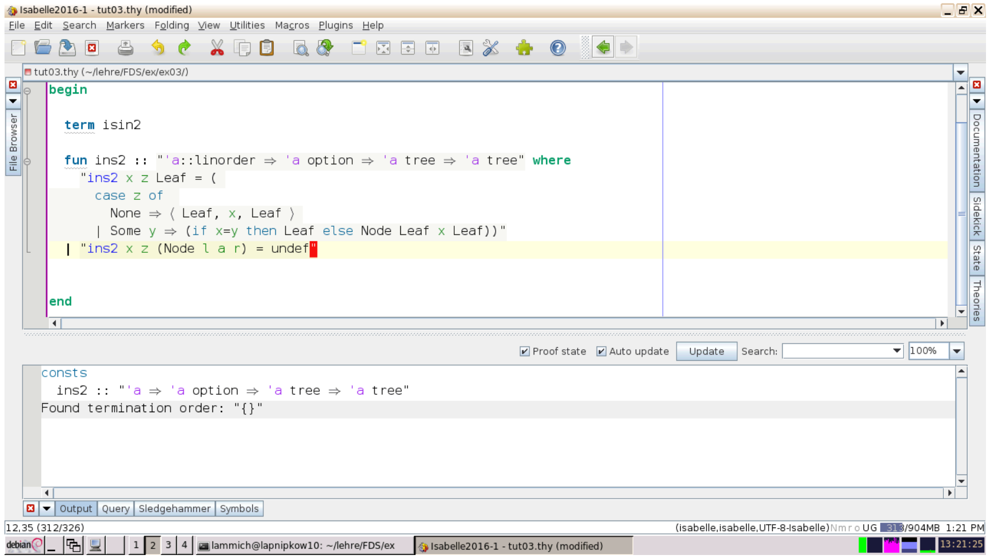

Isabelle2016-1 - tut03.thy
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ Node Leaf x Leaf
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = undefined"
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "{}"

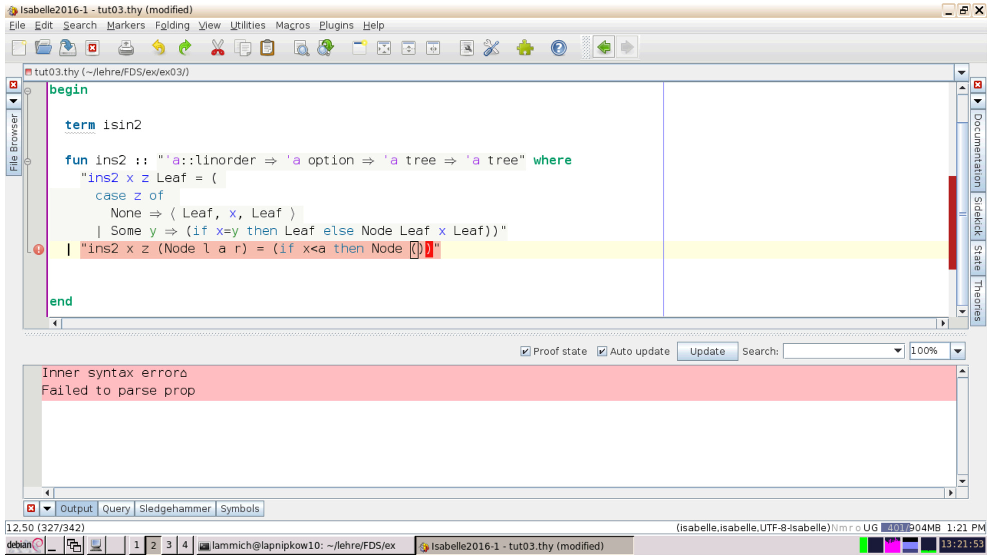

Isabelle2016-1 - tut03.thy
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = undefined"
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "{}"

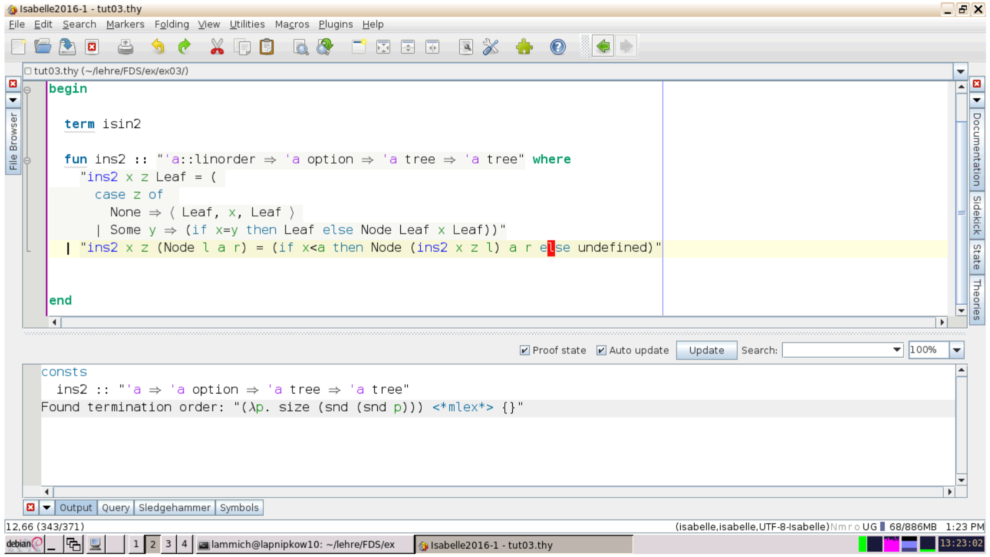

Isabelle2016-1 - tut03.thy (modified)
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩ <
| Some y ⇒ (if x=y then Lea
| "ins2 x z (Node l a r) = unde
end
⟨ (symbol "\<langle>")
← (symbol "\<leftarrow>")
↩ (symbol "\<hookleftarrow>")
↼ (symbol "\<leftharpoonup>")
↽ (symbol "\<leftharpoondown>")
⇐ (symbol "\<Leftarrow>")
⇚ (symbol "\<Lleftarrow>")
⟵ (symbol "\<longlonglongleftarrow>")
Inner syntax error
Failed to parse prop

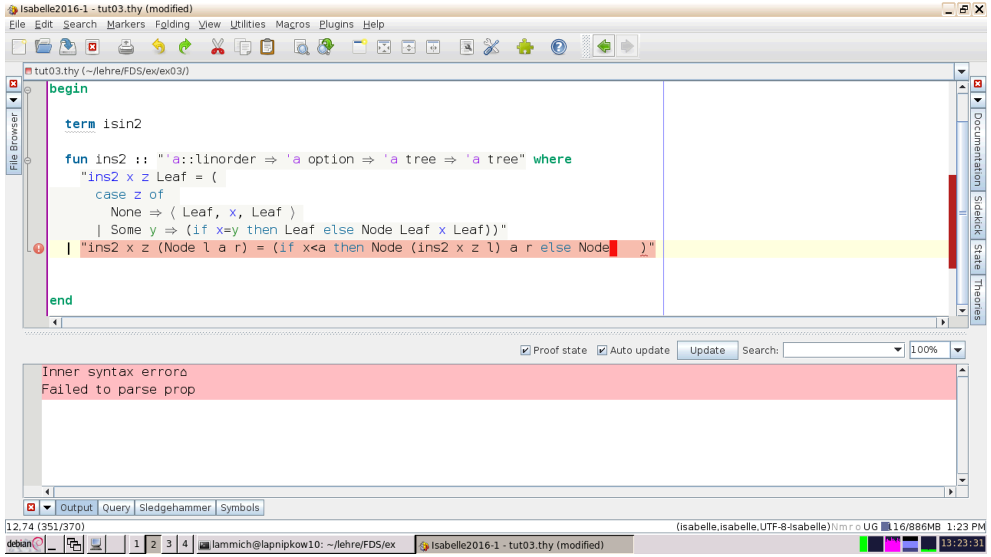

Isabelle2016-1 - tut03.thy (modified)
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ (Leaf, x, Leaf)
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = undefined"
end
Inner syntax error
Failed to parse prop

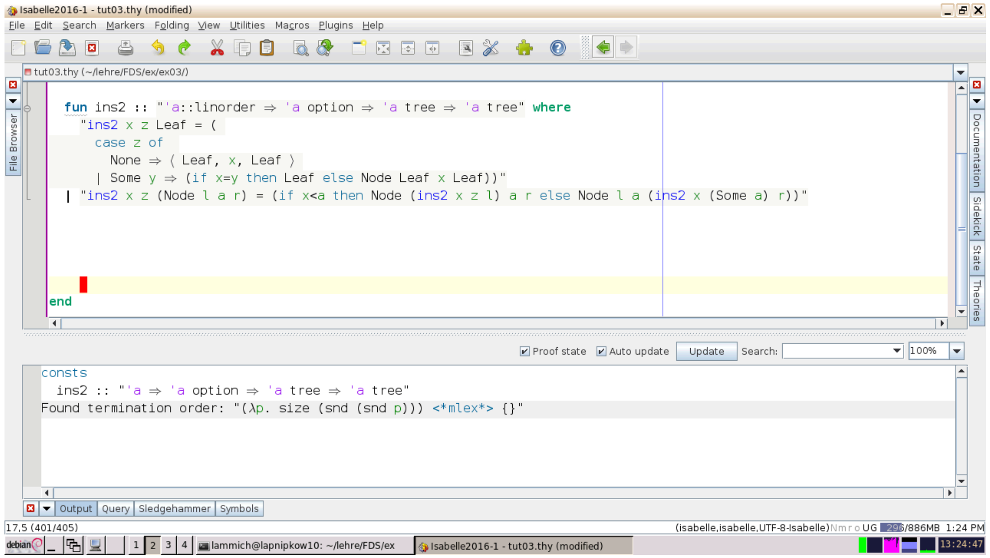

Isabelle2016-1 - tut03.thy
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ (Leaf, x, Leaf)
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = undefined"
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "{}"

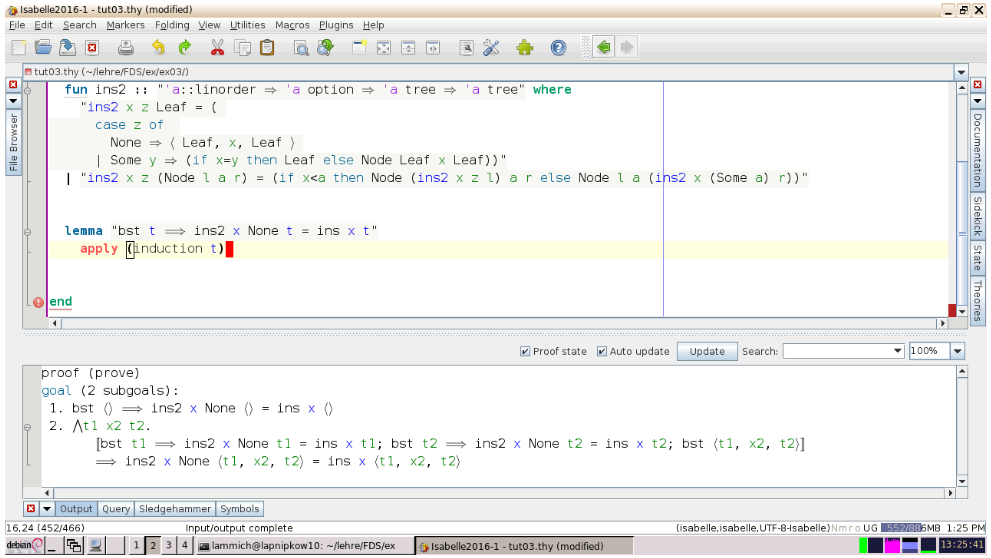

Isabelle2016-1 - tut03.thy (modified)
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ (Leaf, x, Leaf)
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = undef
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "{}"

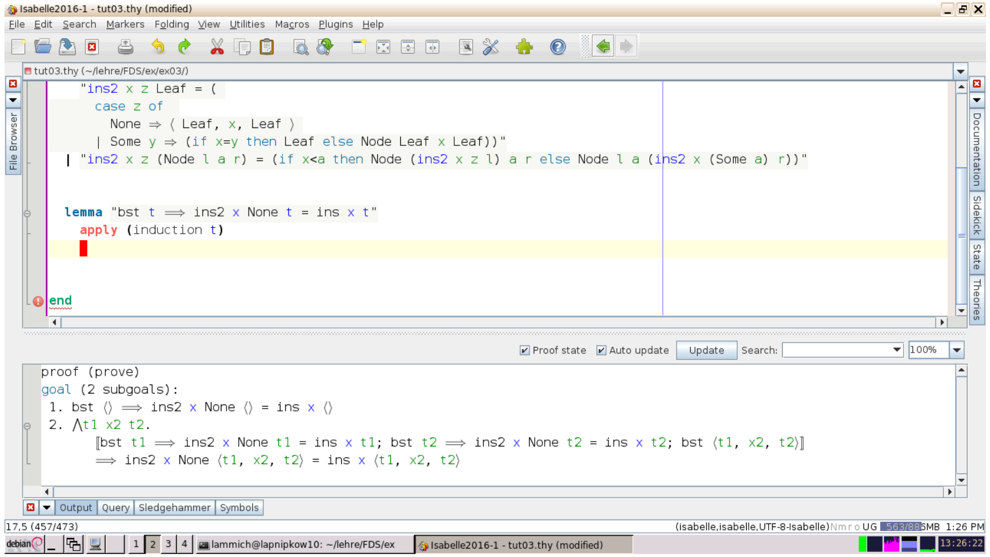
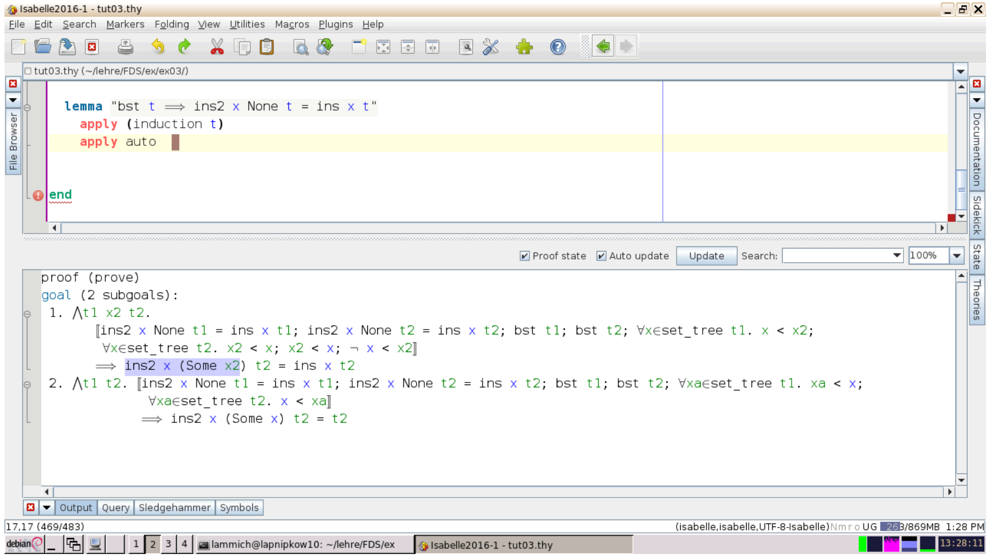
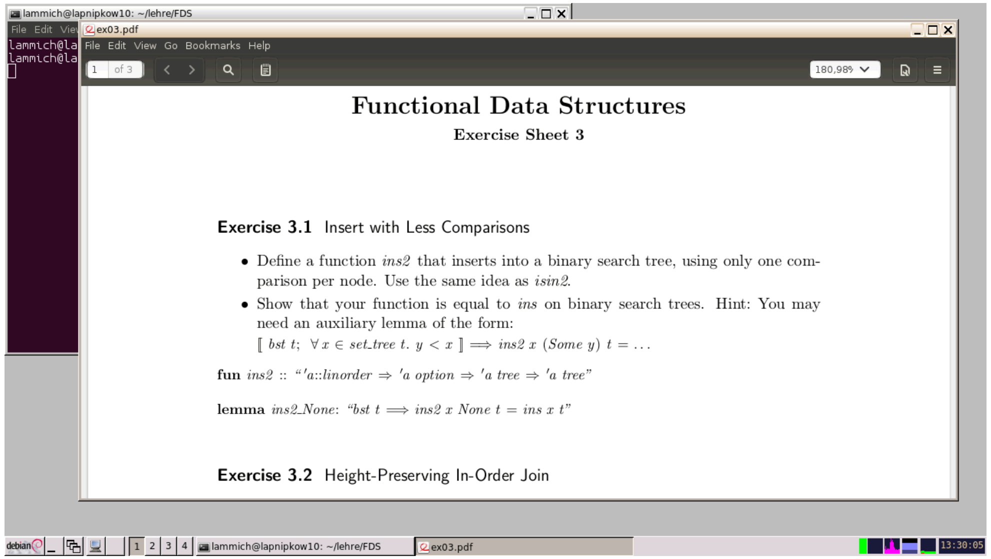
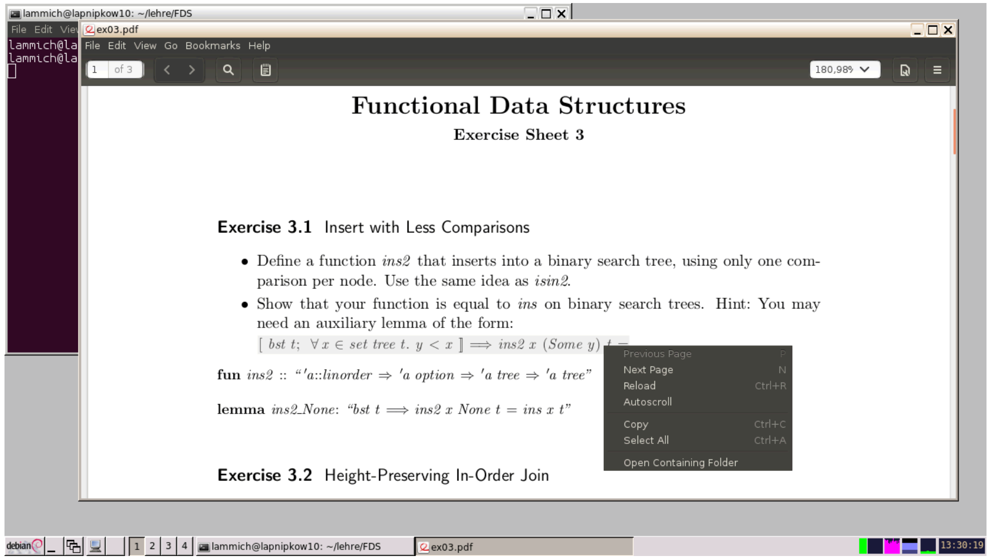

Isabelle2016-1 - tut03.thy (modified)
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ (Leaf, x, Leaf)
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node ()"
end
Inner syntax error
Failed to parse prop

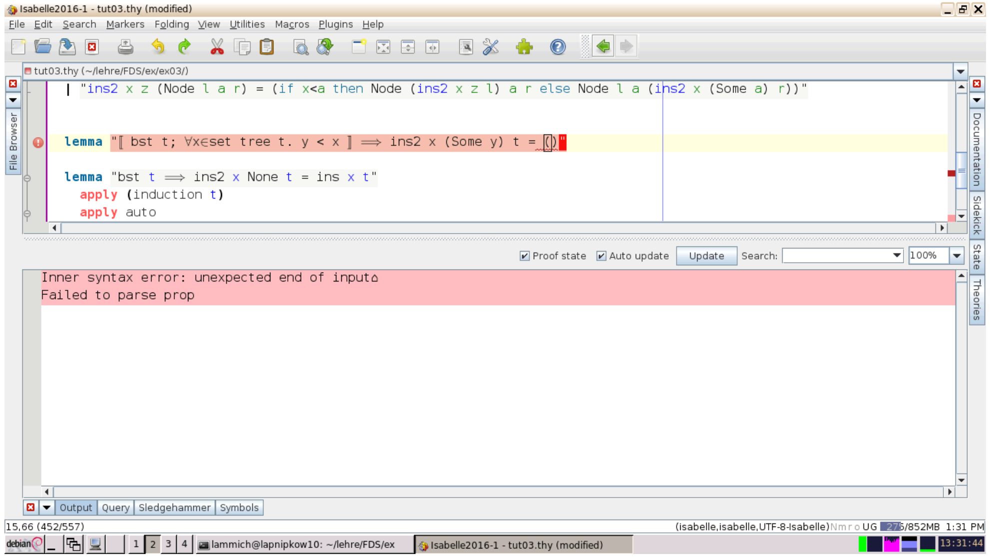

Isabelle2016-1 - tut03.thy
tut03.thy (~/lehre/FDS/ex/ex03/)
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else undefined)"
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> {}"
Output Query Sledgehammer Symbols

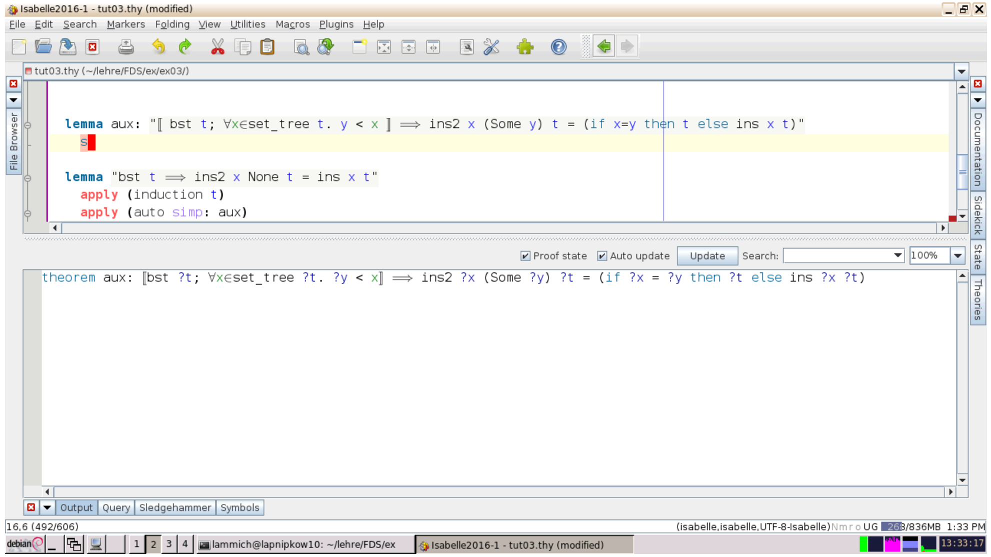

Isabelle2016-1 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/ex/ex03/)
begin
term isin2
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else Node)"
end
Inner syntax error
Failed to parse prop
Output Query Sledgehammer Symbols

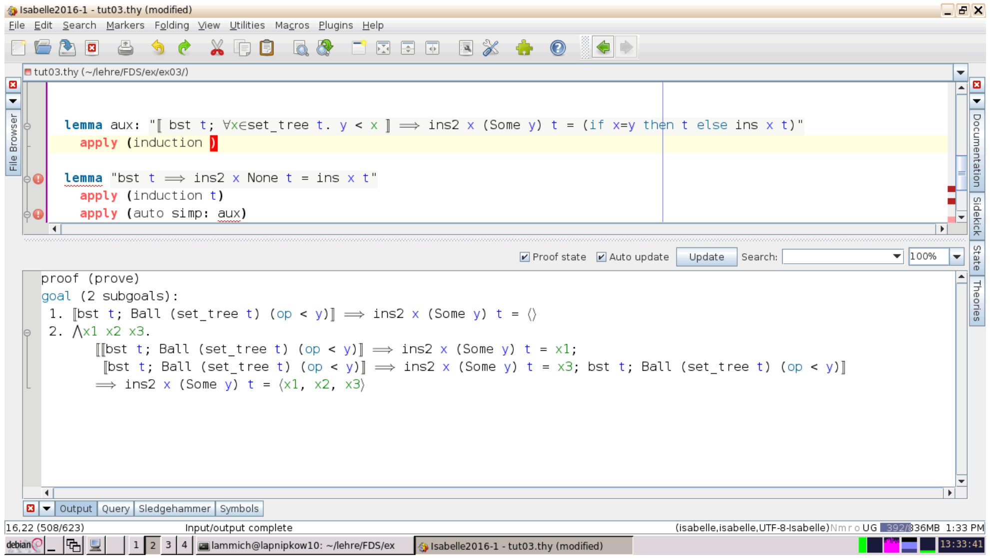

Isabelle2016-1 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/ex/ex03/)
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else Node l a (ins2 x (Some a) r))"
end
consts
ins2 :: "'a ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
Found termination order: "(λp. size (snd (snd p))) <*mlex*> {}"
Output Query Sledgehammer Symbols

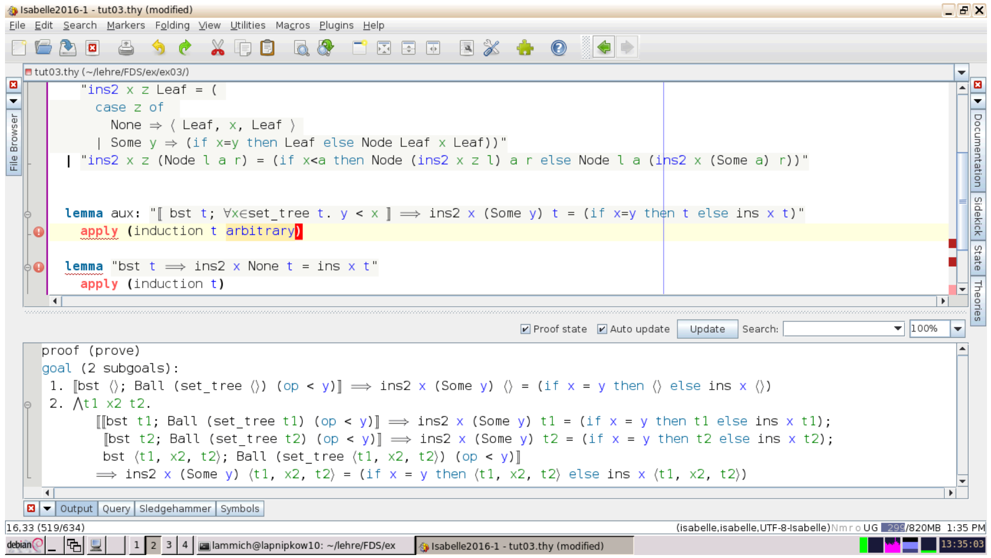

Isabelle2016-1 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/ex/ex03/)
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree" where
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else Node l a (ins2 x (Some a) r))"
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
end
proof (prove)
goal (2 subgoals):
1. bst ⟨⟩ ⟹ ins2 x None ⟨⟩ = ins x ⟨⟩
2. ⋀t1 x2 t2.
⟦bst t1 ⟹ ins2 x None t1 = ins x t1; bst t2 ⟹ ins2 x None t2 = ins x t2; bst ⟨t1, x2, t2⟩⟧
⟹ ins2 x None ⟨t1, x2, t2⟩ = ins x ⟨t1, x2, t2⟩
Output Query Sledgehammer Symbols

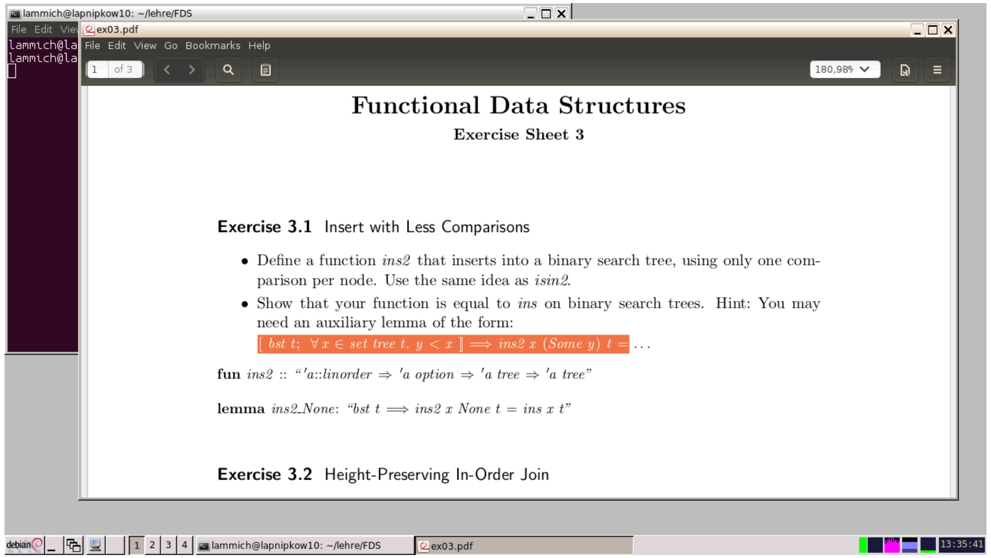

Isabelle2016-1 - tut03.thy (modified)
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else Node l a (ins2 x (Some a) r))"
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
end
proof (prove)
goal (2 subgoals):
1. bst ⟨⟩ ⟹ ins2 x None ⟨⟩ = ins x ⟨⟩
2. ⋀t1 x2 t2.
⟦bst t1 ⟹ ins2 x None t1 = ins x t1; bst t2 ⟹ ins2 x None t2 = ins x t2; bst ⟨t1, x2, t2⟩⟧
⟹ ins2 x None ⟨t1, x2, t2⟩ = ins x ⟨t1, x2, t2⟩

Isabelle2016-1 - tut03.thy
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
apply auto
end
proof (prove)
goal (2 subgoals):
1. ⋀t1 x2 t2.
⟦ins2 x None t1 = ins x t1; ins2 x None t2 = ins x t2; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x; x2 < x; ¬ x < x2⟧
⟹ ins2 x (Some x2) t2 = ins x t2
2. ⋀t1 t2. ⟦ins2 x None t1 = ins x t1; ins2 x None t2 = ins x t2; bst t1; bst t2; ∀xa∈set_tree t1. xa < x;
∀xa∈set_tree t2. x < xa⟧
⟹ ins2 x (Some x) t2 = t2

ex03.pdf
Functional Data Structures
Exercise Sheet 3
Exercise 3.1 Insert with Less Comparisons
• Define a function ins2 that inserts into a binary search tree, using only one comparison per node. Use the same idea as isin2.
• Show that your function is equal to ins on binary search trees. Hint: You may need an auxiliary lemma of the form:
⟦ bst t; ∀x ∈ set_tree t. y < x ⟧ ⟹ ins2 x (Some y) t = ...
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
lemma ins2_None: "bst t ⟹ ins2 x None t = ins x t"
Exercise 3.2 Height-Preserving In-Order Join

ex03.pdf
Functional Data Structures
Exercise Sheet 3
Exercise 3.1 Insert with Less Comparisons
• Define a function ins2 that inserts into a binary search tree, using only one comparison per node. Use the same idea as isin2.
• Show that your function is equal to ins on binary search trees. Hint: You may need an auxiliary lemma of the form:
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
lemma ins2_None: "bst t ⟹ ins2 x None t = ins x t"
Exercise 3.2 Height-Preserving In-Order Join
Previous Page
Next Page
Reload
Autoscroll
Copy
Select All
Open Containing Folder

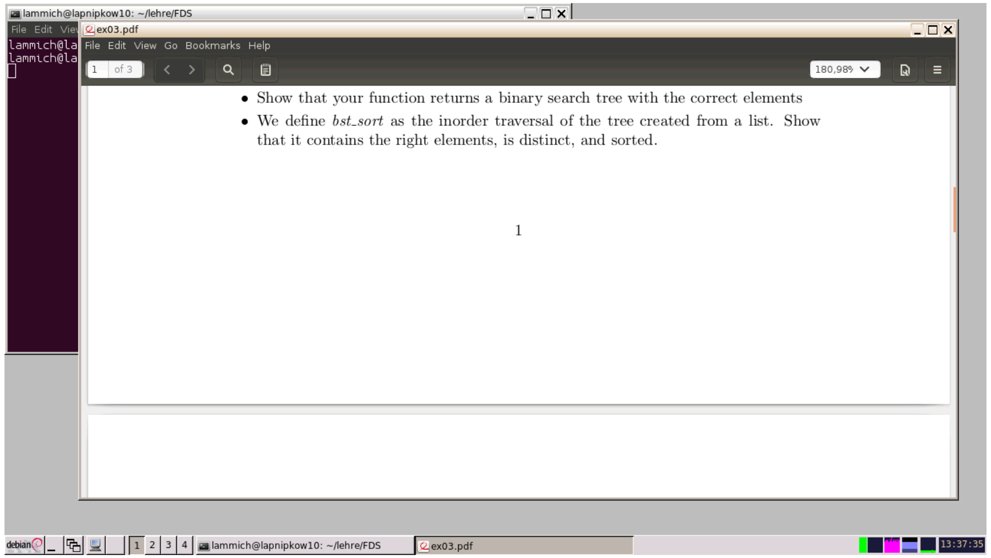

Isabelle2016-1 - tut03.thy (modified)
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else Node l a (ins2 x (Some a) r))"
lemma "⟦ bst t; ∀x∈set tree t. y < x ⟧ ⟹ ins2 x (Some y) t = ⟨⟩"
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
apply auto
Inner syntax error: unexpected end of input
Failed to parse prop

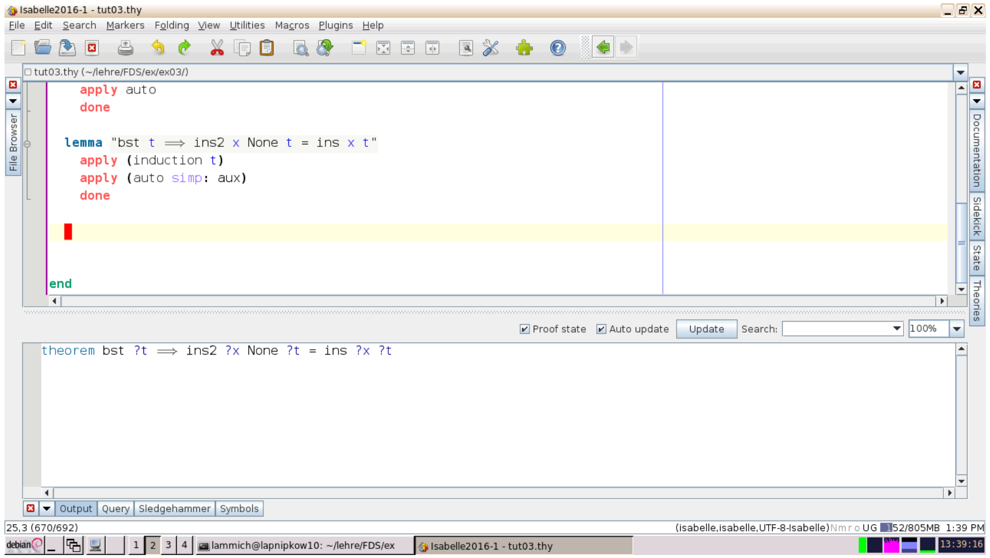

Isabelle2016-1 - tut03.thy (modified)
lemma aux: "⟦ bst t; ∀x∈set_tree t. y < x ⟧ ⟹ ins2 x (Some y) t = (if x=y then t else ins x t)"
s
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
apply (auto simp: aux)
theorem aux: ⟦bst ?t; ∀x∈set_tree ?t. ?y < x⟧ ⟹ ins2 ?x (Some ?y) ?t = (if ?x = ?y then ?t else ins ?x ?t)

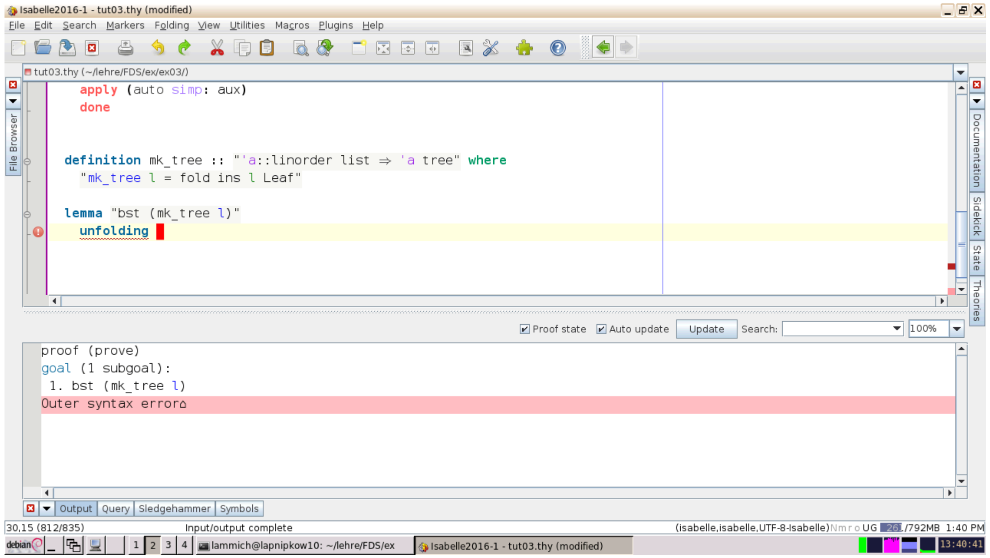

Isabelle2016-1 - tut03.thy (modified)
lemma aux: "⟦ bst t; ∀x∈set_tree t. y < x ⟧ ⟹ ins2 x (Some y) t = (if x=y then t else ins x t)"
apply (induction t)
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
apply (auto simp: aux)
proof (prove)
goal (2 subgoals):
1. ⟦bst t; Ball (set_tree t) (op < y)⟧ ⟹ ins2 x (Some y) t = ⟨⟩
2. ⋀x1 x2 x3.
⟦⟦bst t; Ball (set_tree t) (op < y)⟧ ⟹ ins2 x (Some y) t = x1;
⟦bst t; Ball (set_tree t) (op < y)⟧ ⟹ ins2 x (Some y) t = x3; bst t; Ball (set_tree t) (op < y)⟧
⟹ ins2 x (Some y) t = ⟨x1, x2, x3⟩

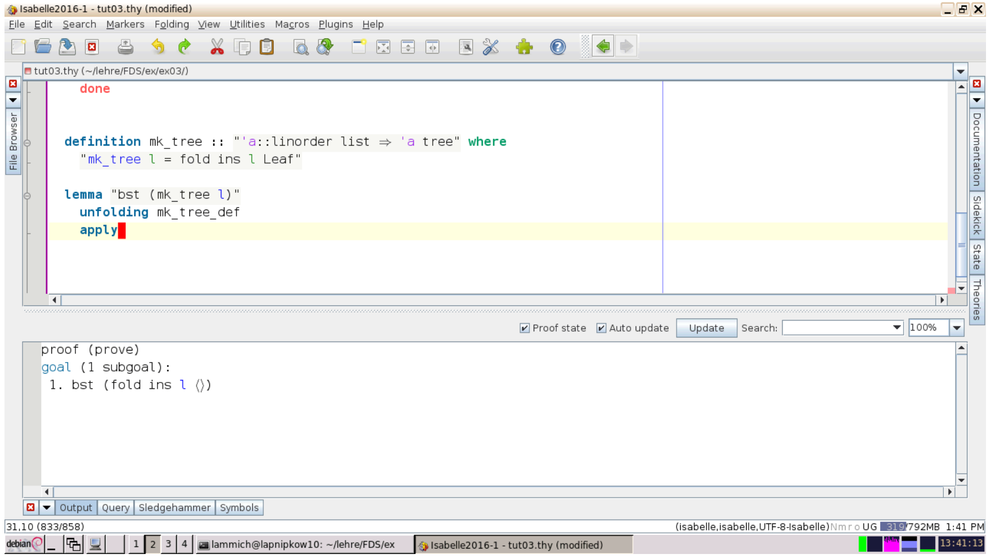

Isabelle2016-1 - tut03.thy (modified)
"ins2 x z Leaf = (
case z of
None ⇒ ⟨ Leaf, x, Leaf ⟩
| Some y ⇒ (if x=y then Leaf else Node Leaf x Leaf))"
| "ins2 x z (Node l a r) = (if x<a then Node (ins2 x z l) a r else Node l a (ins2 x (Some a) r))"
lemma aux: "⟦ bst t; ∀x∈set_tree t. y < x ⟧ ⟹ ins2 x (Some y) t = (if x=y then t else ins x t)"
apply (induction t arbitrary)
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
proof (prove)
goal (2 subgoals):
1. ⟦bst ⟨⟩; Ball (set_tree ⟨⟩) (op < y)⟧ ⟹ ins2 x (Some y) ⟨⟩ = (if x = y then ⟨⟩ else ins x ⟨⟩)
2. ⋀t1 x2 t2.
⟦⟦bst t1; Ball (set_tree t1) (op < y)⟧ ⟹ ins2 x (Some y) t1 = (if x = y then t1 else ins x t1);
⟦bst t2; Ball (set_tree t2) (op < y)⟧ ⟹ ins2 x (Some y) t2 = (if x = y then t2 else ins x t2);
bst ⟨t1, x2, t2⟩; Ball (set_tree ⟨t1, x2, t2⟩) (op < y)⟧
⟹ ins2 x (Some y) ⟨t1, x2, t2⟩ = (if x = y then ⟨t1, x2, t2⟩ else ins x ⟨t1, x2, t2⟩)

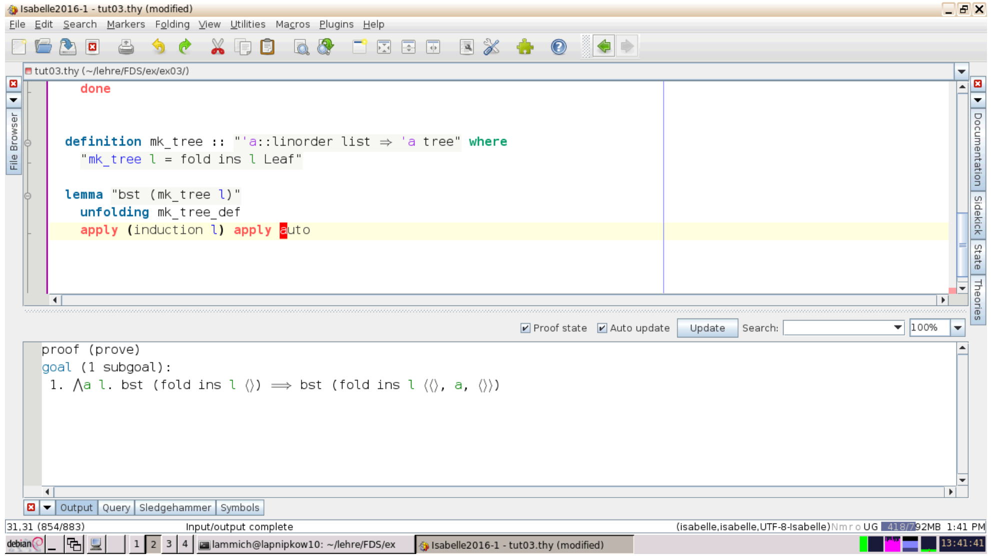

lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
File Edit View Go Bookmarks Help
Functional Data Structures
Exercise Sheet 3
Exercise 3.1 Insert with Less Comparisons
• Define a function ins2 that inserts into a binary search tree, using only one comparison per node. Use the same idea as isin2.
• Show that your function is equal to ins on binary search trees. Hint: You may need an auxiliary lemma of the form:
⟦ bst t; ∀x ∈ set tree t. y < x ⟧ ⟹ ins2 x (Some y) t = . . .
fun ins2 :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"
lemma ins2_None: "bst t ⟹ ins2 x None t = ins x t"
Exercise 3.2 Height-Preserving In-Order Join
13:35:41

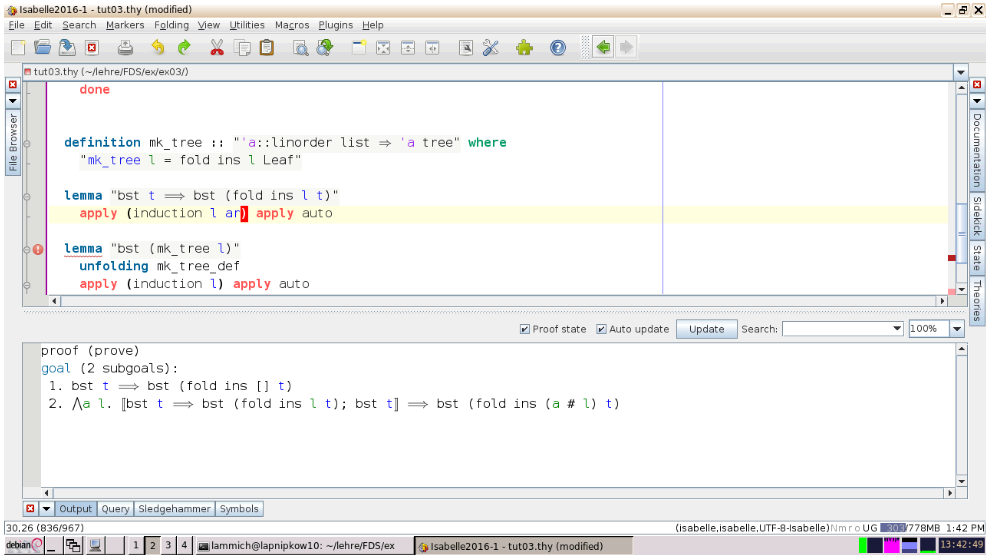

lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
File Edit View Go Bookmarks Help
• Show that your function returns a binary search tree with the correct elements
• We define bst_sort as the inorder traversal of the tree created from a list. Show that it contains the right elements, is distinct, and sorted.
1
13:37:35

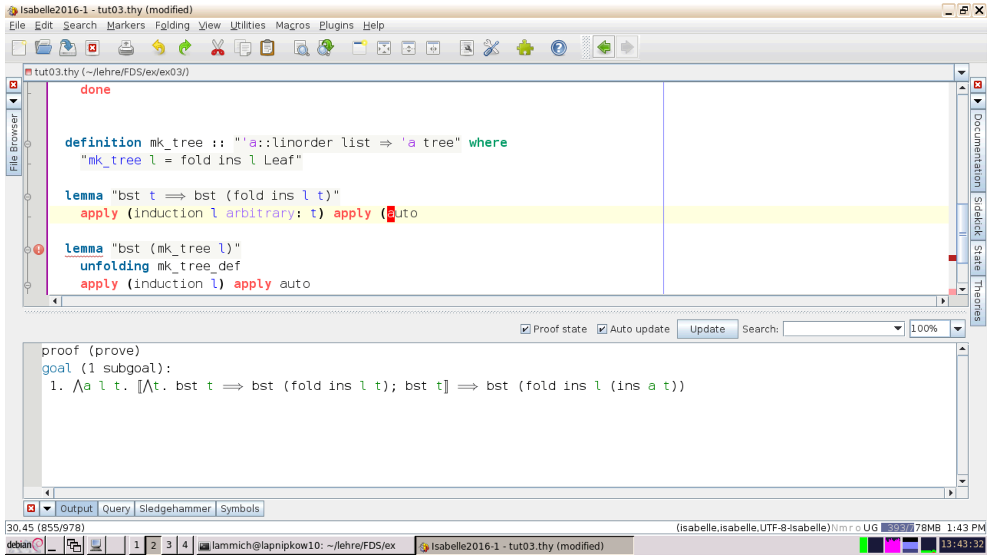

Isabelle2016-1 - tut03.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut03.thy (~/lehre/FDS/ex/ex03/)
apply auto
done
lemma "bst t ⟹ ins2 x None t = ins x t"
apply (induction t)
apply (auto simp: aux)
done
end
Proof state Auto update Update Search: 100%
theorem bst ?t ⟹ ins2 ?x None ?t = ins ?x ?t
Output Query Sledgehammer Symbols
25,3 (670/692)
13:39:16

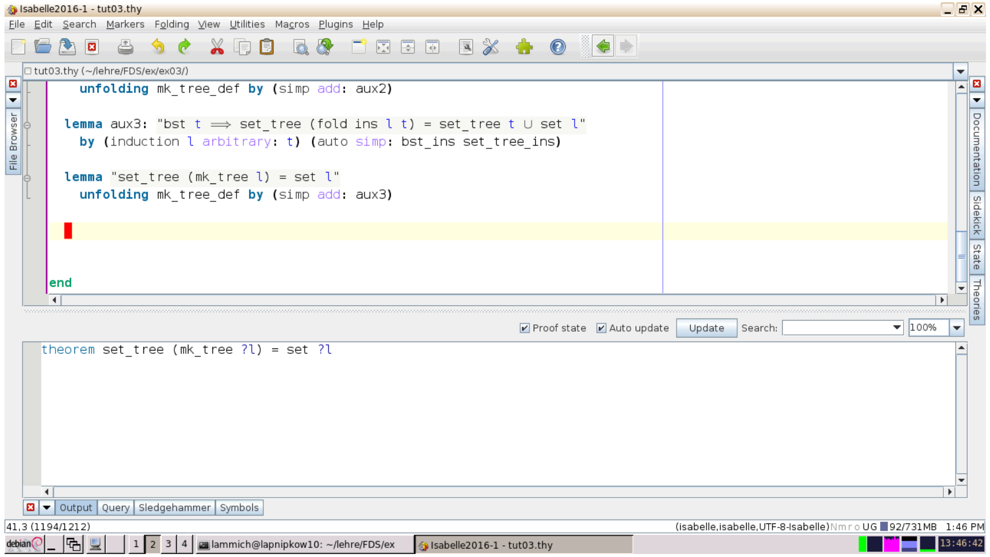

Isabelle2016-1 - tut03.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut03.thy (~/lehre/FDS/ex/ex03/)
apply (auto simp: aux)
done
definition mk_tree :: "'a::linorder list ⇒ 'a tree" where
"mk_tree l = fold ins l Leaf"
lemma "bst (mk_tree l)"
unfolding
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. bst (mk_tree l)
Outer syntax error
Output Query Sledgehammer Symbols
30,15 (812/835)
Input/output complete
13:40:41

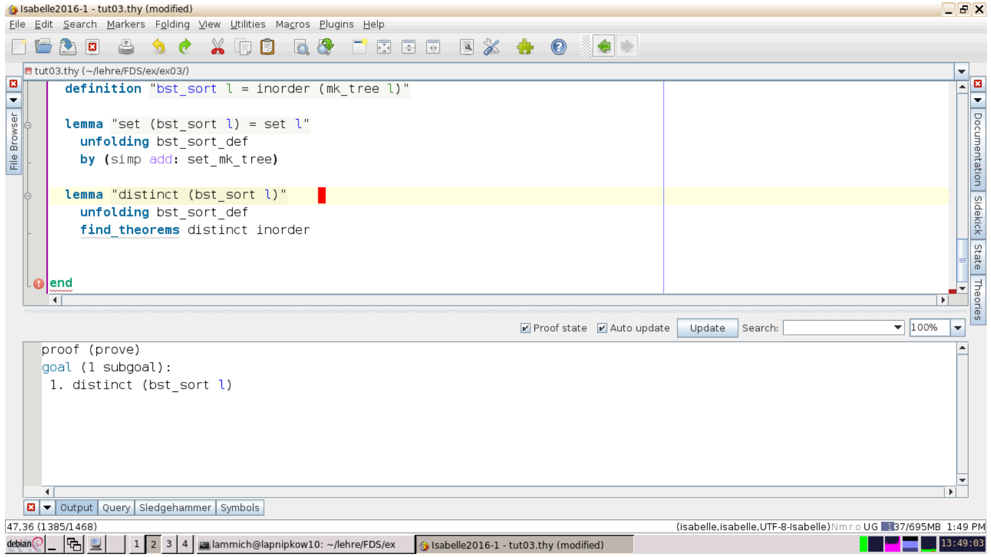

Isabelle2016-1 - tut03.thy (modified)
done
definition mk_tree :: "'a::linorder list ⇒ 'a tree" where
"mk_tree l = fold ins l Leaf"
lemma "bst (mk_tree l)"
unfolding mk_tree_def
apply
proof (prove)
goal (1 subgoal):
1. bst (fold ins l ⟨⟩)

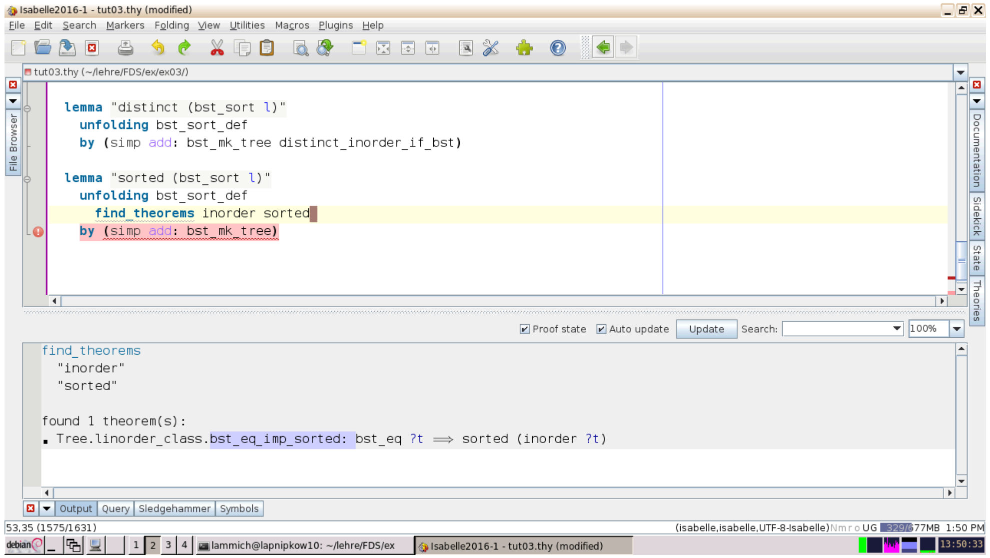

Isabelle2016-1 - tut03.thy (modified)
done
definition mk_tree :: "'a::linorder list ⇒ 'a tree" where
"mk_tree l = fold ins l Leaf"
lemma "bst (mk_tree l)"
unfolding mk_tree_def
apply (induction l) apply auto
proof (prove)
goal (1 subgoal):
1. ⋀a l. bst (fold ins l ⟨⟩) ⟹ bst (fold ins l ⟨⟨⟩, a, ⟨⟩⟩)

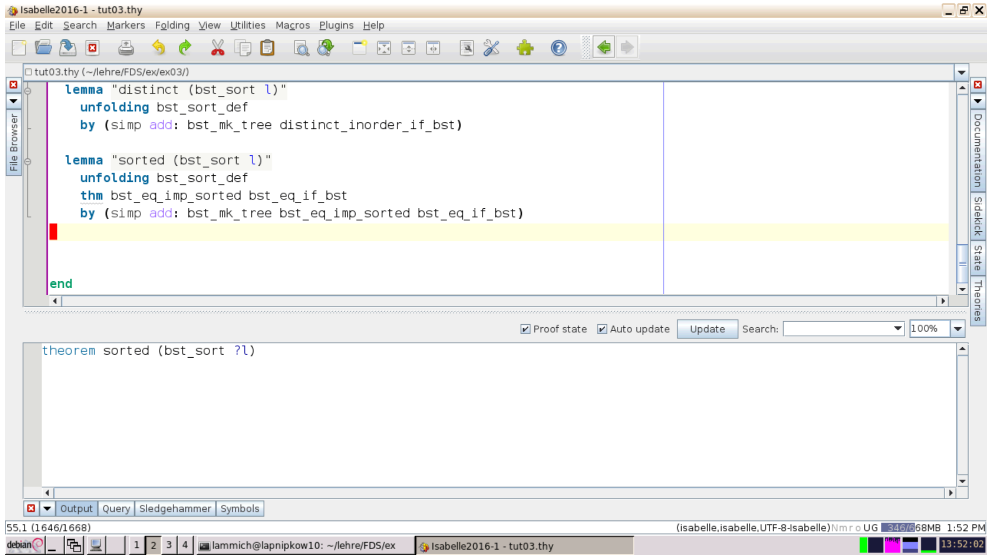

Isabelle2016-1 - tut03.thy (modified)
done
definition mk_tree :: "'a::linorder list ⇒ 'a tree" where
"mk_tree l = fold ins l Leaf"
lemma "bst t ⟹ bst (fold ins l t)"
apply (induction l ar apply auto
lemma "bst (mk_tree l)"
unfolding mk_tree_def
apply (induction l) apply auto
proof (prove)
goal (2 subgoals):
1. bst t ⟹ bst (fold ins [] t)
2. ⋀a l. ⟦bst t ⟹ bst (fold ins l t); bst t⟧ ⟹ bst (fold ins (a # l) t)

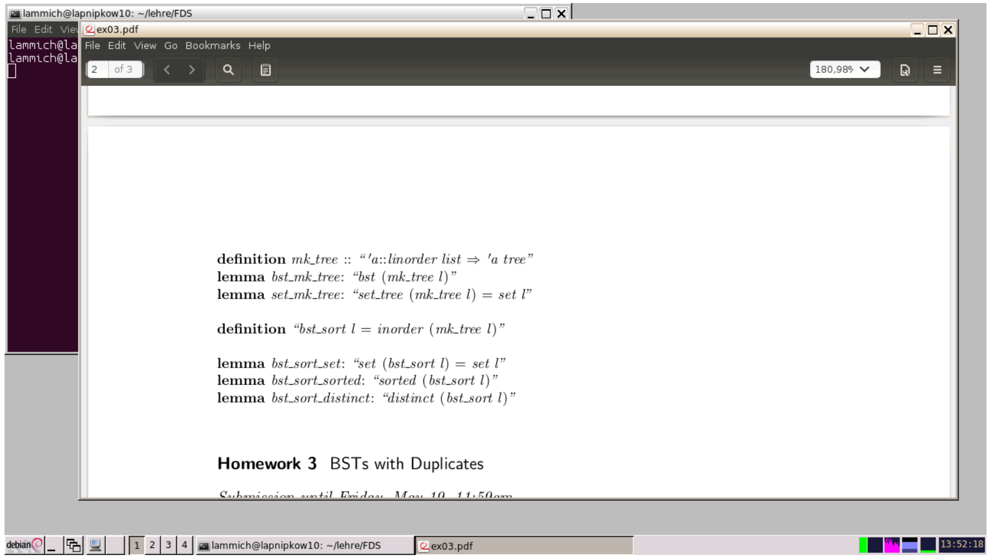

Isabelle2016-1 - tut03.thy (modified)
done
definition mk_tree :: "'a::linorder list ⇒ 'a tree" where
"mk_tree l = fold ins l Leaf"
lemma "bst t ⟹ bst (fold ins l t)"
apply (induction l arbitrary: t) apply (auto
lemma "bst (mk_tree l)"
unfolding mk_tree_def
apply (induction l) apply auto
proof (prove)
goal (1 subgoal):
1. ⋀a l t. ⟦⋀t. bst t ⟹ bst (fold ins l t); bst t⟧ ⟹ bst (fold ins l (ins a t))


Isabelle2016-1 - tut03.thy
tut03.thy (~/lehre/FDS/ex/ex03/)
unfolding mk_tree_def by (simp add: aux2)
lemma aux3: "bst t ⟹ set_tree (fold ins l t) = set_tree t ∪ set l"
by (induction l arbitrary: t) (auto simp: bst_ins set_tree_ins)
lemma "set_tree (mk_tree l) = set l"
unfolding mk_tree_def by (simp add: aux3)
end
theorem set_tree (mk_tree ?l) = set ?l
41,3 (1194/1212)


Isabelle2016-1 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/ex/ex03/)
definition "bst_sort l = inorder (mk_tree l)"
lemma "set (bst_sort l) = set l"
unfolding bst_sort_def
by (simp add: set_mk_tree)
lemma "distinct (bst_sort l)"
unfolding bst_sort_def
find_theorems distinct inorder
end
proof (prove)
goal (1 subgoal):
1. distinct (bst_sort l)
47,36 (1385/1468)


Isabelle2016-1 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/ex/ex03/)
lemma "distinct (bst_sort l)"
unfolding bst_sort_def
by (simp add: bst_mk_tree distinct_inorder_if_bst)
lemma "sorted (bst_sort l)"
unfolding bst_sort_def
find_theorems inorder sorted
by (simp add: bst_mk_tree)
find_theorems
"inorder"
"sorted"
found 1 theorem(s):
Tree.linorder_class.bst_eq_imp_sorted: bst_eq ?t ⟹ sorted (inorder ?t)
53,35 (1575/1631)


Isabelle2016-1 - tut03.thy
tut03.thy (~/lehre/FDS/ex/ex03/)
lemma "distinct (bst_sort l)"
unfolding bst_sort_def
by (simp add: bst_mk_tree distinct_inorder_if_bst)
lemma "sorted (bst_sort l)"
unfolding bst_sort_def
thm bst_eq_imp_sorted bst_eq_if_bst
by (simp add: bst_mk_tree bst_eq_imp_sorted bst_eq_if_bst)
end
theorem sorted (bst_sort ?l)
55,1 (1646/1668)

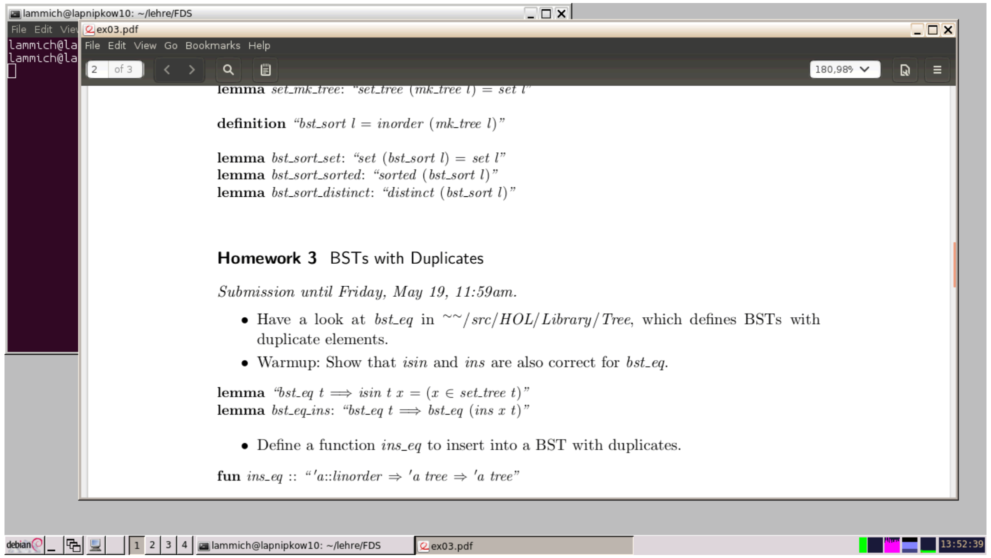


definition *mk_tree* :: "'a::linorder list ⇒ 'a tree"
lemma *bst_mk_tree*: "bst (mk_tree l)"
lemma *set_mk_tree*: "set_tree (mk_tree l) = set l"

definition "bst_sort l = inorder (mk_tree l)"

lemma *bst_sort_set*: "set (bst_sort l) = set l"
lemma *bst_sort_sorted*: "sorted (bst_sort l)"
lemma *bst_sort_distinct*: "distinct (bst_sort l)"

Homework 3 BSTs with Duplicates



definition "bst_sort l = inorder (mk_tree l)"

lemma *bst_sort_set*: "set (bst_sort l) = set l"
lemma *bst_sort_sorted*: "sorted (bst_sort l)"
lemma *bst_sort_distinct*: "distinct (bst_sort l)"

Homework 3 BSTs with Duplicates

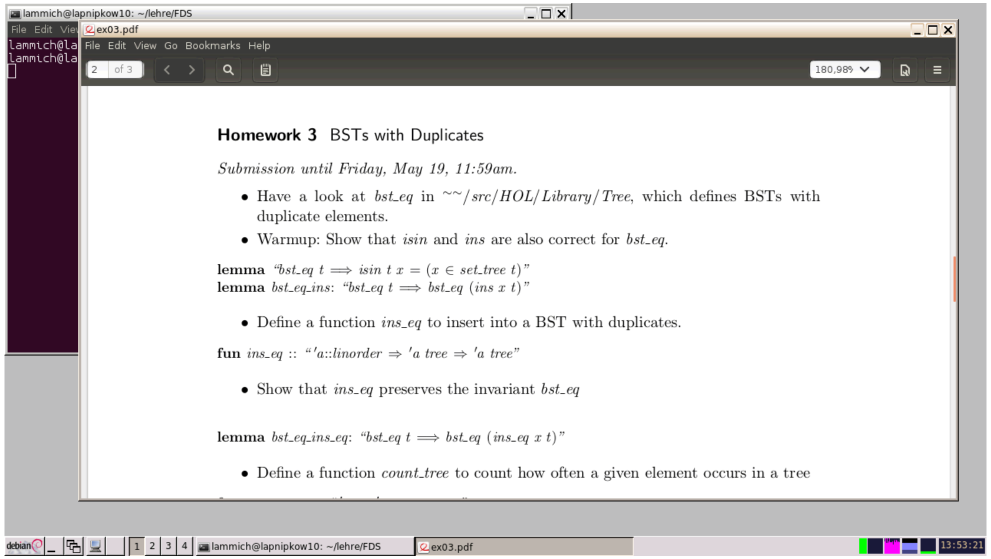
Submission until Friday, May 19, 11:59am.

- Have a look at *bst_eq* in *~~/src/HOL/Library/Tree*, which defines BSTs with duplicate elements.
- Warmup: Show that *isin* and *ins* are also correct for *bst_eq*.

lemma "bst_eq t ⟹ isin t x = (x ∈ set_tree t)"
lemma *bst_eq_ins*: "bst_eq t ⟹ bst_eq (ins x t)"

- Define a function *ins_eq* to insert into a BST with duplicates.

fun *ins_eq* :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"



Homework 3 BSTs with Duplicates

Submission until Friday, May 19, 11:59am.

- Have a look at *bst_eq* in *~~/src/HOL/Library/Tree*, which defines BSTs with duplicate elements.
- Warmup: Show that *isin* and *ins* are also correct for *bst_eq*.

lemma "bst_eq t ⟹ isin t x = (x ∈ set_tree t)"
lemma *bst_eq_ins*: "bst_eq t ⟹ bst_eq (ins x t)"

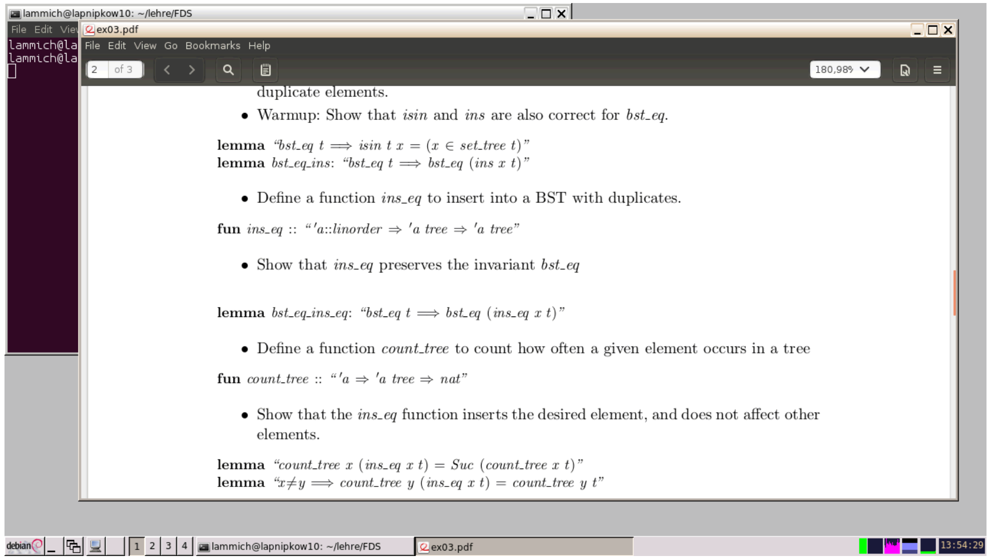
- Define a function *ins_eq* to insert into a BST with duplicates.

fun *ins_eq* :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"

- Show that *ins_eq* preserves the invariant *bst_eq*

lemma *bst_eq_ins_eq*: "bst_eq t ⟹ bst_eq (ins_eq x t)"

- Define a function *count_tree* to count how often a given element occurs in a tree



duplicate elements.
- Warmup: Show that *isin* and *ins* are also correct for *bst_eq*.

lemma "bst_eq t ⟹ isin t x = (x ∈ set_tree t)"
lemma *bst_eq_ins*: "bst_eq t ⟹ bst_eq (ins x t)"

- Define a function *ins_eq* to insert into a BST with duplicates.

fun *ins_eq* :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"

- Show that *ins_eq* preserves the invariant *bst_eq*

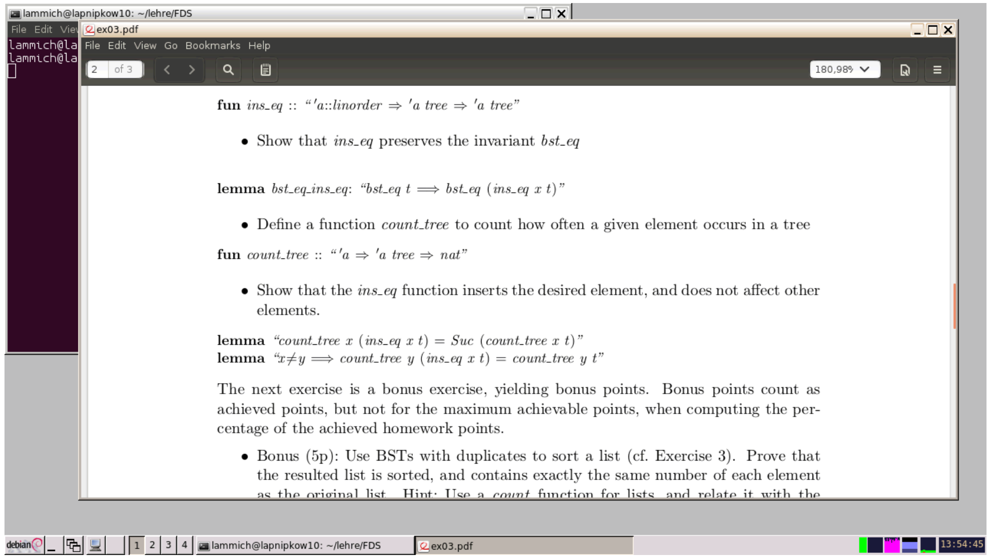
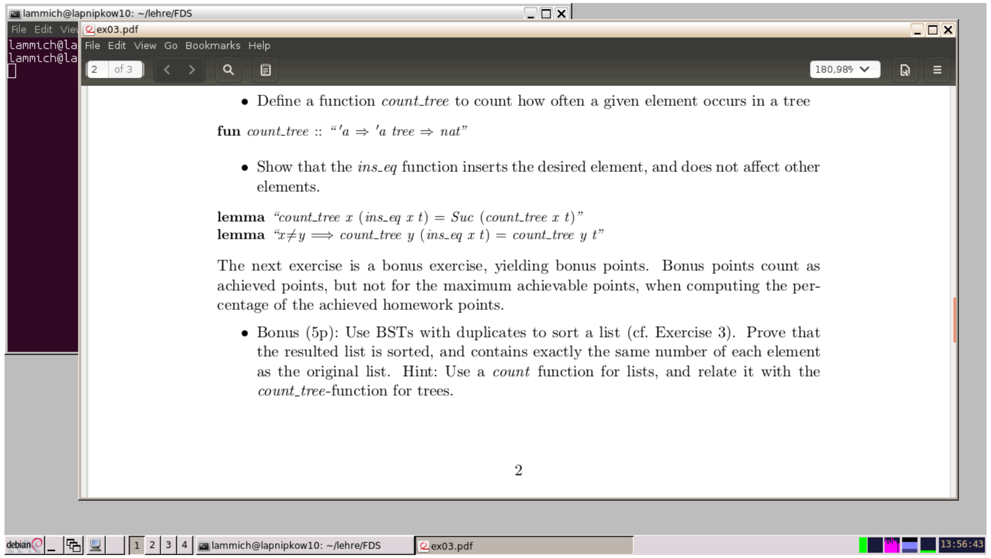
lemma *bst_eq_ins_eq*: "bst_eq t ⟹ bst_eq (ins_eq x t)"

- Define a function *count_tree* to count how often a given element occurs in a tree

fun *count_tree* :: "'a ⇒ 'a tree ⇒ nat"

- Show that the *ins_eq* function inserts the desired element, and does not affect other elements.

lemma "count_tree x (ins_eq x t) = Suc (count_tree x t)"
lemma "x≠y ⟹ count_tree y (ins_eq x t) = count_tree y t"


lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
File Edit View Go Bookmarks Help
2 of 3
180,98%
fun ins_eq :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
• Show that ins_eq preserves the invariant bst_eq
lemma bst_eq_ins_eq: "bst_eq t ⟹ bst_eq (ins_eq x t)"
• Define a function count_tree to count how often a given element occurs in a tree
fun count_tree :: "'a ⇒ 'a tree ⇒ nat"
• Show that the ins_eq function inserts the desired element, and does not affect other elements.
lemma "count_tree x (ins_eq x t) = Suc (count_tree x t)"
lemma "x≠y ⟹ count_tree y (ins_eq x t) = count_tree y t"
The next exercise is a bonus exercise, yielding bonus points. Bonus points count as achieved points, but not for the maximum achievable points, when computing the percentage of the achieved homework points.
• Bonus (5p): Use BSTs with duplicates to sort a list (cf. Exercise 3). Prove that the resulted list is sorted, and contains exactly the same number of each element
debian
1 2 3 4
lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
13:54:45

lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
File Edit View Go Bookmarks Help
2 of 3
180,98%
• Define a function count_tree to count how often a given element occurs in a tree
fun count_tree :: "'a ⇒ 'a tree ⇒ nat"
• Show that the ins_eq function inserts the desired element, and does not affect other elements.
lemma "count_tree x (ins_eq x t) = Suc (count_tree x t)"
lemma "x≠y ⟹ count_tree y (ins_eq x t) = count_tree y t"
The next exercise is a bonus exercise, yielding bonus points. Bonus points count as achieved points, but not for the maximum achievable points, when computing the percentage of the achieved homework points.
• Bonus (5p): Use BSTs with duplicates to sort a list (cf. Exercise 3). Prove that the resulted list is sorted, and contains exactly the same number of each element as the original list. Hint: Use a count function for lists, and relate it with the count_tree-function for trees.
2
debian
1 2 3 4
lammich@lapnipkow10: ~/lehre/FDS
ex03.pdf
13:56:43