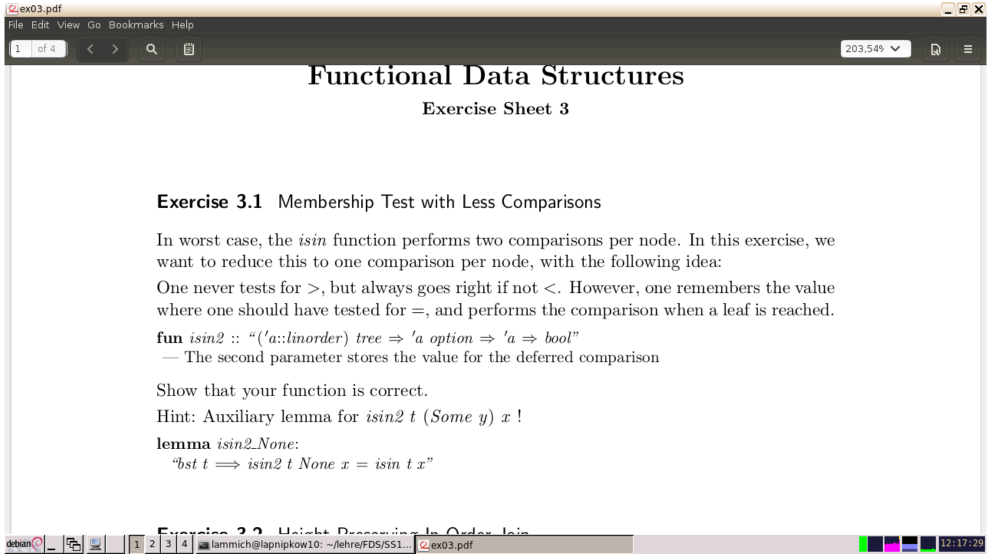
Script generated by TTT

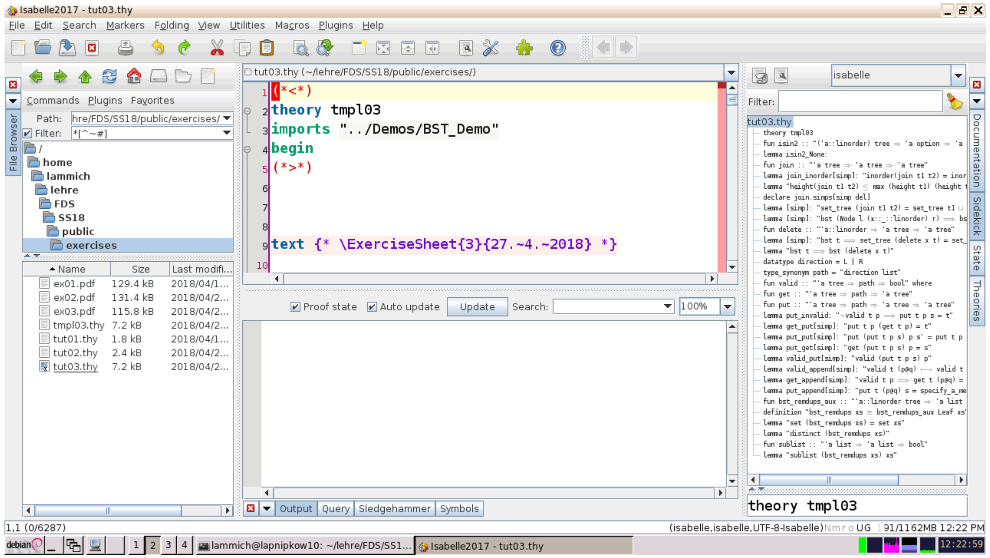
Title: Lammich: FDS Tutorial (27.04.2018)

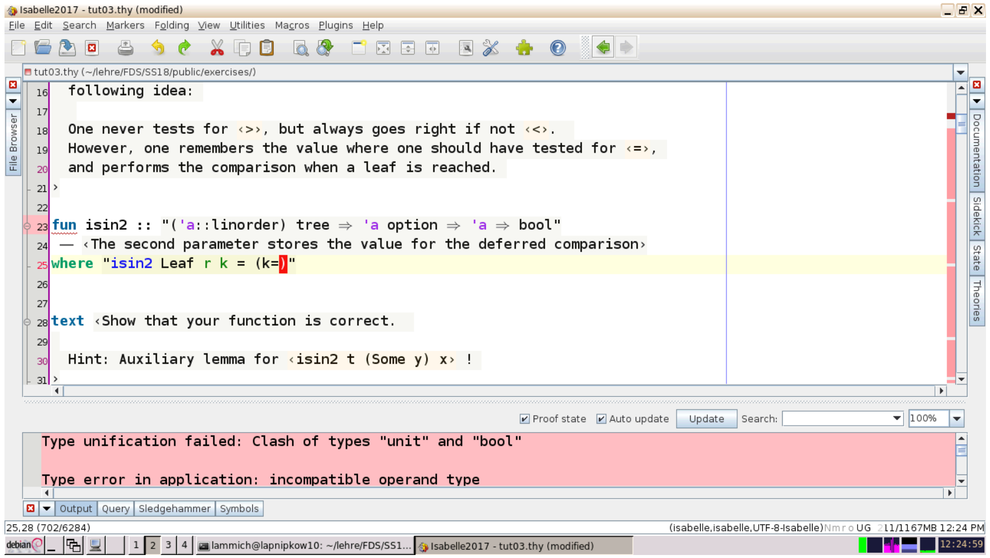
Date: Fri Apr 27 12:17:29 CEST 2018

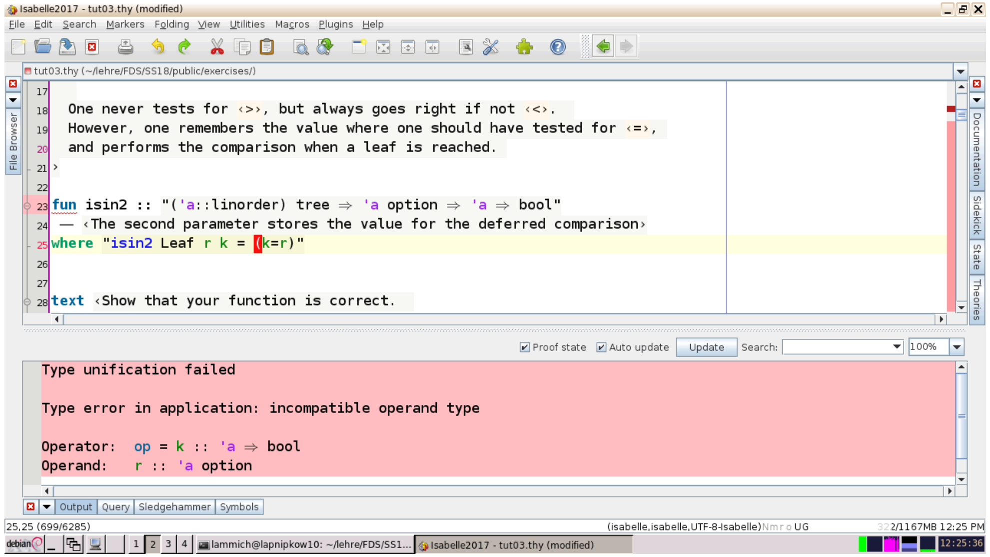
Duration: 91:08 min

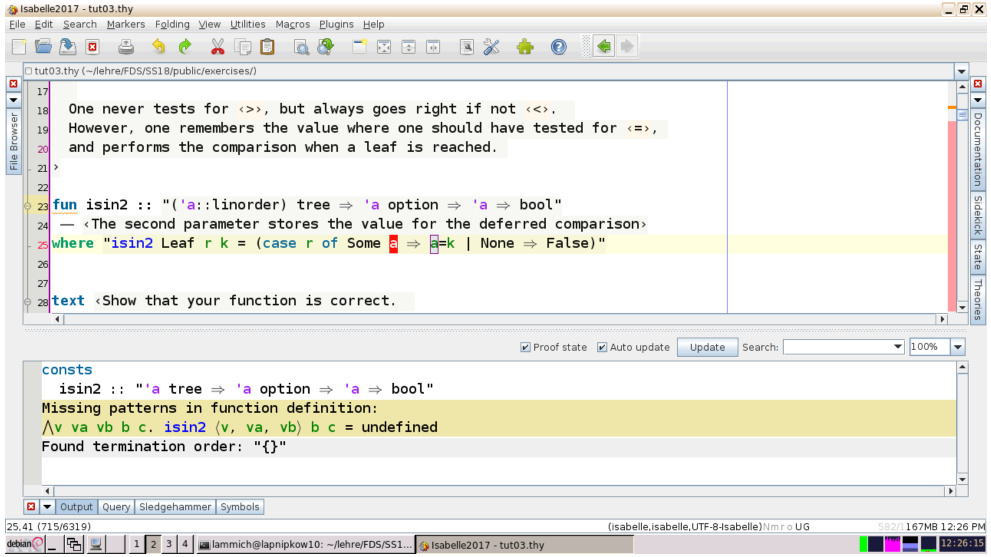
Pages: 74

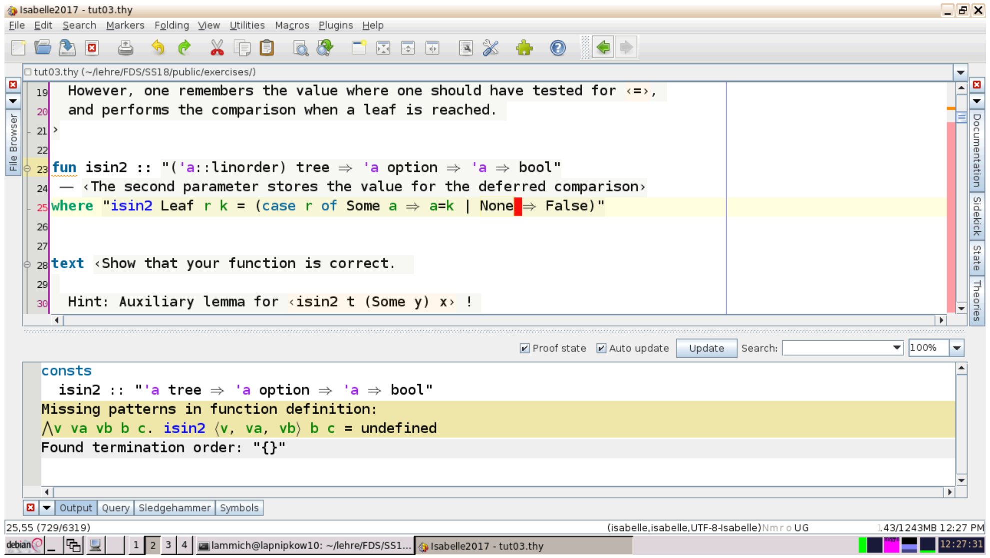


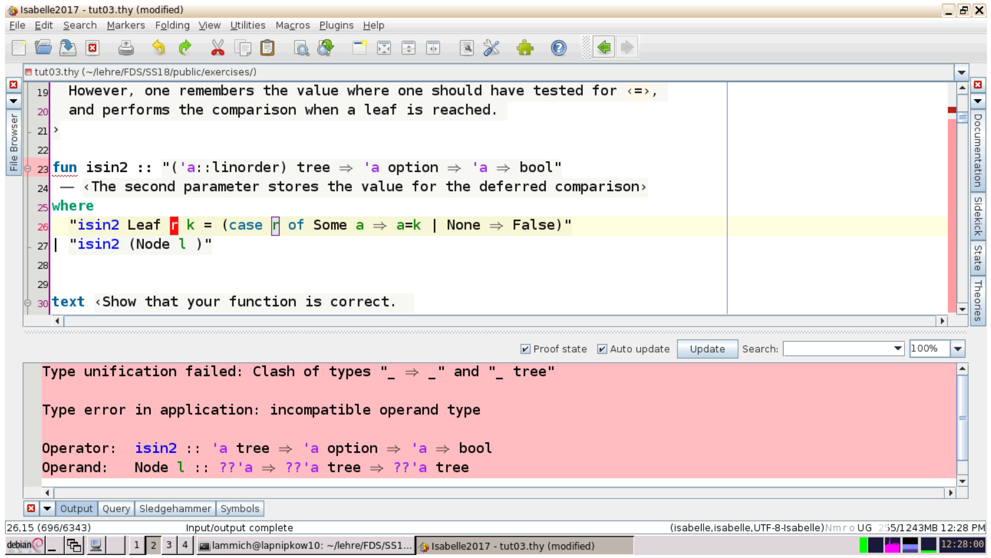


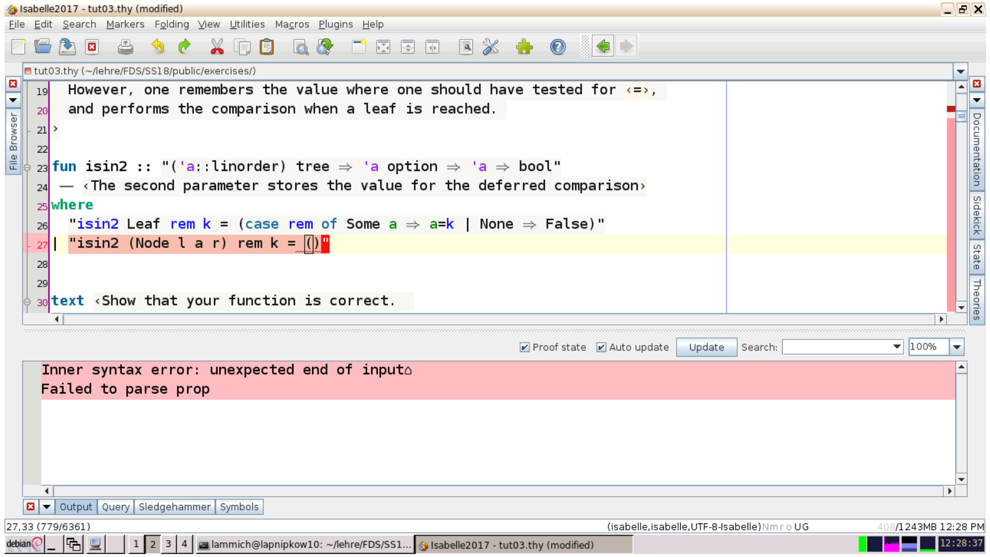


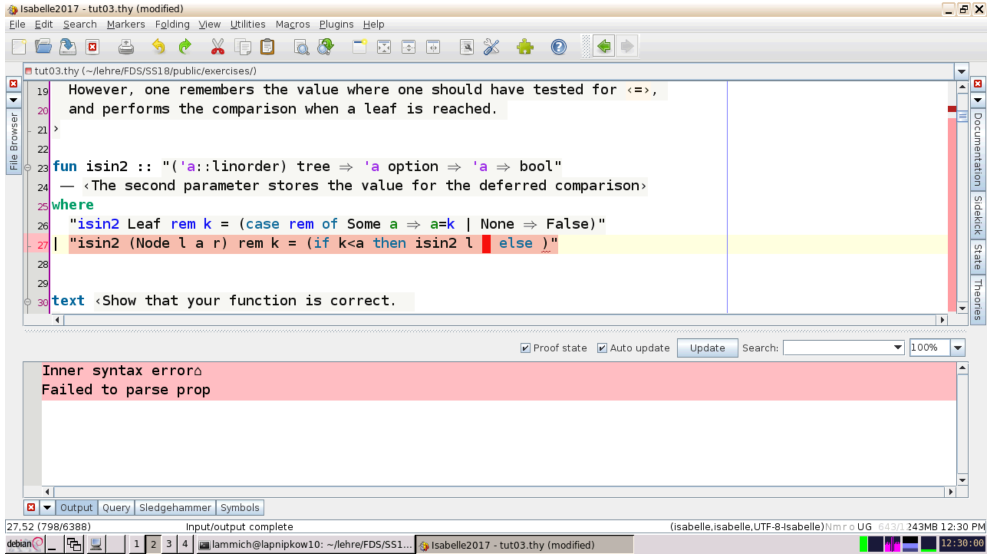

Isabelle2017 - tut03.thy (modified)
One never tests for ‹>›, but always goes right if not ‹<›.
However, one remembers the value where one should have tested for ‹=›,
and performs the comparison when a leaf is reached.
›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where "isin2 Leaf r k = (k=r)"
text ‹Show that your function is correct.
Type unification failed
Type error in application: incompatible operand type
Operator: op = k :: 'a ⇒ bool
Operand: r :: 'a option

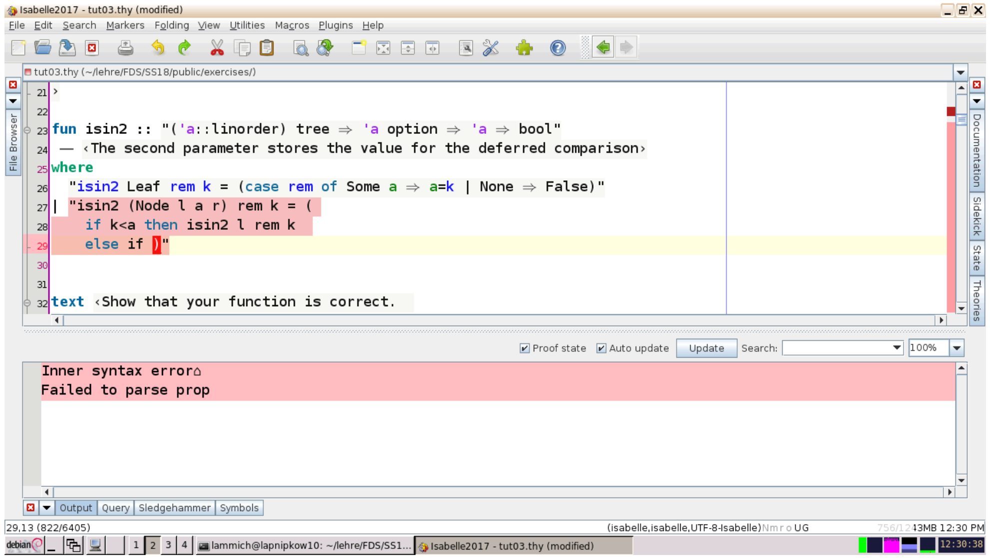

Isabelle2017 - tut03.thy
One never tests for ‹>›, but always goes right if not ‹<›.
However, one remembers the value where one should have tested for ‹=›,
and performs the comparison when a leaf is reached.
›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where "isin2 Leaf r k = (case r of Some a ⇒ a=k | None ⇒ False)"
text ‹Show that your function is correct.
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Missing patterns in function definition:
⋀v va vb b c. isin2 ⟨v, va, vb⟩ b c = undefined
Found termination order: "{}"

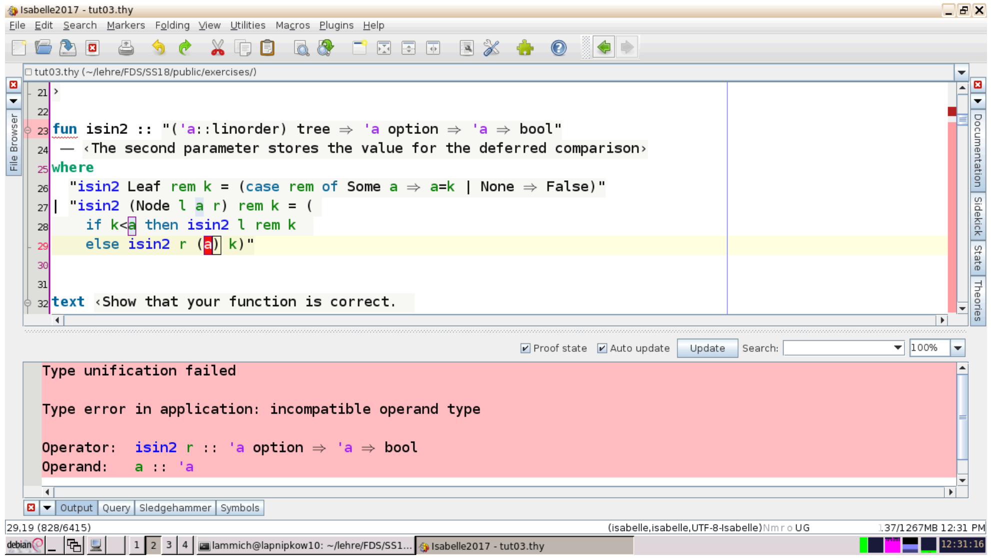

Isabelle2017 - tut03.thy
However, one remembers the value where one should have tested for ‹=›,
and performs the comparison when a leaf is reached.
›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where "isin2 Leaf r k = (case r of Some a ⇒ a=k | None ⇒ False)"
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Missing patterns in function definition:
⋀v va vb b c. isin2 ⟨v, va, vb⟩ b c = undefined
Found termination order: "{}"

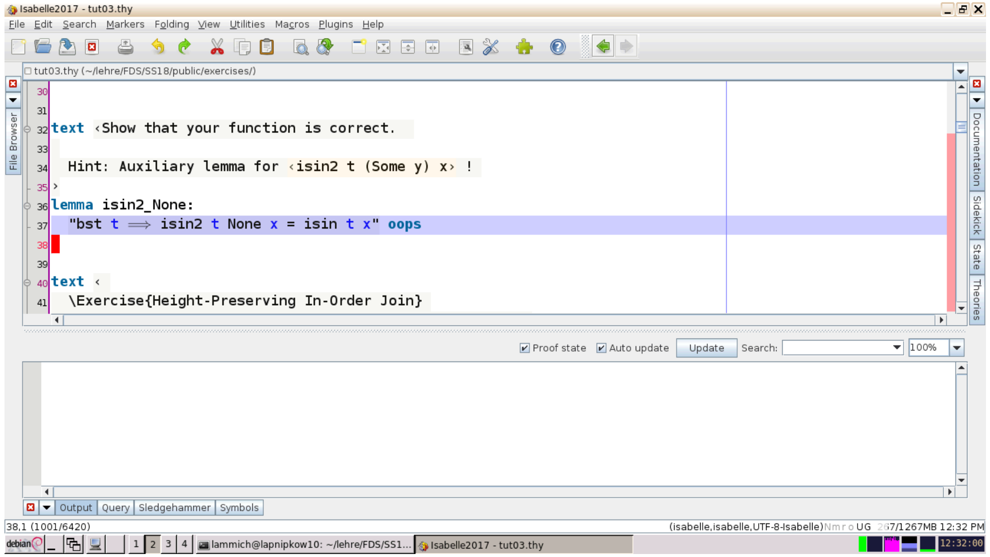

Isabelle2017 - tut03.thy (modified)
However, one remembers the value where one should have tested for ‹=›,
and performs the comparison when a leaf is reached.
›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf r k = (case r of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l)"
text ‹Show that your function is correct.
Type unification failed: Clash of types "_ ⇒ _" and "_ tree"
Type error in application: incompatible operand type
Operator: isin2 :: 'a tree ⇒ 'a option ⇒ 'a ⇒ bool
Operand: Node l :: ??'a ⇒ ??'a tree ⇒ ??'a tree

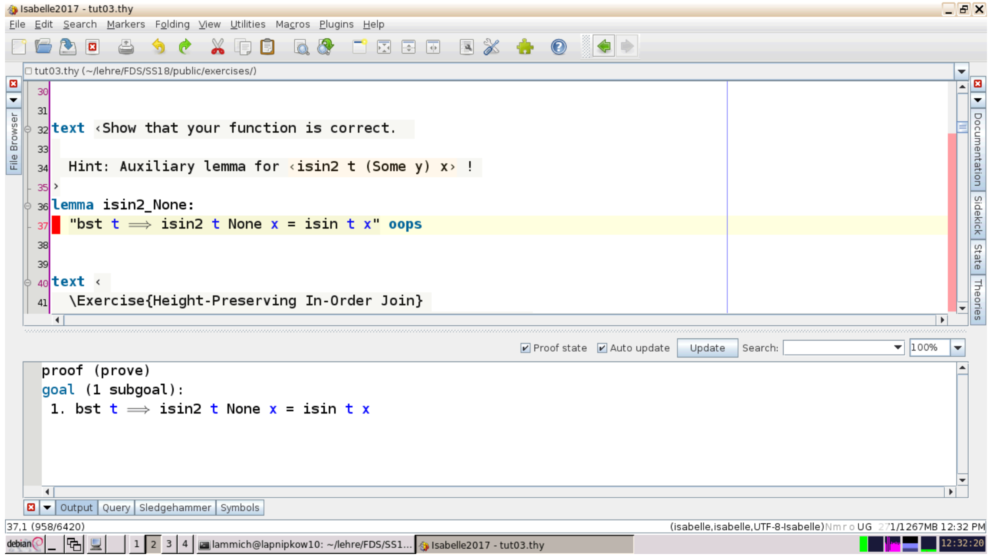

Isabelle2017 - tut03.thy (modified)
However, one remembers the value where one should have tested for ‹=›,
and performs the comparison when a leaf is reached.
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = ()"
text ‹Show that your function is correct.
Inner syntax error: unexpected end of input
Failed to parse prop

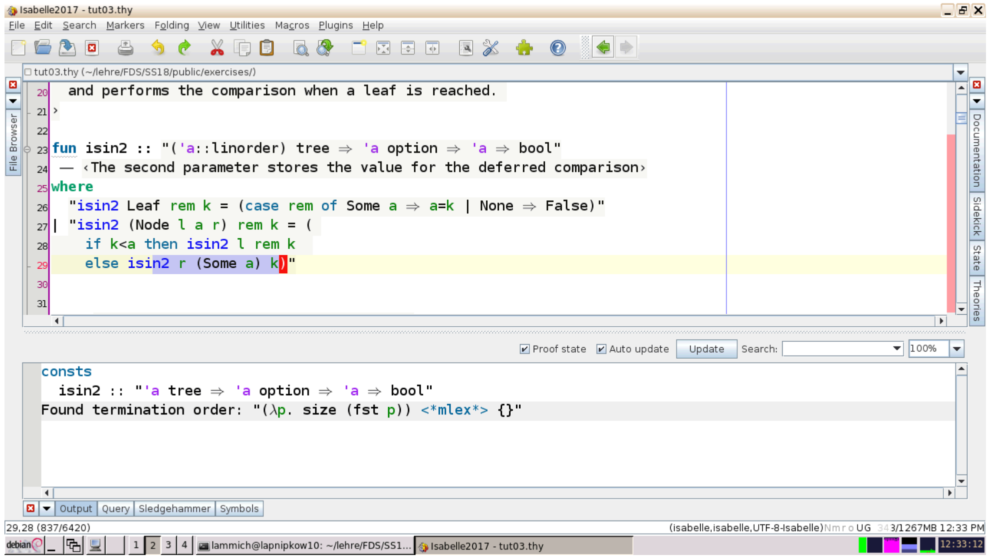

Isabelle2017 - tut03.thy (modified)
However, one remembers the value where one should have tested for ‹=›,
and performs the comparison when a leaf is reached.
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = (if k<a then isin2 l else)"
text ‹Show that your function is correct.
Inner syntax error
Failed to parse prop

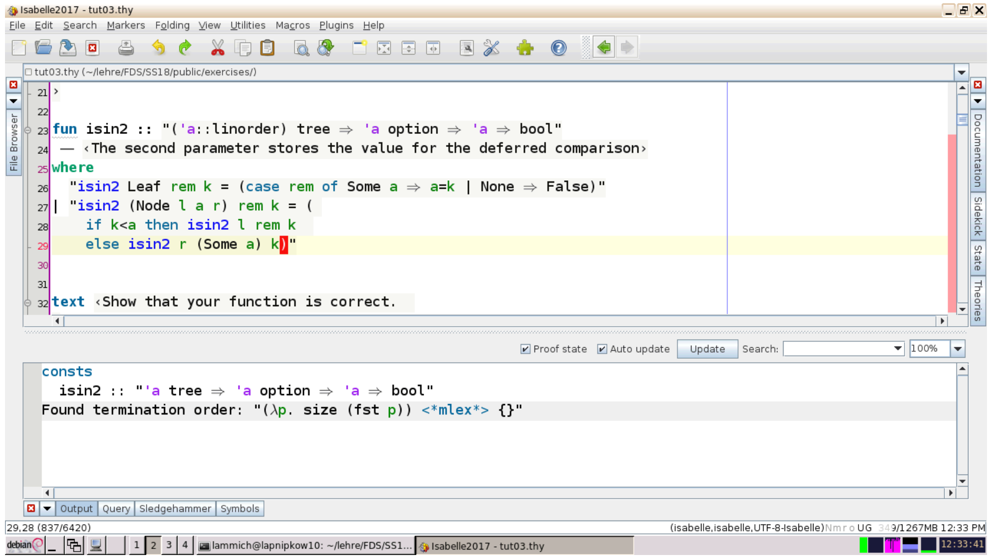

Isabelle2017 - tut03.thy (modified)
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = (
if k<a then isin2 l rem k
else if)"
text ‹Show that your function is correct.
Inner syntax error
Failed to parse prop

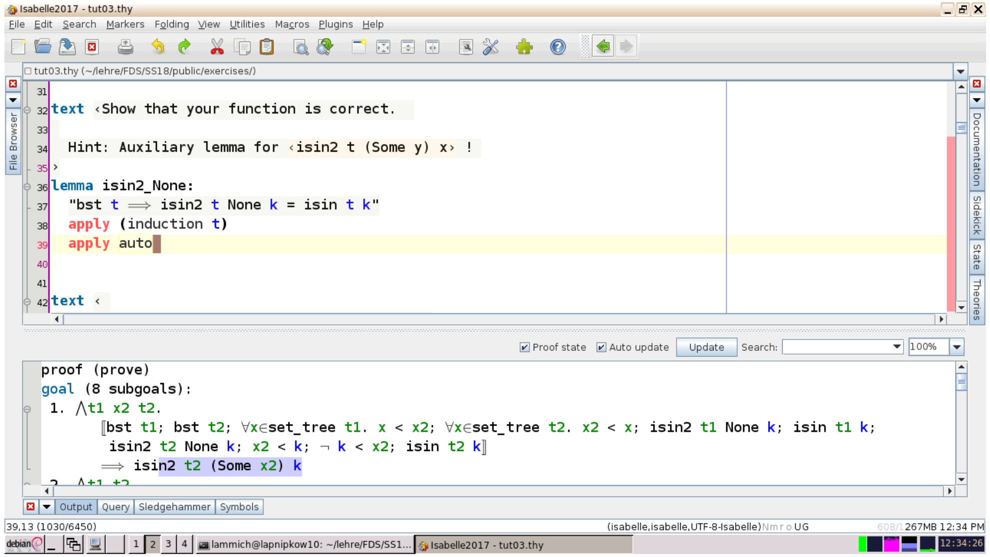

Isabelle2017 - tut03.thy
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = (
if k<a then isin2 l rem k
else isin2 r (a) k)"
text ‹Show that your function is correct.
Type unification failed
Type error in application: incompatible operand type
Operator: isin2 r :: 'a option ⇒ 'a ⇒ bool
Operand: a :: 'a

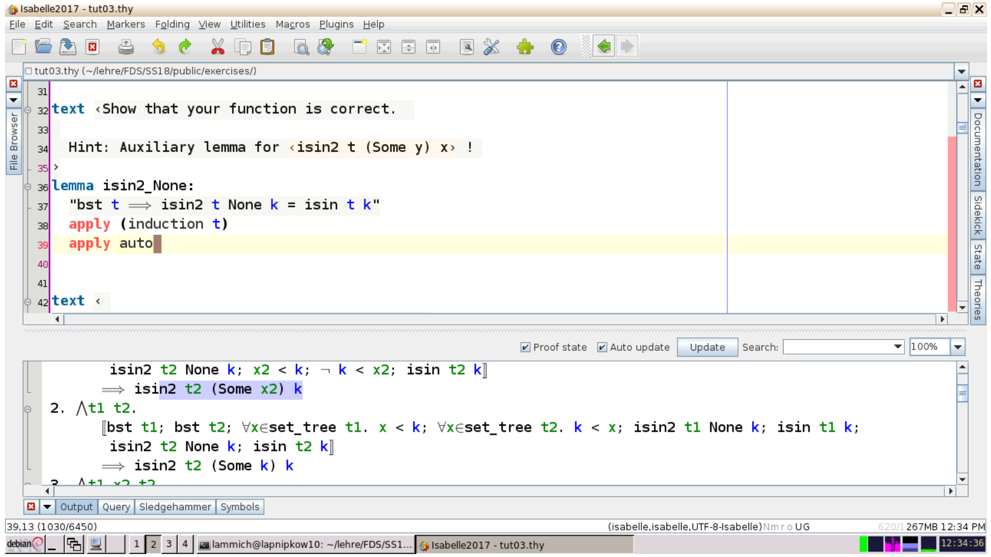

Isabelle2017 - tut03.thy
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma isin2_None:
"bst t ⟹ isin2 t None x = isin t x" oops
text ‹
\Exercise{Height-Preserving In-Order Join}

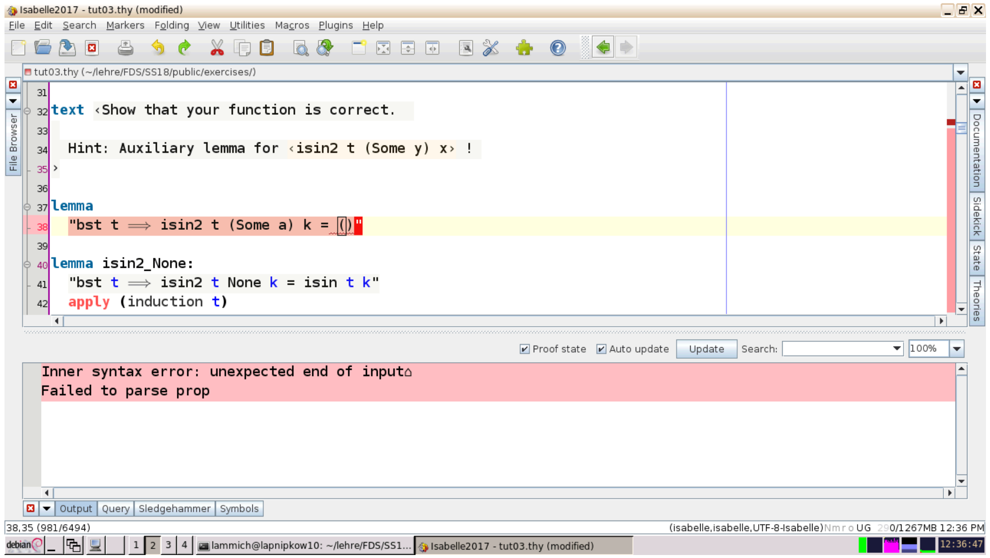

Isabelle2017 - tut03.thy
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma isin2_None:
"bst t ⟹ isin2 t None x = isin t x" oops
text ‹
\Exercise{Height-Preserving In-Order Join}
proof (prove)
goal (1 subgoal):
1. bst t ⟹ isin2 t None x = isin t x

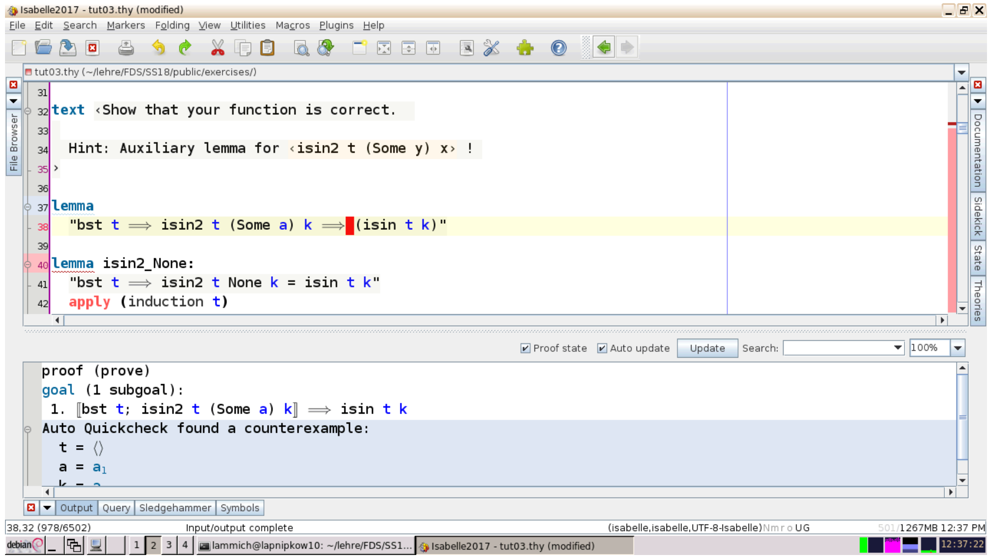

Isabelle2017 - tut03.thy
and performs the comparison when a leaf is reached.
›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = (
if k<a then isin2 l rem k
else isin2 r (Some a) k)"
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

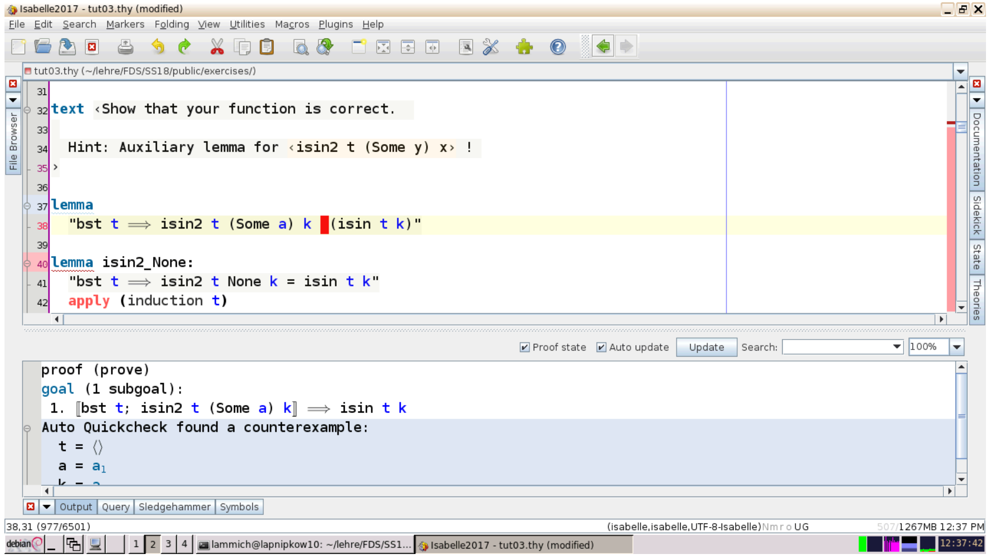

Isabelle2017 - tut03.thy
›
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— ‹The second parameter stores the value for the deferred comparison›
where
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = (
if k<a then isin2 l rem k
else isin2 r (Some a) k)"
text ‹Show that your function is correct.
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

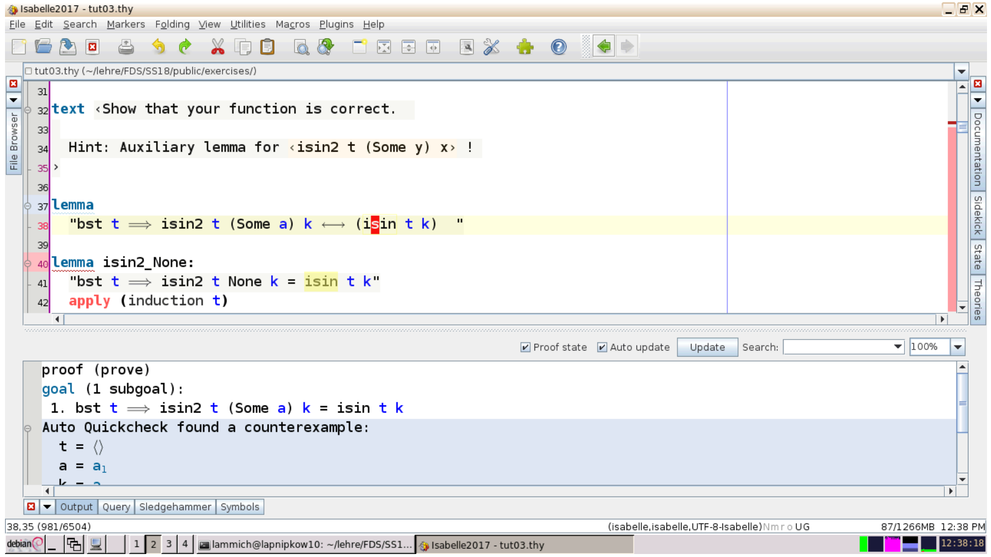

Isabelle2017 - tut03.thy
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
apply auto
text ‹
proof (prove)
goal (8 subgoals):
1. ⋀t1 x2 t2.
⟦bst t1; bst t2; ∀x∈set_tree t1. x < x2; ∀x∈set_tree t2. x2 < x; isin2 t1 None k; isin t1 k;
isin2 t2 None k; x2 < k; ¬ k < x2; isin t2 k⟧
⟹ isin2 t2 (Some x2) k

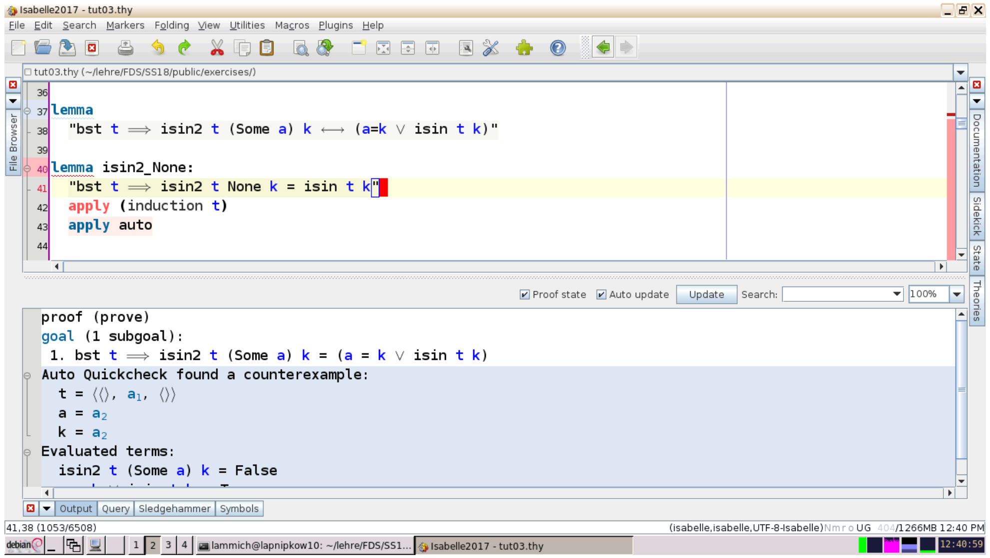

Isabelle2017 - tut03.thy
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
apply auto
text ‹
isin2 t2 None k; x2 < k; ¬ k < x2; isin2 t2 k⟧
⟹ isin2 t2 (Some x2) k
2. ⋀t1 t2.
⟦bst t1; bst t2; ∀x∈set_tree t1. k < x; ∀x∈set_tree t2. k < x; isin2 t1 None k; isin t1 k;
isin2 t2 None k; isin t2 k⟧
⟹ isin2 t2 (Some k) k

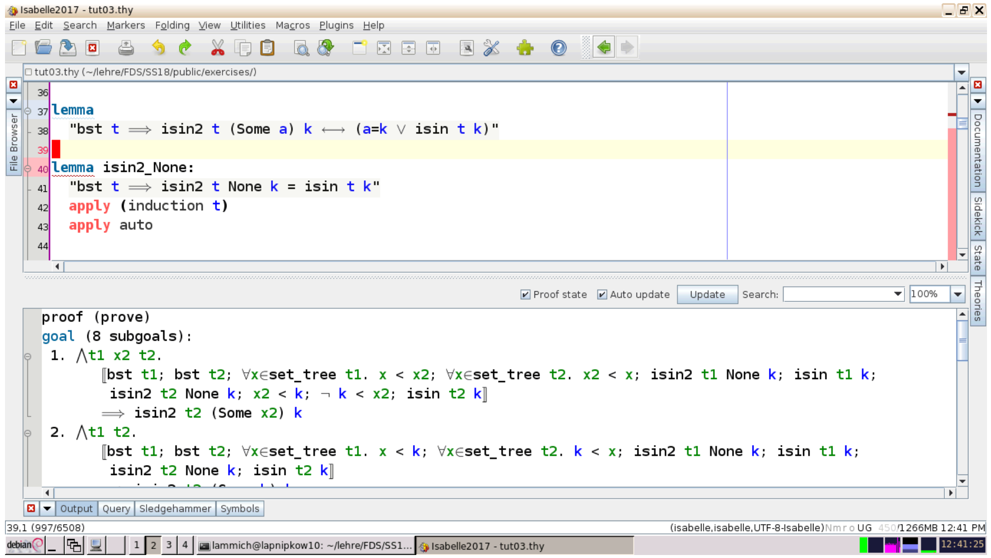

Isabelle2017 - tut03.thy (modified)
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma
"bst t ⟹ isin2 t (Some a) k = ()"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
Inner syntax error: unexpected end of input
Failed to parse prop

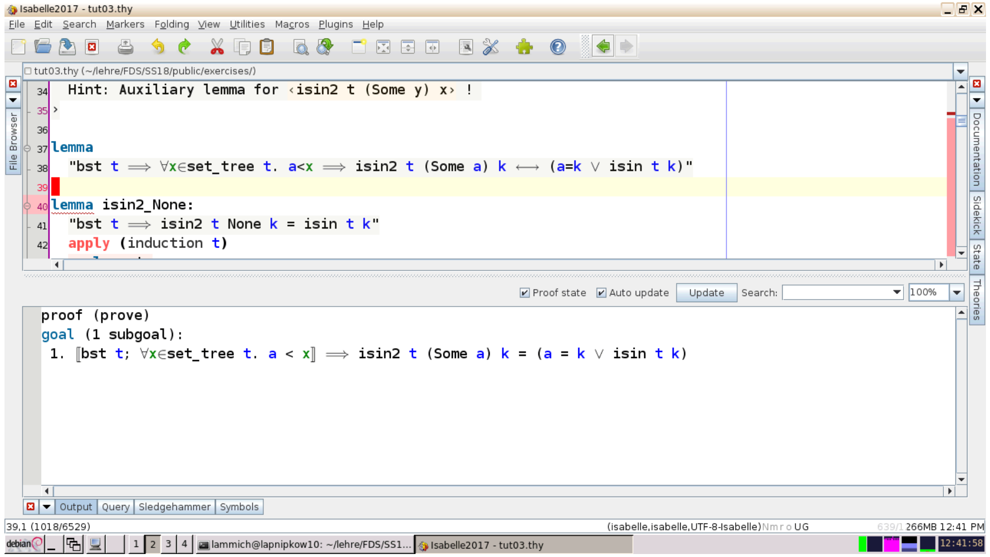
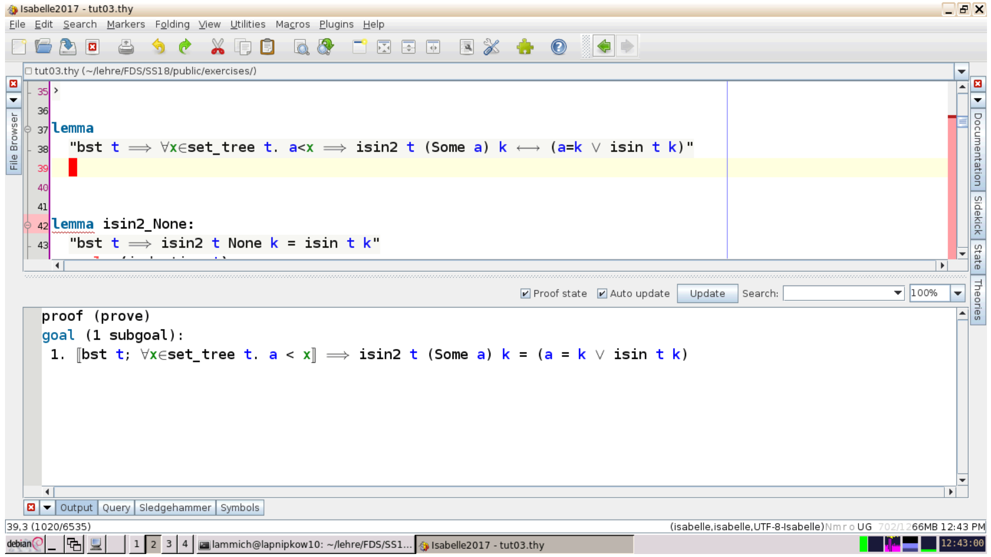
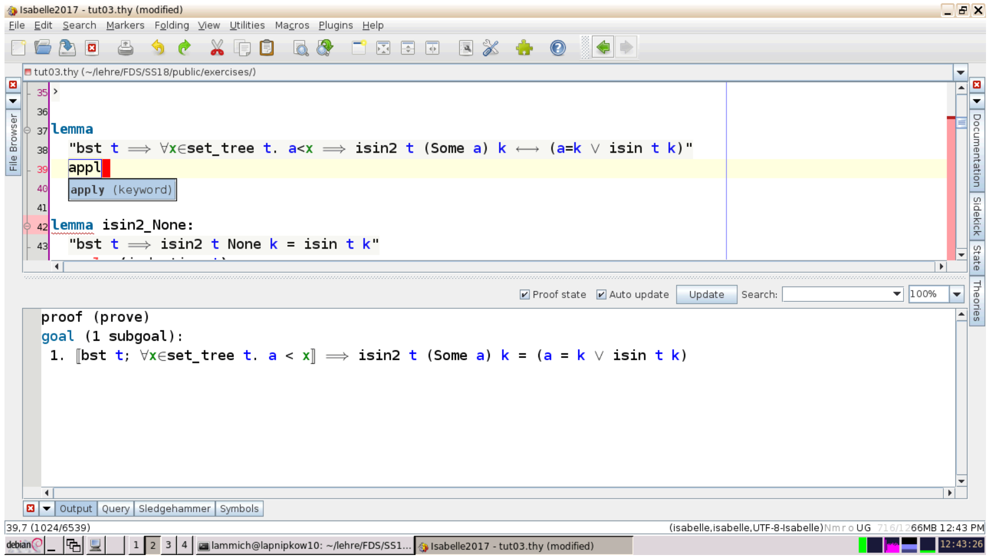
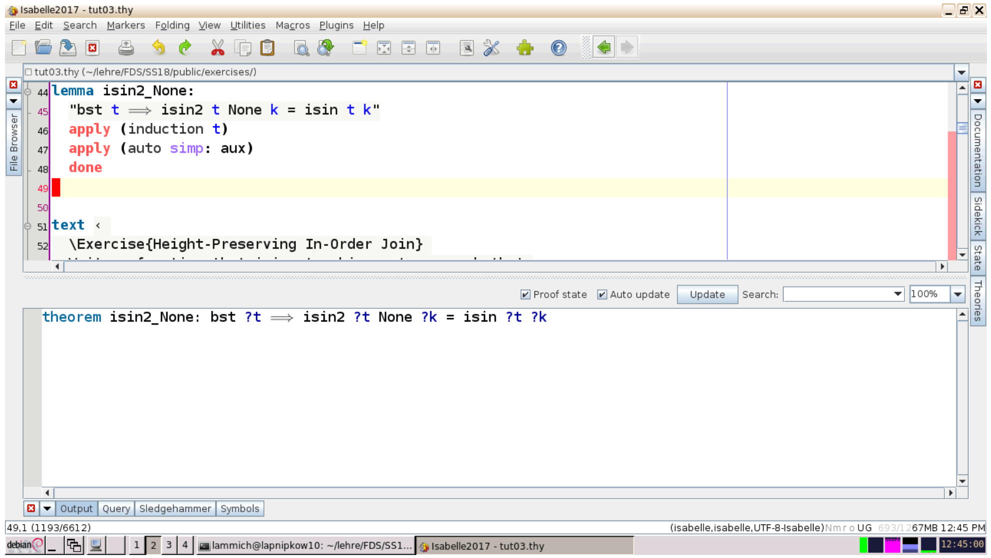

Isabelle2017 - tut03.thy (modified)
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma
"bst t ⟹ isin2 t (Some a) k ⟹ (isin t k)"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
proof (prove)
goal (1 subgoal):
1. ⟦bst t; isin2 t (Some a) k⟧ ⟹ isin t k
Auto Quickcheck found a counterexample:
t = ⟨⟩
a = a1

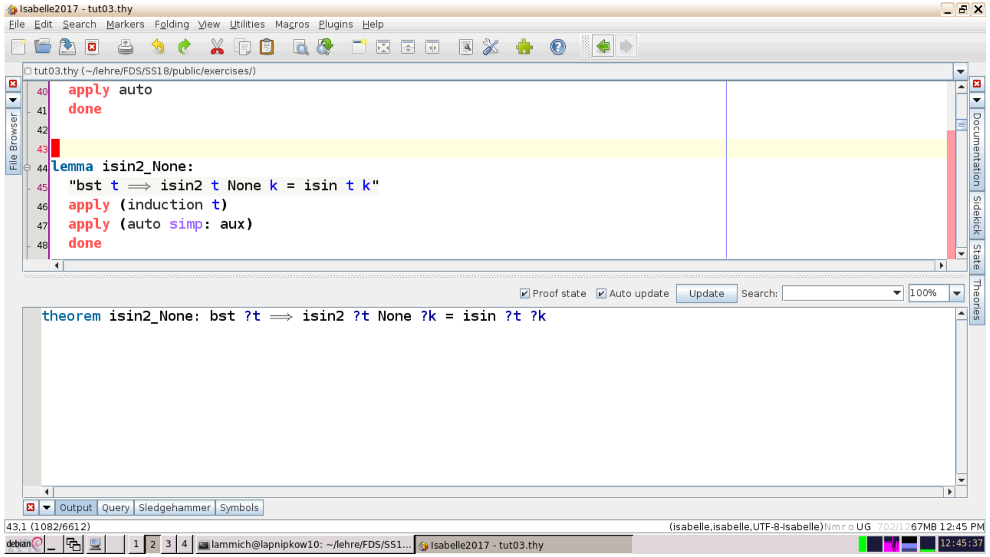

Isabelle2017 - tut03.thy (modified)
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma
"bst t ⟹ isin2 t (Some a) k (isin t k)"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
proof (prove)
goal (1 subgoal):
1. ⟦bst t; isin2 t (Some a) k⟧ ⟹ isin t k
Auto Quickcheck found a counterexample:
t = ⟨⟩
a = a₁

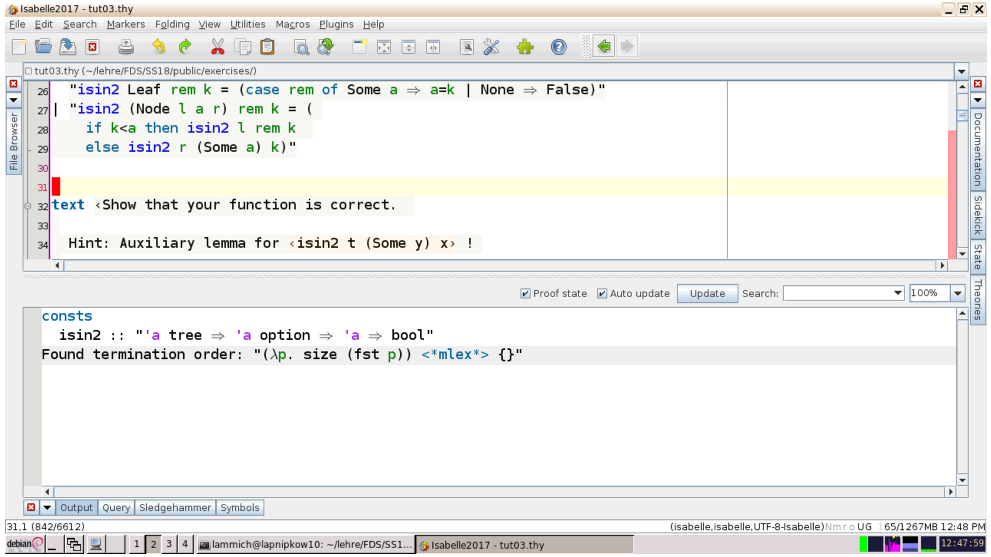

Isabelle2017 - tut03.thy
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma
"bst t ⟹ isin2 t (Some a) k ⟷ (isin t k) "
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
proof (prove)
goal (1 subgoal):
1. bst t ⟹ isin2 t (Some a) k = isin t k
Auto Quickcheck found a counterexample:
t = ⟨⟩
a = a₁

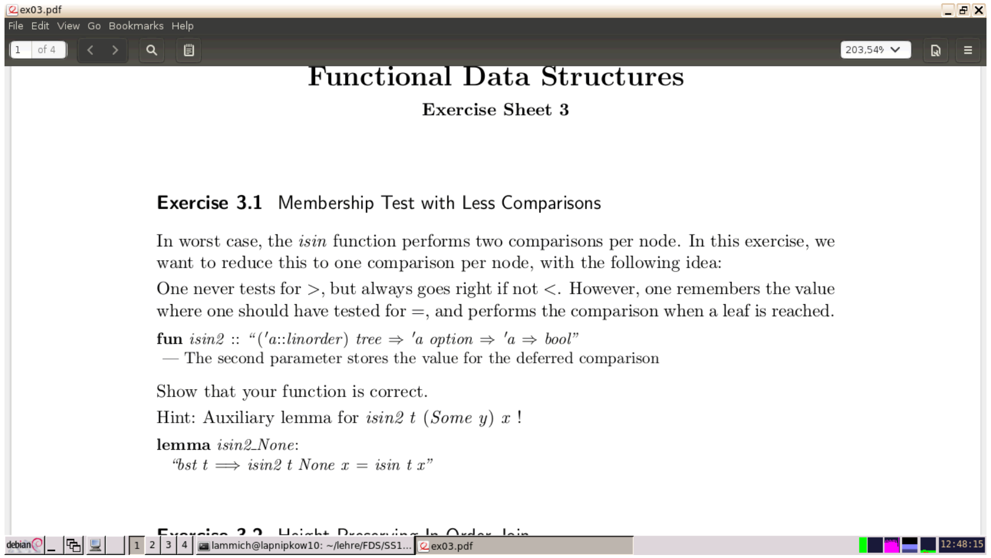

Isabelle2017 - tut03.thy
lemma
"bst t ⟹ isin2 t (Some a) k ⟷ (a=k ∨ isin t k)"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
apply auto
proof (prove)
goal (1 subgoal):
1. bst t ⟹ isin2 t (Some a) k = (a = k ∨ isin t k)
Auto Quickcheck found a counterexample:
t = ⟨⟨⟩, a₁, ⟨⟩⟩
a = a₂
k = a₂
Evaluated terms:
isin2 t (Some a) k = False

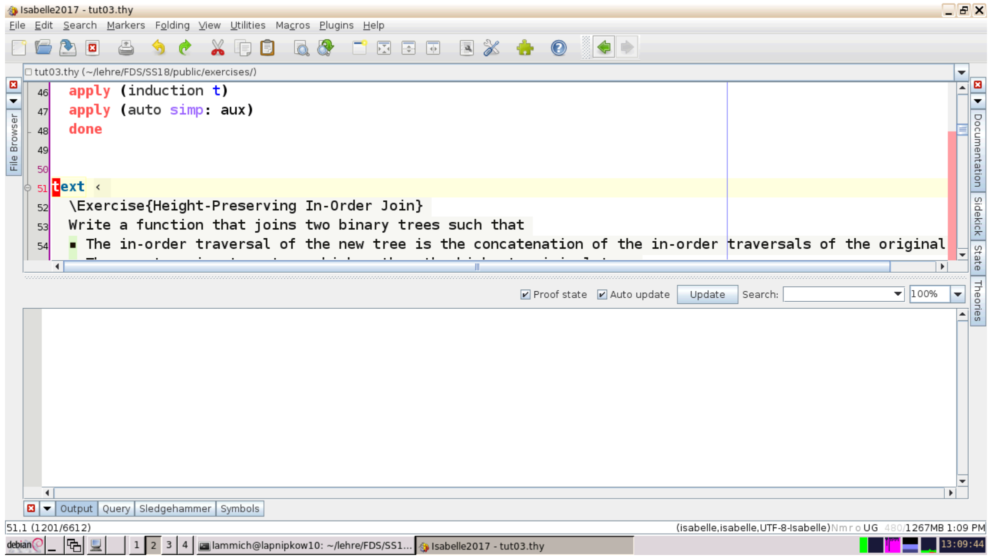

Isabelle2017 - tut03.thy
lemma
"bst t ⟹ isin2 t (Some a) k ⟷ (a=k ∨ isin t k)"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
apply auto
proof (prove)
goal (8 subgoals):
1. ⋀t1 x2 t2.
⟦bst t1; bst t2; ∀x∈set_tree t1. x < x2; ∀x∈set_tree t2. x2 < x; isin2 t1 None k; isin t1 k;
isin2 t2 None k; x2 < k; ¬ k < x2; isin t2 k⟧
⟹ isin2 t2 (Some x2) k
2. ⋀t1 t2.
⟦bst t1; bst t2; ∀x∈set_tree t1. x < k; ∀x∈set_tree t2. k < x; isin2 t1 None k; isin t1 k;
isin2 t2 None k; isin t2 k⟧

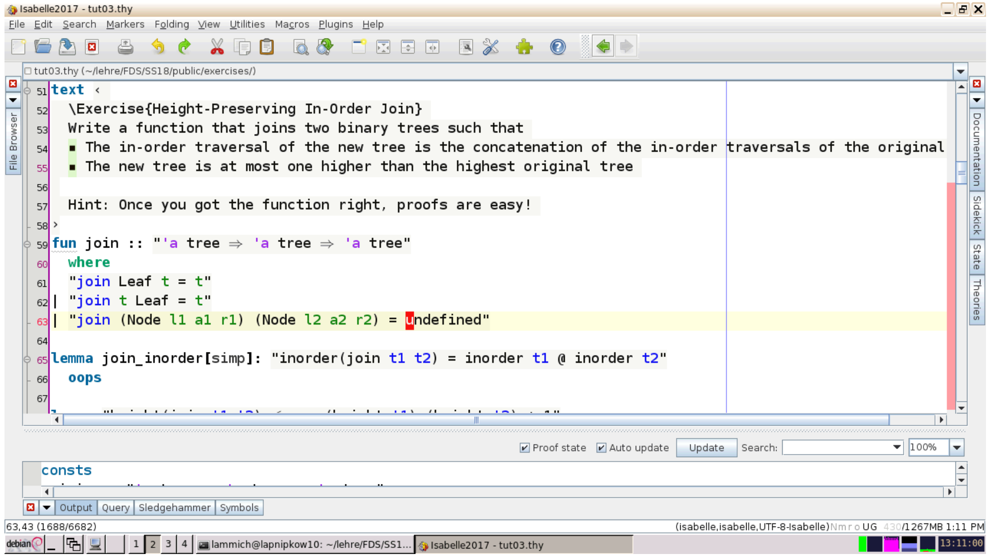

Isabelle2017 - tut03.thy
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
›
lemma
"bst t ⟹ ∀x∈set_tree t. a<x ⟹ isin2 t (Some a) k ⟷ (a=k ∨ isin t k)"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
proof (prove)
goal (1 subgoal):
1. ⟦bst t; ∀x∈set_tree t. a < x⟧ ⟹ isin2 t (Some a) k = (a = k ∨ isin t k)

Isabelle2017 - tut03.thy
›
lemma
"bst t ⟹ ∀x∈set_tree t. a<x ⟹ isin2 t (Some a) k ⟷ (a=k ∨ isin t k)"
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
proof (prove)
goal (1 subgoal):
1. ⟦bst t; ∀x∈set_tree t. a < x⟧ ⟹ isin2 t (Some a) k = (a = k ∨ isin t k)

Isabelle2017 - tut03.thy (modified)
›
lemma
"bst t ⟹ ∀x∈set_tree t. a<x ⟹ isin2 t (Some a) k ⟷ (a=k ∨ isin t k)"
appl
apply (keyword)
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
proof (prove)
goal (1 subgoal):
1. ⟦bst t; ∀x∈set_tree t. a < x⟧ ⟹ isin2 t (Some a) k = (a = k ∨ isin t k)

Isabelle2017 - tut03.thy
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
apply (auto simp: aux)
done
text ‹
\Exercise{Height-Preserving In-Order Join}
theorem isin2_None: bst ?t ⟹ isin2 ?t None ?k = isin ?t ?k

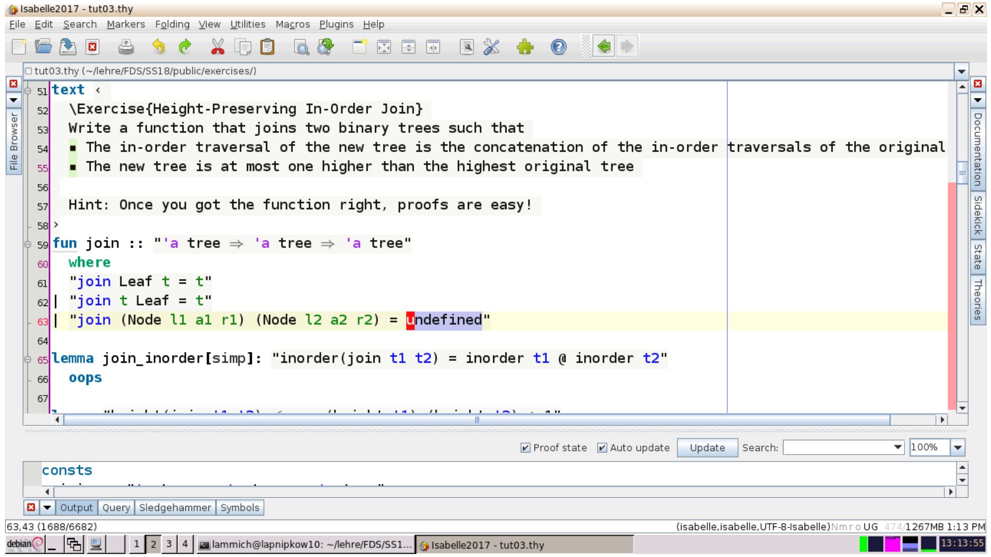

Isabelle2017 - tut03.thy
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
apply auto
done
lemma isin2_None:
"bst t ⟹ isin2 t None k = isin t k"
apply (induction t)
apply (auto simp: aux)
done
theorem isin2_None: bst ?t ⟹ isin2 ?t None ?k = isin ?t ?k
Output Query Sledgehammer Symbols

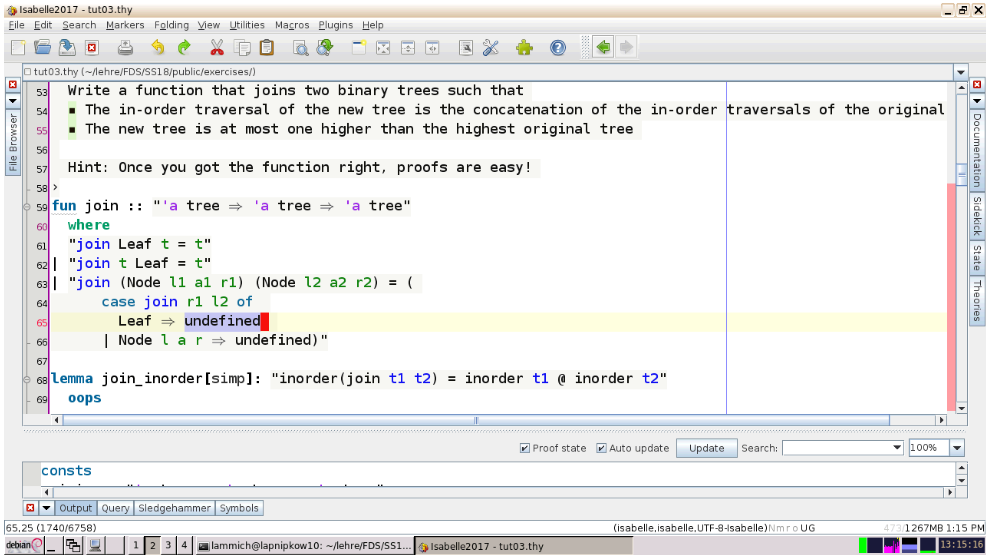

Isabelle2017 - tut03.thy
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
"isin2 Leaf rem k = (case rem of Some a ⇒ a=k | None ⇒ False)"
| "isin2 (Node l a r) rem k = (
if k<a then isin2 l rem k
else isin2 r (Some a) k)"
text ‹Show that your function is correct.
Hint: Auxiliary lemma for ‹isin2 t (Some y) x› !
consts
isin2 :: "'a tree ⇒ 'a option ⇒ 'a ⇒ bool"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"
Output Query Sledgehammer Symbols

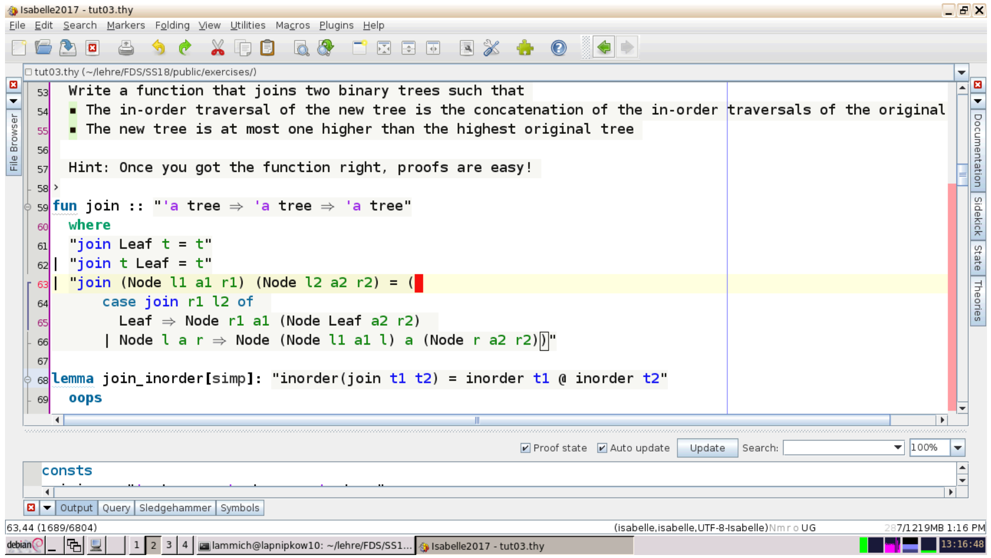

ex03.pdf
Functional Data Structures
Exercise Sheet 3
Exercise 3.1 Membership Test with Less Comparisons
In worst case, the isin function performs two comparisons per node. In this exercise, we want to reduce this to one comparison per node, with the following idea:
One never tests for >, but always goes right if not <. However, one remembers the value where one should have tested for =, and performs the comparison when a leaf is reached.
fun isin2 :: "('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
— The second parameter stores the value for the deferred comparison
Show that your function is correct.
Hint: Auxiliary lemma for isin2 t (Some y) x !
lemma isin2_None:
"bst t ⟹ isin2 t None x = isin t x"

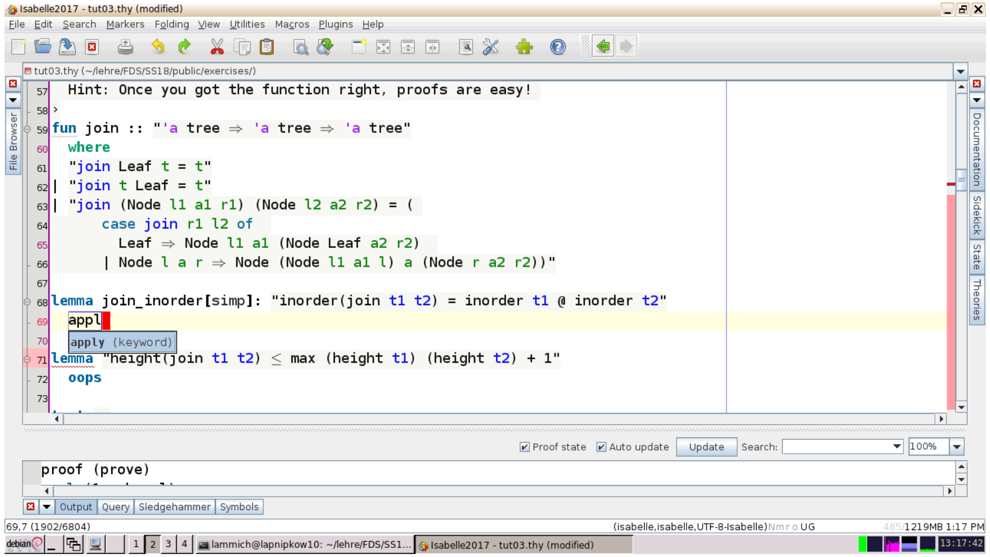
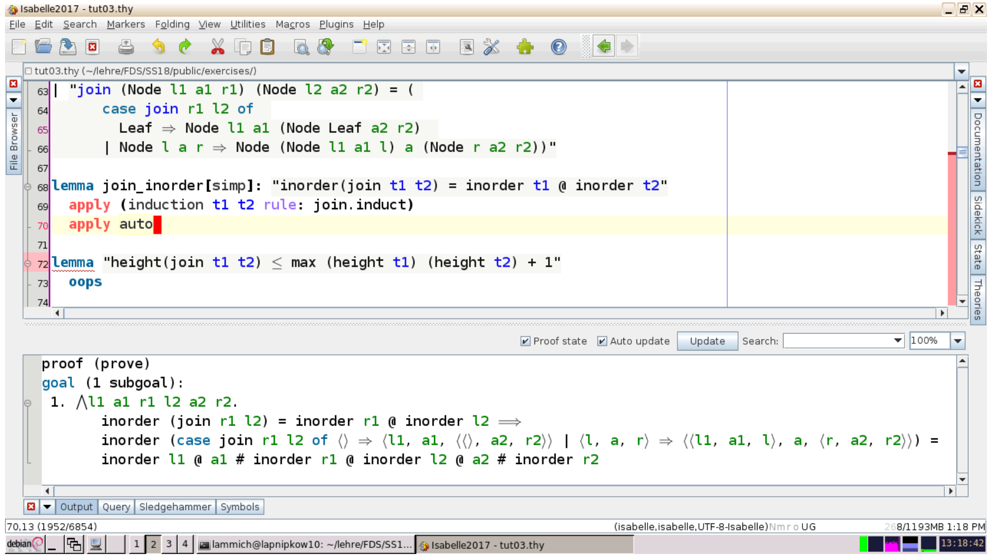
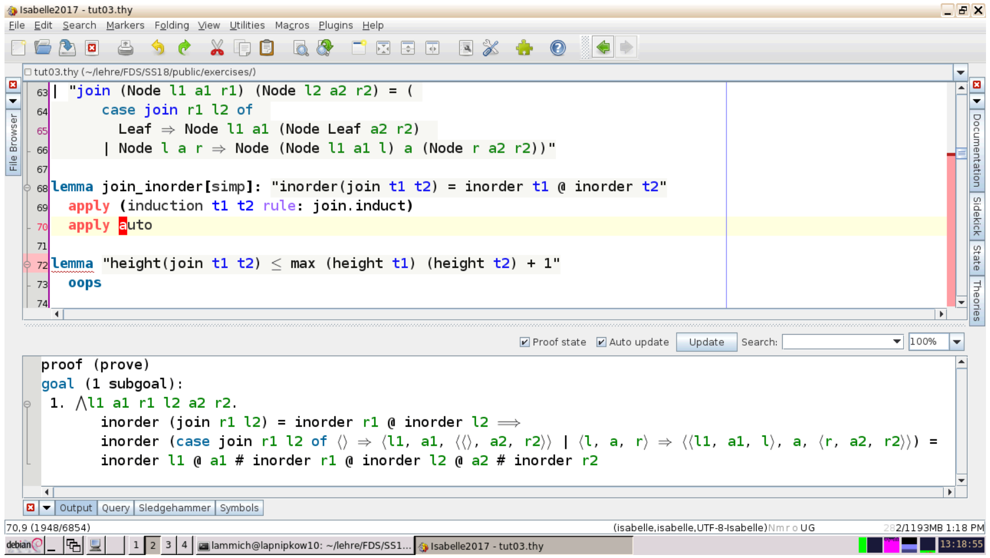
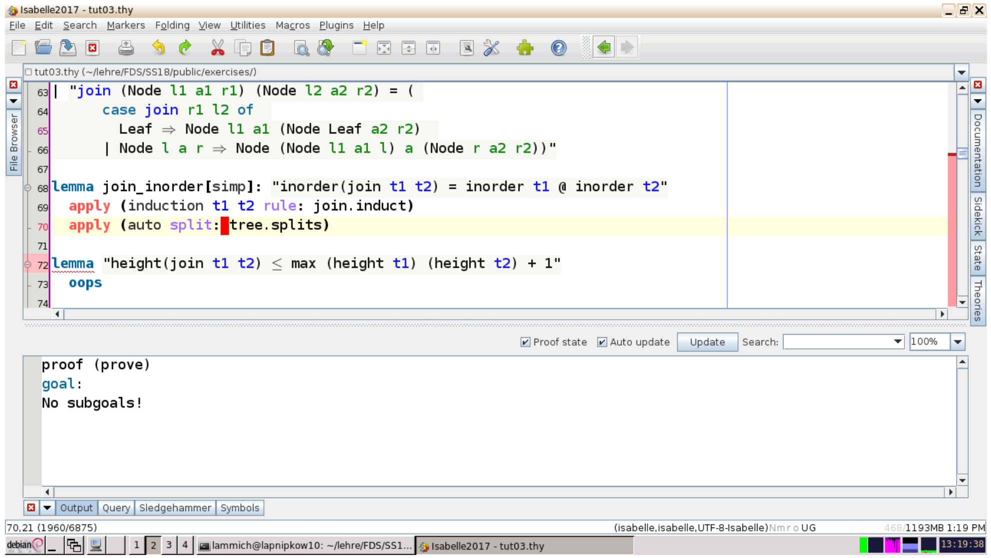

Isabelle2017 - tut03.thy
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
apply (induction t)
apply (auto simp: aux)
done
text ‹
\Exercise{Height-Preserving In-Order Join}
Write a function that joins two binary trees such that
▪ The in-order traversal of the new tree is the concatenation of the in-order traversals of the original
Output Query Sledgehammer Symbols

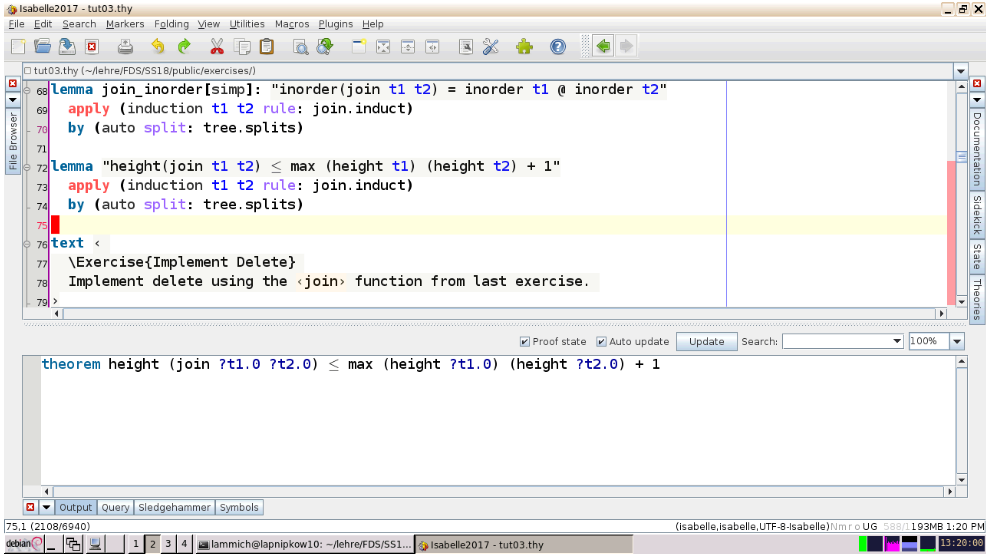

Isabelle2017 - tut03.thy
text ‹
\Exercise{Height-Preserving In-Order Join}
Write a function that joins two binary trees such that
▪ The in-order traversal of the new tree is the concatenation of the in-order traversals of the original
▪ The new tree is at most one higher than the highest original tree
Hint: Once you got the function right, proofs are easy!
›
fun join :: "'a tree ⇒ 'a tree ⇒ 'a tree"
where
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = undefined"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
oops
consts

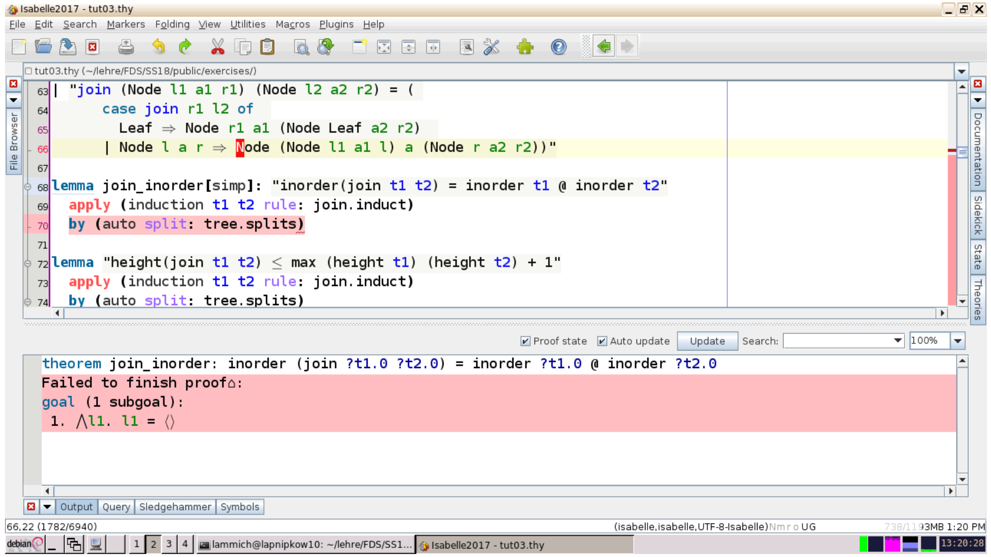

Isabelle2017 - tut03.thy
text ‹
\Exercise{Height-Preserving In-Order Join}
Write a function that joins two binary trees such that
▪ The in-order traversal of the new tree is the concatenation of the in-order traversals of the original
▪ The new tree is at most one higher than the highest original tree
Hint: Once you got the function right, proofs are easy!
›
fun join :: "'a tree ⇒ 'a tree ⇒ 'a tree"
where
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = undefined"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
oops
consts

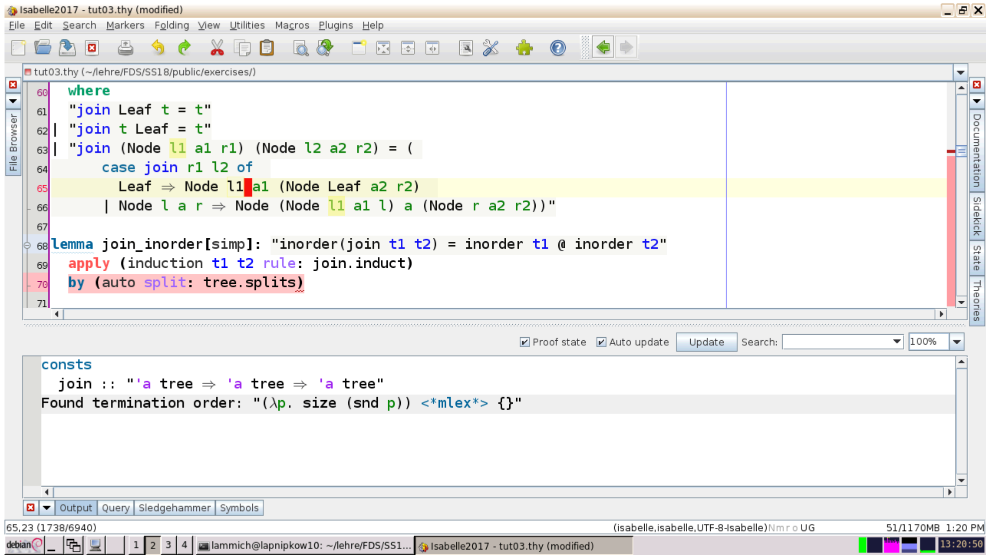

Isabelle2017 - tut03.thy
Write a function that joins two binary trees such that
▪ The in-order traversal of the new tree is the concatenation of the in-order traversals of the original
▪ The new tree is at most one higher than the highest original tree
Hint: Once you got the function right, proofs are easy!
›
fun join :: "'a tree ⇒ 'a tree ⇒ 'a tree"
where
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ undefined
| Node l a r ⇒ undefined)"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
oops
consts

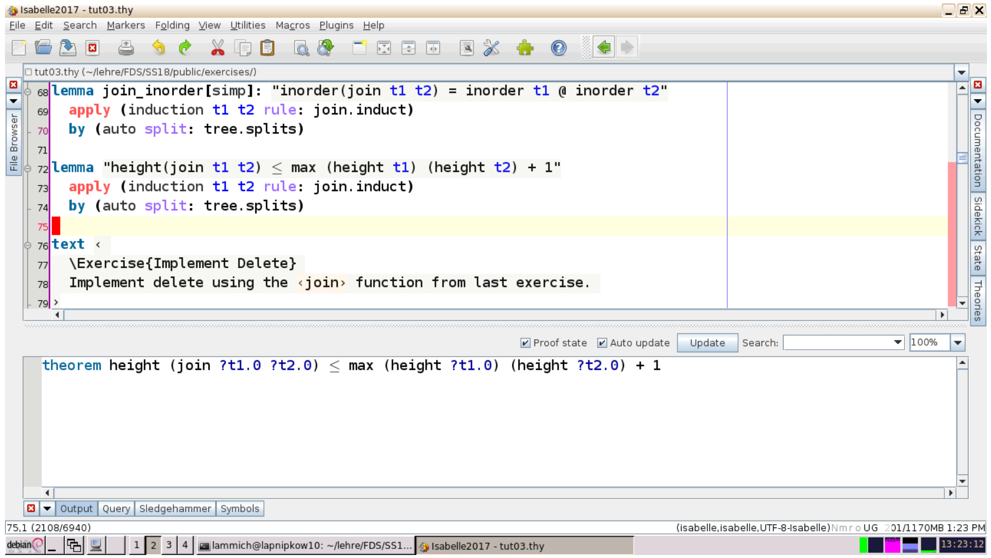

Isabelle2017 - tut03.thy
Write a function that joins two binary trees such that
▪ The in-order traversal of the new tree is the concatenation of the in-order traversals of the original
▪ The new tree is at most one higher than the highest original tree
Hint: Once you got the function right, proofs are easy!
›
fun join :: "'a tree ⇒ 'a tree ⇒ 'a tree"
where
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node r1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
oops
consts

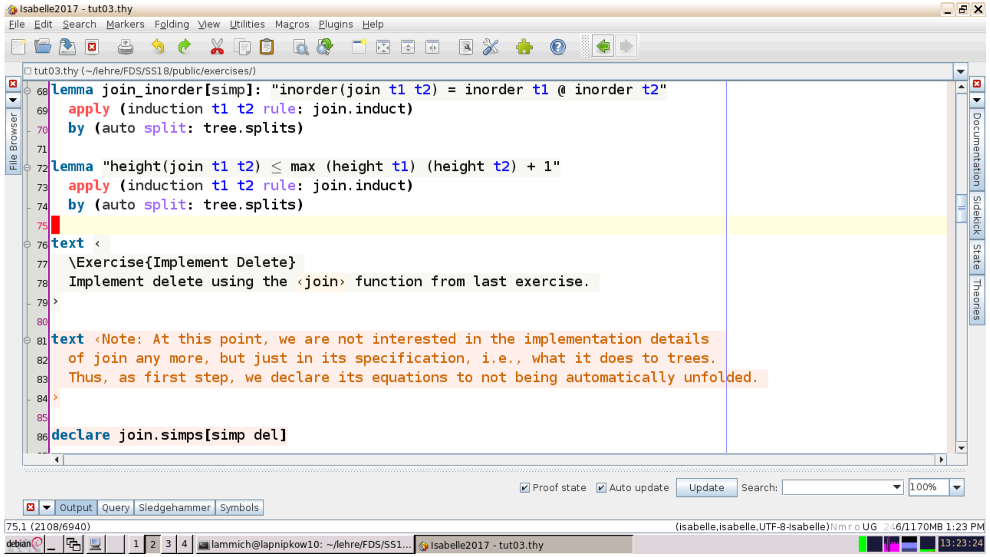

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
Hint: Once you got the function right, proofs are easy!
›
fun join :: "'a tree ⇒ 'a tree ⇒ 'a tree"
where
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
appl
apply (keyword)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
oops
proof (prove)

Isabelle2017 - tut03.thy
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
apply auto
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
oops
proof (prove)
goal (1 subgoal):
1. ⋀l1 a1 r1 l2 a2 r2.
inorder (join r1 l2) = inorder r1 @ inorder l2 ⟹
inorder (case join r1 l2 of ⟨⟩ ⇒ ⟨l1, a1, ⟨⟨⟩, a2, r2⟩⟩ | ⟨l, a, r⟩ ⇒ ⟨⟨l1, a1, l⟩, a, ⟨r, a2, r2⟩⟩) =
inorder l1 @ a1 # inorder r1 @ inorder l2 @ a2 # inorder r2

Isabelle2017 - tut03.thy
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
apply auto
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
oops
proof (prove)
goal (1 subgoal):
1. ⋀l1 a1 r1 l2 a2 r2.
inorder (join r1 l2) = inorder r1 @ inorder l2 ⟹
inorder (case join r1 l2 of ⟨⟩ ⇒ ⟨l1, a1, ⟨⟨⟩, a2, r2⟩⟩ | ⟨l, a, r⟩ ⇒ ⟨⟨l1, a1, l⟩, a, ⟨r, a2, r2⟩⟩) =
inorder l1 @ a1 # inorder r1 @ inorder l2 @ a2 # inorder r2

Isabelle2017 - tut03.thy
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
apply (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
oops
proof (prove)
goal:
No subgoals!

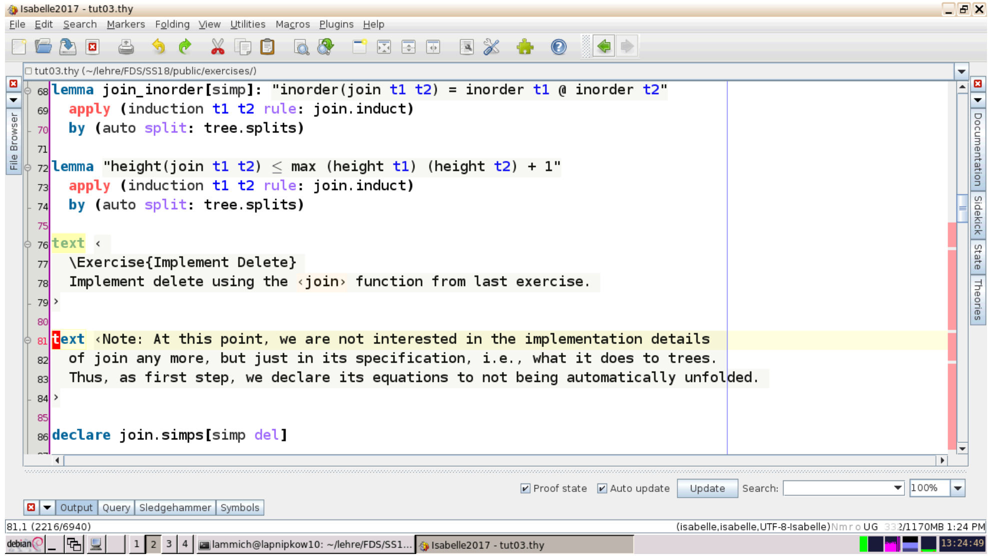

Isabelle2017 - tut03.thy
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
text ‹
\Exercise{Implement Delete}
Implement delete using the ‹join› function from last exercise.
›
theorem height (join ?t1.0 ?t2.0) ≤ max (height ?t1.0) (height ?t2.0) + 1

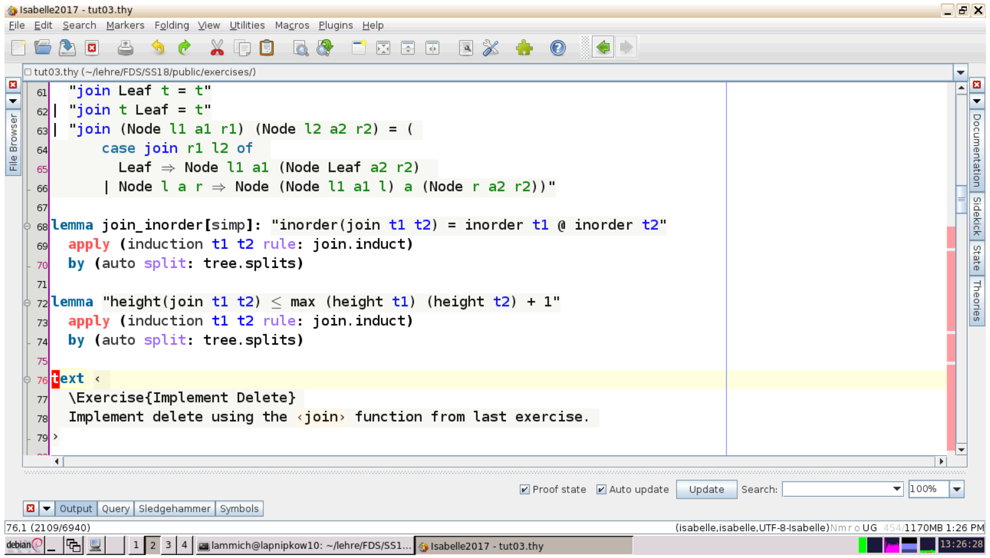

Isabelle2017 - tut03.thy
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node r1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
theorem join_inorder: inorder (join ?t1.0 ?t2.0) = inorder ?t1.0 @ inorder ?t2.0
Failed to finish proof⌂:
goal (1 subgoal):
1. ⋀l1. l1 = ⟨⟩

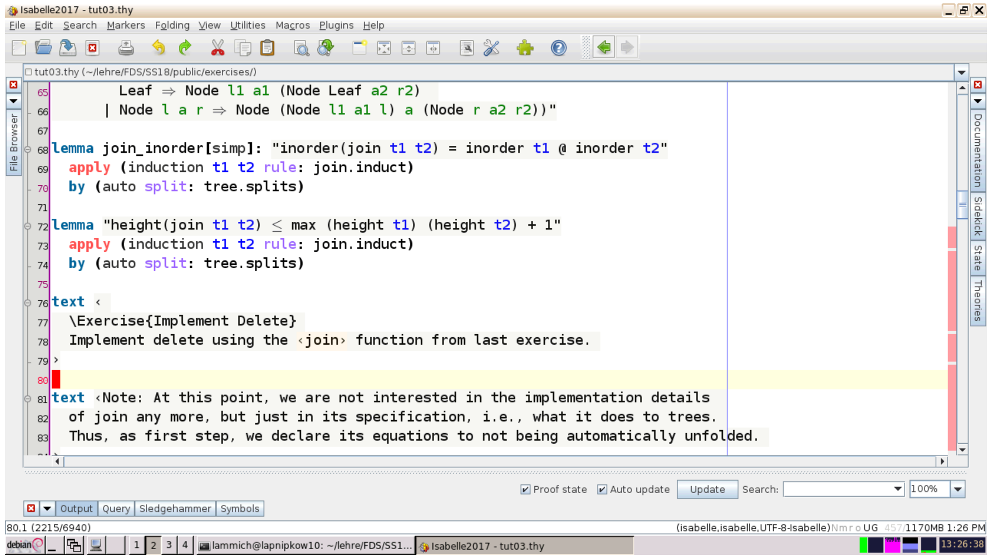

Isabelle2017 - tut03.thy (modified)
where
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
consts
join :: "'a tree ⇒ 'a tree ⇒ 'a tree"
Found termination order: "(λp. size (snd p)) <*mlex*> {}"

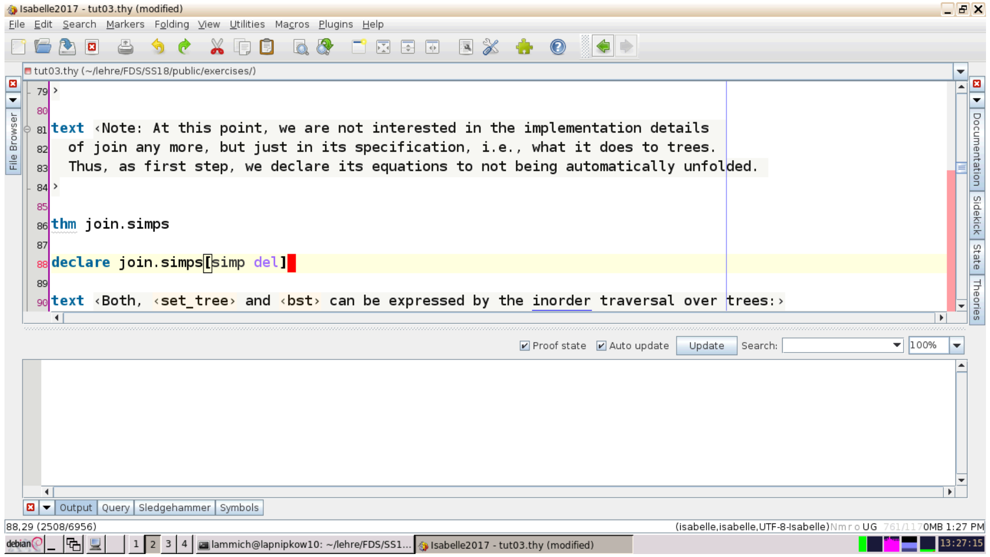

Isabelle2017 - tut03.thy
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
text ‹
\Exercise{Implement Delete}
Implement delete using the ‹join› function from last exercise.
›
theorem height (join ?t1.0 ?t2.0) ≤ max (height ?t1.0) (height ?t2.0) + 1

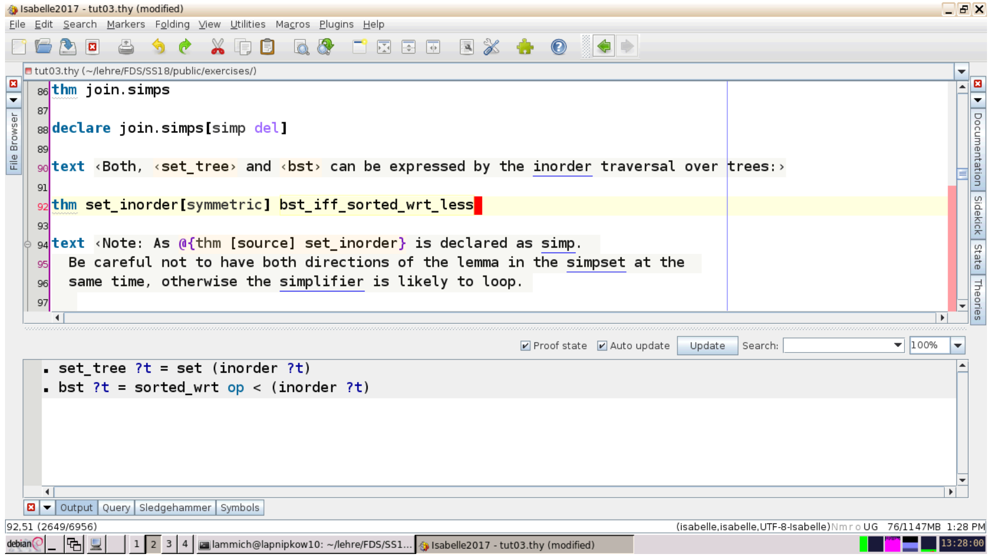

Isabelle2017 - tut03.thy
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
text ‹
\Exercise{Implement Delete}
Implement delete using the ‹join› function from last exercise.
›
text ‹Note: At this point, we are not interested in the implementation details
of join any more, but just in its specification, i.e., what it does to trees.
Thus, as first step, we declare its equations to not being automatically unfolded.
›
declare join.simps[simp del]

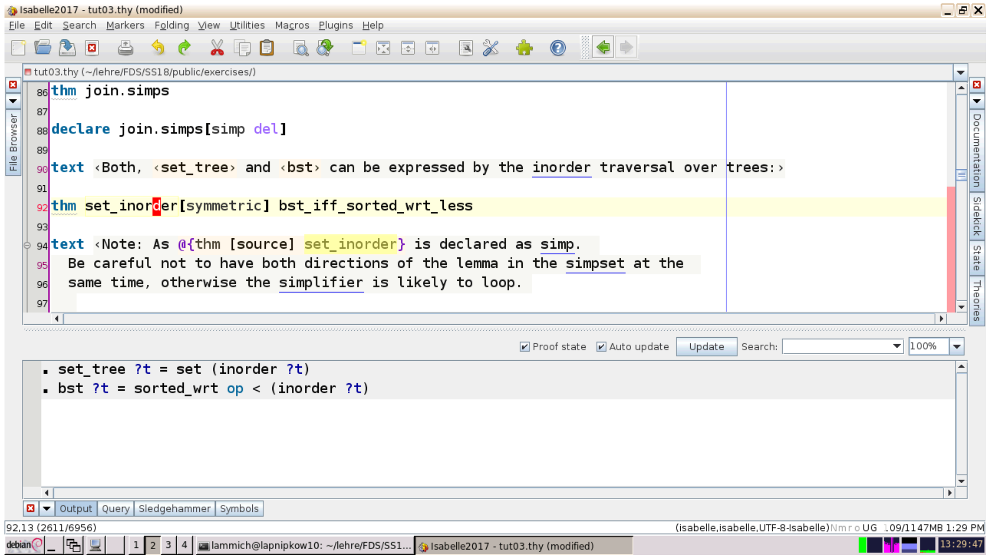

Isabelle2017 - tut03.thy
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
text ‹
\Exercise{Implement Delete}
Implement delete using the ‹join› function from last exercise.
›
text ‹Note: At this point, we are not interested in the implementation details
of join any more, but just in its specification, i.e., what it does to trees.
Thus, as first step, we declare its equations to not being automatically unfolded.
›
declare join.simps[simp del]

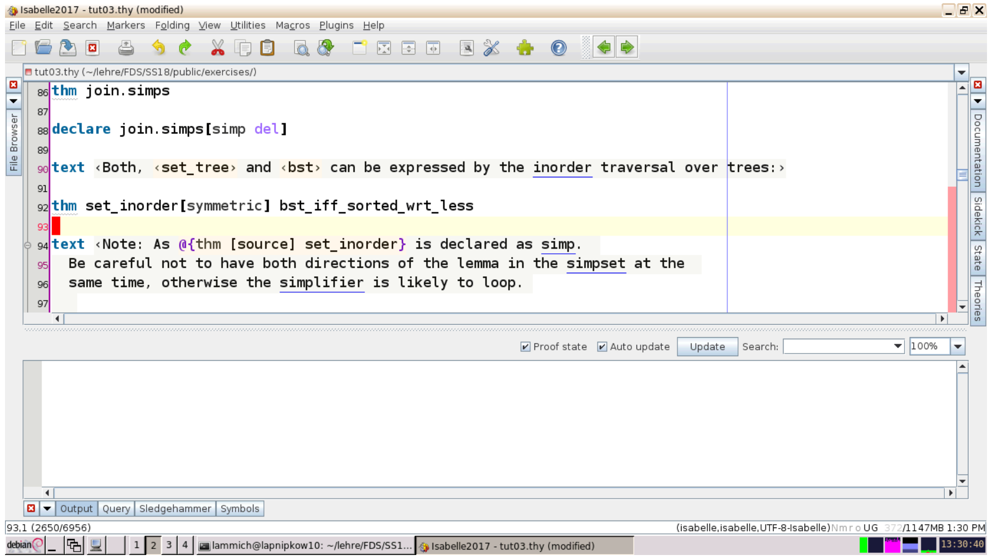

Isabelle2017 - tut03.thy
"join Leaf t = t"
| "join t Leaf = t"
| "join (Node l1 a1 r1) (Node l2 a2 r2) = (
case join r1 l2 of
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
text ‹
\Exercise{Implement Delete}
Implement delete using the ‹join› function from last exercise.
›

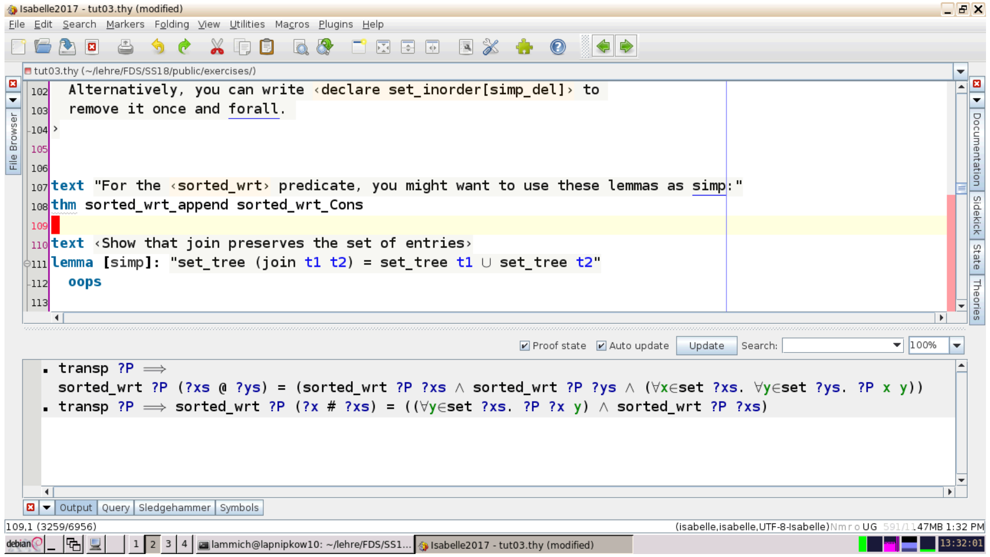

Isabelle2017 - tut03.thy
Leaf ⇒ Node l1 a1 (Node Leaf a2 r2)
| Node l a r ⇒ Node (Node l1 a1 l) a (Node r a2 r2))"
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"
apply (induction t1 t2 rule: join.induct)
by (auto split: tree.splits)
text ‹
\Exercise{Implement Delete}
Implement delete using the ‹join› function from last exercise.
›
text ‹Note: At this point, we are not interested in the implementation details
of join any more, but just in its specification, i.e., what it does to trees.
Thus, as first step, we declare its equations to not being automatically unfolded.

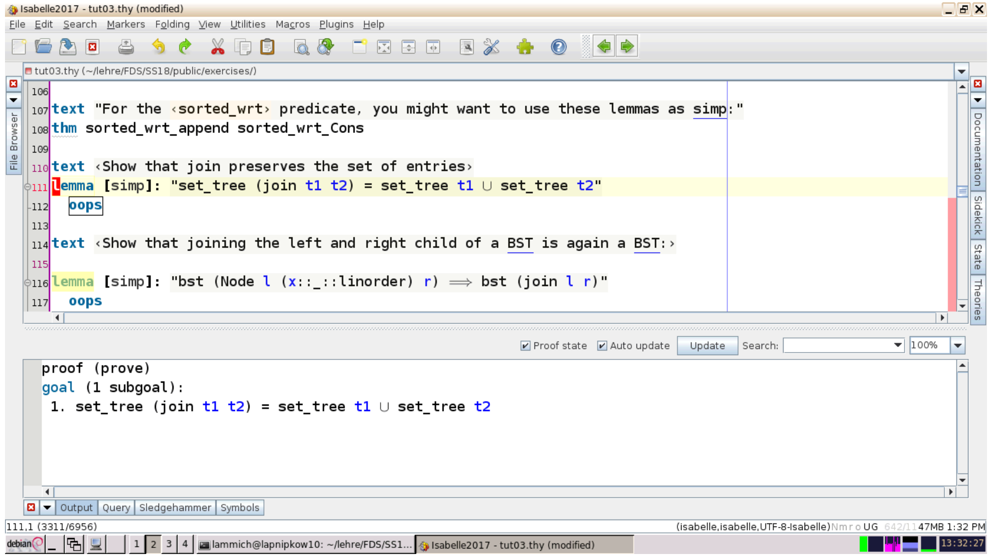

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Note: At this point, we are not interested in the implementation details
of join any more, but just in its specification, i.e., what it does to trees.
Thus, as first step, we declare its equations to not being automatically unfolded.
›
thm join.simps
declare join.simps[simp del]
text ‹Both, ‹set_tree› and ‹bst› can be expressed by the inorder traversal over trees:›
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
88,29 (2508/6956)
(isabelle,isabelle,UTF-8-Isabelle)
13:27:15

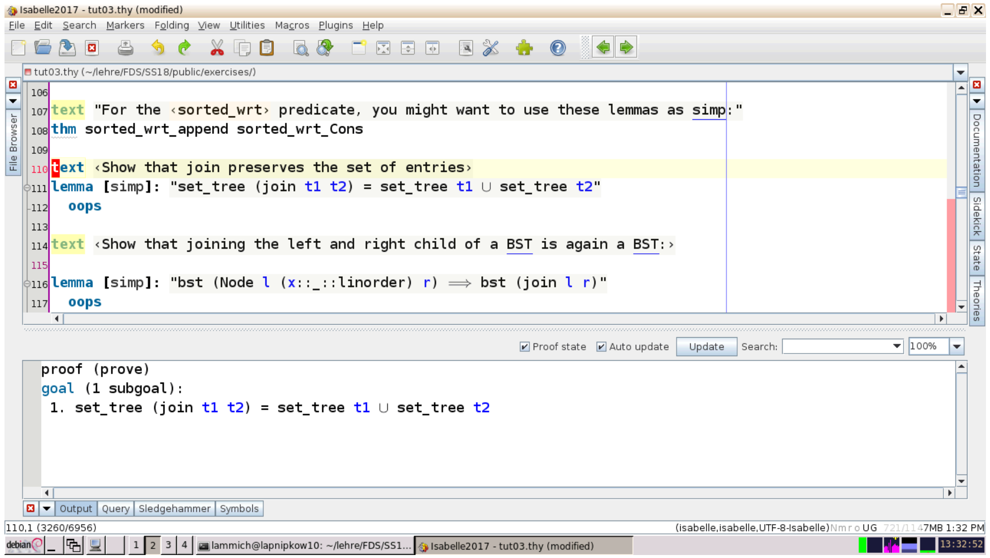

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
thm join.simps
declare join.simps[simp del]
text ‹Both, ‹set_tree› and ‹bst› can be expressed by the inorder traversal over trees:›
thm set_inorder[symmetric] bst_iff_sorted_wrt_less
text ‹Note: As @{thm [source] set_inorder} is declared as simp.
Be careful not to have both directions of the lemma in the simpset at the
same time, otherwise the simplifier is likely to loop.
set_tree ?t = set (inorder ?t)
bst ?t = sorted_wrt op < (inorder ?t)
Output
Query
Sledgehammer
Symbols
92,51 (2649/6956)
(isabelle,isabelle,UTF-8-Isabelle)
13:28:00

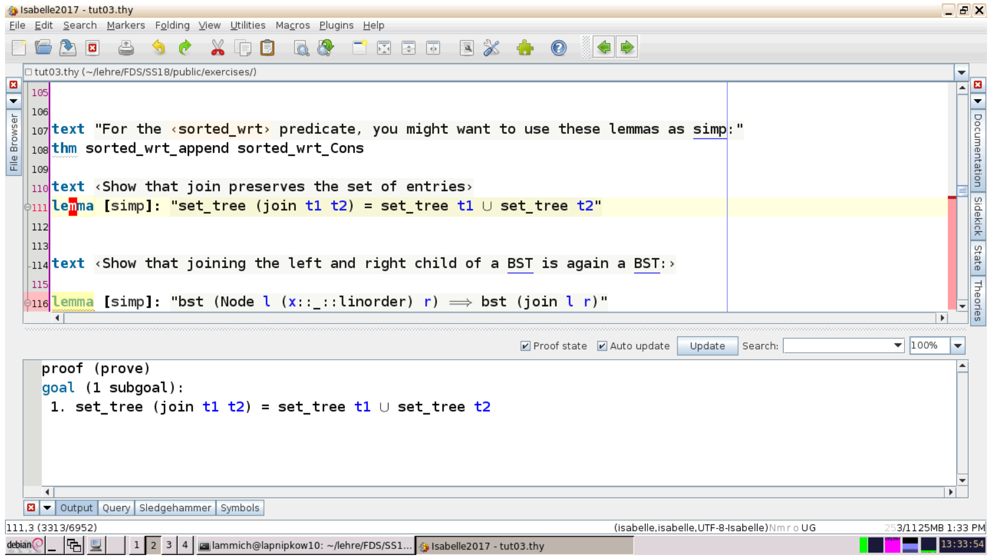

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
thm join.simps
declare join.simps[simp del]
text ‹Both, ‹set_tree› and ‹bst› can be expressed by the inorder traversal over trees:›
thm set_inorder[symmetric] bst_iff_sorted_wrt_less
text ‹Note: As @{thm [source] set_inorder} is declared as simp.
Be careful not to have both directions of the lemma in the simpset at the
same time, otherwise the simplifier is likely to loop.
set_tree ?t = set (inorder ?t)
bst ?t = sorted_wrt op < (inorder ?t)
Output
Query
Sledgehammer
Symbols
92,13 (2611/6956)
(isabelle,isabelle,UTF-8-Isabelle)
13:29:47

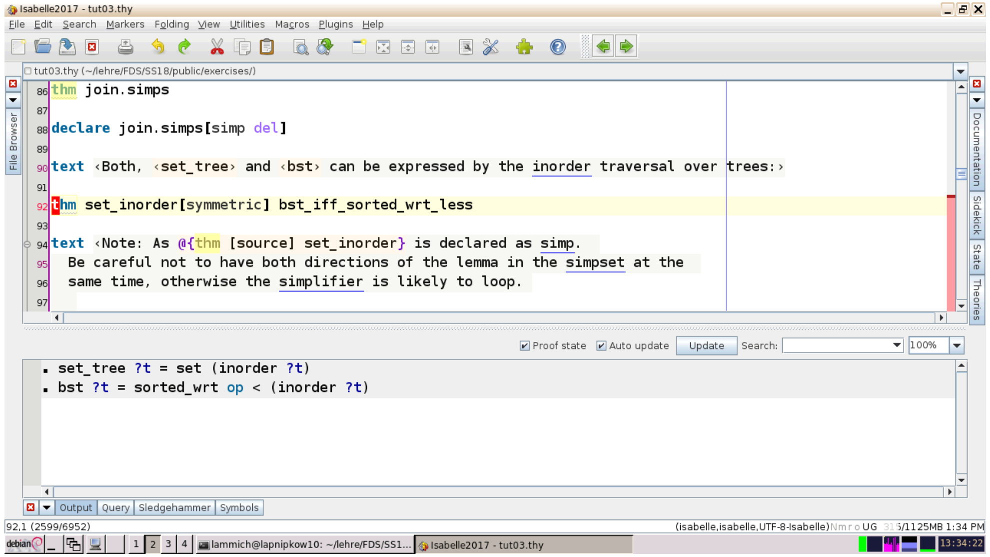
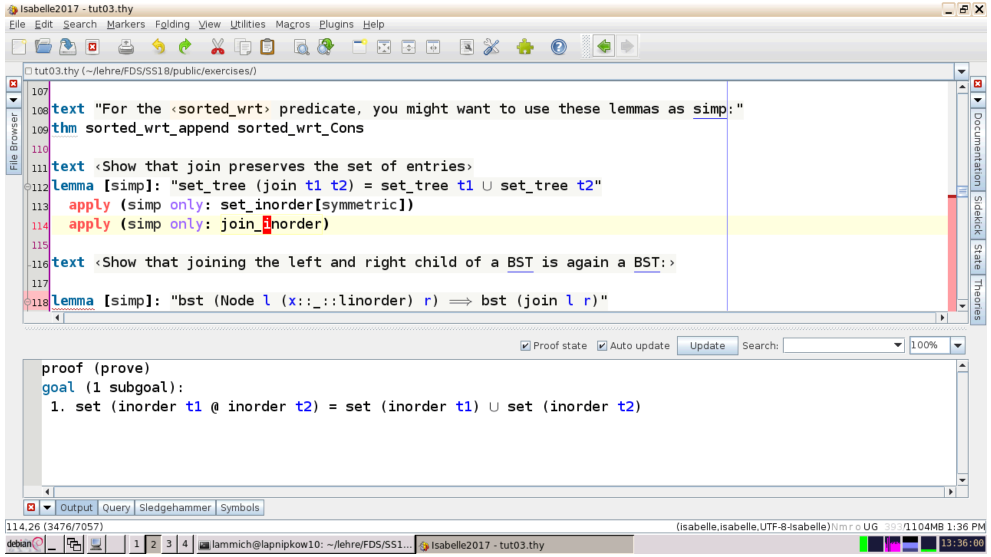
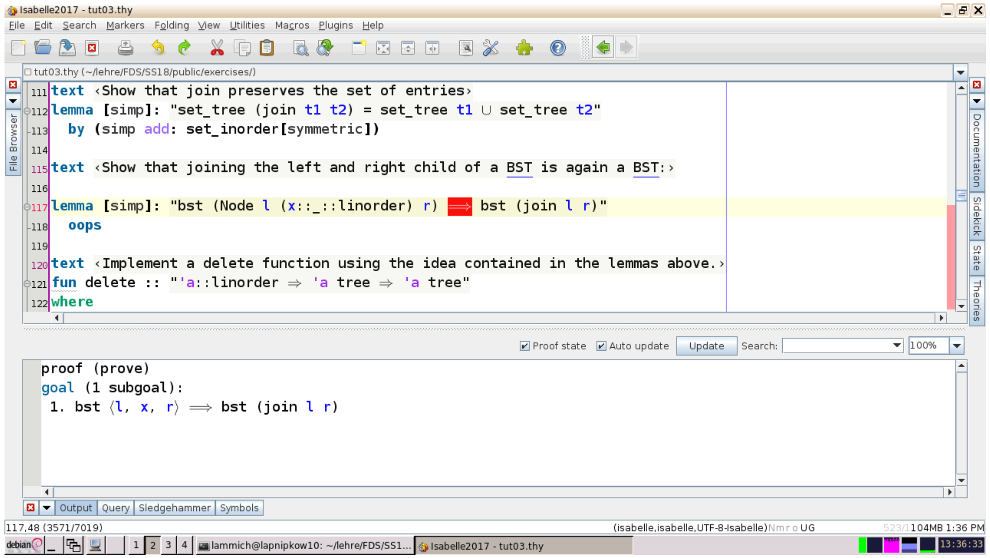
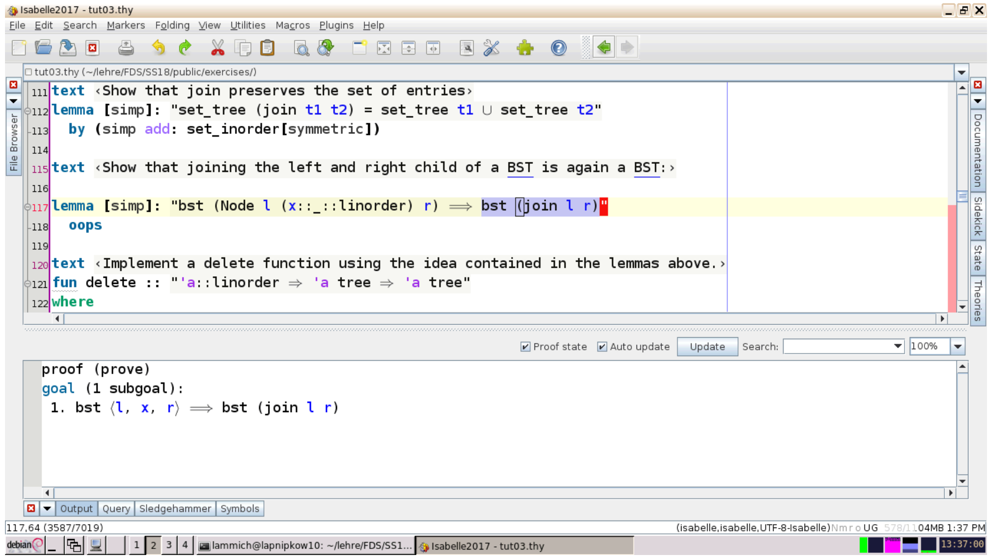

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
thm join.simps
declare join.simps[simp del]
text ‹Both, ‹set_tree› and ‹bst› can be expressed by the inorder traversal over trees:›
thm set_inorder[symmetric] bst_iff_sorted_wrt_less
text ‹Note: As @{thm [source] set_inorder} is declared as simp.
Be careful not to have both directions of the lemma in the simpset at the
same time, otherwise the simplifier is likely to loop.
Output
Query
Sledgehammer
Symbols
93,1 (2650/6956)
(isabelle,isabelle,UTF-8-Isabelle)
13:30:40

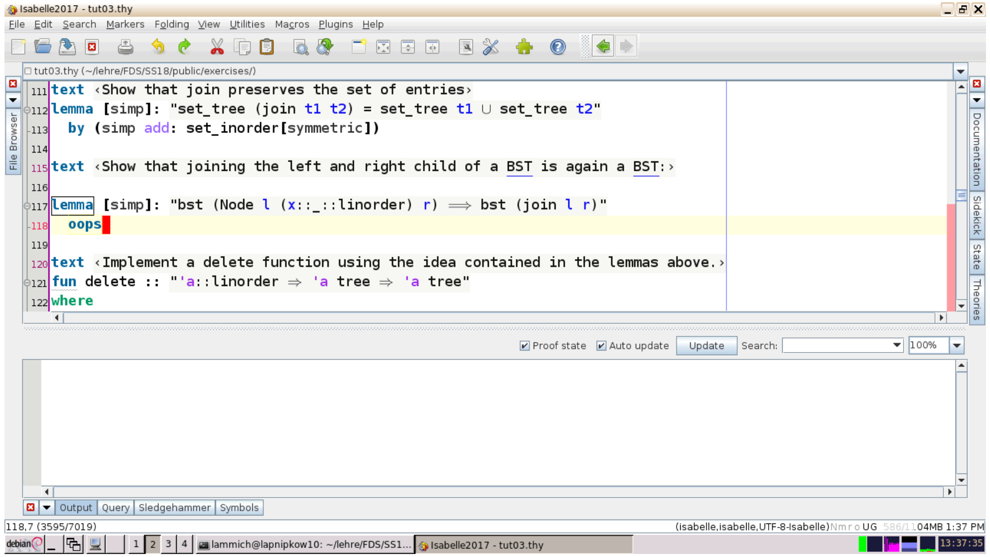

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
Alternatively, you can write ‹declare set_inorder[simp_del]› to
remove it once and forall.
›
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
oops
• transp ?P ⟹
sorted_wrt ?P (?xs @ ?ys) = (sorted_wrt ?P ?xs ∧ sorted_wrt ?P ?ys ∧ (∀x∈set ?xs. ∀y∈set ?ys. ?P x y))
• transp ?P ⟹ sorted_wrt ?P (?x # ?xs) = ((∀y∈set ?xs. ?P ?x y) ∧ sorted_wrt ?P ?xs)
109.1 (3259/6956)

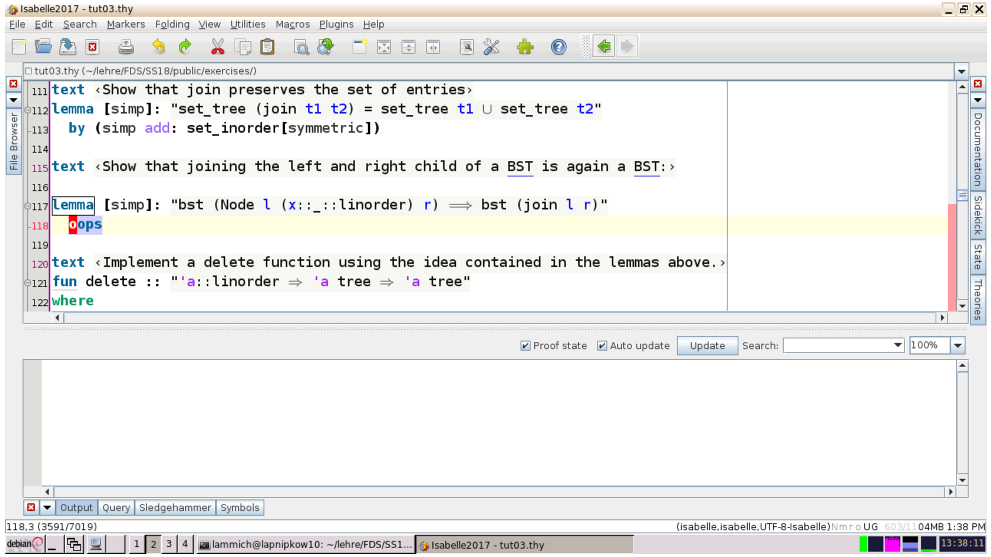

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
oops
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
oops
proof (prove)
goal (1 subgoal):
1. set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2
111.1 (3311/6956)

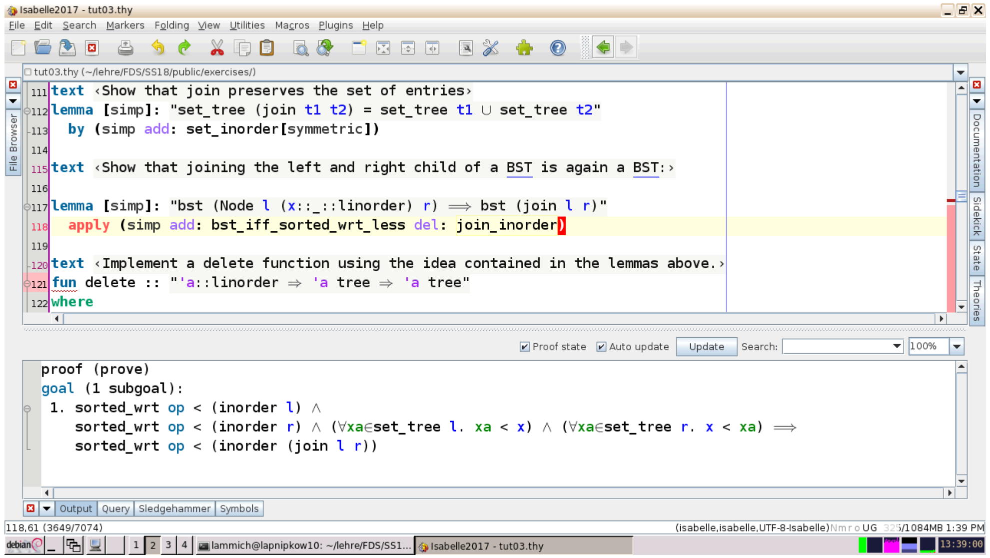

Isabelle2017 - tut03.thy (modified)
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
oops
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
oops
proof (prove)
goal (1 subgoal):
1. set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2
110.1 (3260/6956)

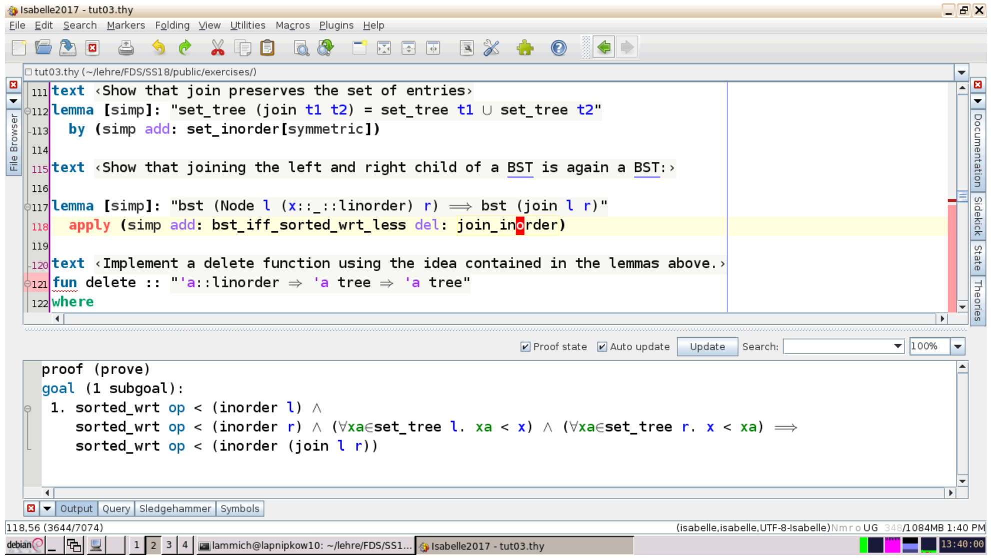

Isabelle2017 - tut03.thy
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
proof (prove)
goal (1 subgoal):
1. set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2
111.3 (3313/6952)

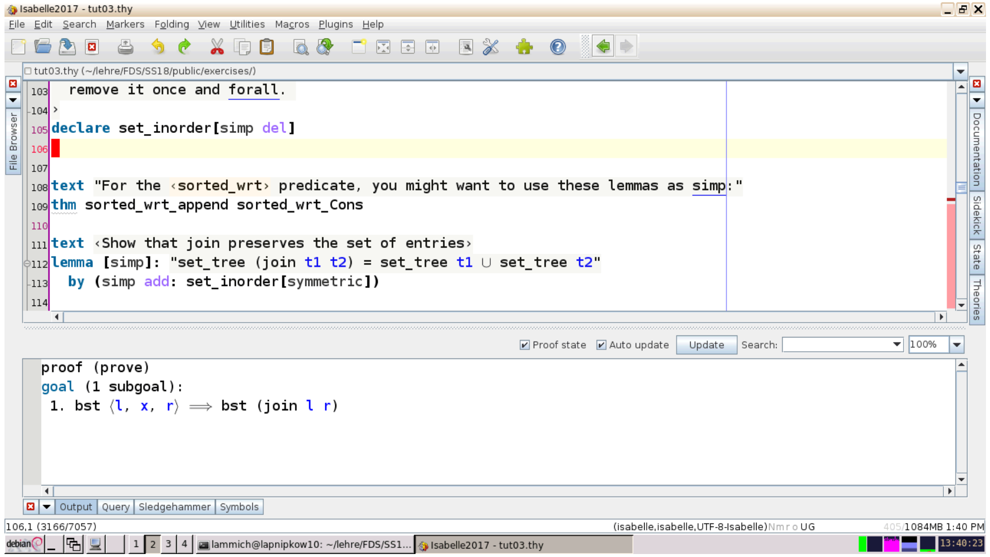

Isabelle2017 - tut03.thy
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
thm join.simps
declare join.simps[simp del]
text ‹Both, ‹set_tree› and ‹bst› can be expressed by the inorder traversal over trees:›
thm set_inorder[symmetric] bst_iff_sorted_wrt_less
text ‹Note: As @{thm [source] set_inorder} is declared as simp.
Be careful not to have both directions of the lemma in the simpset at the
same time, otherwise the simplifier is likely to loop.
• set_tree ?t = set (inorder ?t)
• bst ?t = sorted_wrt op < (inorder ?t)

Isabelle2017 - tut03.thy
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
apply (simp only: set_inorder[symmetric])
apply (simp only: join_inorder)
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
proof (prove)
goal (1 subgoal):
1. set (inorder t1 @ inorder t2) = set (inorder t1) ∪ set (inorder t2)

Isabelle2017 - tut03.thy
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
oops
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
proof (prove)
goal (1 subgoal):
1. bst ⟨l, x, r⟩ ⟹ bst (join l r)

Isabelle2017 - tut03.thy
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
oops
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
proof (prove)
goal (1 subgoal):
1. bst ⟨l, x, r⟩ ⟹ bst (join l r)

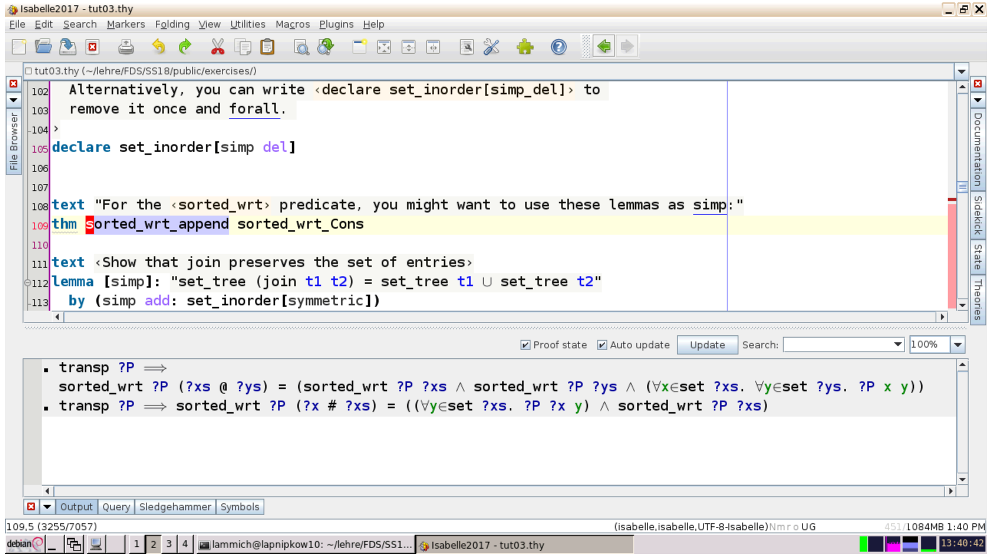

Isabelle2017 - tut03.thy
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
oops
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where

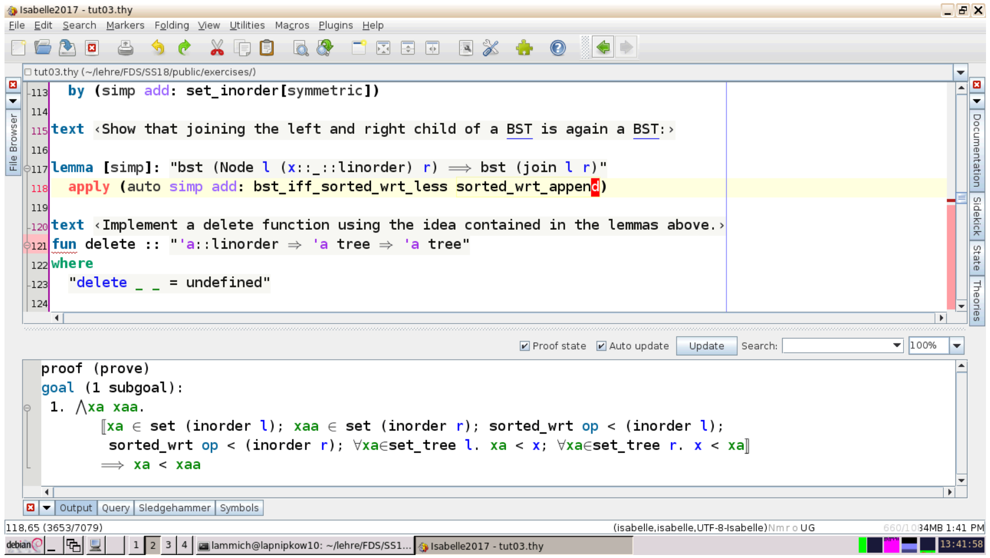

Isabelle2017 - tut03.thy
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
oops
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where

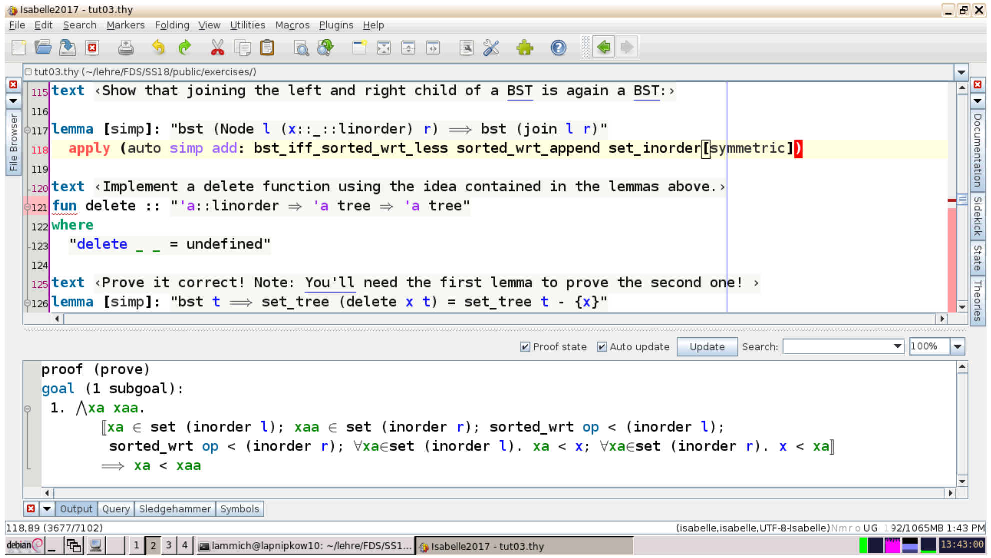

Isabelle2017 - tut03.thy
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (simp add: bst_iff_sorted_wrt_less del: join_inorder)
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
proof (prove)
goal (1 subgoal):
1. sorted_wrt op < (inorder l) ∧
sorted_wrt op < (inorder r) ∧ (∀xa∈set_tree l. xa < x) ∧ (∀xa∈set_tree r. x < xa) ⟹
sorted_wrt op < (inorder (join l r))

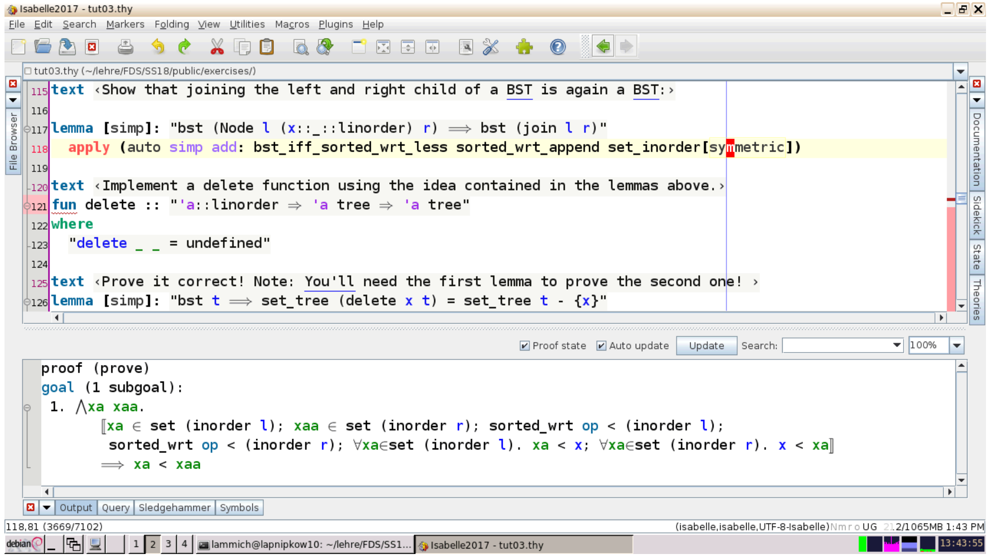

Isabelle2017 - tut03.thy
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (simp add: bst_iff_sorted_wrt_less del: join_inorder)
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
proof (prove)
goal (1 subgoal):
1. sorted_wrt op < (inorder l) ∧
sorted_wrt op < (inorder r) ∧ (∀xa∈set_tree l. xa < x) ∧ (∀xa∈set_tree r. x < xa) ⟹
sorted_wrt op < (inorder (join l r))

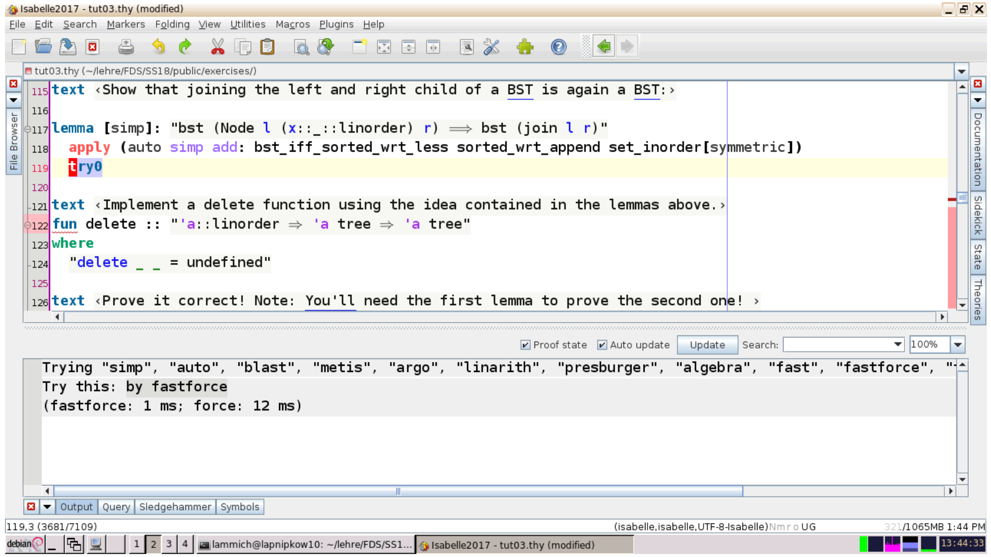

Isabelle2017 - tut03.thy
remove it once and forall.
›
declare set_inorder[simp del]
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
proof (prove)
goal (1 subgoal):
1. bst ⟨l, x, r⟩ ⟹ bst (join l r)

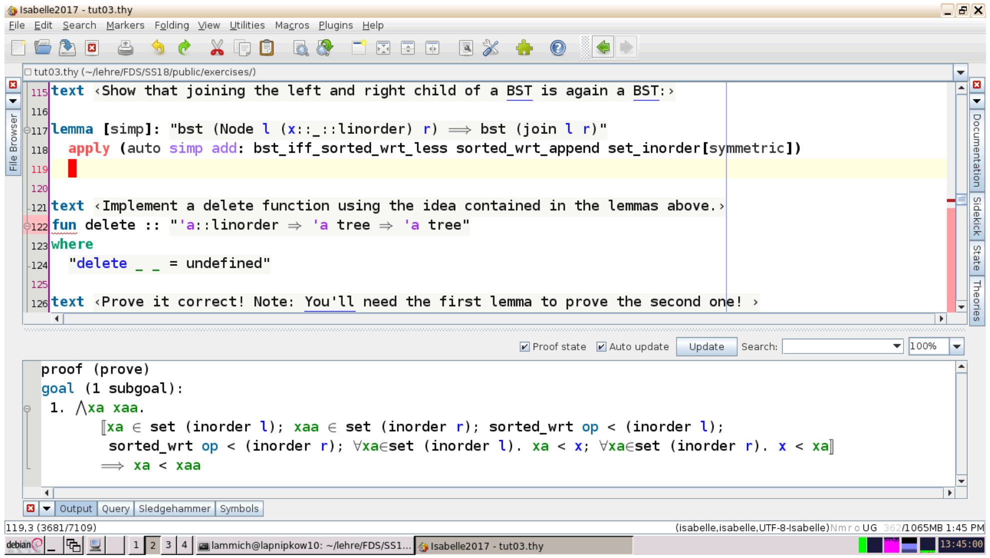

Isabelle2017 - tut03.thy
Alternatively, you can write ‹declare set_inorder[simp_del]› to
remove it once and forall.
›
declare set_inorder[simp del]
text "For the ‹sorted_wrt› predicate, you might want to use these lemmas as simp:"
thm sorted_wrt_append sorted_wrt_Cons
text ‹Show that join preserves the set of entries›
lemma [simp]: "set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"
by (simp add: set_inorder[symmetric])
transp ?P ⟹
sorted_wrt ?P (?xs @ ?ys) = (sorted_wrt ?P ?xs ∧ sorted_wrt ?P ?ys ∧ (∀x∈set ?xs. ∀y∈set ?ys. ?P x y))
transp ?P ⟹ sorted_wrt ?P (?x # ?xs) = ((∀y∈set ?xs. ?P ?x y) ∧ sorted_wrt ?P ?xs)

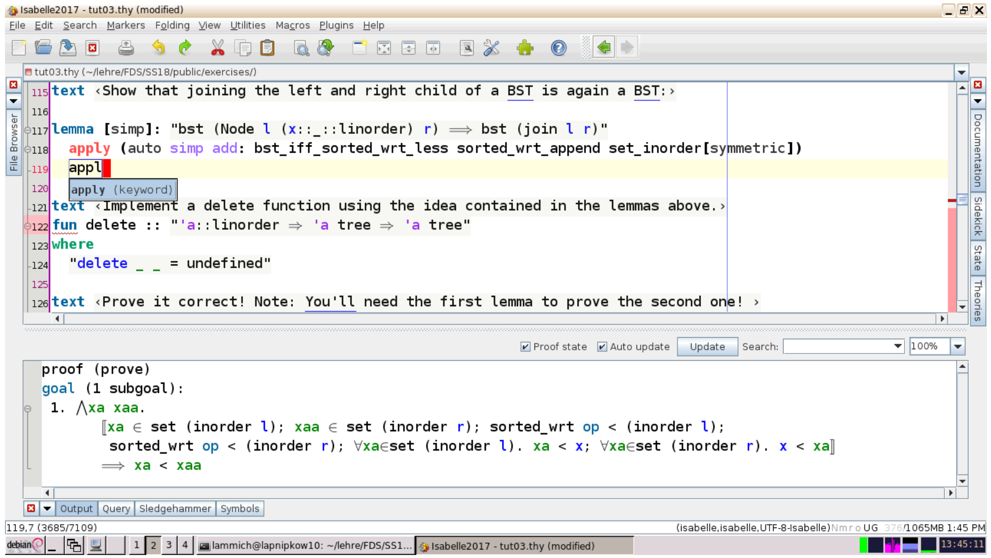

Isabelle2017 - tut03.thy
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append)
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
proof (prove)
goal (1 subgoal):
1. ⋀xa xaa.
⟦xa ∈ set (inorder l); xaa ∈ set (inorder r); sorted_wrt op < (inorder l);
sorted_wrt op < (inorder r); ∀xa∈set_tree l. xa < x; ∀xa∈set_tree r. x < xa⟧
⟹ xa < xaa

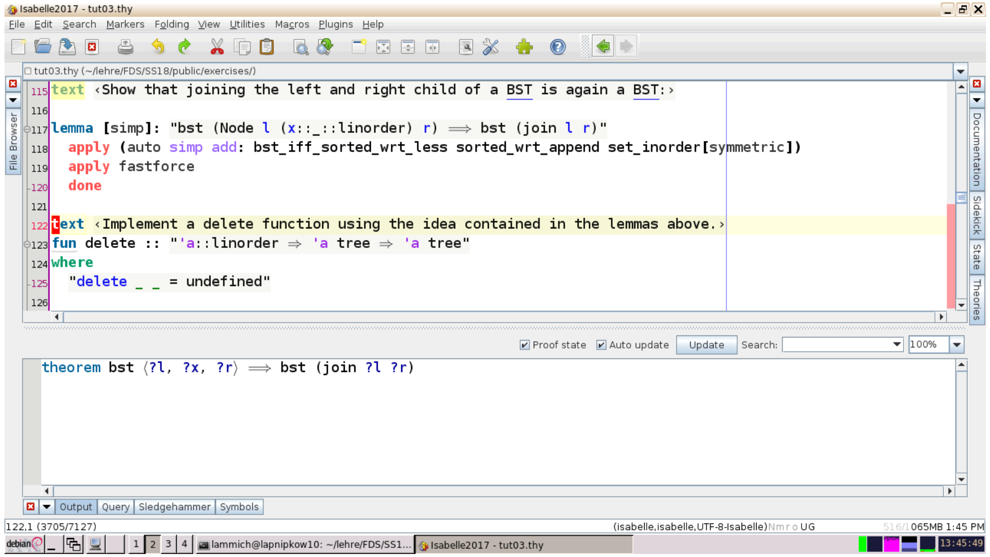

Isabelle2017 - tut03.thy
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
text ‹Prove it correct! Note: You'll need the first lemma to prove the second one! ›
lemma [simp]: "bst t ⟹ set_tree (delete x t) = set_tree t - {x}"
proof (prove)
goal (1 subgoal):
1. ⋀xa xaa.
⟦xa ∈ set (inorder l); xaa ∈ set (inorder r); sorted_wrt op < (inorder l);
sorted_wrt op < (inorder r); ∀xa∈set (inorder l). xa < x; ∀xa∈set (inorder r). x < xa⟧
⟹ xa < xaa

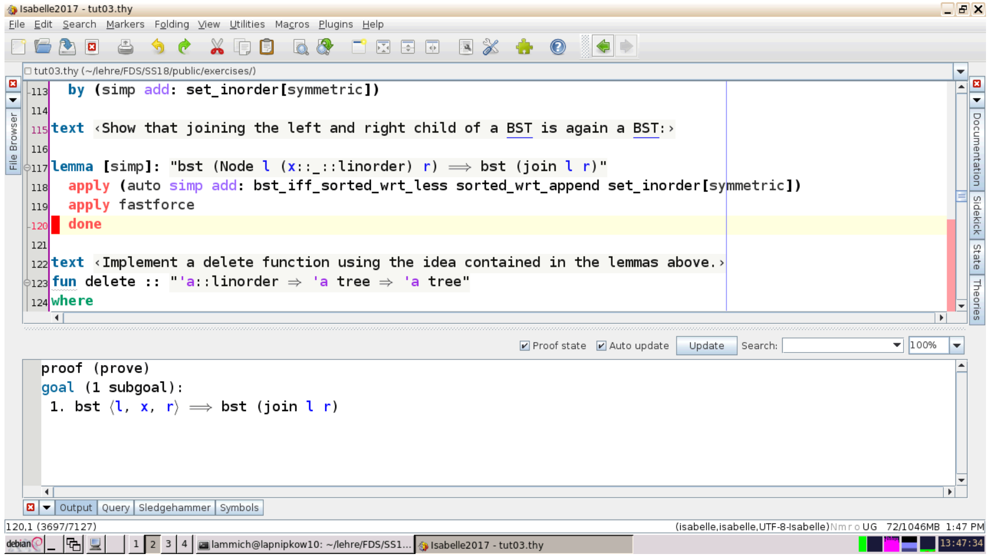

Isabelle2017 - tut03.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
text ‹Prove it correct! Note: You'll need the first lemma to prove the second one! ›
lemma [simp]: "bst t ⟹ set_tree (delete x t) = set_tree t - {x}"
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀xa xaa.
⟦xa ∈ set (inorder l); xaa ∈ set (inorder r); sorted_wrt op < (inorder l);
sorted_wrt op < (inorder r); ∀xa∈set (inorder l). xa < x; ∀xa∈set (inorder r). x < xa⟧
⟹ xa < xaa
Output Query Sledgehammer Symbols
118,81 (3669/7102)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut03.thy
13:43:55

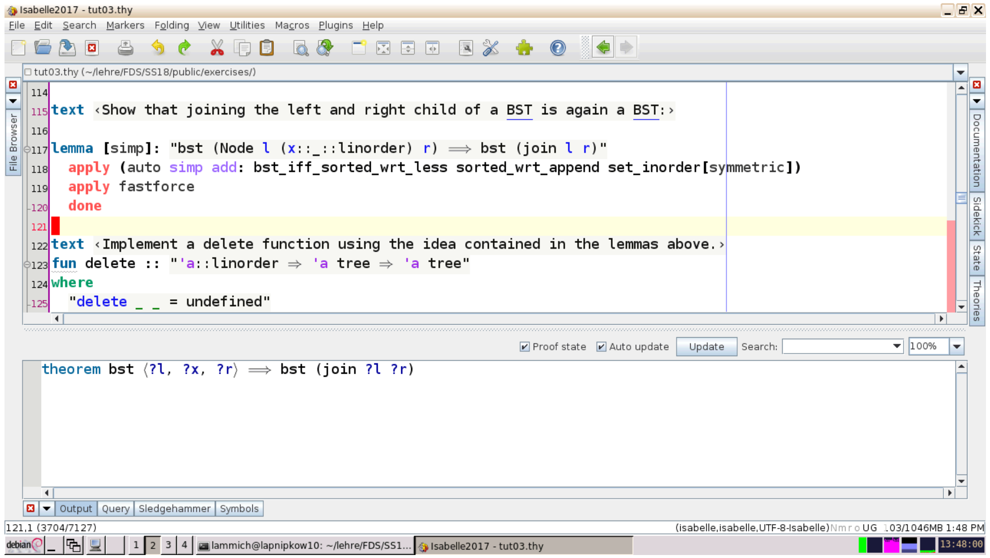

Isabelle2017 - tut03.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
try0
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
text ‹Prove it correct! Note: You'll need the first lemma to prove the second one! ›
Proof state Auto update Update Search: 100%
Trying "simp", "auto", "blast", "metis", "argo", "linarith", "presburger", "algebra", "fast", "fastforce",
Try this: by fastforce
(fastforce: 1 ms; force: 12 ms)
Output Query Sledgehammer Symbols
119,3 (3681/7109)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut03.thy (modified)
13:44:33


Isabelle2017 - tut03.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
text ‹Prove it correct! Note: You'll need the first lemma to prove the second one! ›
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀xa xaa.
⟦xa ∈ set (inorder l); xaa ∈ set (inorder r); sorted_wrt op < (inorder l);
sorted_wrt op < (inorder r); ∀xa∈set (inorder l). xa < x; ∀xa∈set (inorder r). x < xa⟧
⟹ xa < xaa
Output Query Sledgehammer Symbols
119,3 (3681/7109)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut03.thy
13:45:00


Isabelle2017 - tut03.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut03.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
appl
apply (keyword)
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
text ‹Prove it correct! Note: You'll need the first lemma to prove the second one! ›
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. ⋀xa xaa.
⟦xa ∈ set (inorder l); xaa ∈ set (inorder r); sorted_wrt op < (inorder l);
sorted_wrt op < (inorder r); ∀xa∈set (inorder l). xa < x; ∀xa∈set (inorder r). x < xa⟧
⟹ xa < xaa
Output Query Sledgehammer Symbols
119,7 (3685/7109)
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut03.thy (modified)
13:45:11


Isabelle2017 - tut03.thy
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
apply fastforce
done
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
theorem bst ⟨?l, ?x, ?r⟩ ⟹ bst (join ?l ?r)


Isabelle2017 - tut03.thy
by (simp add: set_inorder[symmetric])
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
apply fastforce
done
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
proof (prove)
goal (1 subgoal):
1. bst ⟨l, x, r⟩ ⟹ bst (join l r)


Isabelle2017 - tut03.thy
text ‹Show that joining the left and right child of a BST is again a BST:›
lemma [simp]: "bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"
apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])
apply fastforce
done
text ‹Implement a delete function using the idea contained in the lemmas above.›
fun delete :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"
where
"delete _ _ = undefined"
theorem bst ⟨?l, ?x, ?r⟩ ⟹ bst (join ?l ?r)