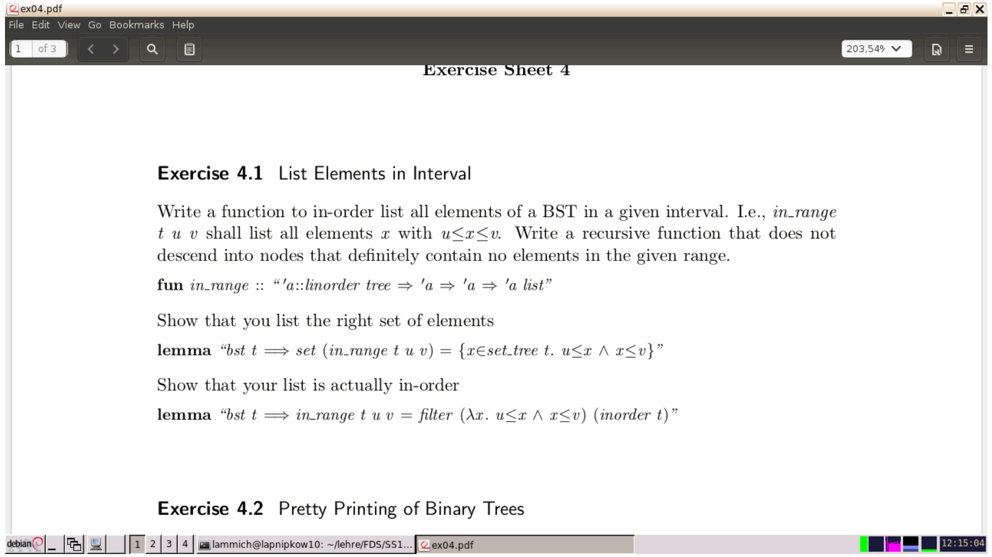
Script generated by TTT

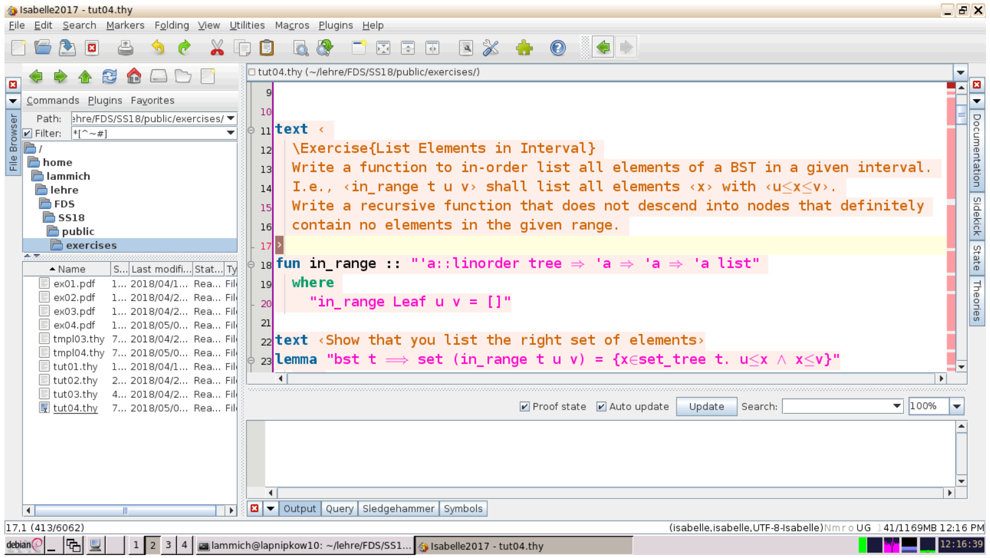
Title: Lammich: FDS Tutorial (04.05.2018)

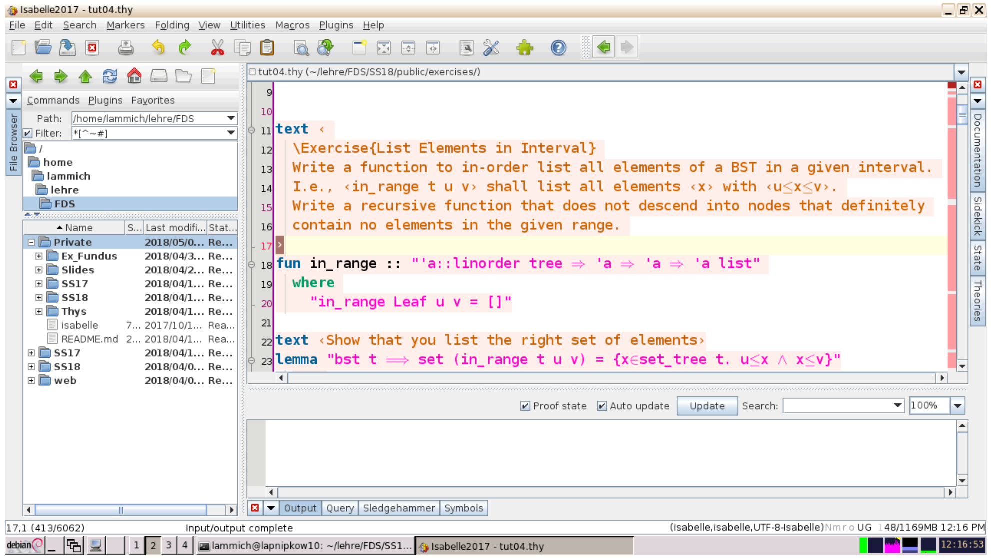
Date: Fri May 04 12:15:04 CEST 2018

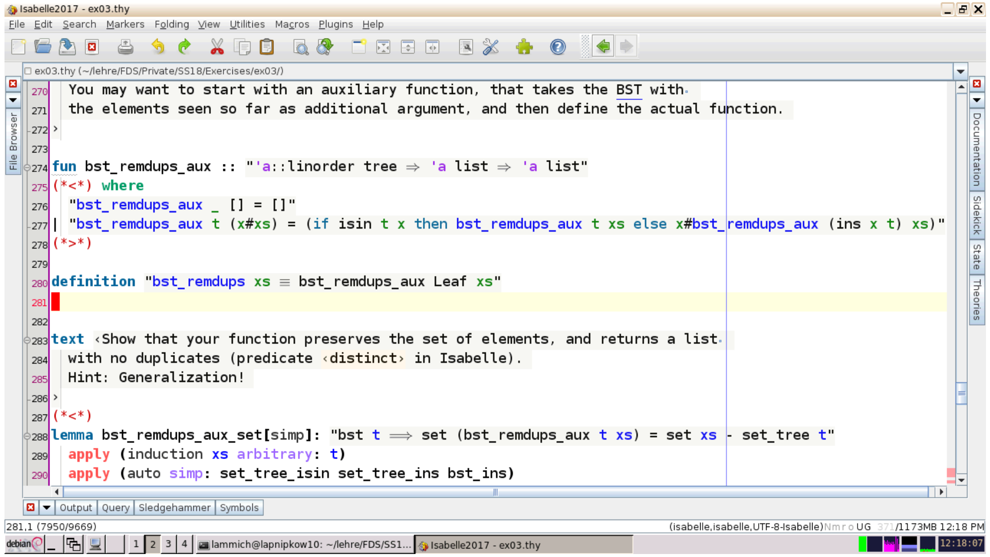
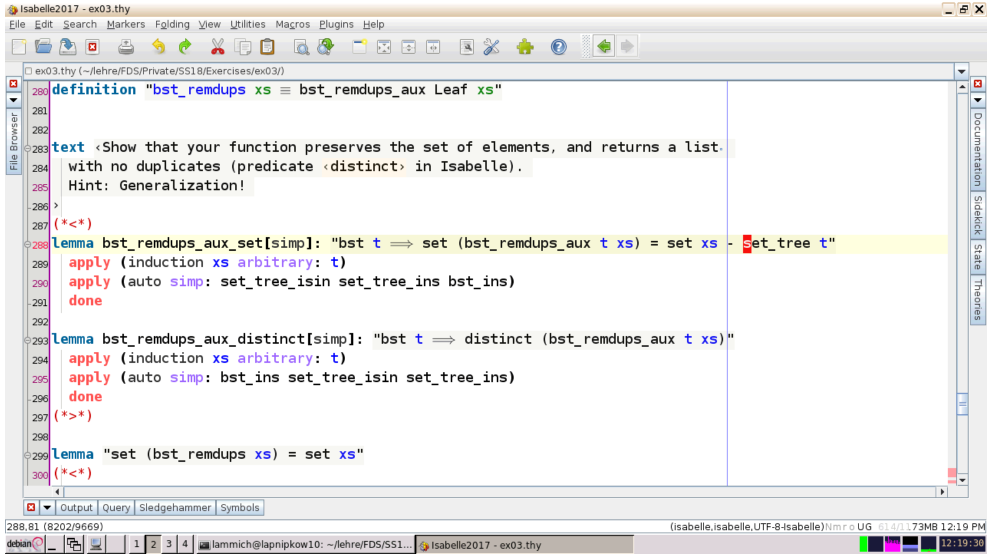
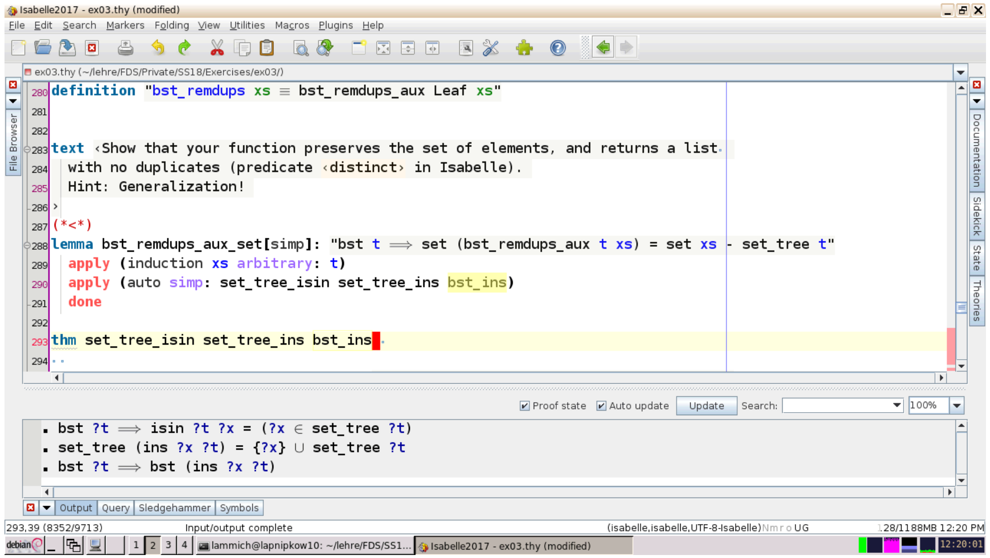
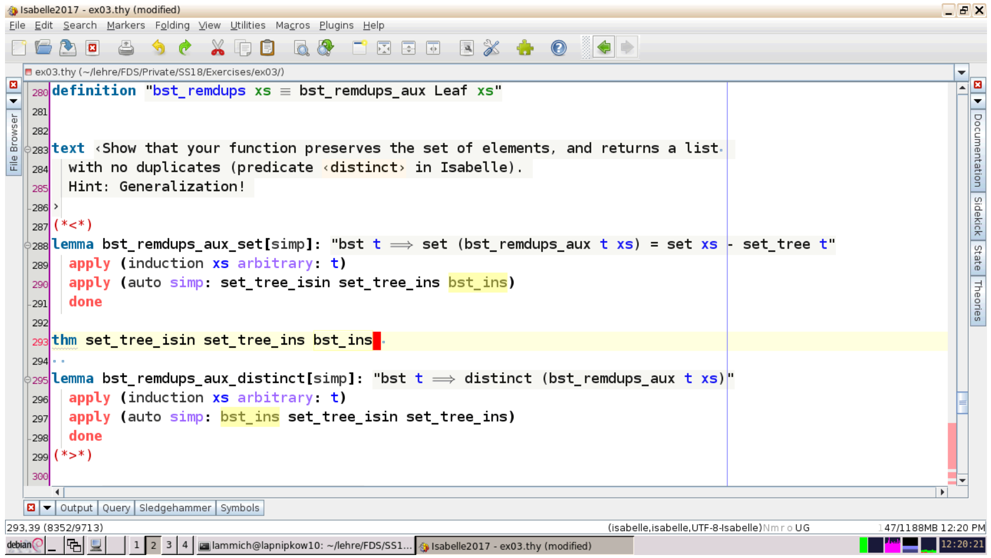
Duration: 109:05 min

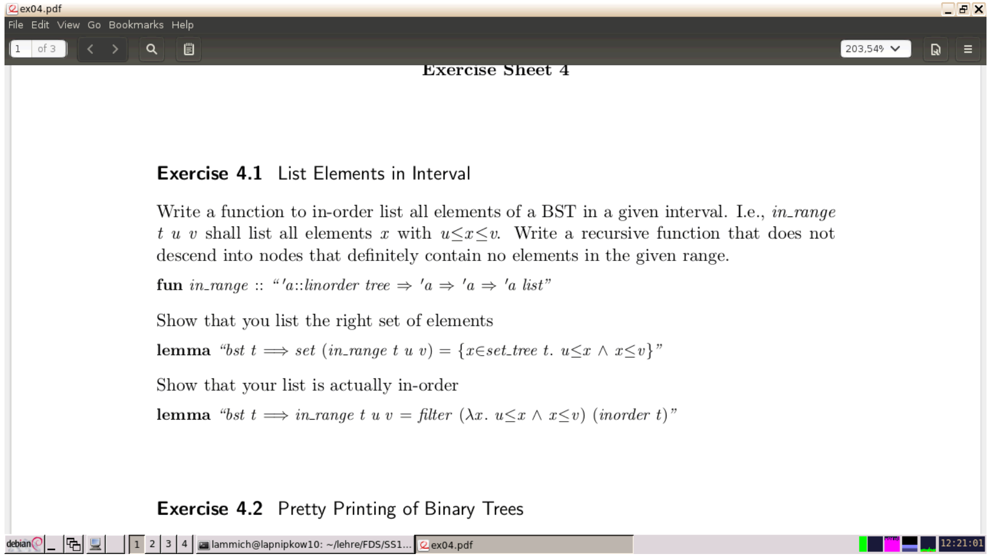
Pages: 108

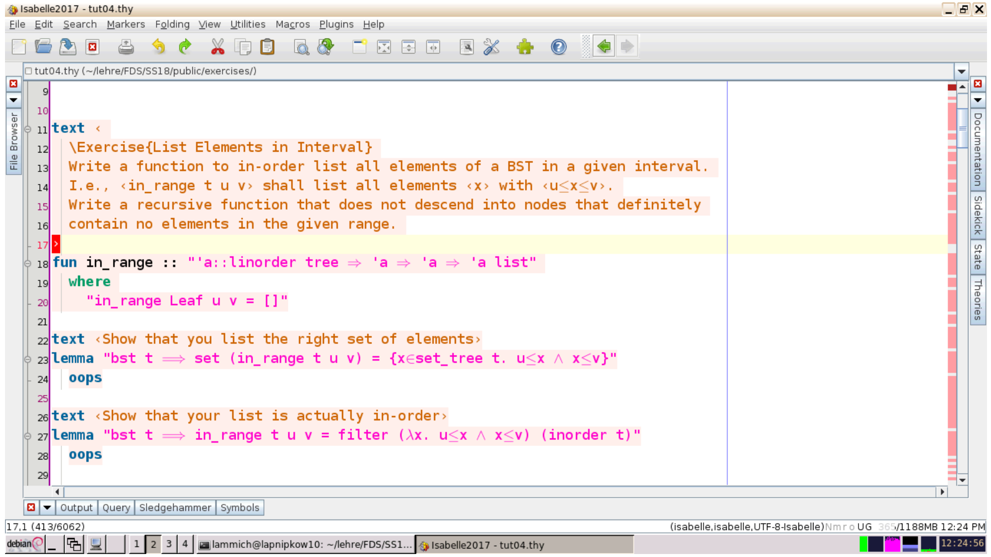


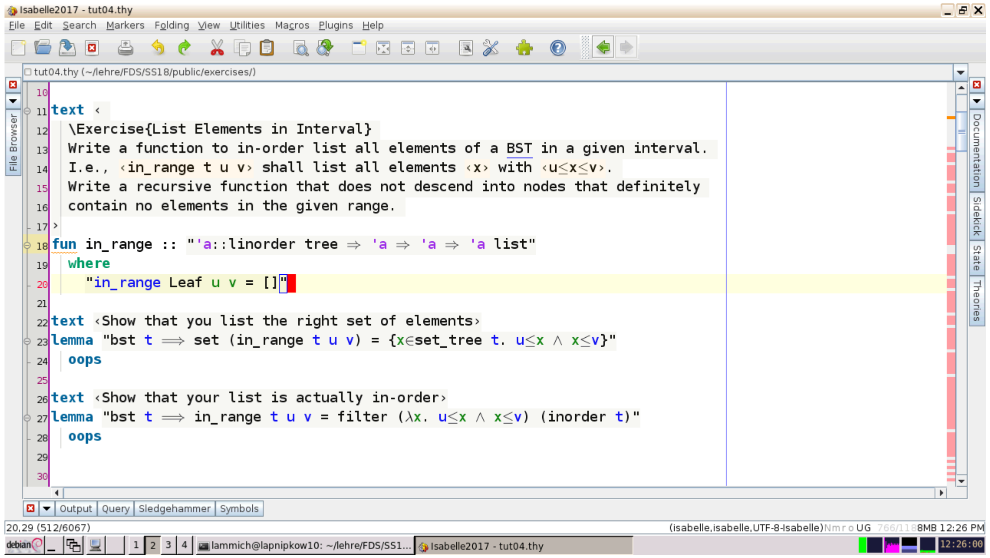


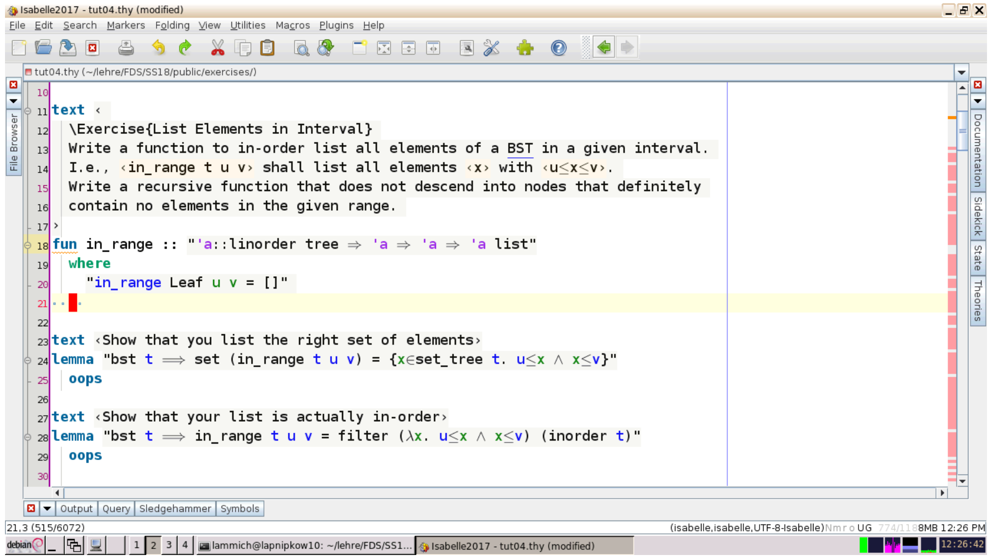


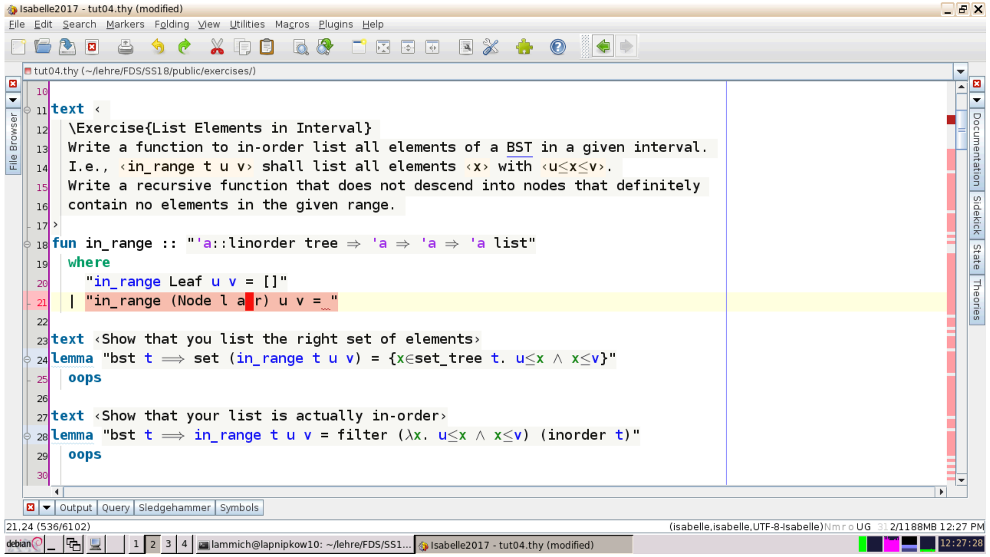

Isabelle2017 - ex03.thy
You may want to start with an auxiliary function, that takes the BST with
the elements seen so far as additional argument, and then define the actual function.
fun bst_remdups_aux :: "'a::linorder tree ⇒ 'a list ⇒ 'a list"
(*<*) where
"bst_remdups_aux _ [] = []"
| "bst_remdups_aux t (x#xs) = (if isin t x then bst_remdups_aux t xs else x#bst_remdups_aux (ins x t) xs)"
(*>*)
definition "bst_remdups xs ≡ bst_remdups_aux Leaf xs"
text ‹Show that your function preserves the set of elements, and returns a list
with no duplicates (predicate ‹distinct› in Isabelle).
Hint: Generalization!
›
(*<*)
lemma bst_remdups_aux_set[simp]: "bst t ⟹ set (bst_remdups_aux t xs) = set xs - set_tree t"
apply (induction xs arbitrary: t)
apply (auto simp: set_tree_isin set_tree_ins bst_ins)

Isabelle2017 - ex03.thy
definition "bst_remdups xs ≡ bst_remdups_aux Leaf xs"
text ‹Show that your function preserves the set of elements, and returns a list
with no duplicates (predicate ‹distinct› in Isabelle).
Hint: Generalization!
›
(*<*)
lemma bst_remdups_aux_set[simp]: "bst t ⟹ set (bst_remdups_aux t xs) = set xs - set_tree t"
apply (induction xs arbitrary: t)
apply (auto simp: set_tree_isin set_tree_ins bst_ins)
done
lemma bst_remdups_aux_distinct[simp]: "bst t ⟹ distinct (bst_remdups_aux t xs)"
apply (induction xs arbitrary: t)
apply (auto simp: bst_ins set_tree_isin set_tree_ins)
done
(*>*)
lemma "set (bst_remdups xs) = set xs"
(*<*)

Isabelle2017 - ex03.thy (modified)
definition "bst_remdups xs ≡ bst_remdups_aux Leaf xs"
text ‹Show that your function preserves the set of elements, and returns a list
with no duplicates (predicate ‹distinct› in Isabelle).
Hint: Generalization!
›
(*<*)
lemma bst_remdups_aux_set[simp]: "bst t ⟹ set (bst_remdups_aux t xs) = set xs - set_tree t"
apply (induction xs arbitrary: t)
apply (auto simp: set_tree_isin set_tree_ins bst_ins)
done
thm set_tree_isin set_tree_ins bst_ins
bst ?t ⟹ isin ?t ?x = (?x ∈ set_tree ?t)
set_tree (ins ?x ?t) = {?x} ∪ set_tree ?t
bst ?t ⟹ bst (ins ?x ?t)

Isabelle2017 - ex03.thy (modified)
definition "bst_remdups xs ≡ bst_remdups_aux Leaf xs"
text ‹Show that your function preserves the set of elements, and returns a list
with no duplicates (predicate ‹distinct› in Isabelle).
Hint: Generalization!
›
(*<*)
lemma bst_remdups_aux_set[simp]: "bst t ⟹ set (bst_remdups_aux t xs) = set xs - set_tree t"
apply (induction xs arbitrary: t)
apply (auto simp: set_tree_isin set_tree_ins bst_ins)
done
thm set_tree_isin set_tree_ins bst_ins
lemma bst_remdups_aux_distinct[simp]: "bst t ⟹ distinct (bst_remdups_aux t xs)"
apply (induction xs arbitrary: t)
apply (auto simp: bst_ins set_tree_isin set_tree_ins)
done
(*>*)

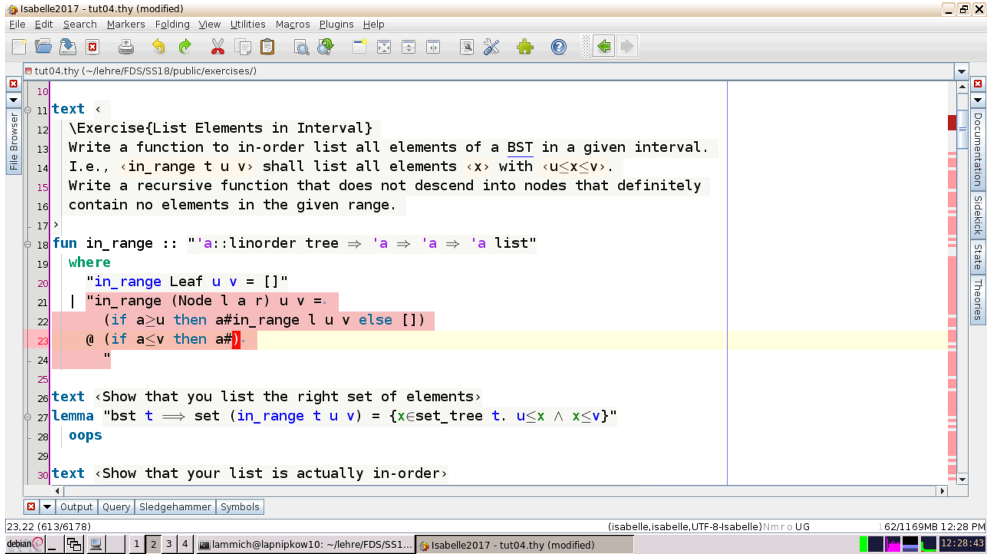

ex04.pdf
Exercise Sheet 4
Exercise 4.1 List Elements in Interval
Write a function to in-order list all elements of a BST in a given interval. I.e., in_range t u v shall list all elements x with u≤x≤v. Write a recursive function that does not descend into nodes that definitely contain no elements in the given range.
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
Show that you list the right set of elements
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
Show that your list is actually in-order
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
Exercise 4.2 Pretty Printing of Binary Trees

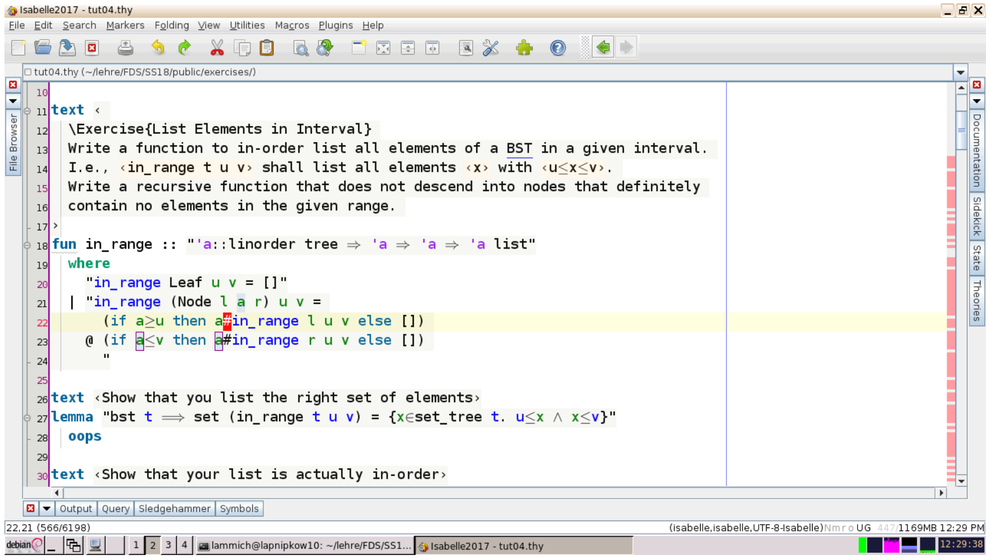

Isabelle2017 - tut04.thy
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
oops

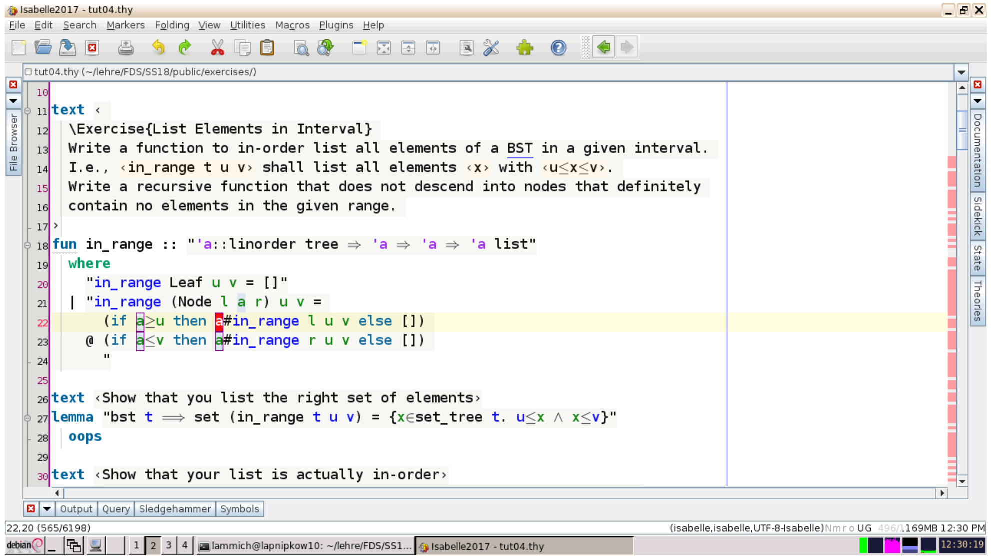

Isabelle2017 - tut04.thy
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
oops

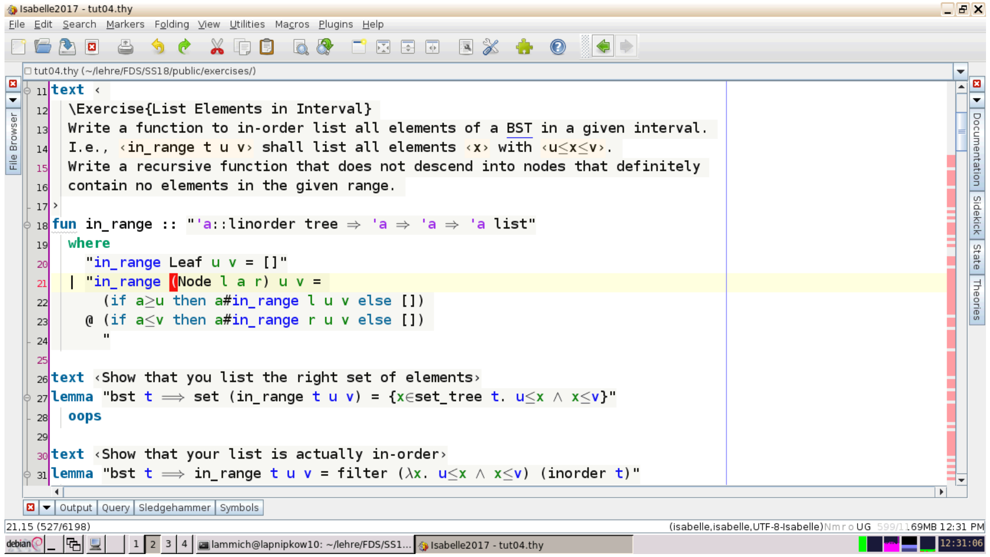

Isabelle2017 - tut04.thy (modified)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
oops

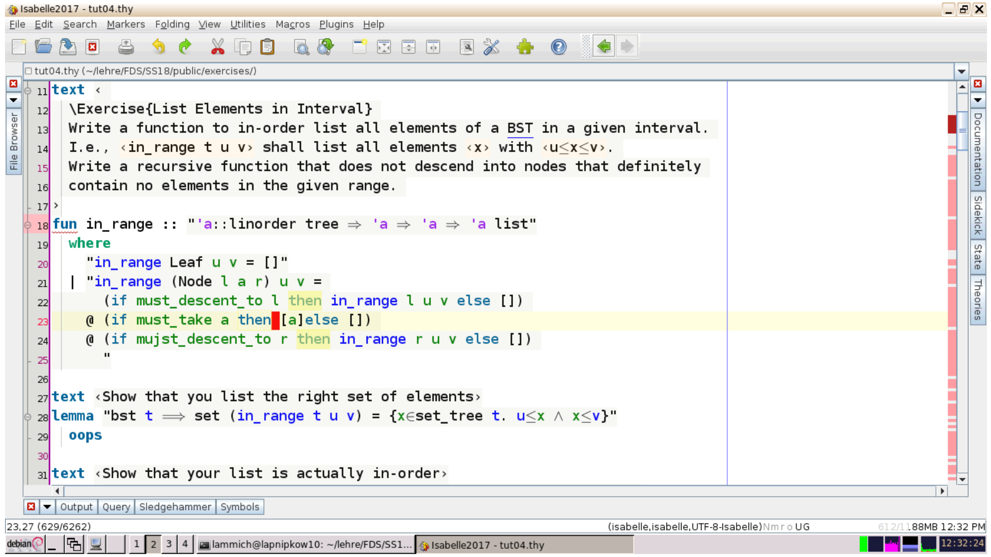

Isabelle2017 - tut04.thy (modified)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v = "
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
oops

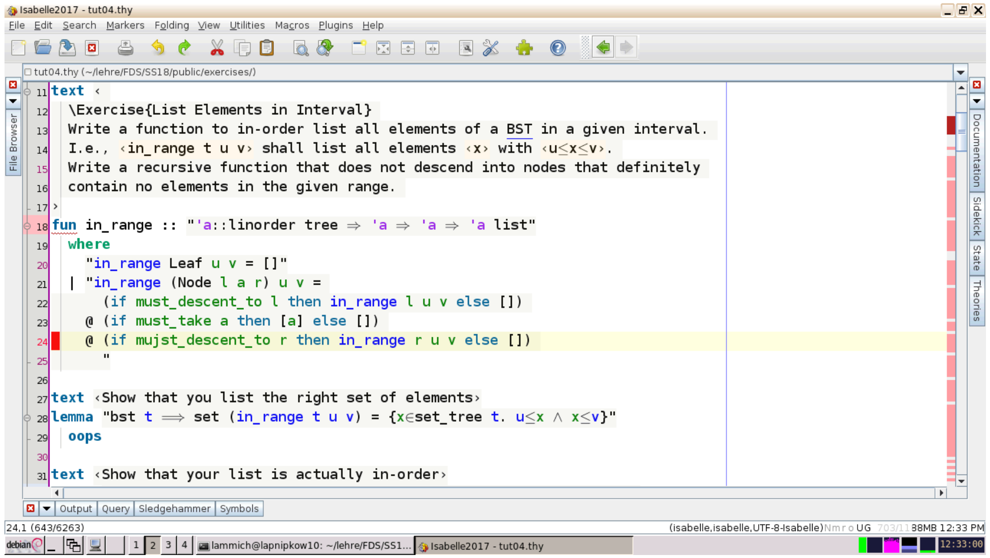

Isabelle2017 - tut04.thy (modified)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if a≥u then a#in_range l u v else [])
@ (if a≤v then a#)
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›

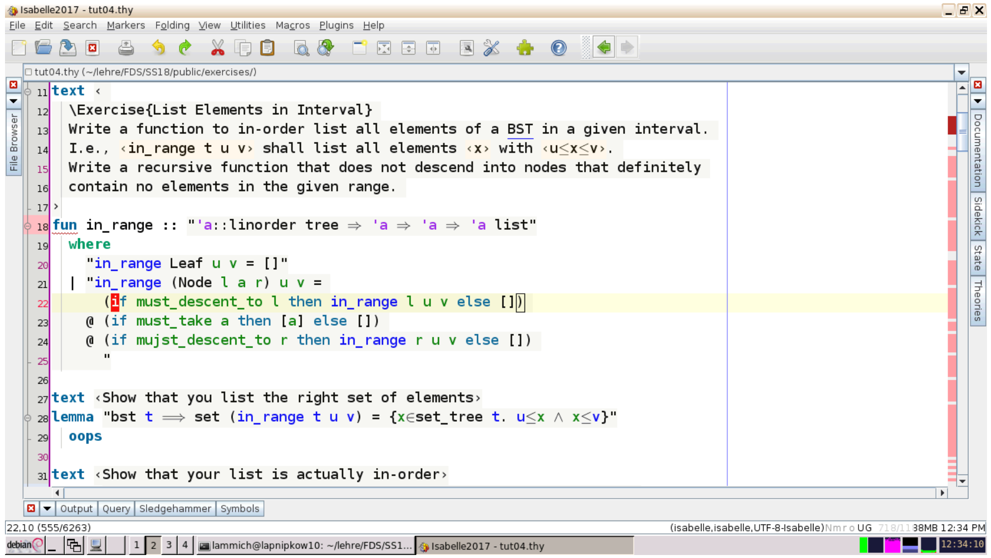

Isabelle2017 - tut04.thy
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if a≥u then a#in_range l u v else [])
@ (if a≤v then a#in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›

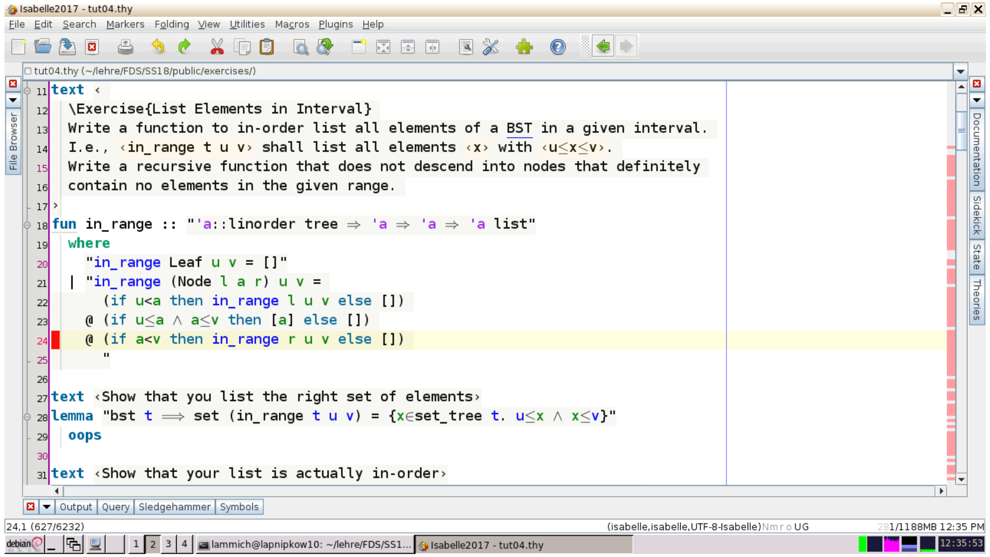

Isabelle2017 - tut04.thy
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if a≥u then a#in_range l u v else [])
@ (if a≤v then a#in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›

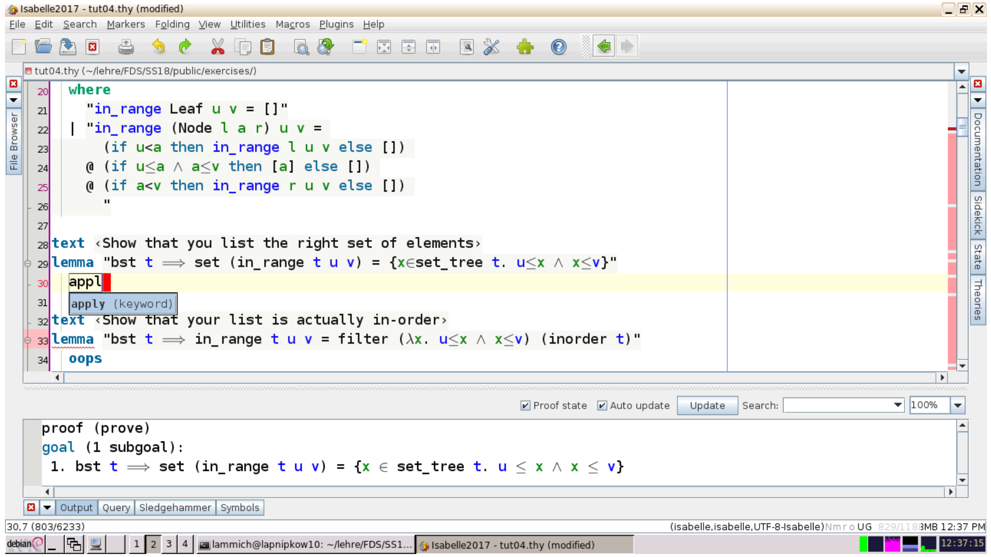

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if a≥u then a#in_range l u v else [])
@ (if a≤v then a#in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
Output Query Sledgehammer Symbols
21,15 (527/6198)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:31:06

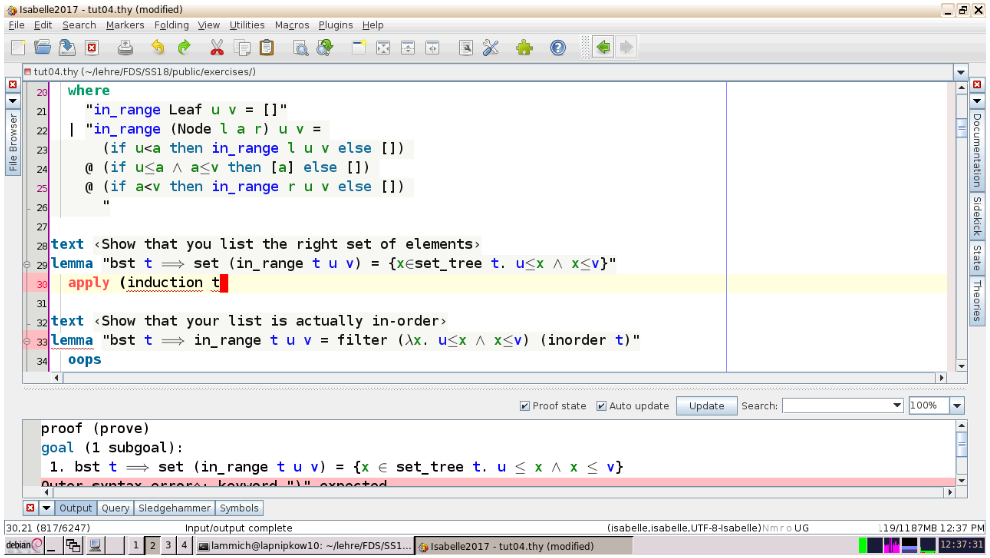

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if must_descent_to l then in_range l u v else [])
@ (if must_take a then[a]else [])
@ (if mujst_descent_to r then in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
Output Query Sledgehammer Symbols
23,27 (629/6262)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:32:24

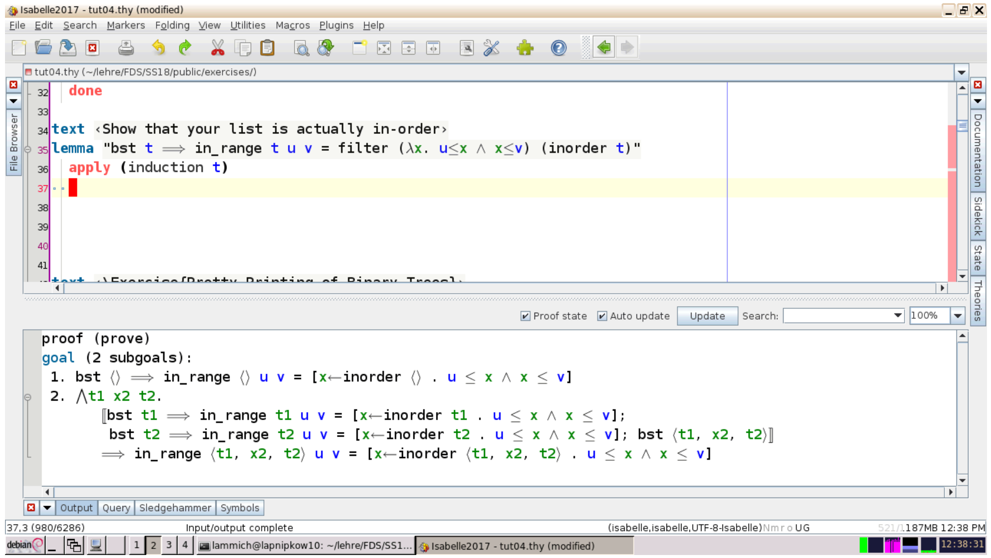

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if must_descent_to l then in_range l u v else [])
@ (if must_take a then [a] else [])
@ (if mujst_descent_to r then in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
Output Query Sledgehammer Symbols
24,1 (643/6263)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:33:00

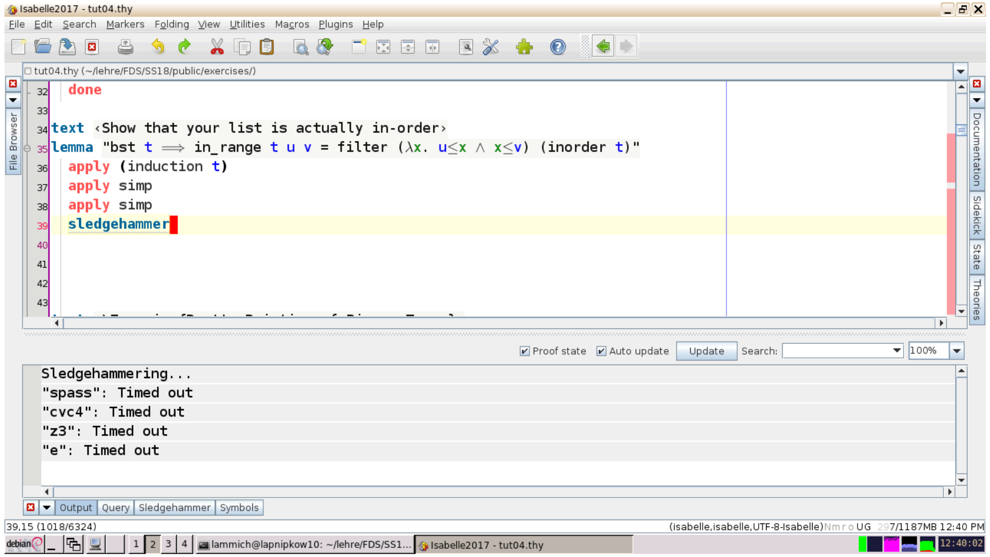

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if must_descent_to l then in_range l u v else [])
@ (if must_take a then [a] else [])
@ (if mujst_descent_to r then in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
Output Query Sledgehammer Symbols
22,10 (555/6263)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:34:10

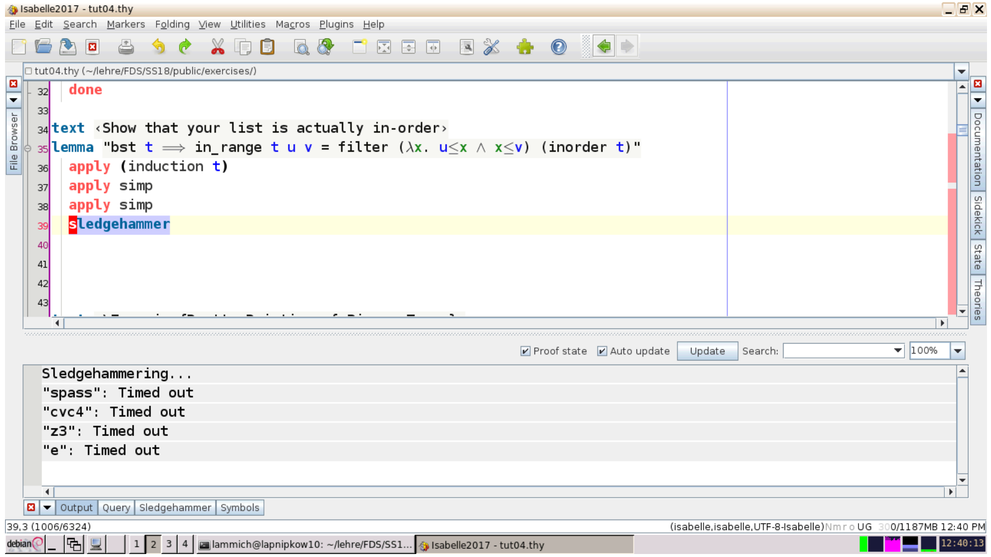

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹
\Exercise{List Elements in Interval}
Write a function to in-order list all elements of a BST in a given interval.
I.e., ‹in_range t u v› shall list all elements ‹x› with ‹u≤x≤v›.
Write a recursive function that does not descend into nodes that definitely
contain no elements in the given range.
›
fun in_range :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if u<a then in_range l u v else [])
@ (if u≤a ∧ a≤v then [a] else [])
@ (if a<v then in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
oops
text ‹Show that your list is actually in-order›
Output Query Sledgehammer Symbols
24,1 (627/6232)
(isabelle,isabelle,UTF-8-Isabelle)
12:35 PM

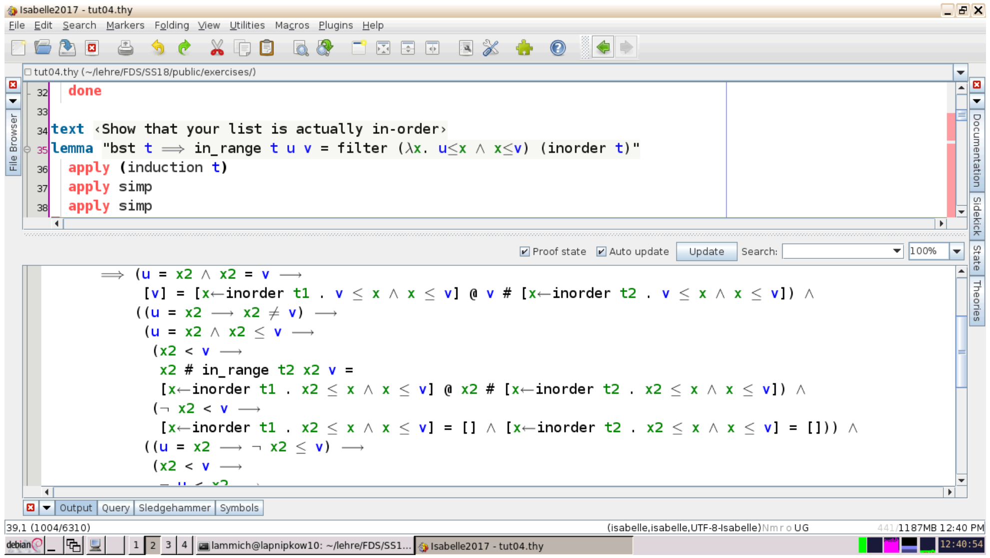

Isabelle2017 - tut04.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if u<a then in_range l u v else [])
@ (if u≤a ∧ a≤v then [a] else [])
@ (if a<v then in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
appl
apply (keyword)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
oops
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. bst t ⟹ set (in_range t u v) = {x ∈ set_tree t. u ≤ x ∧ x ≤ v}
Output Query Sledgehammer Symbols
30,7 (803/6233)
(isabelle,isabelle,UTF-8-Isabelle)
12:37 PM

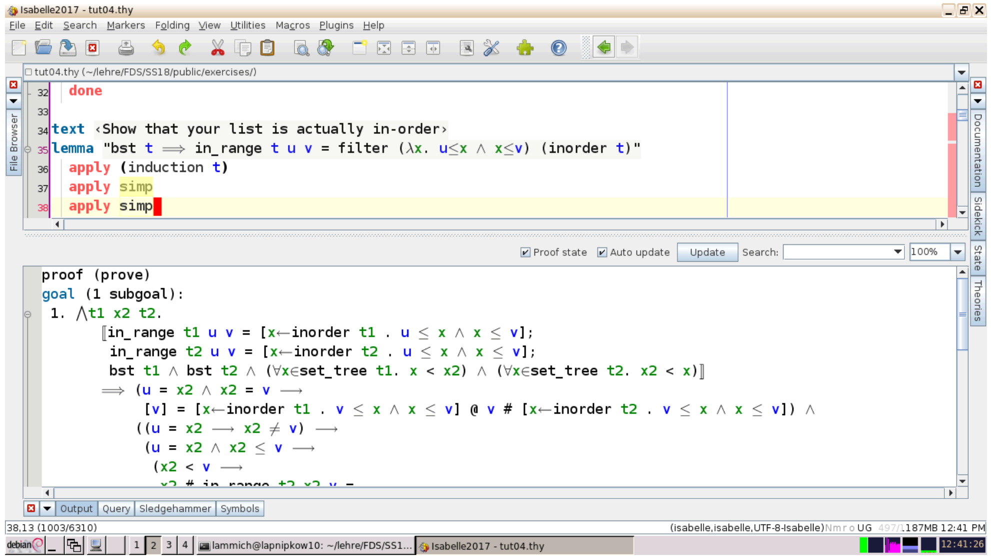

Isabelle2017 - tut04.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
where
"in_range Leaf u v = []"
| "in_range (Node l a r) u v =
(if u<a then in_range l u v else [])
@ (if u≤a ∧ a≤v then [a] else [])
@ (if a<v then in_range r u v else [])
"
text ‹Show that you list the right set of elements›
lemma "bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
apply (induction t
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
oops
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. bst t ⟹ set (in_range t u v) = {x ∈ set_tree t. u ≤ x ∧ x ≤ v}
Outer syntax error: keyword ")" expected
Output Query Sledgehammer Symbols
30,21 (817/6247)
Input/output complete
(isabelle,isabelle,UTF-8-Isabelle)
12:37 PM

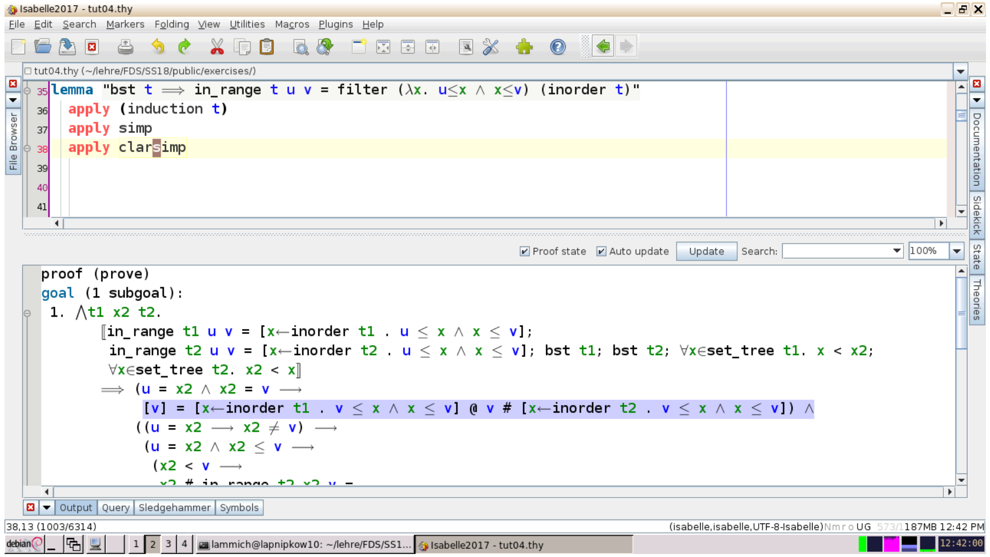

Isabelle2017 - tut04.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
done
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
Proof state Auto update Update Search: 100%
proof (prove)
goal (2 subgoals):
1. bst ⟨⟩ ⟹ in_range ⟨⟩ u v = [x←inorder ⟨⟩ . u ≤ x ∧ x ≤ v]
2. ⋀t1 x2 t2.
⟦bst t1 ⟹ in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
bst t2 ⟹ in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst ⟨t1, x2, t2⟩⟧
⟹ in_range ⟨t1, x2, t2⟩ u v = [x←inorder ⟨t1, x2, t2⟩ . u ≤ x ∧ x ≤ v]
Output Query Sledgehammer Symbols
37,3 (980/6286)
Input/output complete
(isabelle,isabelle,UTF-8-Isabelle)
12:38 PM

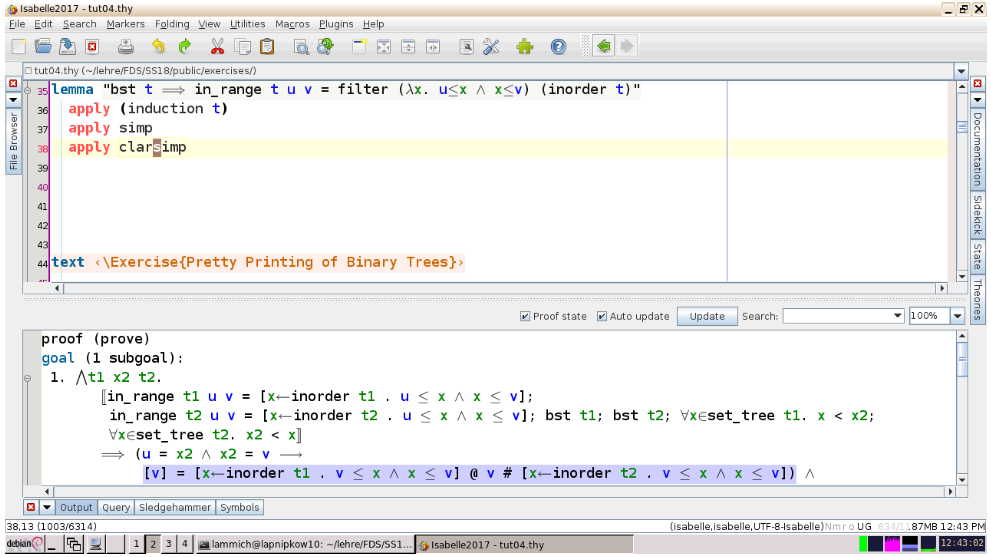

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
done
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply simp
sledgehammer
Sledgehammering...
"spass": Timed out
"cvc4": Timed out
"z3": Timed out
"e": Timed out
Output Query Sledgehammer Symbols

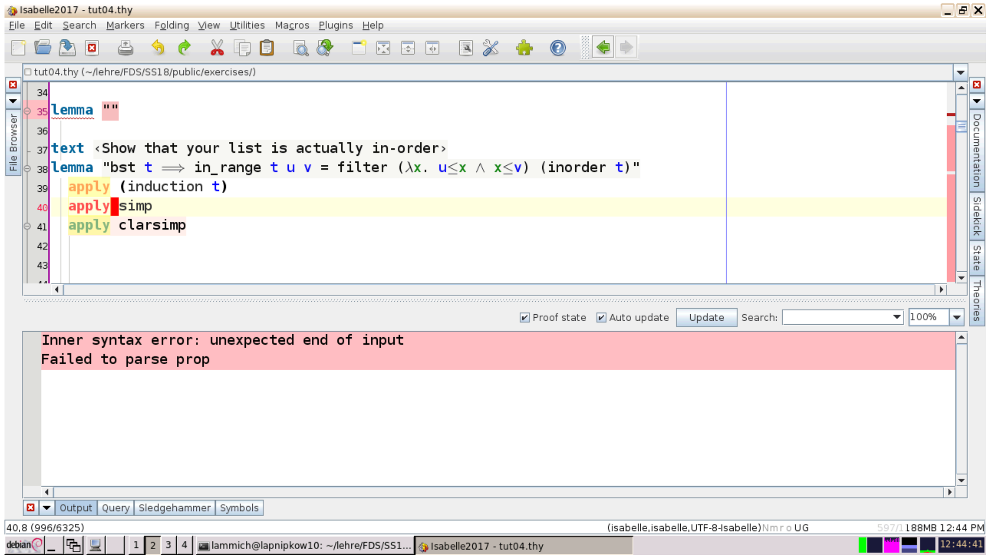

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
done
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply simp
sledgehammer
Sledgehammering...
"spass": Timed out
"cvc4": Timed out
"z3": Timed out
"e": Timed out
Output Query Sledgehammer Symbols

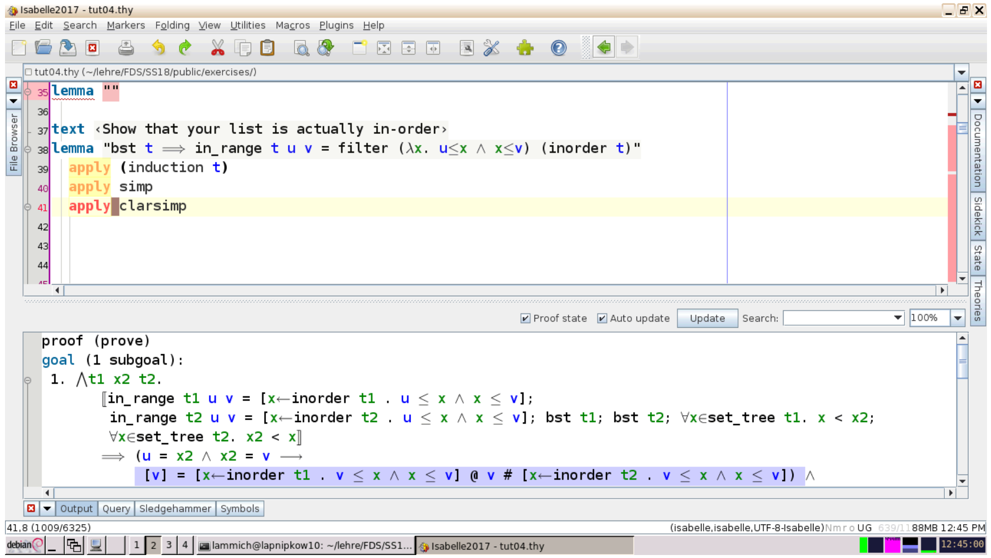

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
done
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply simp
⟹ (u = x2 ∧ x2 = v ⟶
[v] = [x←inorder t1 . v ≤ x ∧ x ≤ v] @ v # [x←inorder t2 . v ≤ x ∧ x ≤ v]) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧
(¬ x2 < v ⟶
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] = [] ∧ [x←inorder t2 . x2 ≤ x ∧ x ≤ v] = [])) ∧
((u = x2 ⟶ ¬ x2 ≤ v) ⟶
(x2 < v ⟶
Output Query Sledgehammer Symbols

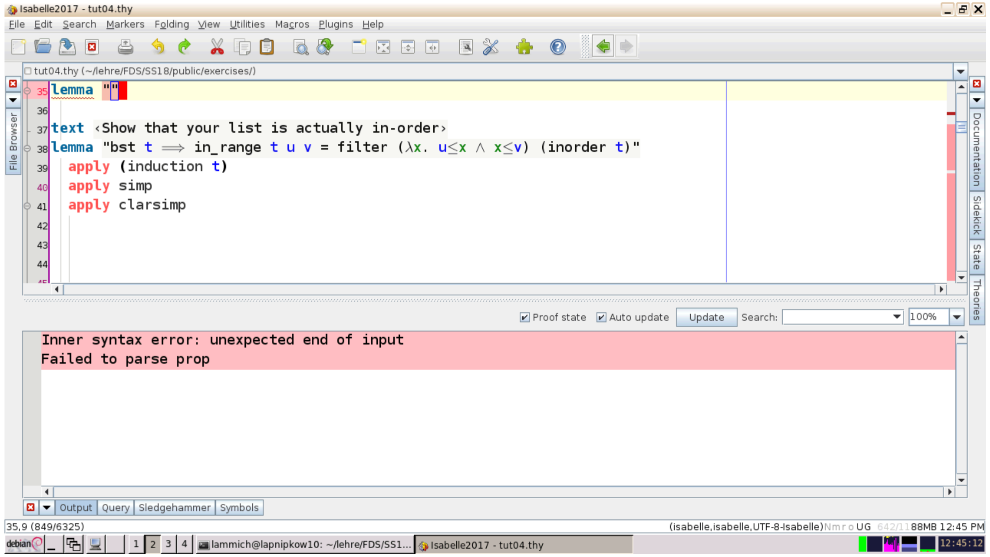

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
done
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply simp
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v];
bst t1 ∧ bst t2 ∧ (∀x∈set_tree t1. x < x2) ∧ (∀x∈set_tree t2. x2 < x)⟧
⟹ (u = x2 ∧ x2 = v ⟶
[v] = [x←inorder t1 . v ≤ x ∧ x ≤ v] @ v # [x←inorder t2 . v ≤ x ∧ x ≤ v]) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
Output Query Sledgehammer Symbols

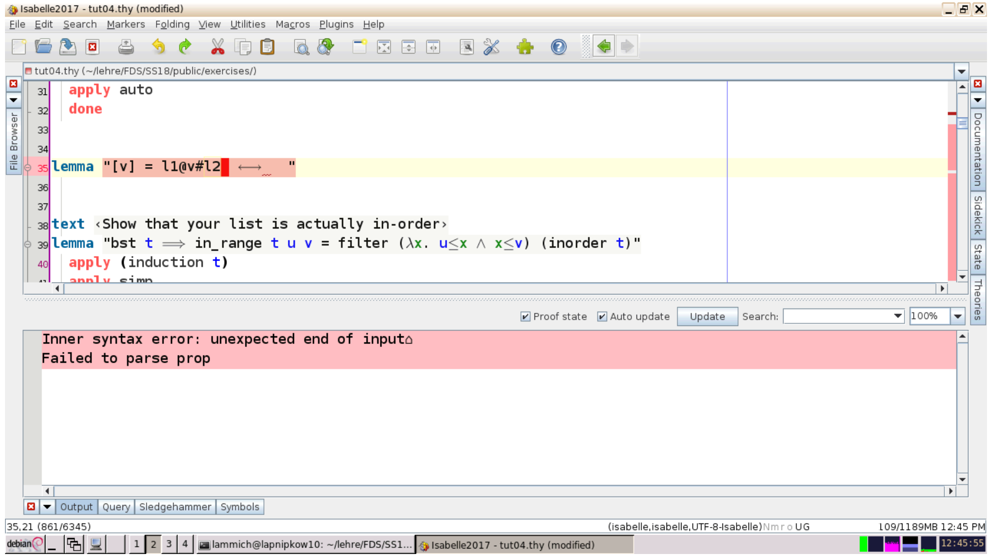

Isabelle2017 - tut04.thy
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[v] = [x←inorder t1 . v ≤ x ∧ x ≤ v] @ v # [x←inorder t2 . v ≤ x ∧ x ≤ v]) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶

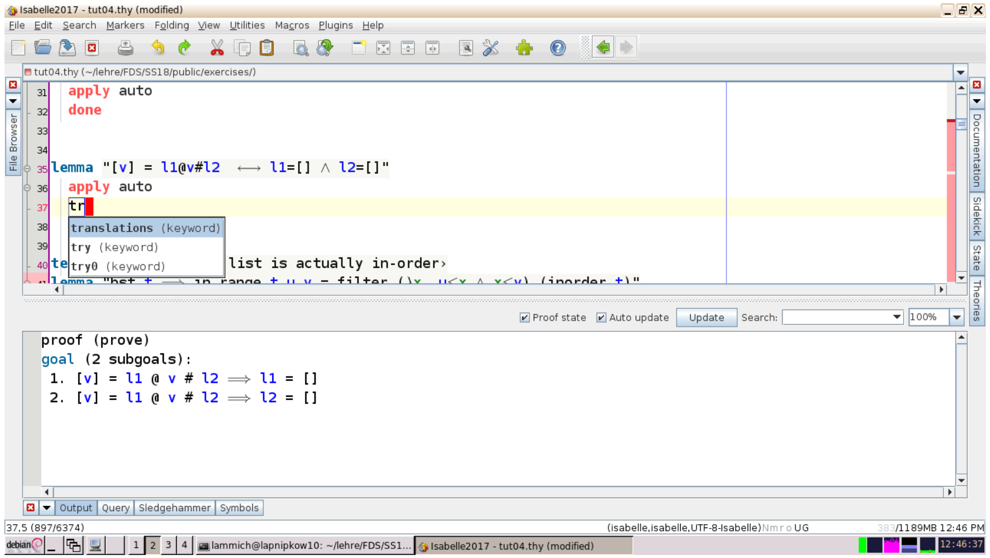

Isabelle2017 - tut04.thy
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
text ‹\Exercise{Pretty Printing of Binary Trees}›
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[v] = [x←inorder t1 . v ≤ x ∧ x ≤ v] @ v # [x←inorder t2 . v ≤ x ∧ x ≤ v]) ∧

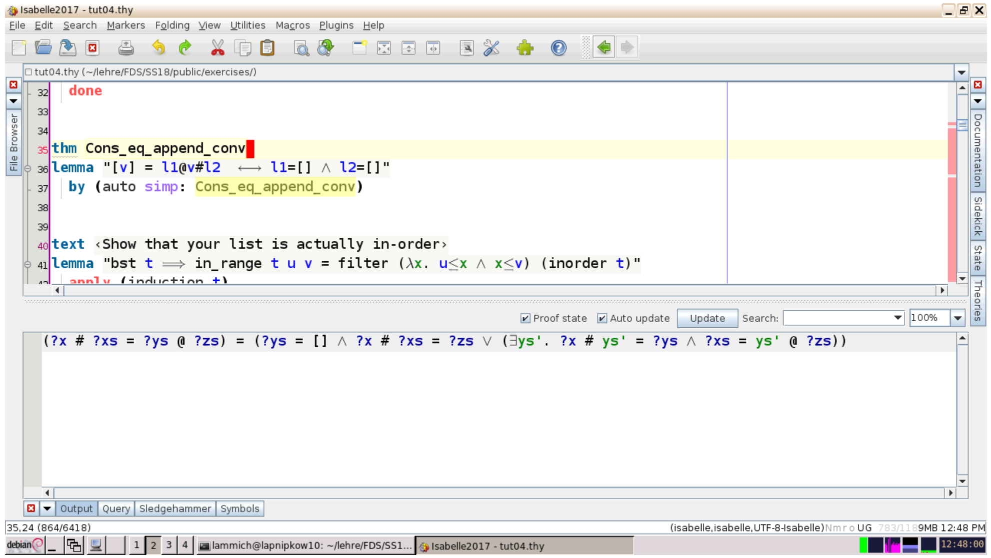

Isabelle2017 - tut04.thy
lemma ""
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
Inner syntax error: unexpected end of input
Failed to parse prop

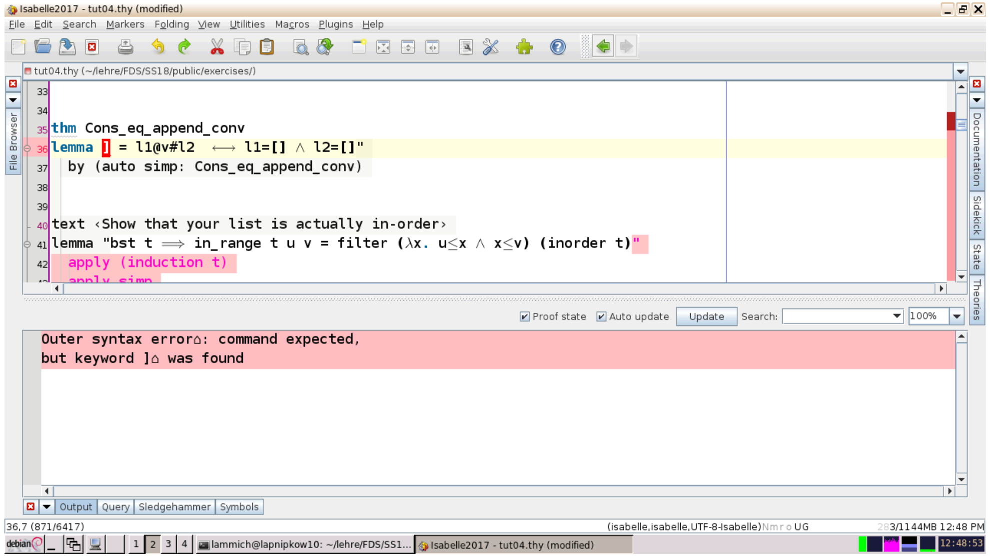

Isabelle2017 - tut04.thy
lemma ""
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[v] = [x←inorder t1 . v ≤ x ∧ x ≤ v] @ v # [x←inorder t2 . v ≤ x ∧ x ≤ v]) ∧

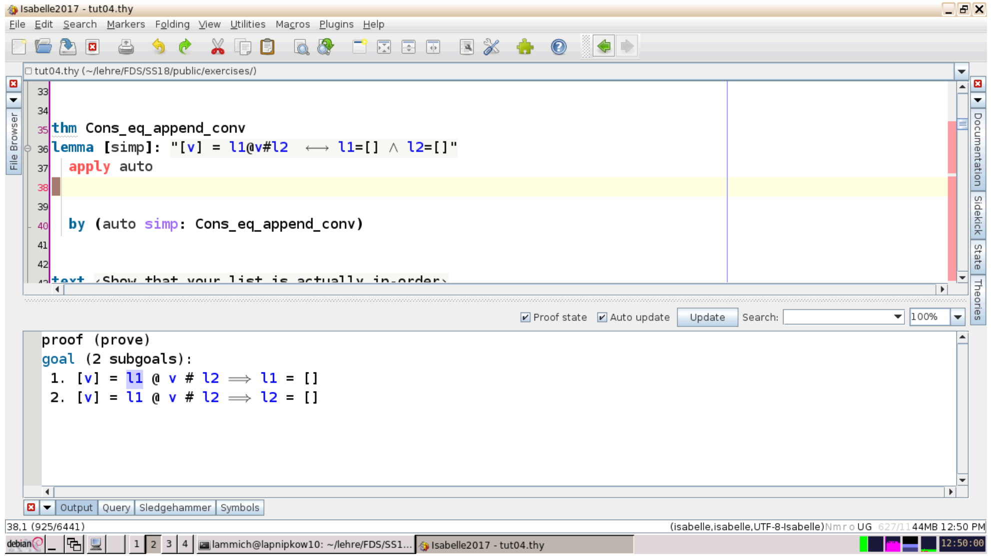

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
lemma ""
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
Inner syntax error: unexpected end of input
Failed to parse prop

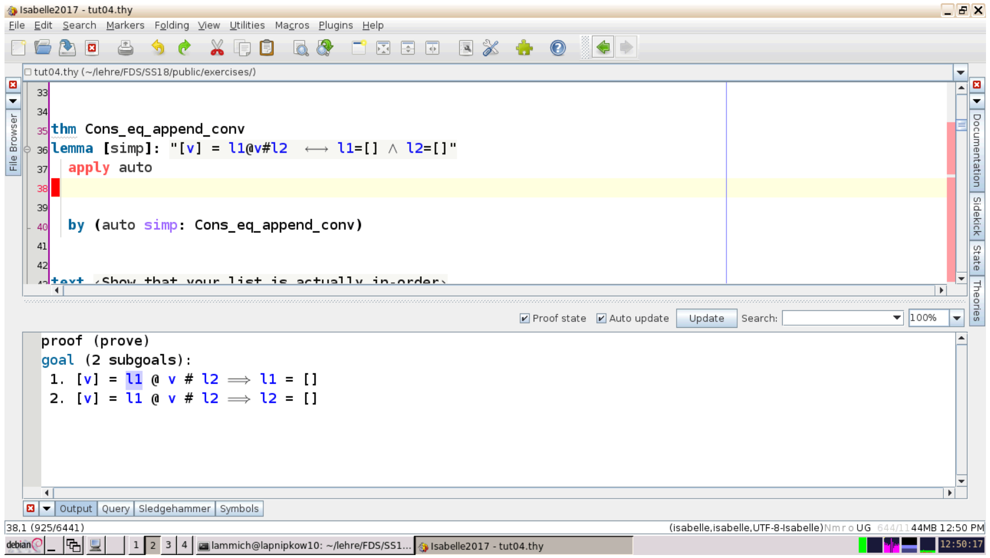

Isabelle2017 - tut04.thy (modified)
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
apply auto
done
lemma "[v] = l1@v#l2 ⟷ "
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
Inner syntax error: unexpected end of input
Failed to parse prop

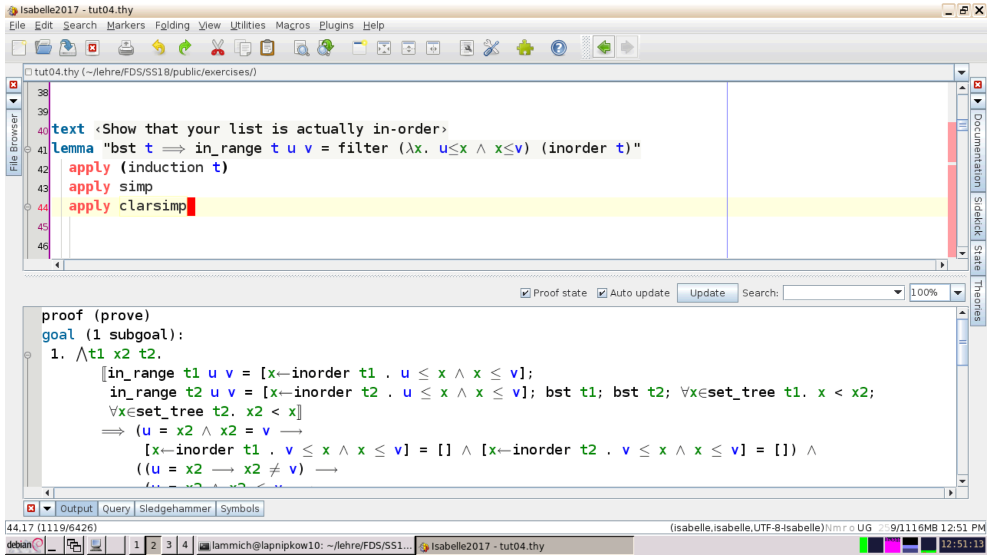

Isabelle2017 - tut04.thy (modified)
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
apply auto
done
lemma "[v] = l1@v#l2 ⟷ l1=[] ∧ l2=[]"
apply auto
tr
translations (keyword)
try (keyword)
try0 (keyword)
list is actually in-order›
proof (prove)
goal (2 subgoals):
1. [v] = l1 @ v # l2 ⟹ l1 = []
2. [v] = l1 @ v # l2 ⟹ l2 = []

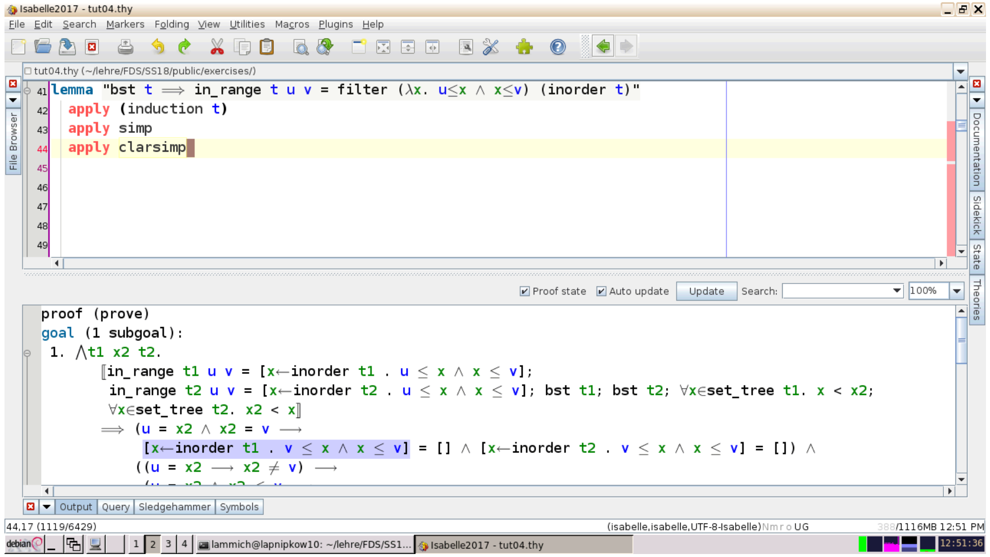

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
done
thm Cons_eq_append_conv
lemma "[v] = l1@v#l2 ⟷ l1=[] ∧ l2=[]"
by (auto simp: Cons_eq_append_conv)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
(?x # ?xs = ?ys @ ?zs) = (?ys = [] ∧ ?x # ?xs = ?zs ∨ (∃ys'. ?x # ys' = ?ys ∧ ?xs = ys' @ ?zs))

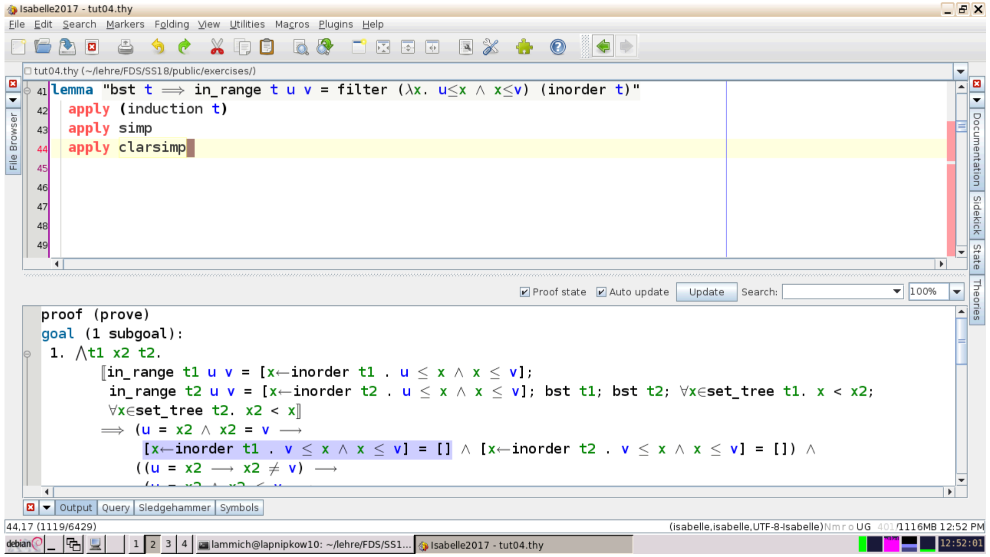

Isabelle2017 - tut04.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
thm Cons_eq_append_conv
lemma] = l1@v#l2 ⟷ l1=[] ∧ l2=[]"
by (auto simp: Cons_eq_append_conv)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
Proof state Auto update Update Search: 100%
Outer syntax error⌂: command expected,
but keyword]⌂ was found
Output Query Sledgehammer Symbols
36,7 (871/6417) (isabelle,isabelle,UTF-8-Isabelle)Nmro UG 3/1144MB 12:48 PM

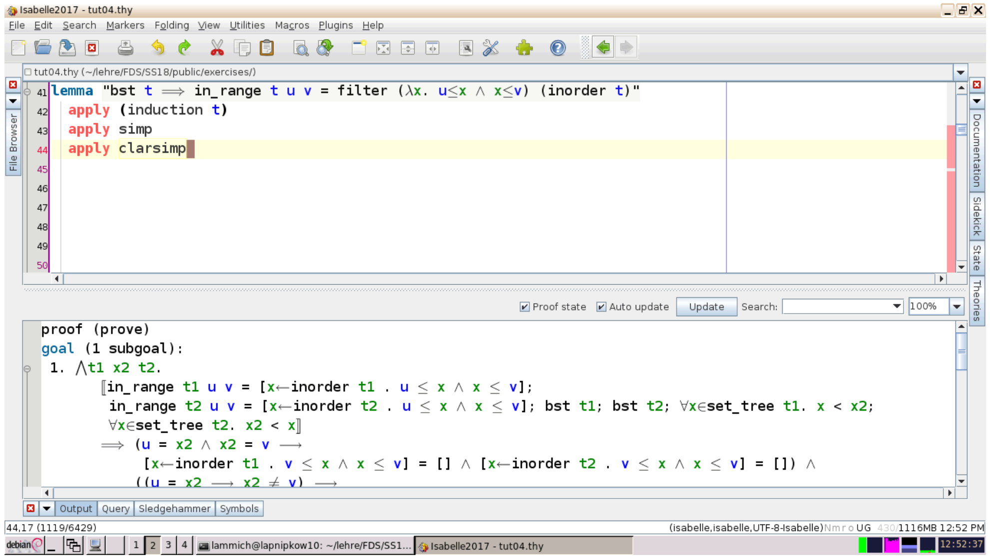

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
thm Cons_eq_append_conv
lemma [simp]: "[v] = l1@v#l2 ⟷ l1=[] ∧ l2=[]"
apply auto
by (auto simp: Cons_eq_append_conv)
text ‹Show that your list is actually in-order›
Proof state Auto update Update Search: 100%
proof (prove)
goal (2 subgoals):
1. [v] = l1 @ v # l2 ⟹ l1 = []
2. [v] = l1 @ v # l2 ⟹ l2 = []
Output Query Sledgehammer Symbols
38,1 (925/6441) (isabelle,isabelle,UTF-8-Isabelle)Nmro UG 44MB 12:50 PM

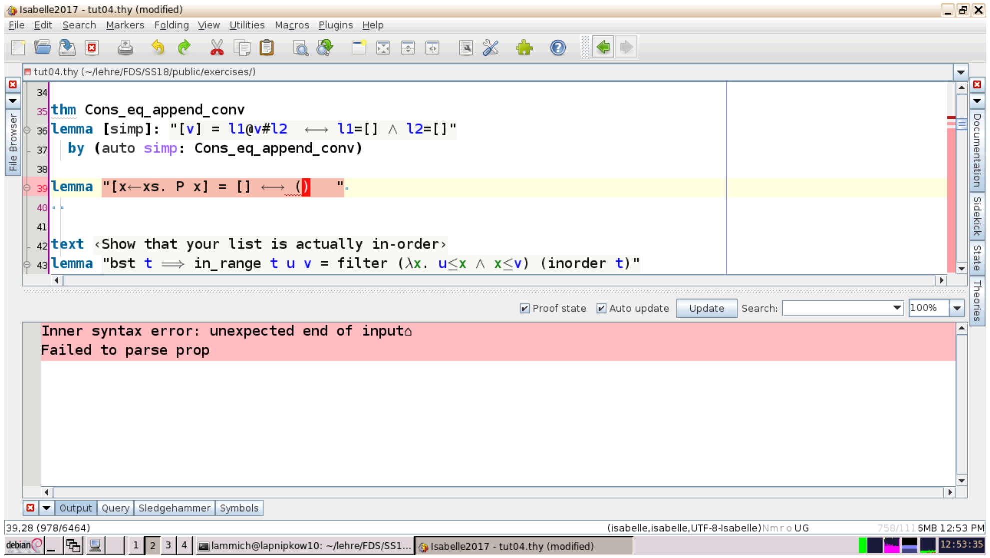

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
thm Cons_eq_append_conv
lemma [simp]: "[v] = l1@v#l2 ⟷ l1=[] ∧ l2=[]"
apply auto
by (auto simp: Cons_eq_append_conv)
text ‹Show that your list is actually in-order›
Proof state Auto update Update Search: 100%
proof (prove)
goal (2 subgoals):
1. [v] = l1 @ v # l2 ⟹ l1 = []
2. [v] = l1 @ v # l2 ⟹ l2 = []
Output Query Sledgehammer Symbols
38,1 (925/6441) (isabelle,isabelle,UTF-8-Isabelle)Nmro UG 44MB 12:50 PM

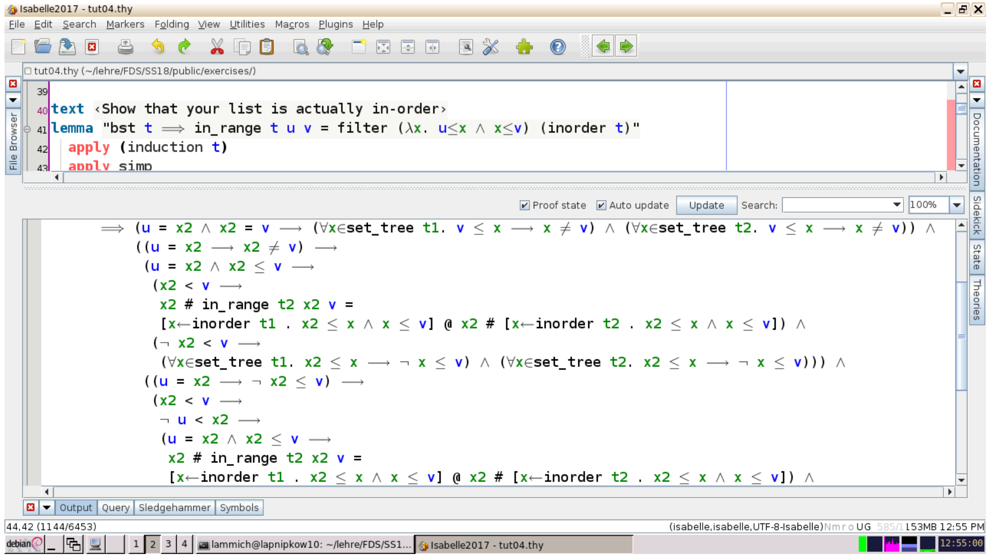

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[x←inorder t1 . v ≤ x ∧ x ≤ v] = [] ∧ [x←inorder t2 . v ≤ x ∧ x ≤ v] = []) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
Output Query Sledgehammer Symbols
44,17 (1119/6426) (isabelle,isabelle,UTF-8-Isabelle)Nmro UG 9/1116MB 12:51 PM

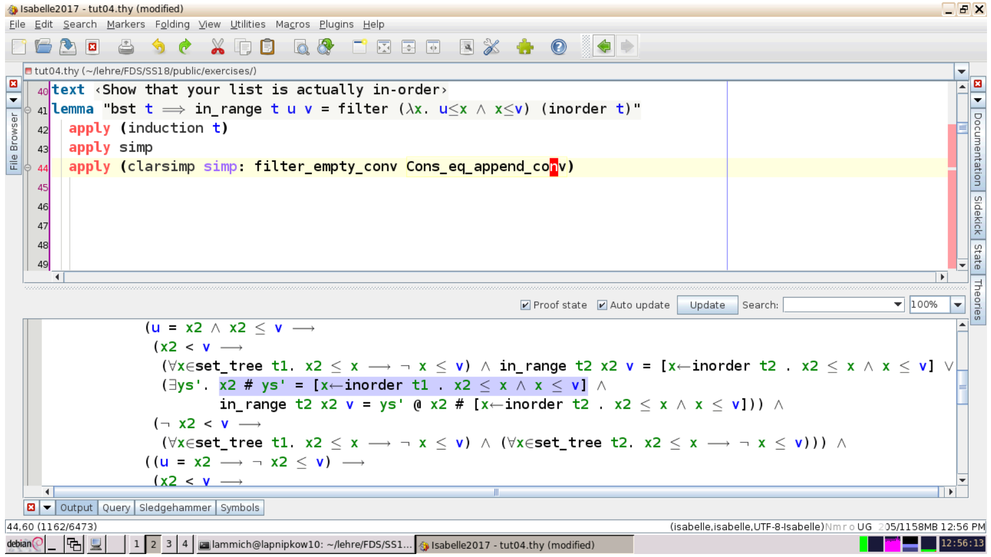

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[x←inorder t1 . v ≤ x ∧ x ≤ v] = [] ∧ [x←inorder t2 . v ≤ x ∧ x ≤ v] = []) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
Output Query Sledgehammer Symbols
44,17 (1119/6429)
(isabelle,isabelle,UTF-8-Isabelle)
1116MB 12:51 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:51:36

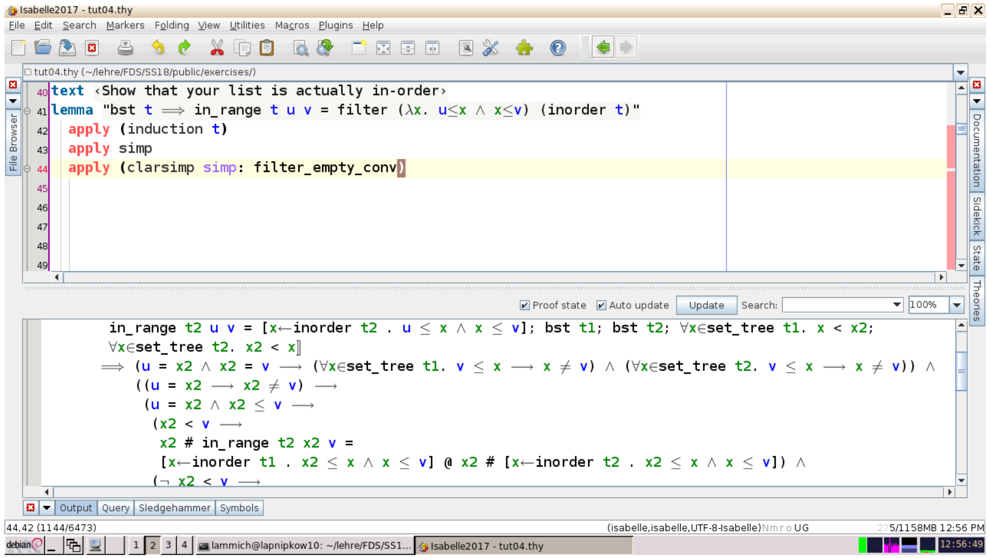

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[x←inorder t1 . v ≤ x ∧ x ≤ v] = [] ∧ [x←inorder t2 . v ≤ x ∧ x ≤ v] = []) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
Output Query Sledgehammer Symbols
44,17 (1119/6429)
(isabelle,isabelle,UTF-8-Isabelle)
1116MB 12:52 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:52:01

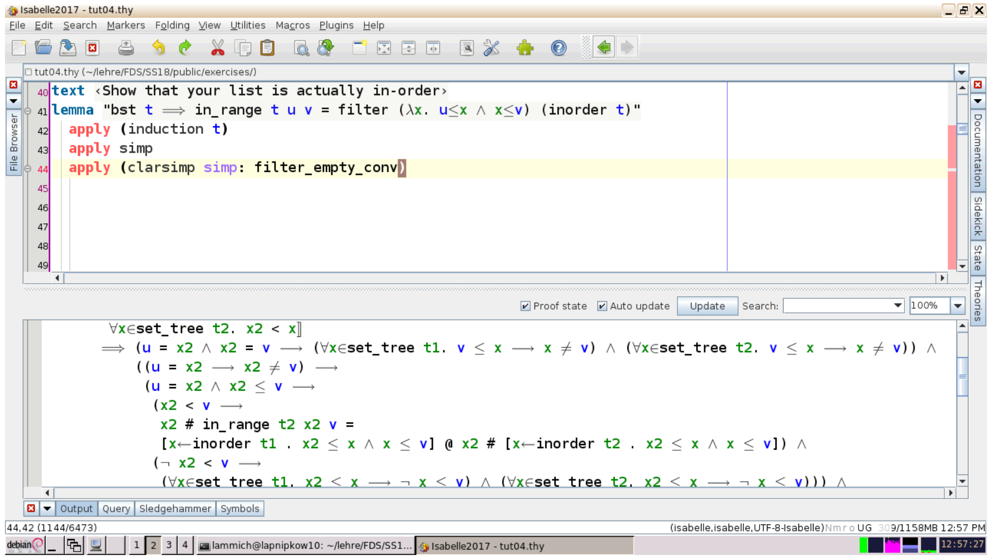

Isabelle2017 - tut04.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply clarsimp
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶
[x←inorder t1 . v ≤ x ∧ x ≤ v] = [] ∧ [x←inorder t2 . v ≤ x ∧ x ≤ v] = []) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
Output Query Sledgehammer Symbols
44,17 (1119/6429)
(isabelle,isabelle,UTF-8-Isabelle)
1116MB 12:52 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy
12:52:37

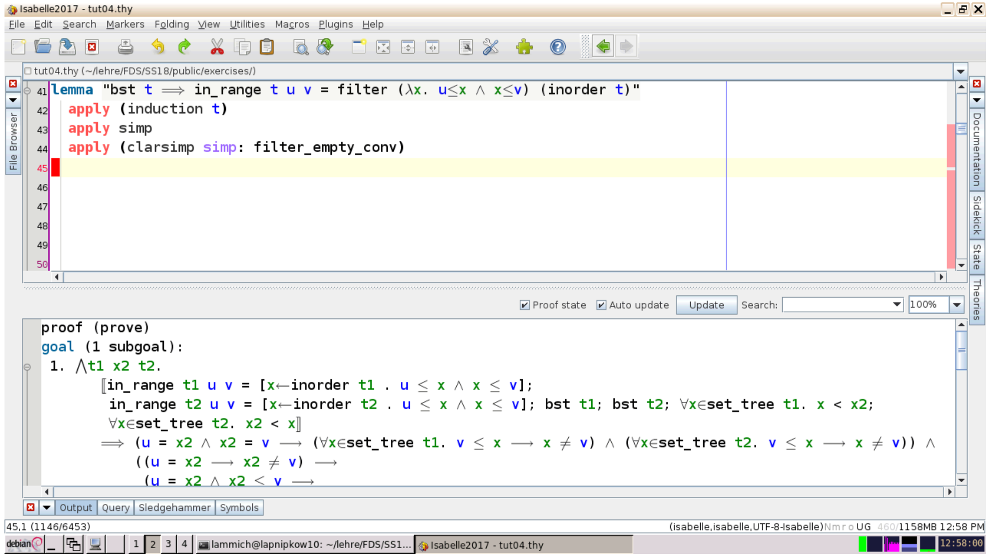
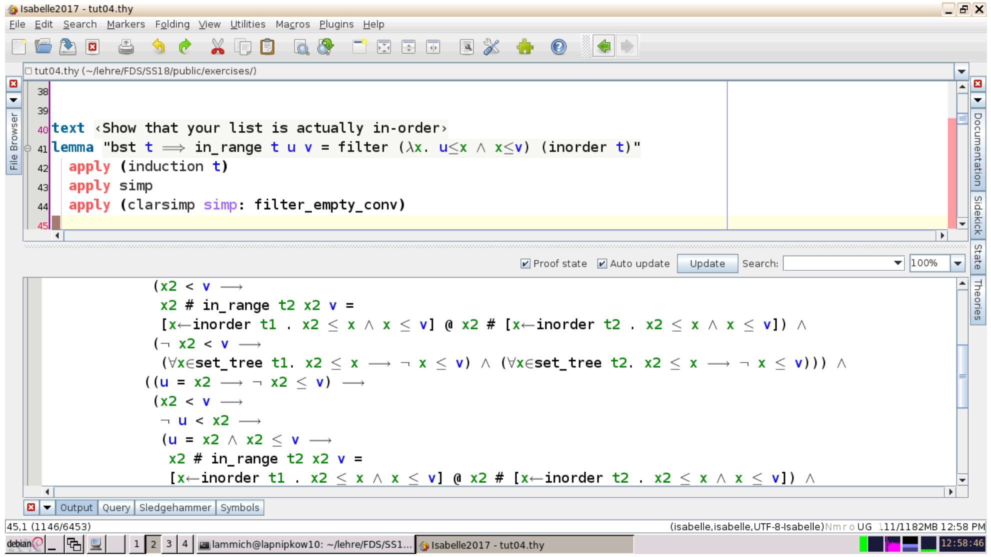
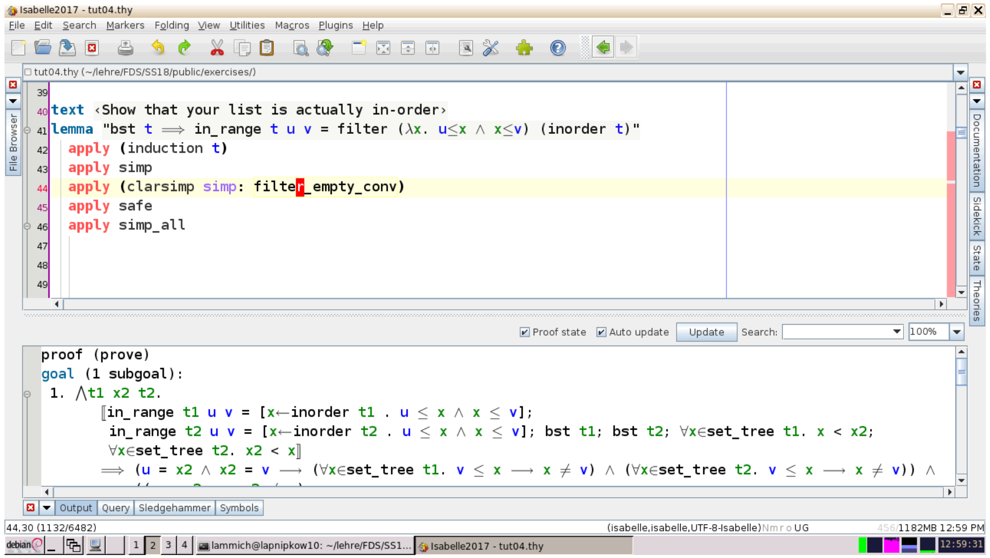
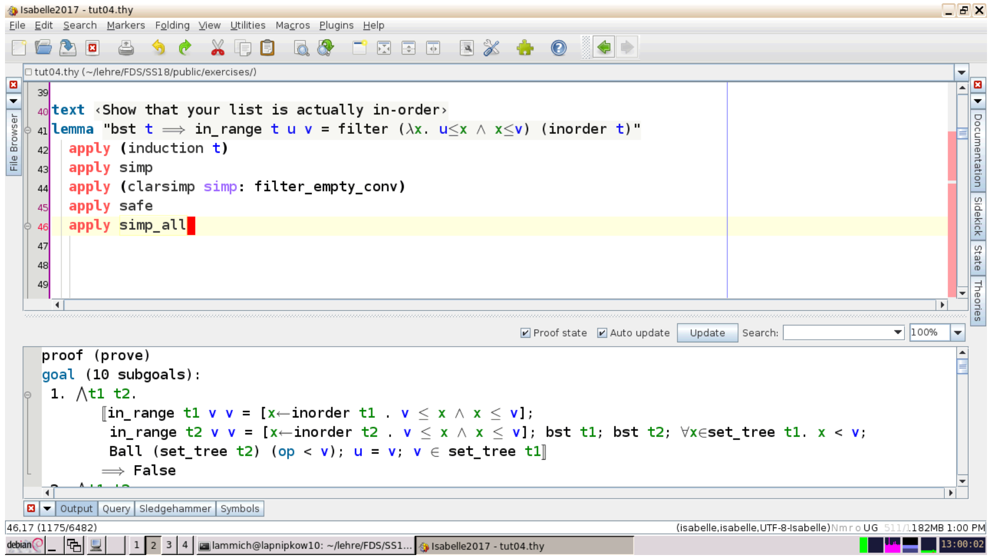

Isabelle2017 - tut04.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
thm Cons_eq_append_conv
lemma [simp]: "[v] = l1@v#l2 ⟷ l1=[] ∧ l2=[]"
by (auto simp: Cons_eq_append_conv)
lemma "[x←xs. P x] = [] ⟷ () "
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
Proof state Auto update Update Search: 100%
Inner syntax error: unexpected end of input
Failed to parse prop
Output Query Sledgehammer Symbols
39,28 (978/6464)
(isabelle,isabelle,UTF-8-Isabelle)
12:53 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut04.thy (modified)
12:53:35

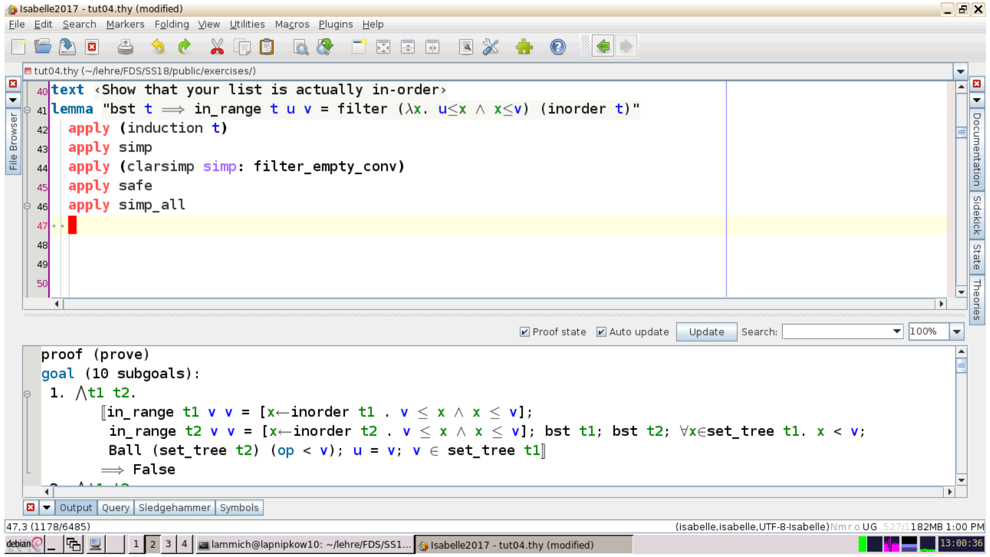

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧
(¬ x2 < v ⟶
(∀x∈set_tree t1. x2 ≤ x ⟶ ¬ x ≤ v) ∧ (∀x∈set_tree t2. x2 ≤ x ⟶ ¬ x ≤ v))) ∧
((u = x2 ⟶ ¬ x2 ≤ v) ⟶
(x2 < v ⟶
¬ u < x2 ⟶
(u = x2 ∧ x2 ≤ v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧

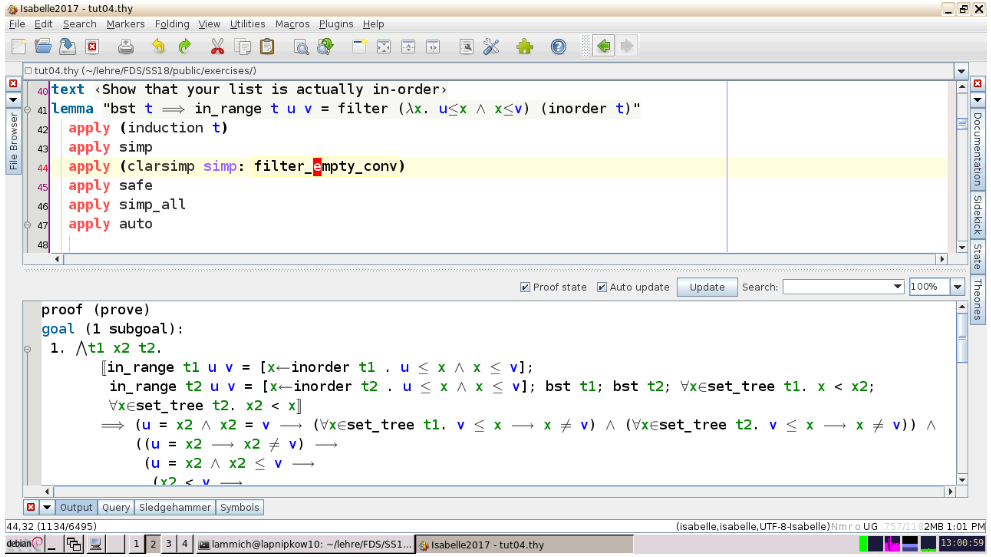

Isabelle2017 - tut04.thy (modified)
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv Cons_eq_append_conv)
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
(∀x∈set_tree t1. x2 ≤ x ⟶ ¬ x ≤ v) ∧ in_range t2 x2 v = [x←inorder t2 . x2 ≤ x ∧ x ≤ v] ∨
(∃ys'. x2 # ys' = [x←inorder t1 . x2 ≤ x ∧ x ≤ v] ∧
in_range t2 x2 v = ys' @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v])) ∧
(¬ x2 < v ⟶
(∀x∈set_tree t1. x2 ≤ x ⟶ ¬ x ≤ v) ∧ (∀x∈set_tree t2. x2 ≤ x ⟶ ¬ x ≤ v))) ∧
((u = x2 ⟶ ¬ x2 ≤ v) ⟶
(x2 < v ⟶

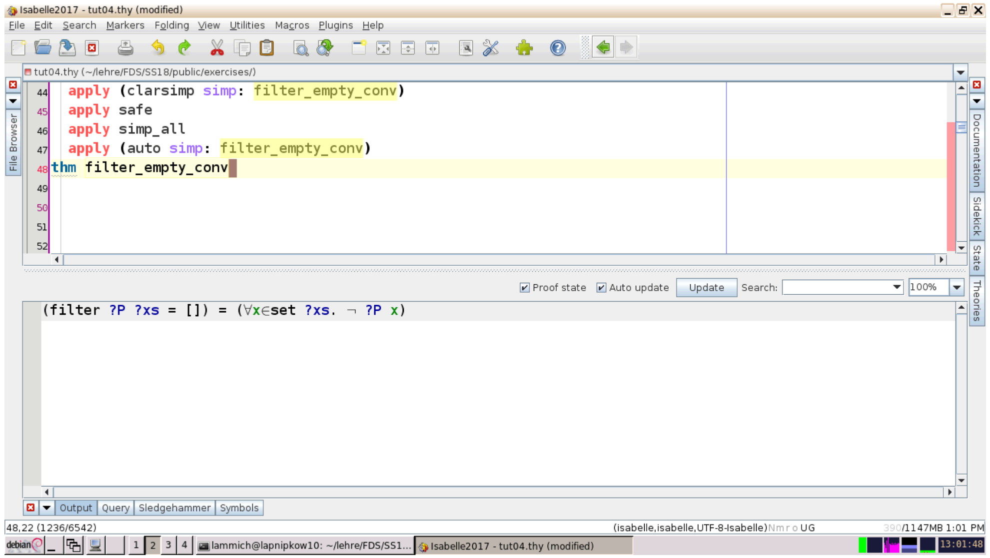

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x]
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧
(¬ x2 < v ⟶

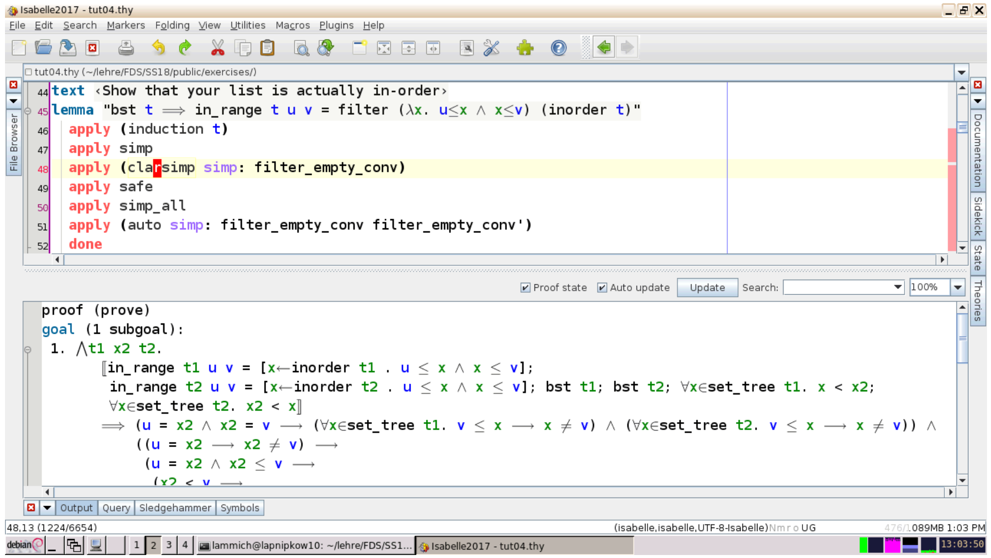

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
∀x∈set_tree t2. x2 < x]
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧
(¬ x2 < v ⟶
(∀x∈set_tree t1. x2 ≤ x ⟶ ¬ x ≤ v) ∧ (∀x∈set_tree t2. x2 ≤ x ⟶ ¬ x ≤ v))) ∧

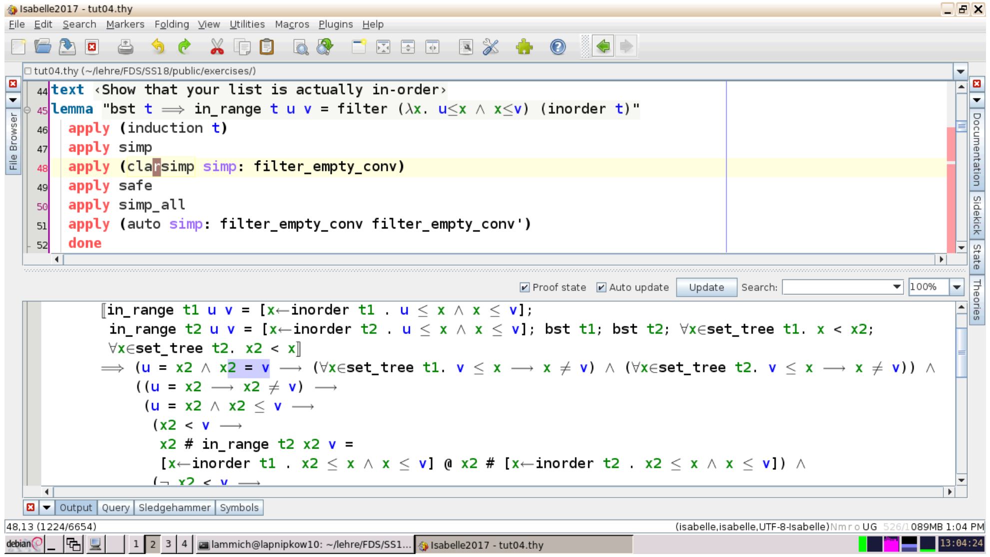

Isabelle2017 - tut04.thy
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
(x2 < v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧
(¬ x2 < v ⟶
(∀x∈set_tree t1. x2 ≤ x ⟶ ¬ x ≤ v) ∧ (∀x∈set_tree t2. x2 ≤ x ⟶ ¬ x ≤ v))) ∧
((u = x2 ⟶ ¬ x2 ≤ v) ⟶
(x2 < v ⟶
¬ u < x2 ⟶
(u = x2 ∧ x2 ≤ v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
⟦in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x⟧
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
proof (prove)
goal (10 subgoals):
1. ⋀t1 t2.
⟦in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
Ball (set_tree t2) (op < v); u = v; v ∈ set_tree t1⟧
⟹ False


Isabelle2017 - tut04.thy (modified)
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
proof (prove)
goal (10 subgoals):
1. ⋀t1 t2.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
Ball (set_tree t2) (op < v); u = v; v ∈ set_tree t1]
⟹ False

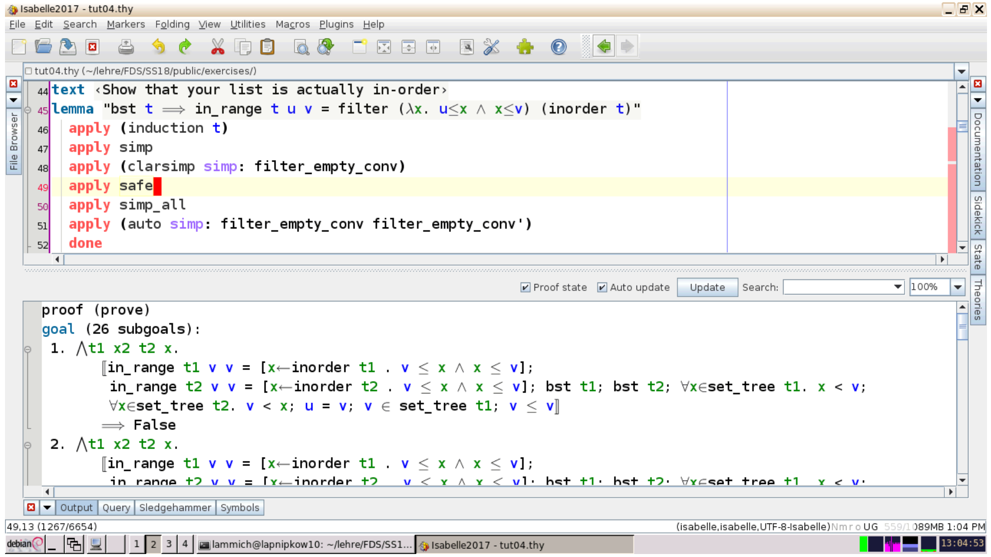

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
apply auto
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
[in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x]
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶


Isabelle2017 - tut04.thy (modified)
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv)
thm filter_empty_conv
(filter ?P ?xs = []) = (∀x∈set ?xs. ¬ ?P x)

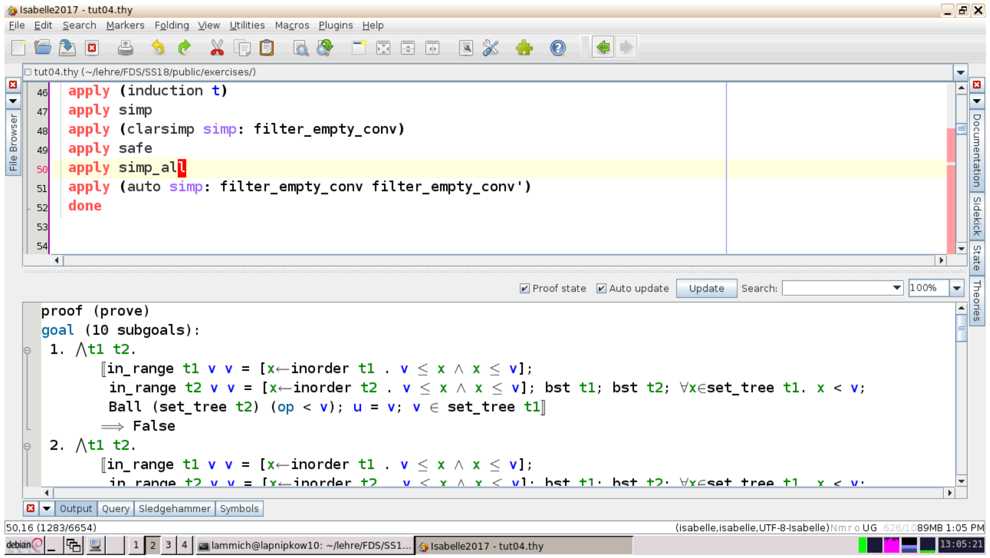

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv filter_empty_conv')
done
proof (prove)
goal (1 subgoal):
1. ⋀t1 x2 t2.
[in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x]
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶



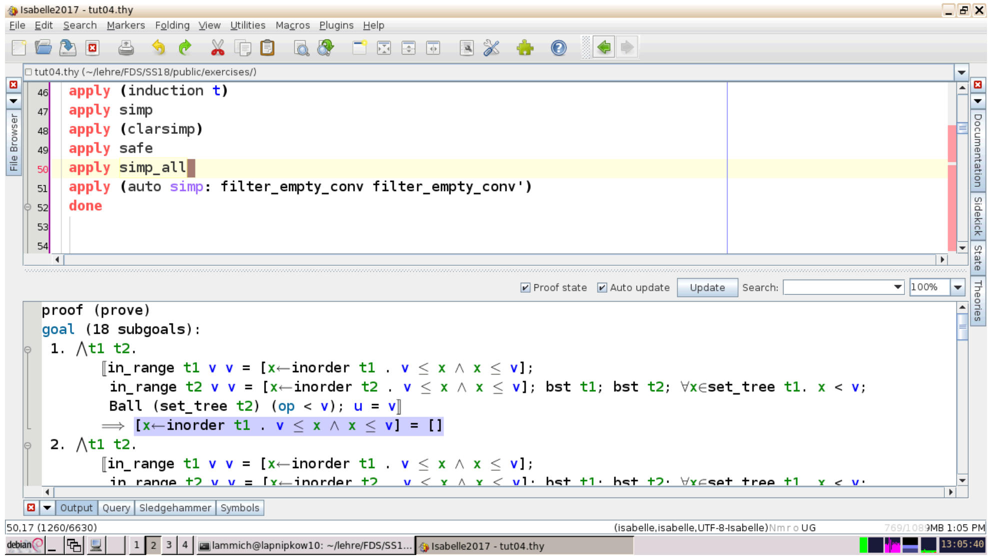
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv filter_empty_conv')
done
[in_range t1 u v = [x←inorder t1 . u ≤ x ∧ x ≤ v];
in_range t2 u v = [x←inorder t2 . u ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < x2;
∀x∈set_tree t2. x2 < x]
⟹ (u = x2 ∧ x2 = v ⟶ (∀x∈set_tree t1. v ≤ x ⟶ x ≠ v) ∧ (∀x∈set_tree t2. v ≤ x ⟶ x ≠ v)) ∧
((u = x2 ⟶ x2 ≠ v) ⟶
(u = x2 ∧ x2 ≤ v ⟶
(x2 < v ⟶
x2 # in_range t2 x2 v =
[x←inorder t1 . x2 ≤ x ∧ x ≤ v] @ x2 # [x←inorder t2 . x2 ≤ x ∧ x ≤ v]) ∧



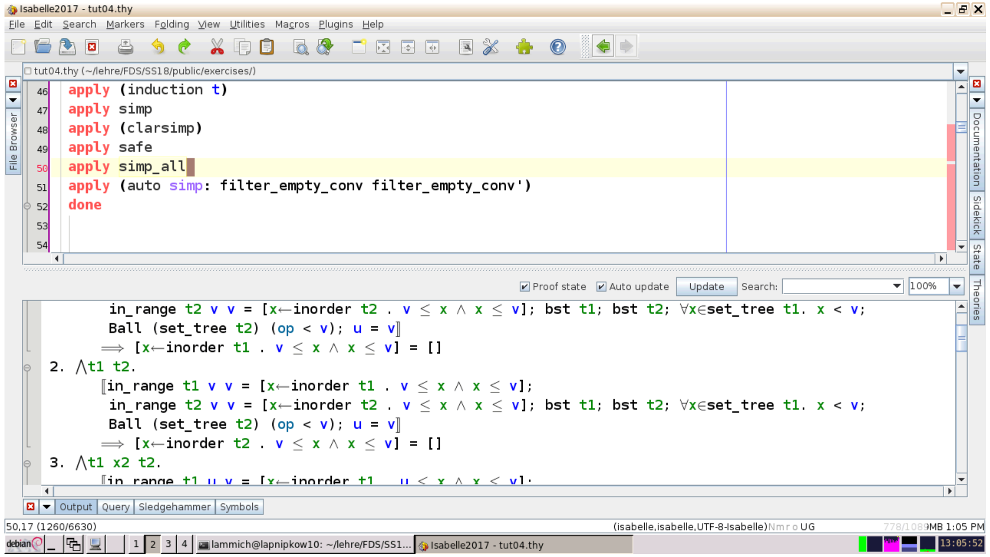
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv filter_empty_conv')
done
proof (prove)
goal (26 subgoals):
1. ⋀t1 x2 t2 x.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
∀x∈set_tree t2. v < x; u = v; v ∈ set_tree t1; v ≤ v]
⟹ False
2. ⋀t1 x2 t2 x.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];

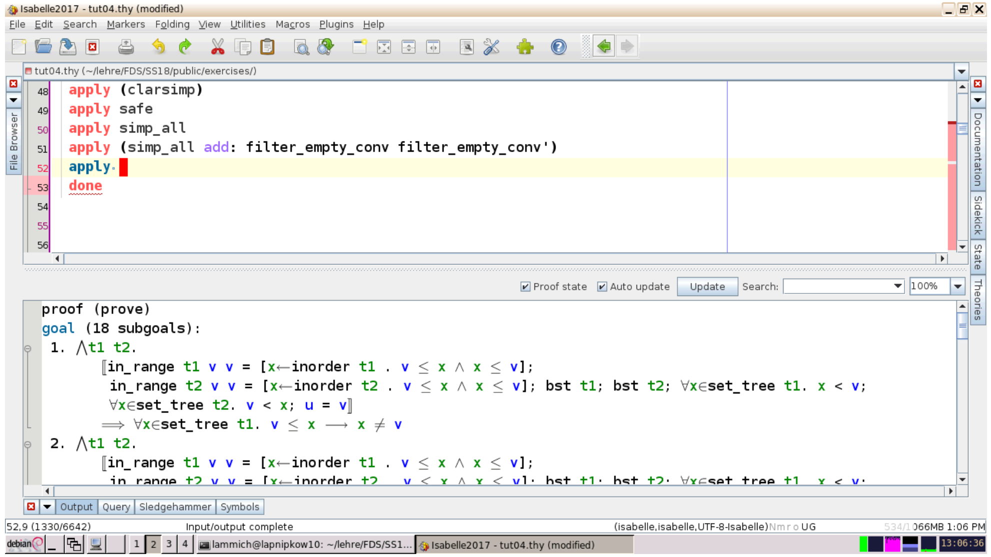


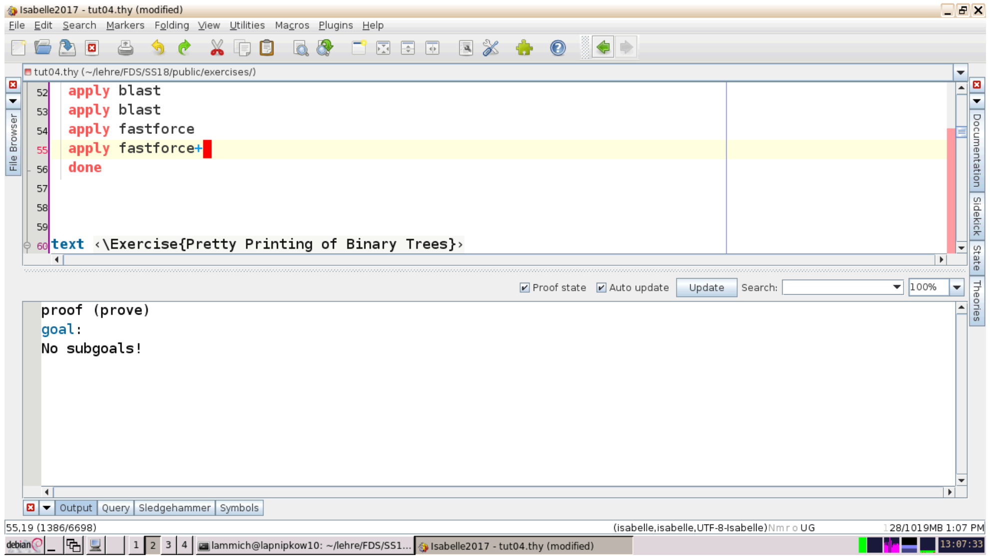
apply (induction t)
apply simp
apply (clarsimp simp: filter_empty_conv)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv filter_empty_conv')
done
proof (prove)
goal (10 subgoals):
1. ⋀t1 t2.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
Ball (set_tree t2) (op < v); u = v; v ∈ set_tree t1]
⟹ False
2. ⋀t1 t2.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];

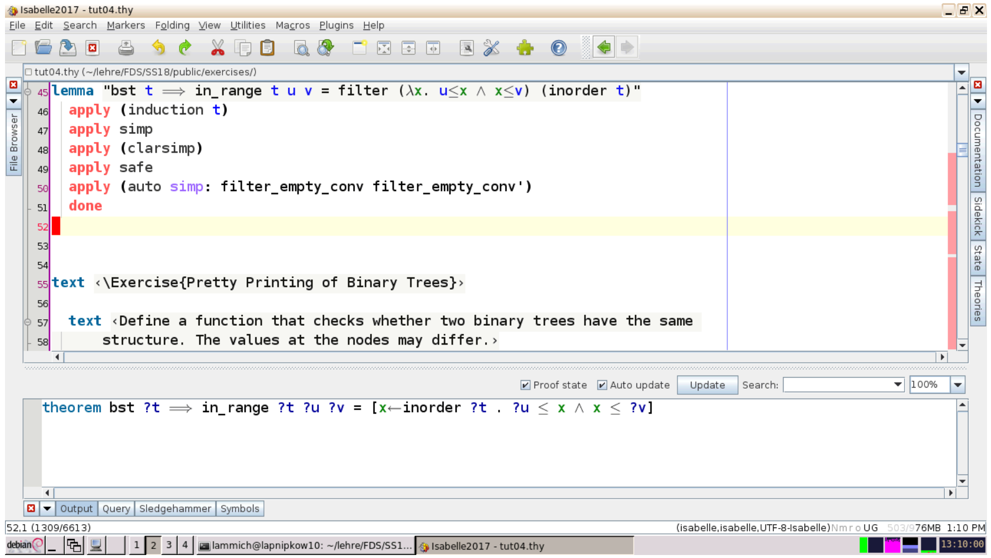


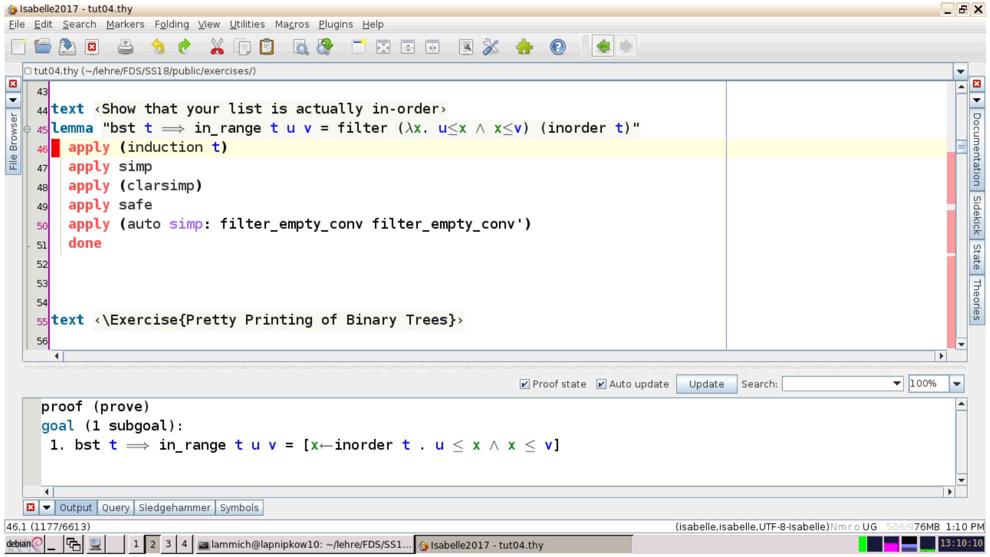
apply (induction t)
apply simp
apply (clarsimp)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv filter_empty_conv')
done
proof (prove)
goal (18 subgoals):
1. ⋀t1 t2.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
Ball (set_tree t2) (op < v); u = v]
⟹ [x←inorder t1 . v ≤ x ∧ x ≤ v] = []
2. ⋀t1 t2.
[in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];

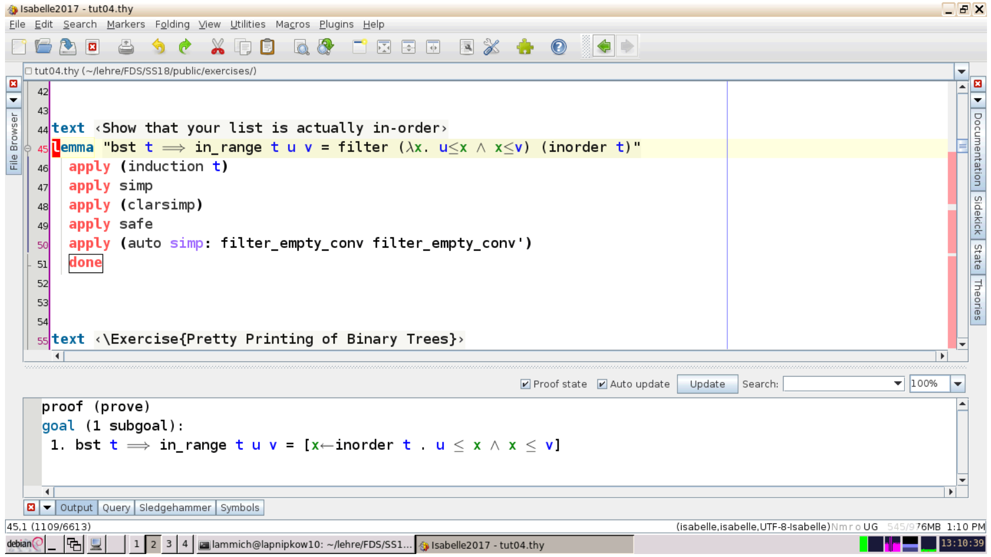

Isabelle2017 - tut04.thy
apply (induction t)
apply simp
apply (clarsimp)
apply safe
apply simp_all
apply (auto simp: filter_empty_conv filter_empty_conv')
done
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
Ball (set_tree t2) (op < v); u = v⟧
⟹ [x←inorder t1 . v ≤ x ∧ x ≤ v] = []
2. ⋀t1 t2.
⟦in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
Ball (set_tree t2) (op < v); u = v⟧
⟹ [x←inorder t2 . v ≤ x ∧ x ≤ v] = []
3. ⋀t1 x2 t2.

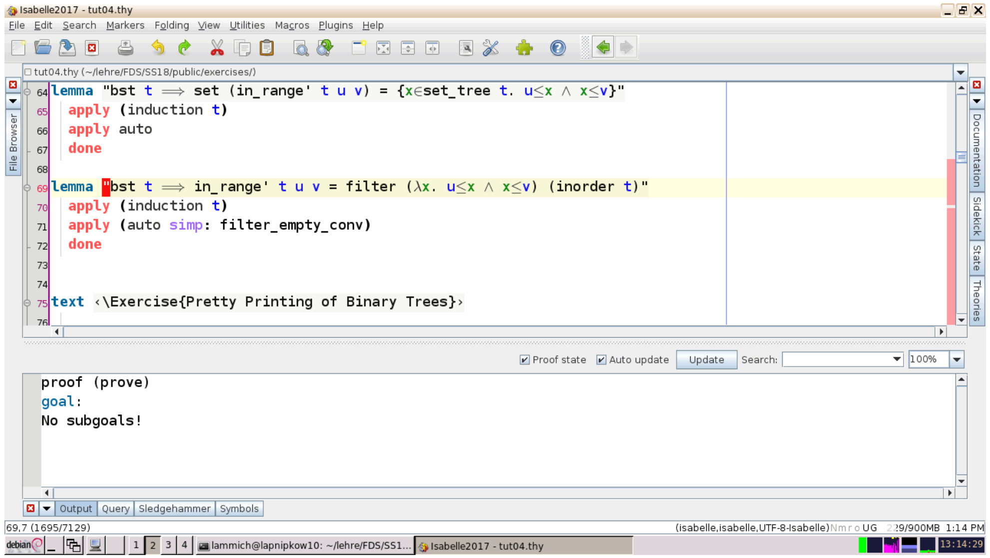

Isabelle2017 - tut04.thy (modified)
apply (clarsimp)
apply safe
apply simp_all
apply (simp_all add: filter_empty_conv filter_empty_conv')
apply
done
proof (prove)
goal (18 subgoals):
1. ⋀t1 t2.
⟦in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
in_range t2 v v = [x←inorder t2 . v ≤ x ∧ x ≤ v]; bst t1; bst t2; ∀x∈set_tree t1. x < v;
∀x∈set_tree t2. v < x; u = v⟧
⟹ ∀x∈set_tree t1. v ≤ x ⟶ x ≠ v
2. ⋀t1 t2.
⟦in_range t1 v v = [x←inorder t1 . v ≤ x ∧ x ≤ v];
Input/output complete

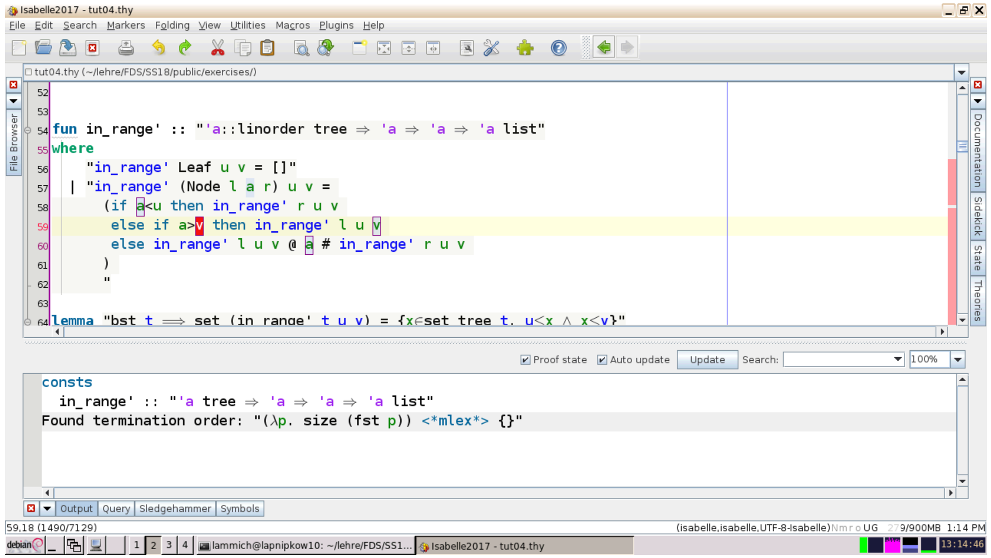

Isabelle2017 - tut04.thy (modified)
apply blast
apply blast
apply fastforce
apply fastforce
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
proof (prove)
goal:
No subgoals!

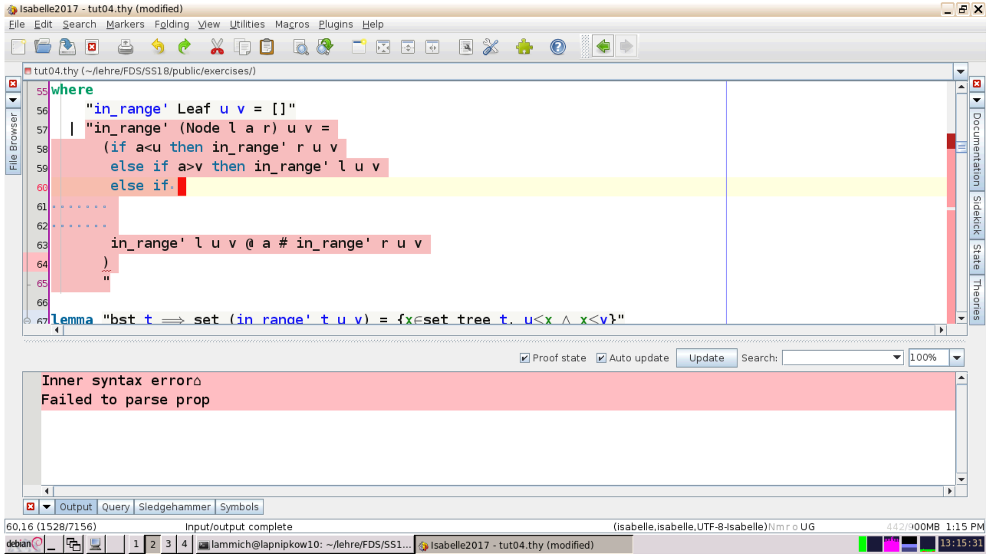

Isabelle2017 - tut04.thy
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp)
apply safe
apply (auto simp: filter_empty_conv filter_empty_conv')
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
text ‹Define a function that checks whether two binary trees have the same
structure. The values at the nodes may differ.›
theorem bst ?t ⟹ in_range ?t ?u ?v = [x←inorder ?t . ?u ≤ x ∧ x ≤ ?v]

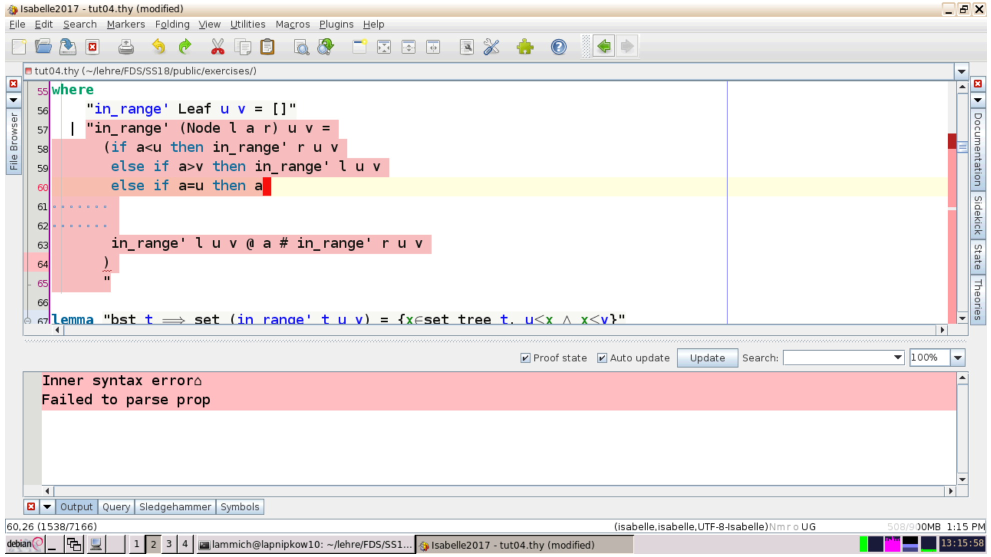

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp)
apply safe
apply (auto simp: filter_empty_conv filter_empty_conv')
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
proof (prove)
goal (1 subgoal):
1. bst t ⟹ in_range t u v = [x←inorder t . u ≤ x ∧ x ≤ v]

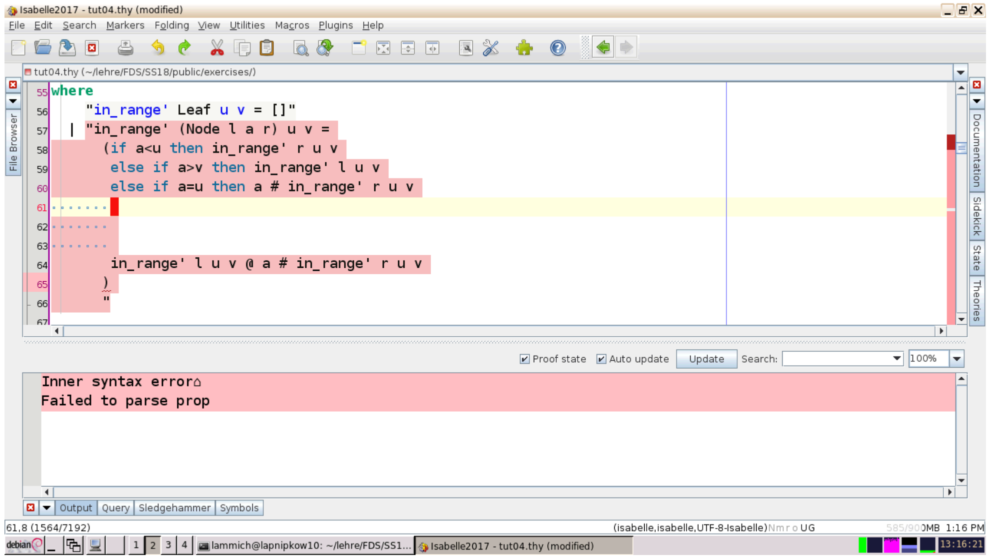

Isabelle2017 - tut04.thy
text ‹Show that your list is actually in-order›
lemma "bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply simp
apply (clarsimp)
apply safe
apply (auto simp: filter_empty_conv filter_empty_conv')
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
proof (prove)
goal (1 subgoal):
1. bst t ⟹ in_range t u v = [x←inorder t . u ≤ x ∧ x ≤ v]

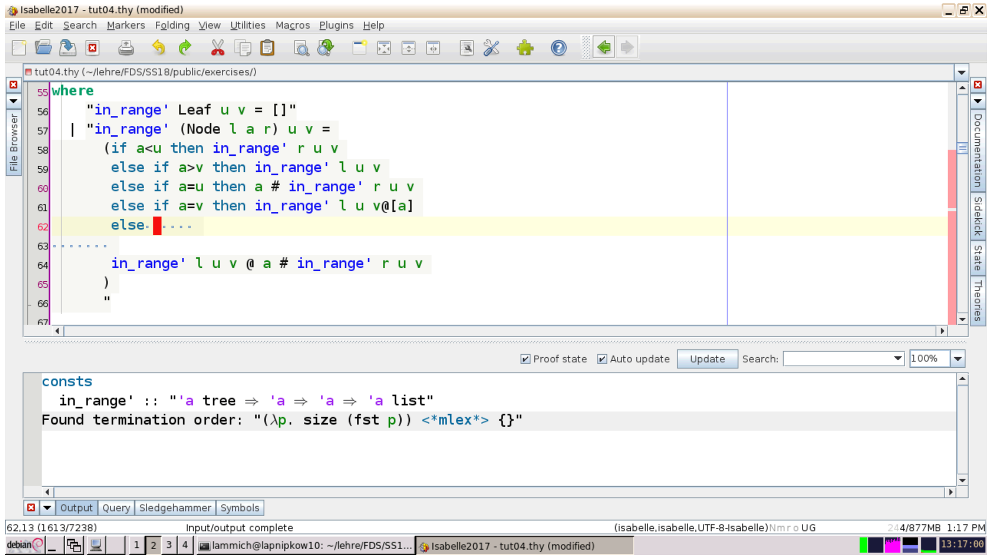

Isabelle2017 - tut04.thy
lemma "bst t ⟹ set (in_range' t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
apply (induction t)
apply auto
done
lemma "bst t ⟹ in_range' t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply (auto simp: filter_empty_conv)
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
proof (prove)
goal:
No subgoals!

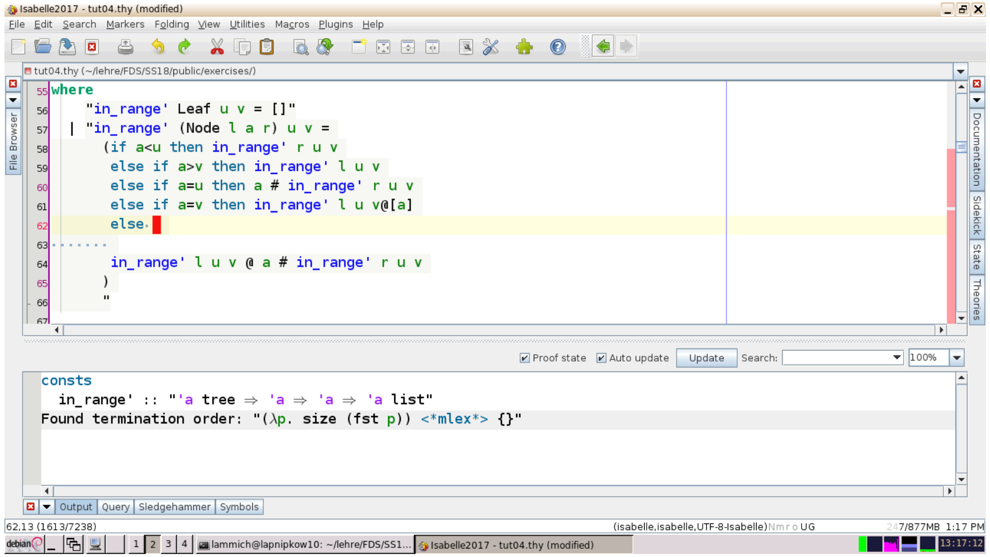

Isabelle2017 - tut04.thy
fun in_range' :: "'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"
where
"in_range' Leaf u v = []"
| "in_range' (Node l a r) u v =
(if a<u then in_range' r u v
else if a>v then in_range' l u v
else in_range' l u v @ a # in_range' r u v
)
"
consts
in_range' :: "'a tree ⇒ 'a ⇒ 'a ⇒ 'a list"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

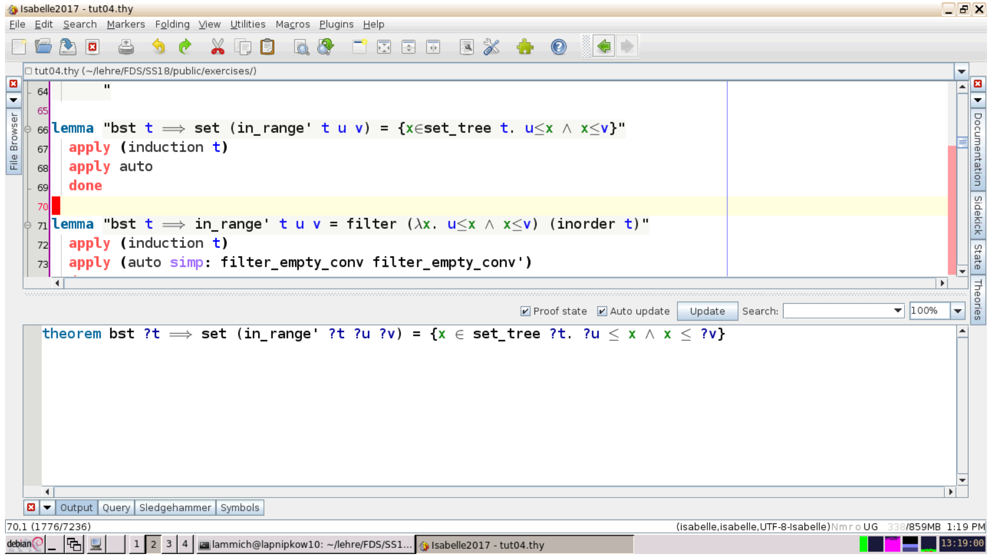

Isabelle2017 - tut04.thy (modified)
where
"in_range' Leaf u v = []"
| "in_range' (Node l a r) u v =
(if a<u then in_range' r u v
else if a>v then in_range' l u v
else if
in_range' l u v @ a # in_range' r u v
)
"
lemma "bst t ⟹ set (in_range' t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
Inner syntax error
Failed to parse prop

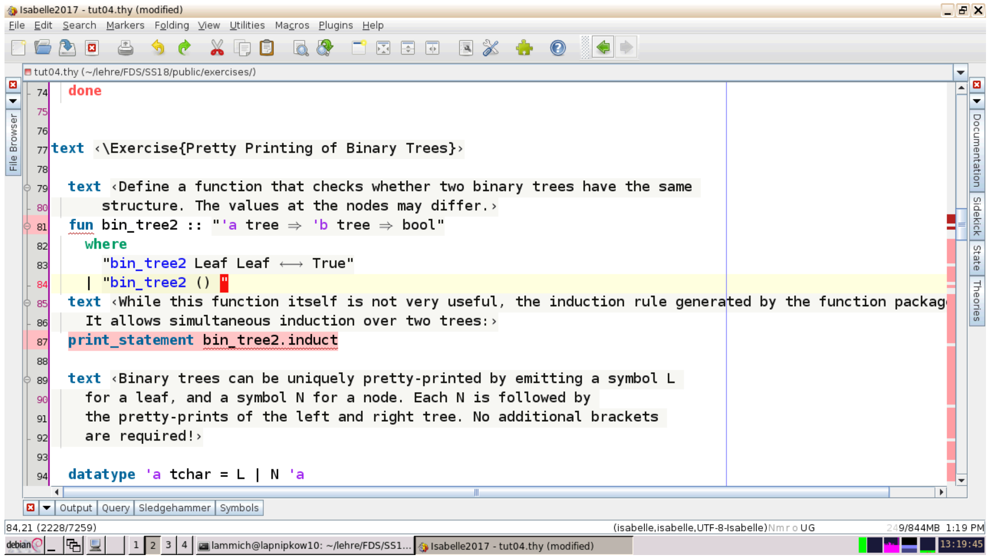

Isabelle2017 - tut04.thy (modified)
where
"in_range' Leaf u v = []"
| "in_range' (Node l a r) u v =
(if a<u then in_range' r u v
else if a>v then in_range' l u v
else if a=u then a
in_range' l u v @ a # in_range' r u v
)
"
lemma "bst t ⟹ set (in_range' t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
Inner syntax error
Failed to parse prop

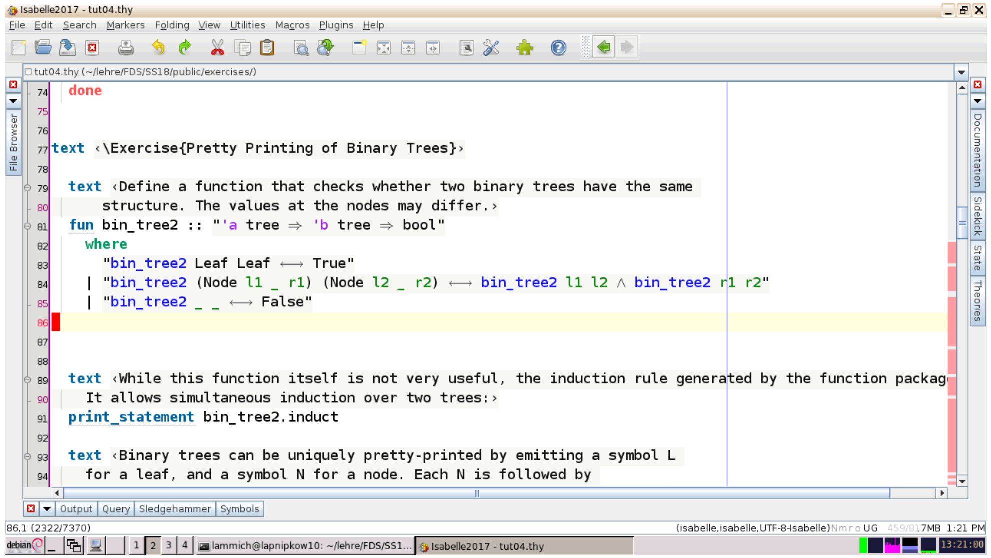

Isabelle2017 - tut04.thy (modified)
where
"in_range' Leaf u v = []"
| "in_range' (Node l a r) u v =
(if a<u then in_range' r u v
else if a>v then in_range' l u v
else if a=u then a # in_range' r u v
in_range' l u v @ a # in_range' r u v
)
"
Inner syntax error
Failed to parse prop

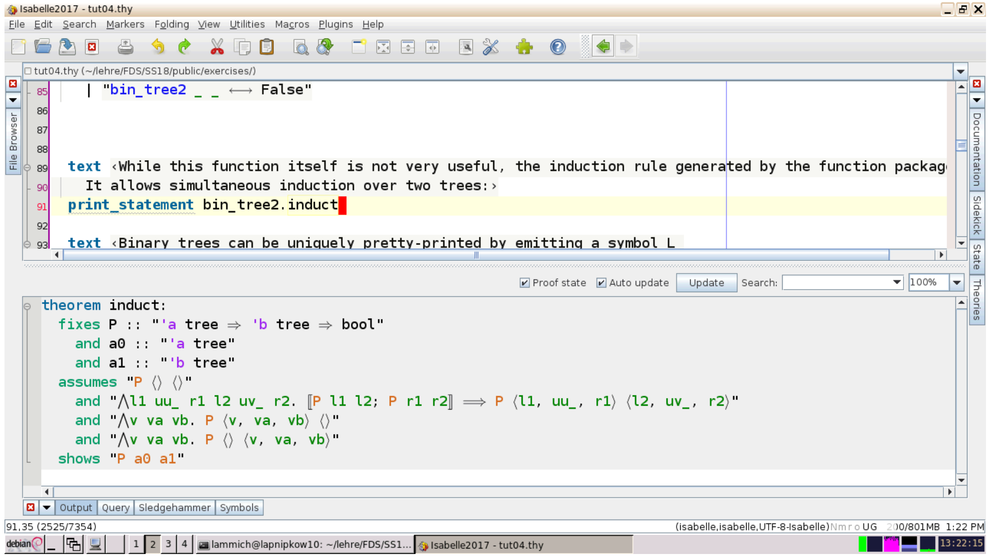

Isabelle2017 - tut04.thy (modified)
where
"in_range' Leaf u v = []"
| "in_range' (Node l a r) u v =
(if a<u then in_range' r u v
else if a>v then in_range' l u v
else if a=u then a # in_range' r u v
else if a=v then in_range' l u v@[a]
else
in_range' l u v @ a # in_range' r u v
)
"
consts
in_range' :: "'a tree ⇒ 'a ⇒ 'a ⇒ 'a list"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

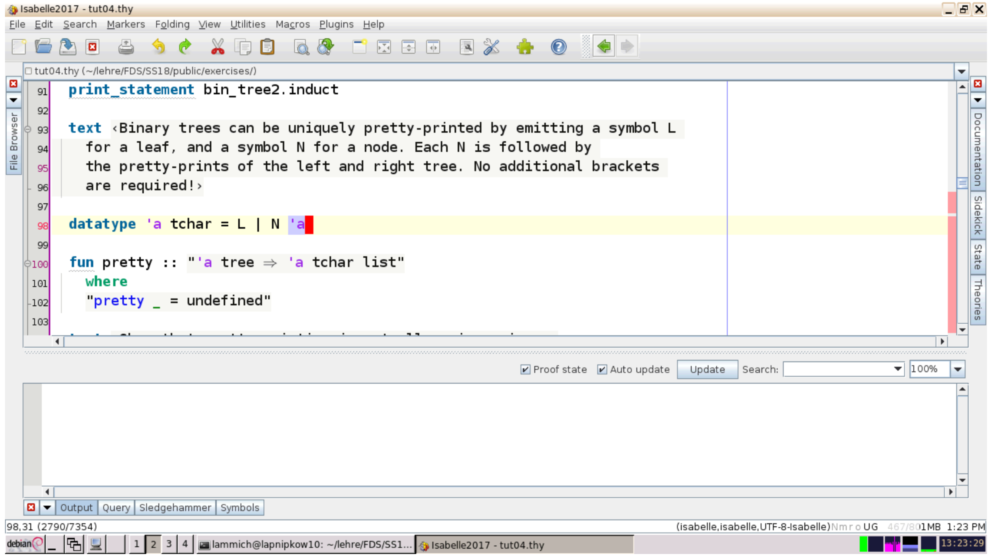

Isabelle2017 - tut04.thy (modified)
where
"in_range' Leaf u v = []"
| "in_range' (Node l a r) u v =
(if a<u then in_range' r u v
else if a>v then in_range' l u v
else if a=u then a # in_range' r u v
else if a=v then in_range' l u v@[a]
else
in_range' l u v @ a # in_range' r u v
)
"
consts
in_range' :: "'a tree ⇒ 'a ⇒ 'a ⇒ 'a list"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

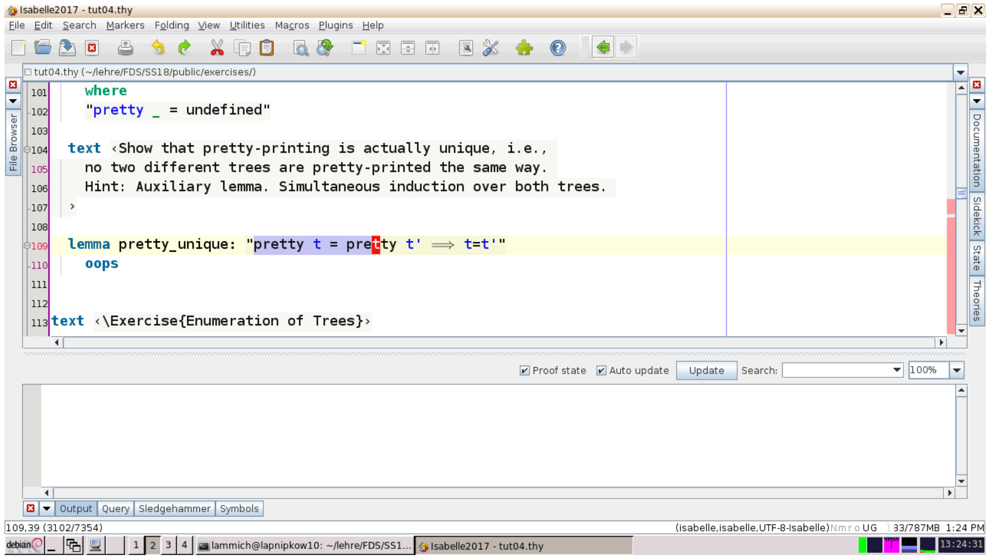

Isabelle2017 - tut04.thy
"
lemma "bst t ⟹ set (in_range' t u v) = {x∈set_tree t. u≤x ∧ x≤v}"
apply (induction t)
apply auto
done
lemma "bst t ⟹ in_range' t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"
apply (induction t)
apply (auto simp: filter_empty_conv filter_empty_conv')
theorem bst ?t ⟹ set (in_range' ?t ?u ?v) = {x ∈ set_tree ?t. ?u ≤ x ∧ x ≤ ?v}

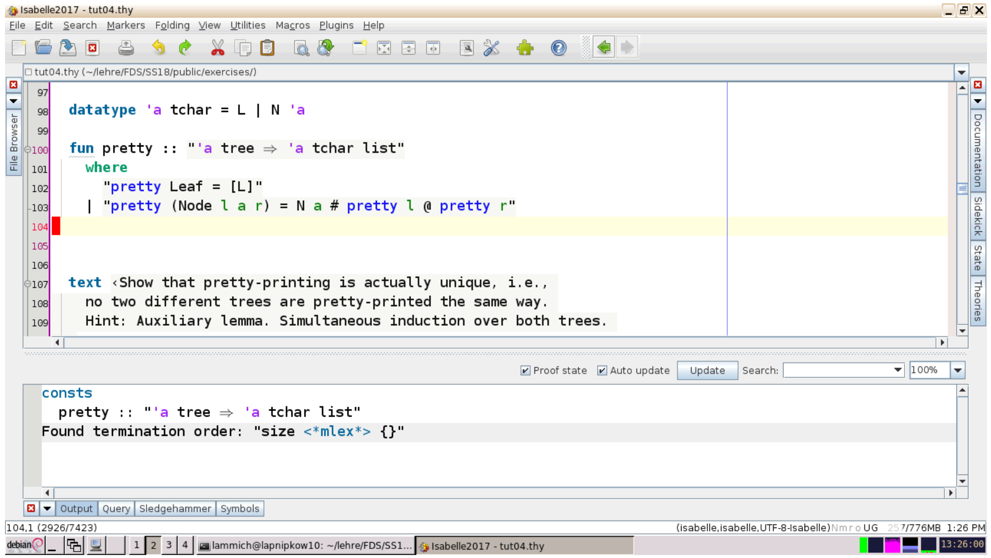

Isabelle2017 - tut04.thy (modified)
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
text ‹Define a function that checks whether two binary trees have the same
structure. The values at the nodes may differ.›
fun bin_tree2 :: "'a tree ⇒ 'b tree ⇒ bool"
where
"bin_tree2 Leaf Leaf ⟷ True"
| "bin_tree2 () "
text ‹While this function itself is not very useful, the induction rule generated by the function packag
It allows simultaneous induction over two trees:›
print_statement bin_tree2.induct
text ‹Binary trees can be uniquely pretty-printed by emitting a symbol L
for a leaf, and a symbol N for a node. Each N is followed by
the pretty-prints of the left and right tree. No additional brackets
are required!›
datatype 'a tchar = L | N 'a

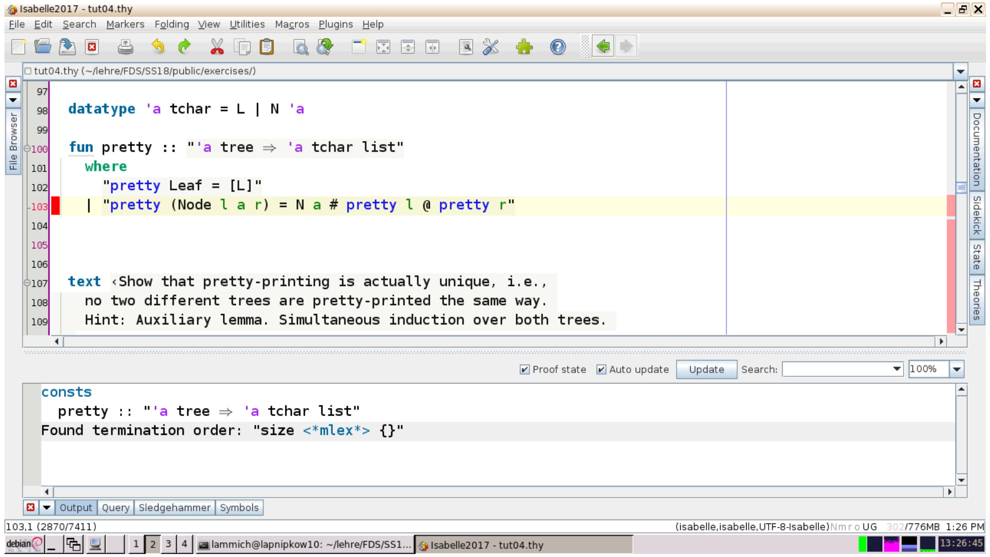

Isabelle2017 - tut04.thy
done
text ‹\Exercise{Pretty Printing of Binary Trees}›
text ‹Define a function that checks whether two binary trees have the same
structure. The values at the nodes may differ.›
fun bin_tree2 :: "'a tree ⇒ 'b tree ⇒ bool"
where
"bin_tree2 Leaf Leaf ⟷ True"
| "bin_tree2 (Node l1 _ r1) (Node l2 _ r2) ⟷ bin_tree2 l1 l2 ∧ bin_tree2 r1 r2"
| "bin_tree2 _ _ ⟷ False"
text ‹While this function itself is not very useful, the induction rule generated by the function packag
It allows simultaneous induction over two trees:›
print_statement bin_tree2.induct
text ‹Binary trees can be uniquely pretty-printed by emitting a symbol L
for a leaf, and a symbol N for a node. Each N is followed by

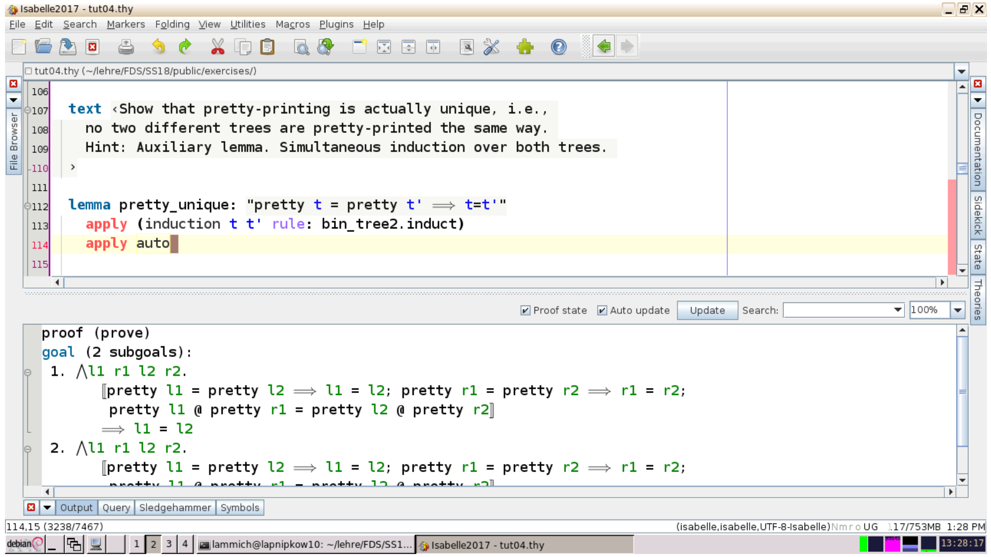

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
| "bin_tree2 _ _ ⟷ False"
text ‹While this function itself is not very useful, the induction rule generated by the function packag
It allows simultaneous induction over two trees:›
print_statement bin_tree2.induct
text ‹Binary trees can be uniquely pretty-printed by emitting a symbol L
Proof state
Auto update
Update
Search:
100%
theorem induct:
fixes P :: "'a tree ⇒ 'b tree ⇒ bool"
and a0 :: "'a tree"
and a1 :: "'b tree"
assumes "P ⟨⟩ ⟨⟩"
and "⋀l1 uu_ r1 l2 uv_ r2. ⟦P l1 l2; P r1 r2⟧ ⟹ P ⟨l1, uu_, r1⟩ ⟨l2, uv_, r2⟩"
and "⋀v va vb. P ⟨v, va, vb⟩ ⟨⟩"
and "⋀v va vb. P ⟨⟩ ⟨v, va, vb⟩"
shows "P a0 a1"
Output
Query
Sledgehammer
Symbols

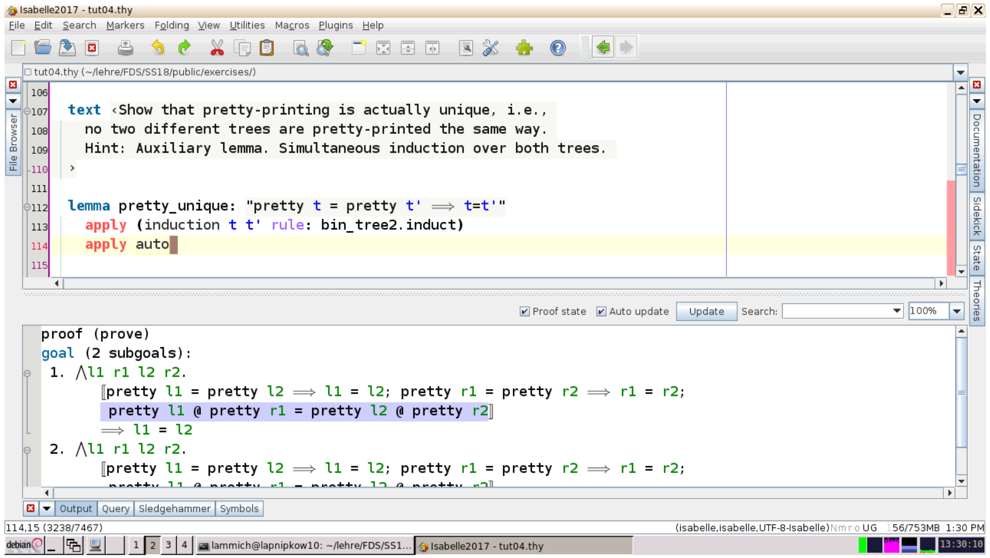

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
print_statement bin_tree2.induct
text ‹Binary trees can be uniquely pretty-printed by emitting a symbol L
for a leaf, and a symbol N for a node. Each N is followed by
the pretty-prints of the left and right tree. No additional brackets
are required!›
datatype 'a tchar = L | N 'a
fun pretty :: "'a tree ⇒ 'a tchar list"
where
"pretty _ = undefined"
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols

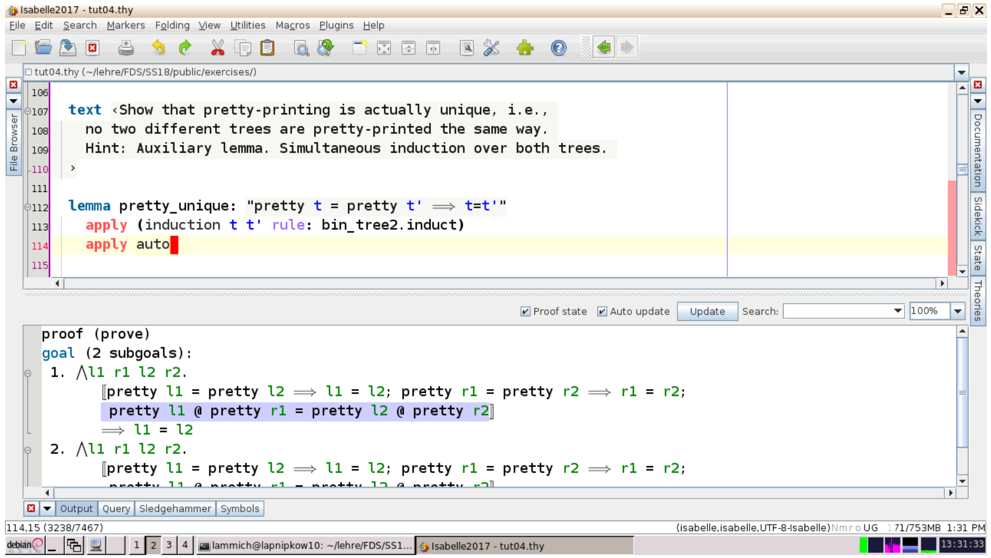

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
where
"pretty _ = undefined"
text ‹Show that pretty-printing is actually unique, i.e.,
no two different trees are pretty-printed the same way.
Hint: Auxiliary lemma. Simultaneous induction over both trees.
›
lemma pretty_unique: "pretty t = pretty t' ⟹ t=t'"
oops
text ‹\Exercise{Enumeration of Trees}›
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols

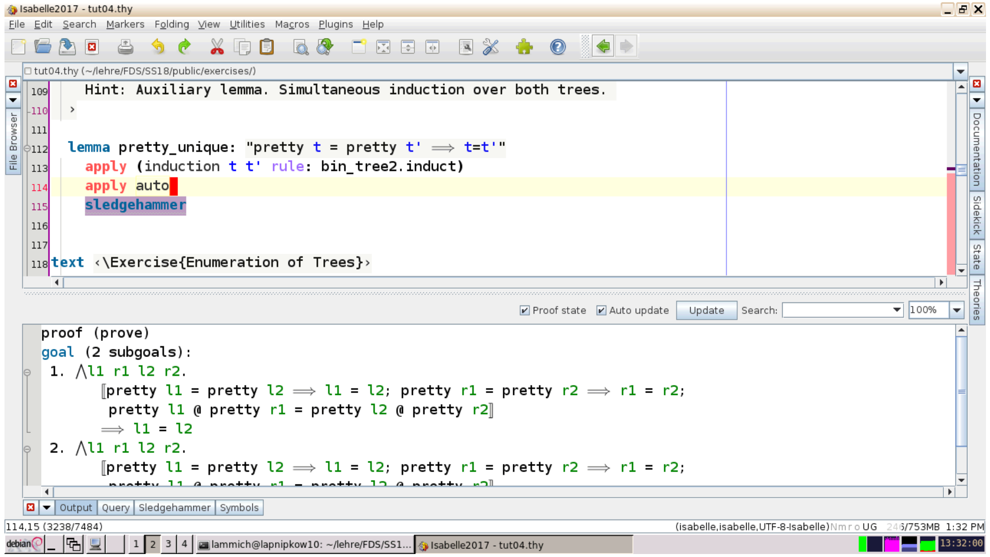

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
datatype 'a tchar = L | N 'a
fun pretty :: "'a tree ⇒ 'a tchar list"
where
"pretty Leaf = [L]"
| "pretty (Node l a r) = N a # pretty l @ pretty r"
text ‹Show that pretty-printing is actually unique, i.e.,
no two different trees are pretty-printed the same way.
Hint: Auxiliary lemma. Simultaneous induction over both trees.
Proof state
Auto update
Update
Search:
100%
consts
pretty :: "'a tree ⇒ 'a tchar list"
Found termination order: "size <*mlex*> {}"
Output
Query
Sledgehammer
Symbols

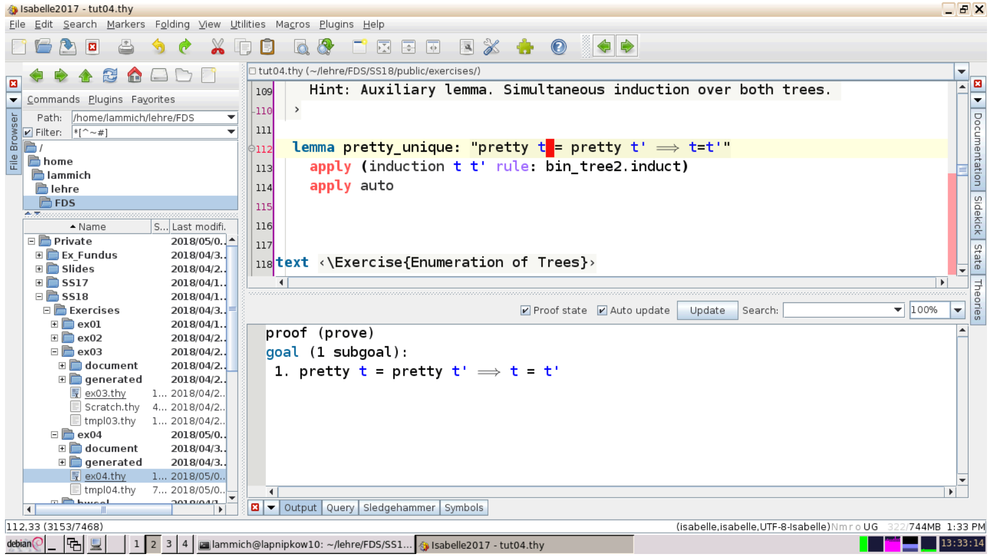

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
datatype 'a tchar = L | N 'a
fun pretty :: "'a tree ⇒ 'a tchar list"
where
"pretty Leaf = [L]"
| "pretty (Node l a r) = N a # pretty l @ pretty r"
text ‹Show that pretty-printing is actually unique, i.e.,
no two different trees are pretty-printed the same way.
Hint: Auxiliary lemma. Simultaneous induction over both trees.
consts
pretty :: "'a tree ⇒ 'a tchar list"
Found termination order: "size <*mlex*> {}"

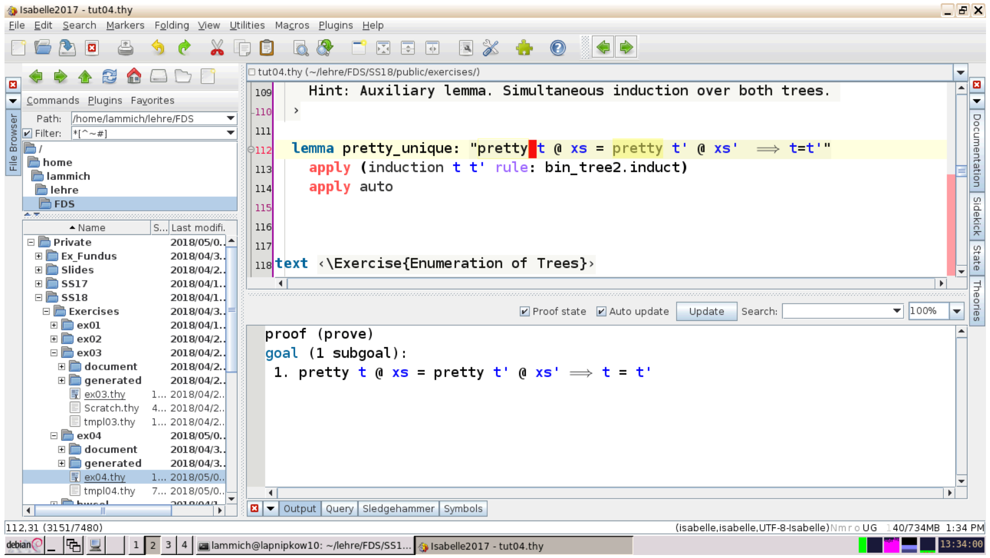

Isabelle2017 - tut04.thy
text ‹Show that pretty-printing is actually unique, i.e.,
no two different trees are pretty-printed the same way.
Hint: Auxiliary lemma. Simultaneous induction over both trees.
›
lemma pretty_unique: "pretty t = pretty t' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
proof (prove)
goal (2 subgoals):
1. ⋀l1 r1 l2 r2.
⟦pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;
pretty l1 @ pretty r1 = pretty l2 @ pretty r2⟧
⟹ l1 = l2
2. ⋀l1 r1 l2 r2.
⟦pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;

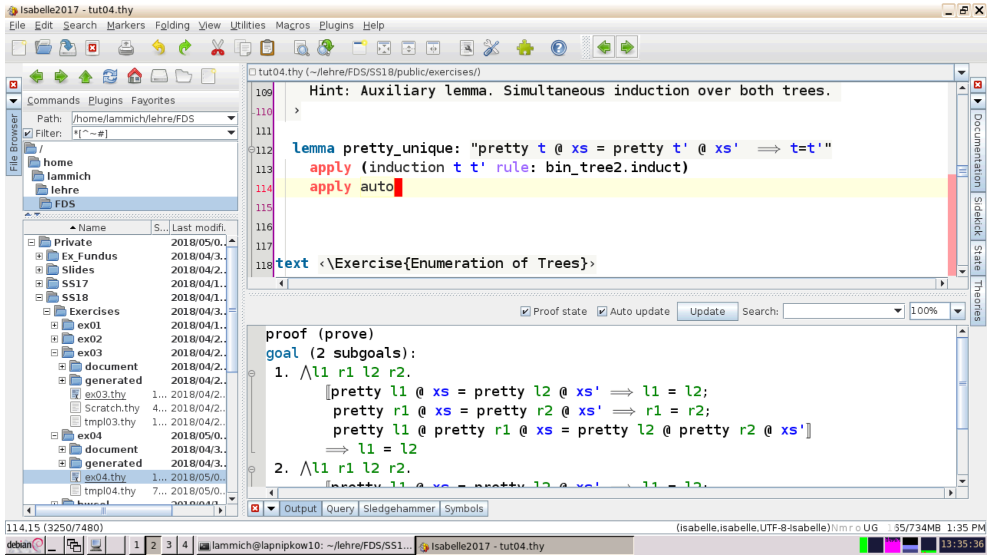

Isabelle2017 - tut04.thy
text ‹Show that pretty-printing is actually unique, i.e.,
no two different trees are pretty-printed the same way.
Hint: Auxiliary lemma. Simultaneous induction over both trees.
›
lemma pretty_unique: "pretty t = pretty t' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
proof (prove)
goal (2 subgoals):
1. ⋀l1 r1 l2 r2.
⟦pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;
pretty l1 @ pretty r1 = pretty l2 @ pretty r2⟧
⟹ l1 = l2
2. ⋀l1 r1 l2 r2.
⟦pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;

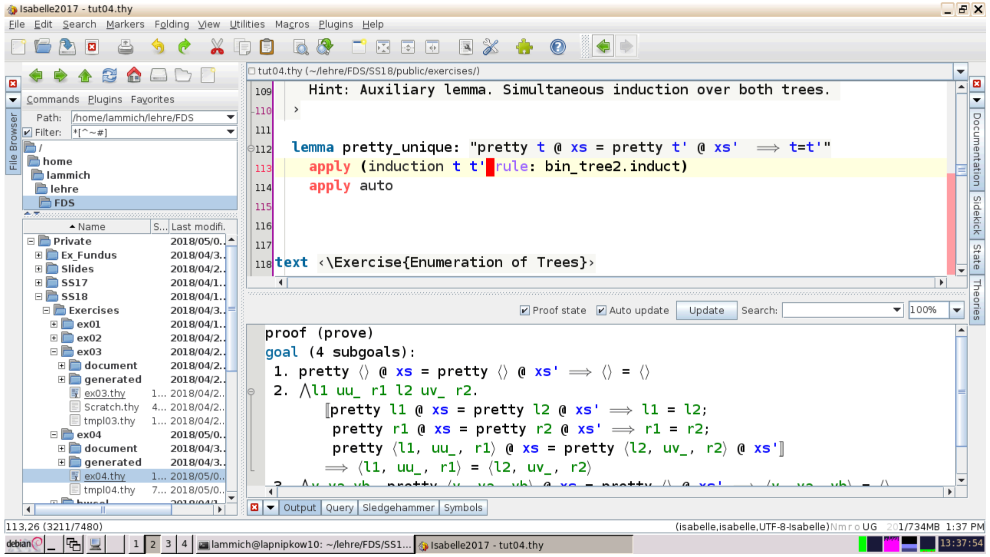

Isabelle2017 - tut04.thy
text ‹Show that pretty-printing is actually unique, i.e.,
no two different trees are pretty-printed the same way.
Hint: Auxiliary lemma. Simultaneous induction over both trees.
›
lemma pretty_unique: "pretty t = pretty t' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
proof (prove)
goal (2 subgoals):
1. ⋀l1 r1 l2 r2.
⟦pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;
pretty l1 @ pretty r1 = pretty l2 @ pretty r2⟧
⟹ l1 = l2
2. ⋀l1 r1 l2 r2.
⟦pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;

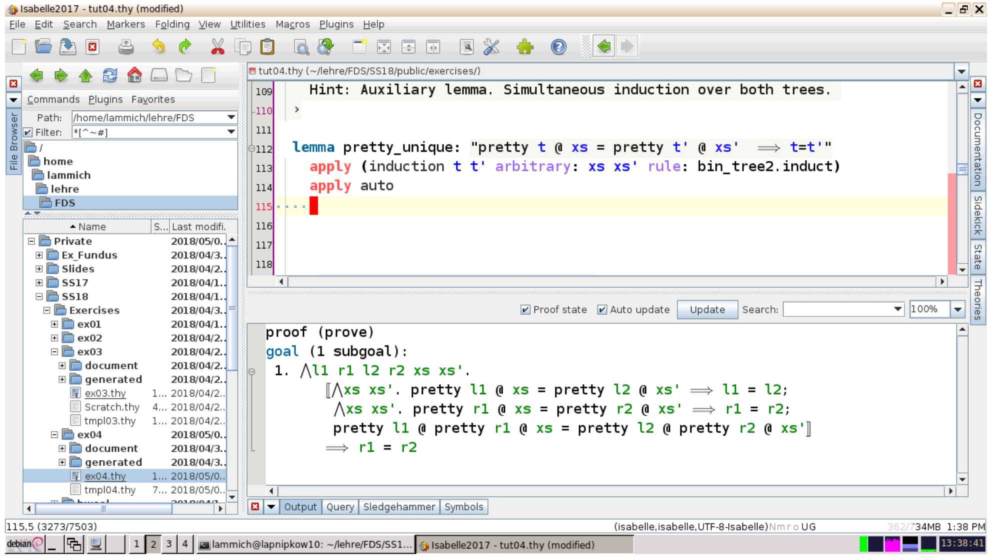

Isabelle2017 - tut04.thy
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t = pretty t' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
sledgehammer
text ‹\Exercise{Enumeration of Trees}›
proof (prove)
goal (2 subgoals):
1. ⋀l1 r1 l2 r2.
[pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;
pretty l1 @ pretty r1 = pretty l2 @ pretty r2]
⟹ l1 = l2
2. ⋀l1 r1 l2 r2.
[pretty l1 = pretty l2 ⟹ l1 = l2; pretty r1 = pretty r2 ⟹ r1 = r2;

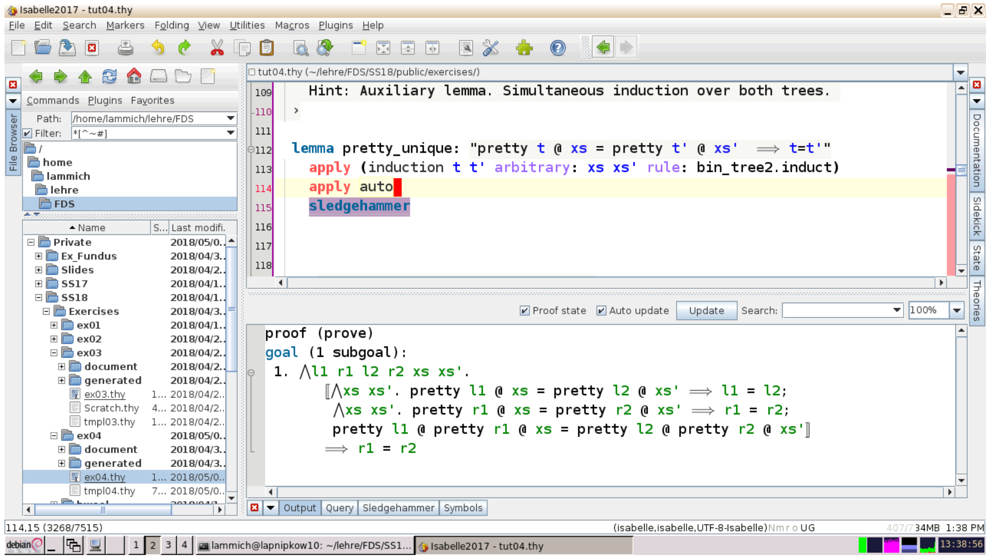

Isabelle2017 - tut04.thy
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t = pretty t' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
text ‹\Exercise{Enumeration of Trees}›
proof (prove)
goal (1 subgoal):
1. pretty t = pretty t' ⟹ t = t'

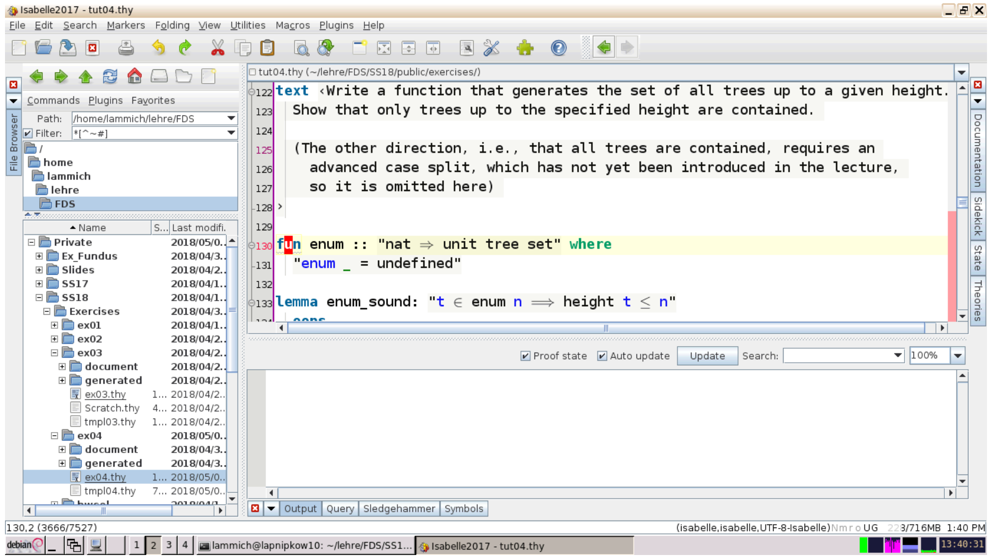

Isabelle2017 - tut04.thy
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t @ xs = pretty t' @ xs' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
text ‹\Exercise{Enumeration of Trees}›
proof (prove)
goal (1 subgoal):
1. pretty t @ xs = pretty t' @ xs' ⟹ t = t'

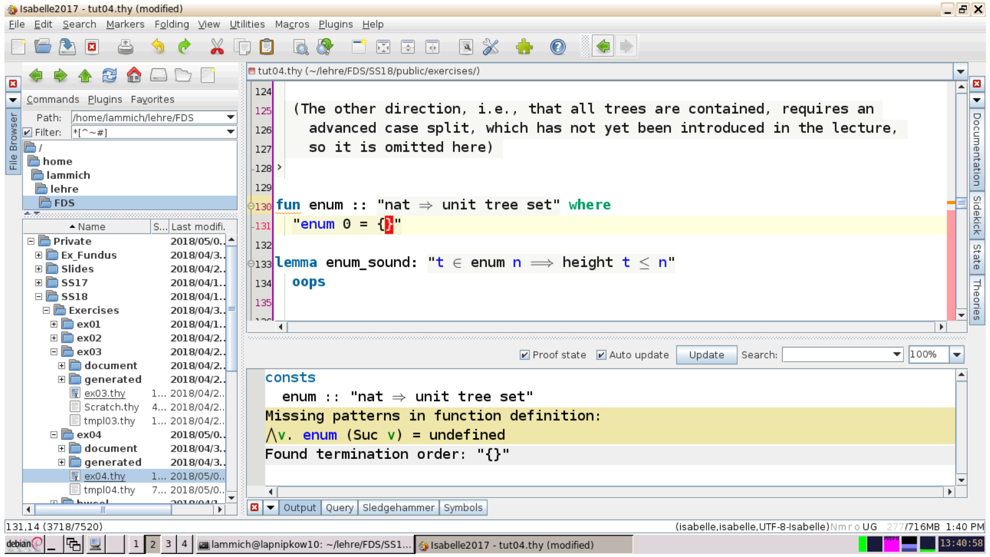

Isabelle2017 - tut04.thy
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t @ xs = pretty t' @ xs' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
text ‹\Exercise{Enumeration of Trees}›
proof (prove)
goal (2 subgoals):
1. ⋀l1 r1 l2 r2.
[pretty l1 @ xs = pretty l2 @ xs' ⟹ l1 = l2;
pretty r1 @ xs = pretty r2 @ xs' ⟹ r1 = r2;
pretty l1 @ pretty r1 @ xs = pretty l2 @ pretty r2 @ xs']
⟹ l1 = l2
2. ⋀l1 r1 l2 r2.

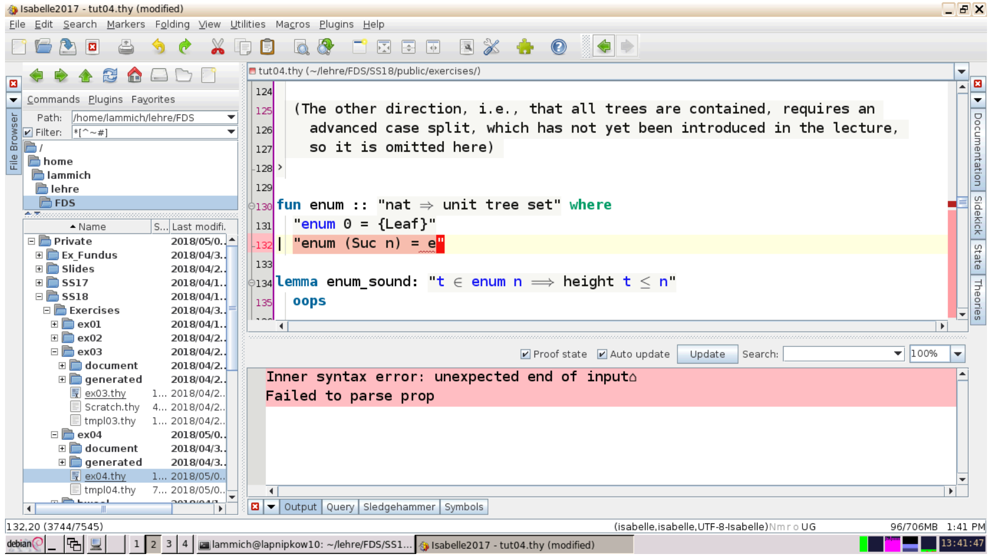

Isabelle2017 - tut04.thy
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t @ xs = pretty t' @ xs' ⟹ t=t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto
text ‹\Exercise{Enumeration of Trees}›
proof (prove)
goal (4 subgoals):
1. pretty ⟨⟩ @ xs = pretty ⟨⟩ @ xs' ⟹ ⟨⟩ = ⟨⟩
2. ⋀l1 uu_ r1 l2 uv_ r2.
⟦pretty l1 @ xs = pretty l2 @ xs' ⟹ l1 = l2;
pretty r1 @ xs = pretty r2 @ xs' ⟹ r1 = r2;
pretty ⟨l1, uu_, r1⟩ @ xs = pretty ⟨l2, uv_, r2⟩ @ xs'⟧
⟹ ⟨l1, uu_, r1⟩ = ⟨l2, uv_, r2⟩

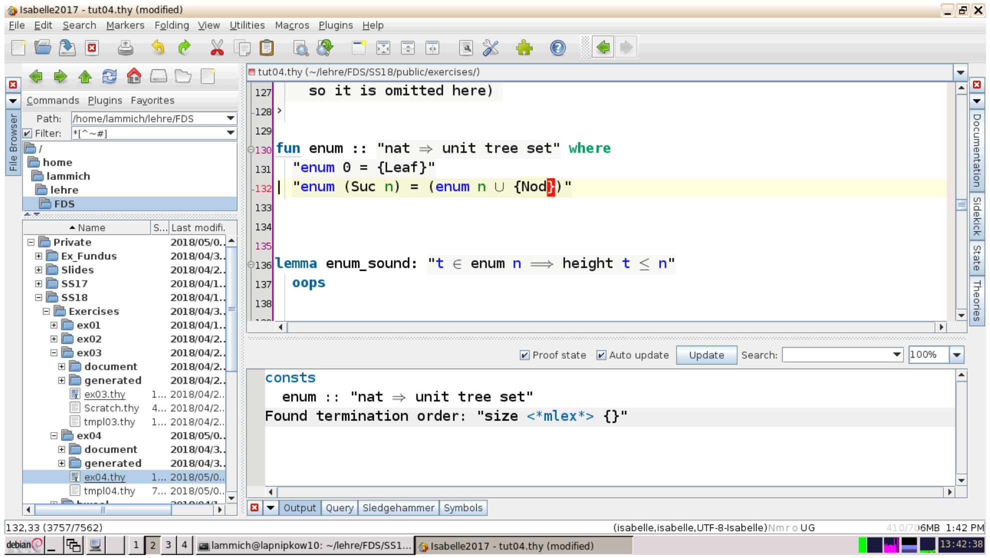

Isabelle2017 - tut04.thy (modified)
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t @ xs = pretty t' @ xs' ⟹ t=t'"
apply (induction t t' arbitrary: xs xs' rule: bin_tree2.induct)
apply auto
proof (prove)
goal (1 subgoal):
1. ⋀l1 r1 l2 r2 xs xs'.
⟦⋀xs xs'. pretty l1 @ xs = pretty l2 @ xs' ⟹ l1 = l2;
⋀xs xs'. pretty r1 @ xs = pretty r2 @ xs' ⟹ r1 = r2;
pretty l1 @ pretty r1 @ xs = pretty l2 @ pretty r2 @ xs'⟧
⟹ r1 = r2

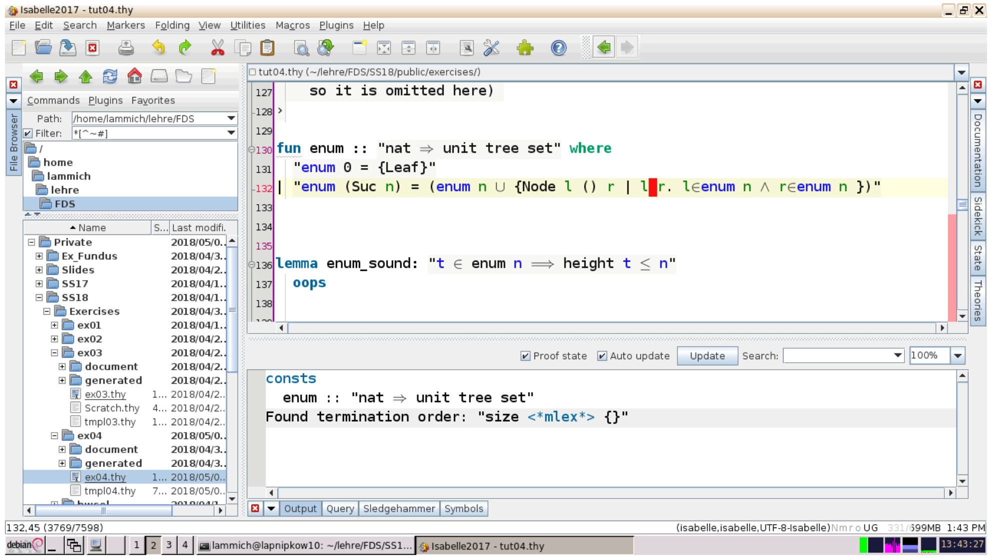

Isabelle2017 - tut04.thy
Hint: Auxiliary lemma. Simultaneous induction over both trees.
lemma pretty_unique: "pretty t @ xs = pretty t' @ xs' ⟹ t=t'"
apply (induction t t' arbitrary: xs xs' rule: bin_tree2.induct)
apply auto
sledgehammer
proof (prove)
goal (1 subgoal):
1. ⋀l1 r1 l2 r2 xs xs'.
⟦⋀xs xs'. pretty l1 @ xs = pretty l2 @ xs' ⟹ l1 = l2;
⋀xs xs'. pretty r1 @ xs = pretty r2 @ xs' ⟹ r1 = r2;
pretty l1 @ pretty r1 @ xs = pretty l2 @ pretty r2 @ xs'⟧
⟹ r1 = r2

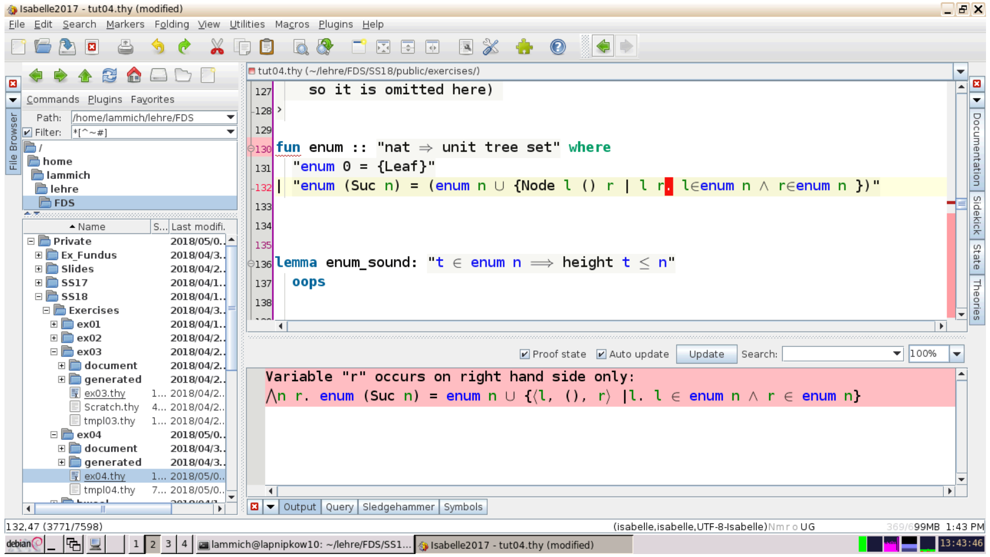

Isabelle2017 - tut04.thy
text ‹Write a function that generates the set of all trees up to a given height.
Show that only trees up to the specified height are contained.
(The other direction, i.e., that all trees are contained, requires an
advanced case split, which has not yet been introduced in the lecture,
so it is omitted here)
fun enum :: "nat ⇒ unit tree set" where
"enum _ = undefined"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"

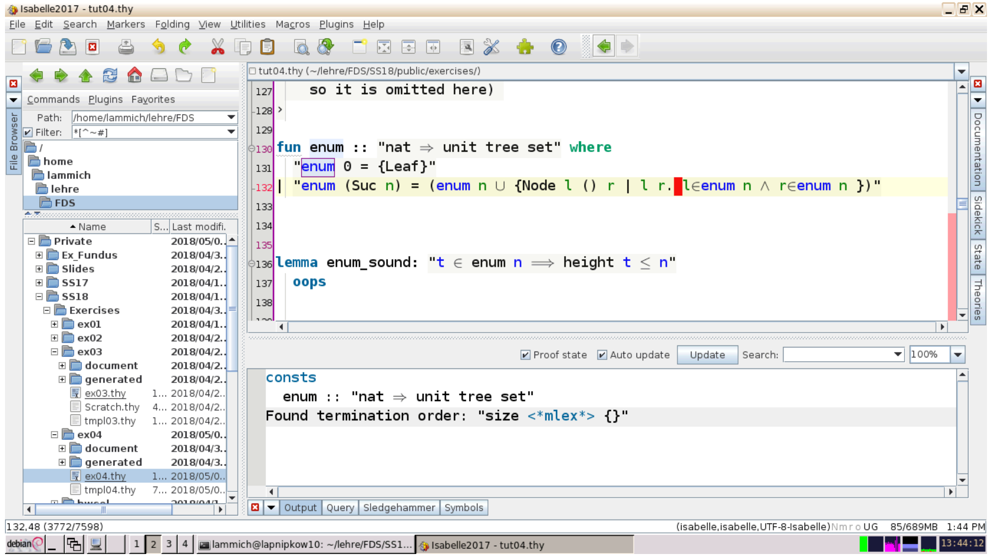

Isabelle2017 - tut04.thy (modified)
tut04.thy (~/lehre/FDS/SS18/public/exercises/)
(The other direction, i.e., that all trees are contained, requires an
advanced case split, which has not yet been introduced in the lecture,
so it is omitted here)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {}"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
oops
consts
enum :: "nat ⇒ unit tree set"
Missing patterns in function definition:
⋀v. enum (Suc v) = undefined
Found termination order: "{}"

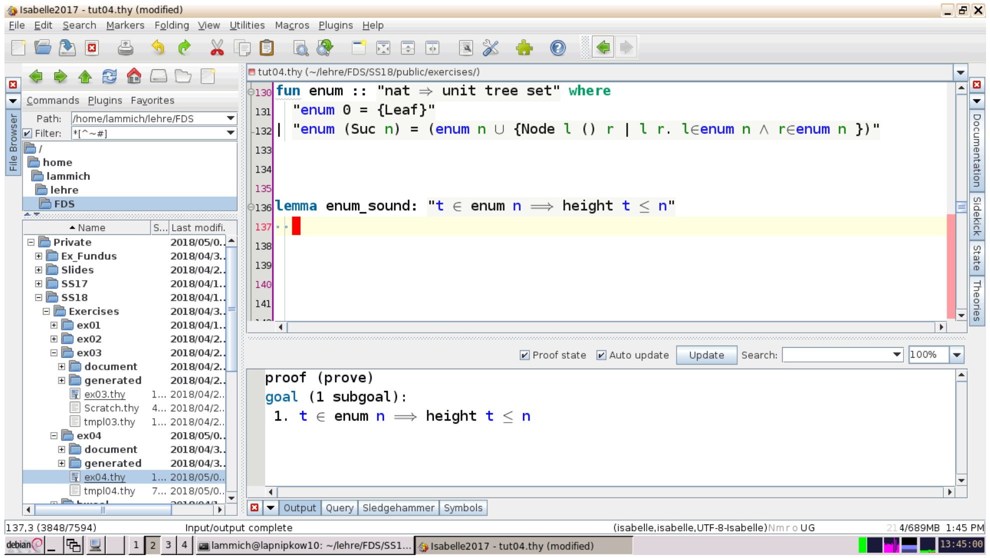

Isabelle2017 - tut04.thy (modified)
(The other direction, i.e., that all trees are contained, requires an
advanced case split, which has not yet been introduced in the lecture,
so it is omitted here)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = e"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
oops
Inner syntax error: unexpected end of input
Failed to parse prop

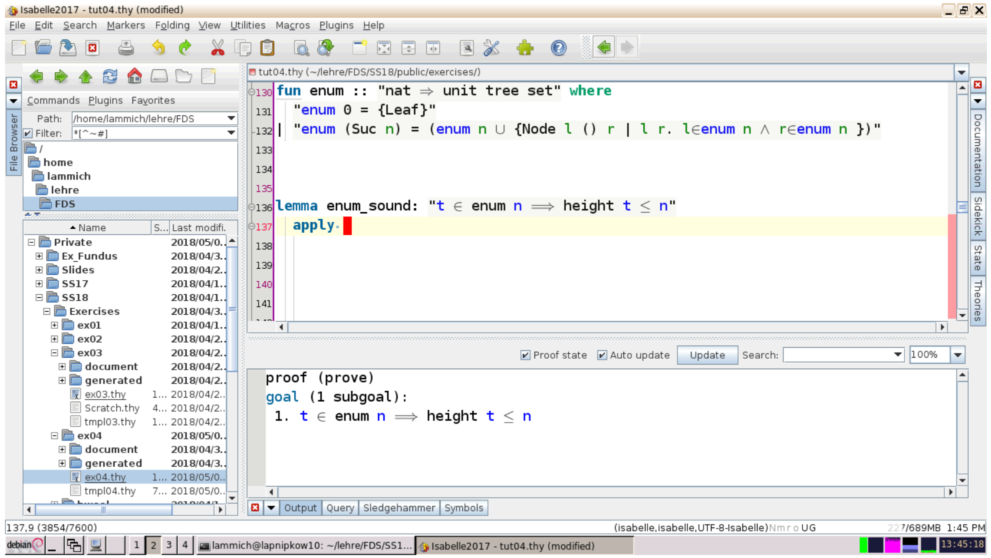

Isabelle2017 - tut04.thy (modified)
so it is omitted here)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Nod})"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
oops
consts
enum :: "nat ⇒ unit tree set"
Found termination order: "size <*mlex*> {}"

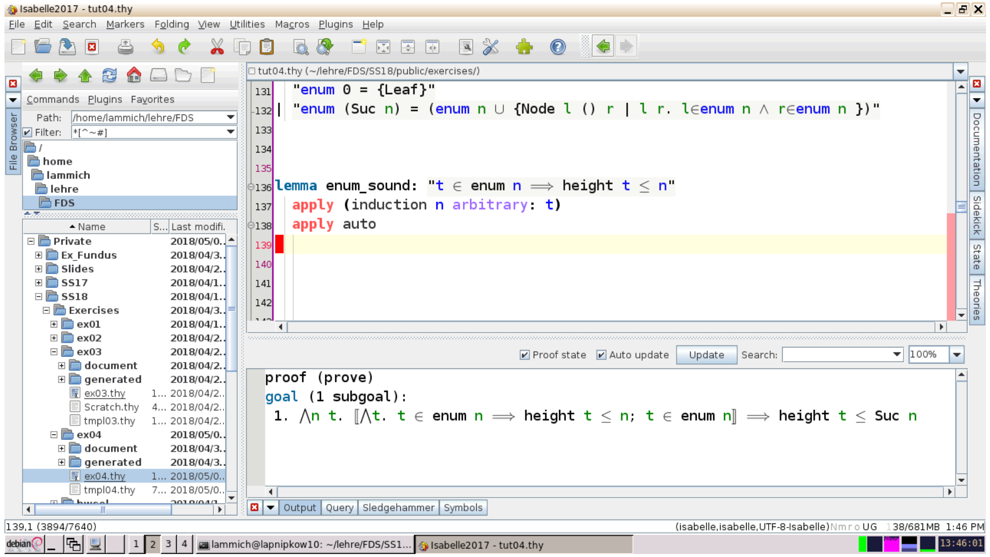

Isabelle2017 - tut04.thy
so it is omitted here)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
oops
consts
enum :: "nat ⇒ unit tree set"
Found termination order: "size <*mlex*> {}"

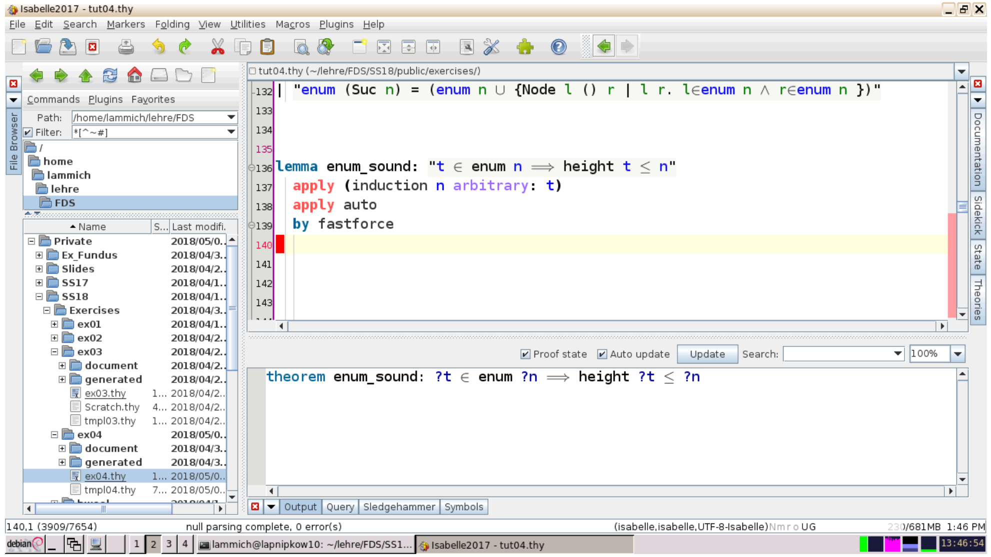

Isabelle2017 - tut04.thy (modified)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
oops
Variable "r" occurs on right hand side only:
⋀n r. enum (Suc n) = enum n ∪ {⟨l, (), r⟩ |l. l ∈ enum n ∧ r ∈ enum n}

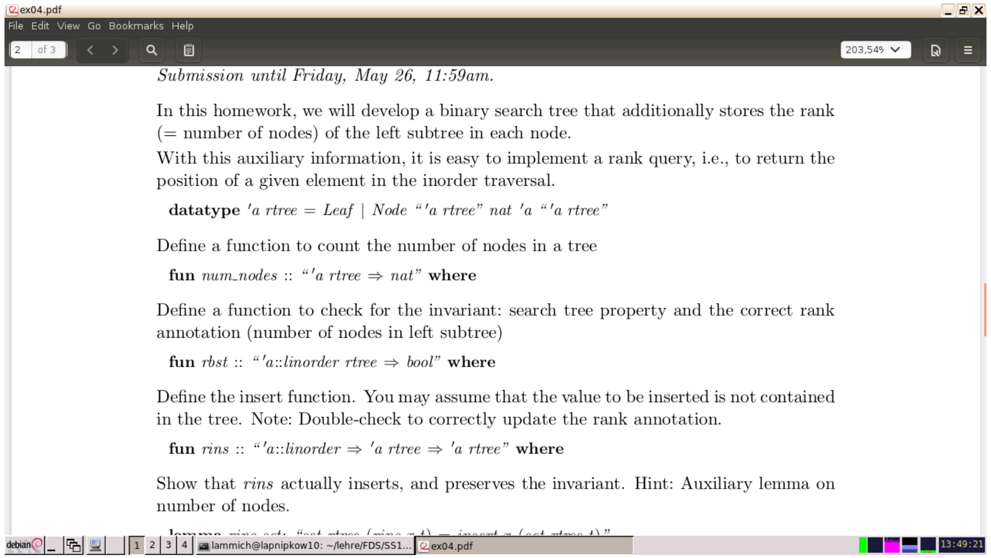

Isabelle2017 - tut04.thy
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
oops
consts
enum :: "nat ⇒ unit tree set"
Found termination order: "size <*mlex*> {}"

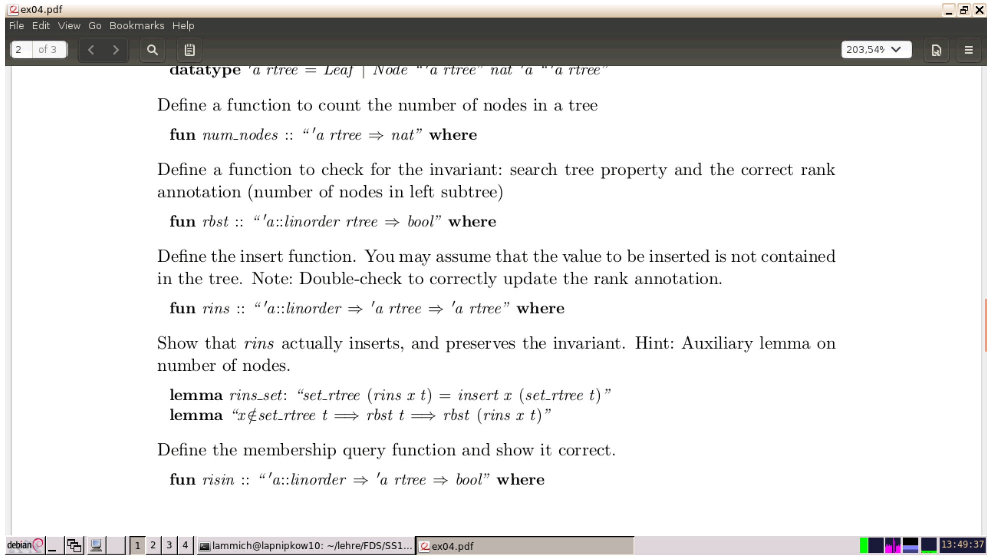

Isabelle2017 - tut04.thy (modified)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
proof (prove)
goal (1 subgoal):
1. t ∈ enum n ⟹ height t ≤ n


Isabelle2017 - tut04.thy (modified)
fun enum :: "nat ⇒ unit tree set" where
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
apply
proof (prove)
goal (1 subgoal):
1. t ∈ enum n ⟹ height t ≤ n


Isabelle2017 - tut04.thy
"enum 0 = {Leaf}"
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
apply (induction n arbitrary: t)
apply auto
proof (prove)
goal (1 subgoal):
1. ⋀n t. ⟦⋀t. t ∈ enum n ⟹ height t ≤ n; t ∈ enum n⟧ ⟹ height t ≤ Suc n


Isabelle2017 - tut04.thy
| "enum (Suc n) = (enum n ∪ {Node l () r | l r. l∈enum n ∧ r∈enum n })"
lemma enum_sound: "t ∈ enum n ⟹ height t ≤ n"
apply (induction n arbitrary: t)
apply auto
by fastforce
theorem enum_sound: ?t ∈ enum ?n ⟹ height ?t ≤ ?n


ex04.pdf
Submission until Friday, May 26, 11:59am.
In this homework, we will develop a binary search tree that additionally stores the rank (= number of nodes) of the left subtree in each node.
With this auxiliary information, it is easy to implement a rank query, i.e., to return the position of a given element in the inorder traversal.
datatype 'a rtree = Leaf | Node "'a rtree" nat 'a "'a rtree"
Define a function to count the number of nodes in a tree
fun num_nodes :: "'a rtree ⇒ nat" where
Define a function to check for the invariant: search tree property and the correct rank annotation (number of nodes in left subtree)
fun rbst :: "'a::linorder rtree ⇒ bool" where
Define the insert function. You may assume that the value to be inserted is not contained in the tree. Note: Double-check to correctly update the rank annotation.
fun rins :: "'a::linorder ⇒ 'a rtree ⇒ 'a rtree" where
Show that rins actually inserts, and preserves the invariant. Hint: Auxiliary lemma on number of nodes.


ex04.pdf
Define a function to count the number of nodes in a tree
fun num_nodes :: "'a rtree ⇒ nat" where
Define a function to check for the invariant: search tree property and the correct rank annotation (number of nodes in left subtree)
fun rbst :: "'a::linorder rtree ⇒ bool" where
Define the insert function. You may assume that the value to be inserted is not contained in the tree. Note: Double-check to correctly update the rank annotation.
fun rins :: "'a::linorder ⇒ 'a rtree ⇒ 'a rtree" where
Show that rins actually inserts, and preserves the invariant. Hint: Auxiliary lemma on number of nodes.
lemma rins_set: "set_rtree (rins x t) = insert x (set_rtree t)"
lemma "x∉set_rtree t ⟹ rbst t ⟹ rbst (rins x t)"
Define the membership query function and show it correct.
fun risin :: "'a::linorder ⇒ 'a rtree ⇒ bool" where

fun *rins* :: "'a::linorder ⇒ 'a rtree ⇒ 'a rtree" **where**

Show that *rins* actually inserts, and preserves the invariant. Hint: Auxiliary lemma on number of nodes.

lemma *rins_set*: "*set_rtree* (*rins x t*) = *insert x* (*set_rtree t*)"
lemma "$x \notin$ *set_rtree t* ⟹ *rbst t* ⟹ *rbst* (*rins x t*)"

Define the membership query function and show it correct.

fun *risin* :: "'a::linorder ⇒ 'a rtree ⇒ bool" **where**

definition "*at_index i l x* ≡ *i<length l* ∧ *l!i=x*"

Show your rank function correct. Hint: Auxiliary lemma relating *num_nodes* and *inorder*.

lemma "*rbst t* ⟹ *x∈set_rtree t* ⟹ *at_index* (*rank x t*) (*inorder t*) *x*"

Define a select function, that returns the ith element of the inorder traversal, and prove it correct.

Only recurse over the tree once, following a single path. In particular, *inorder t* ! *i* is not the desired solution, as it would enumerate all nodes of the tree in a list first, and not exploit the rank annotations at all.

fun *select* :: "nat ⇒ 'a::linorder rtree ⇒ 'a" **where**
lemma *select_correct*: "*rbst t* ⟹ *i<length* (*inorder t*) ⟹ *select i t* = *inorder t* ! *i*"

For 3 **bonus** points, remove the assumption that the inserted element is not yet contained in the tree. Only recurse over the tree once, i.e., do not simply use *if risin x t then t else rins x t*!

Hint: Add an additional return value to the insert function that indicates whether the element was in the tree or not, in order to correctly update the rank annotation. At the end, you must provide a function *rins'* that satisfies the following specification (and only recurses over the tree once, following a single path):

not exploit the rank annotations at all.

fun *select* :: "nat ⇒ 'a::linorder rtree ⇒ 'a" **where**
lemma *select_correct*: "*rbst t* ⟹ *i<length* (*inorder t*) ⟹ *select i t* = *inorder t* ! *i*"

For 3 **bonus** points, remove the assumption that the inserted element is not yet contained in the tree. Only recurse over the tree once, i.e., do not simply use *if risin x t then t else rins x t*!

Hint: Add an additional return value to the insert function that indicates whether the element was in the tree or not, in order to correctly update the rank annotation. At the end, you must provide a function *rins'* that satisfies the following specification (and only recurses over the tree once, following a single path):

definition *rins'* :: "'a::linorder ⇒ 'a rtree ⇒ 'a rtree"
lemma *rins'_set*: "*rbst t* ⟹ *set_rtree* (*rins' x t*) = {*x*} ∪ *set_rtree t*"
lemma *rins'_bst*: "*rbst t* ⟹ *rbst* (*rins' x t*)"

definition "*at_index i l x* ≡ *i<length l* ∧ *l!i=x*"

Show your rank function correct. Hint: Auxiliary lemma relating *num_nodes* and *inorder*.

lemma "*rbst t* ⟹ *x∈set_rtree t* ⟹ *at_index* (*rank x t*) (*inorder t*) *x*"

Define a select function, that returns the ith element of the inorder traversal, and prove it correct.

Only recurse over the tree once, following a single path. In particular, *inorder t* ! *i* is not the desired solution, as it would enumerate all nodes of the tree in a list first, and not exploit the rank annotations at all.

fun *select* :: "nat ⇒ 'a::linorder rtree ⇒ 'a" **where**
lemma *select_correct*: "*rbst t* ⟹ *i<length* (*inorder t*) ⟹ *select i t* = *inorder t* ! *i*"

For 3 **bonus** points, remove the assumption that the inserted element is not yet contained in the tree. Only recurse over the tree once, i.e., do not simply use *if risin x t then t else rins x t*!

Hint: Add an additional return value to the insert function that indicates whether the element was in the tree or not, in order to correctly update the rank annotation. At the end, you must provide a function *rins'* that satisfies the following specification (and only recurses over the tree once, following a single path):

Show that pretty-printing is actually unique, i.e., no two different trees are pretty-printed the same way. Hint: Auxiliary lemma. Simultaneous induction over both trees.

lemma *pretty_unique*: "$pretty\ t = pretty\ t' \Longrightarrow t{=}t'$"

Exercise 4.3 Enumeration of Trees

Write a function that generates the set of all trees up to a given height. Show that only trees up to the specified height are contained.

(The other direction, i.e., that all trees are contained, requires an advanced case split, which has not yet been introduced in the lecture, so it is omitted here)

fun *enum* :: "$nat \Rightarrow unit\ tree\ set$" **where**
lemma *enum_sound*: "$t \in enum\ n \Longrightarrow height\ t \le n$"

Homework 4 Rank Annotated Trees

Submission until Friday, May 26, 11:59am.