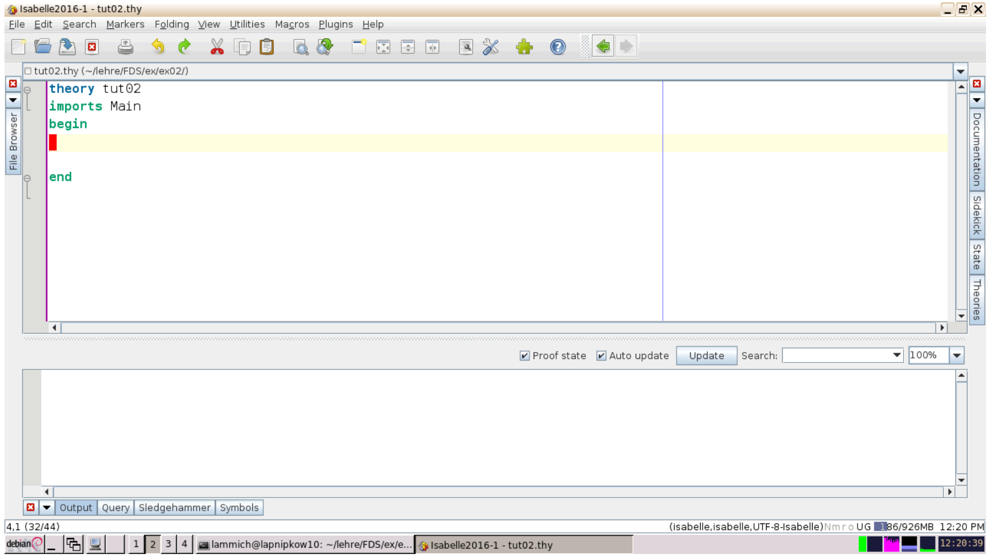
Script generated by TTT

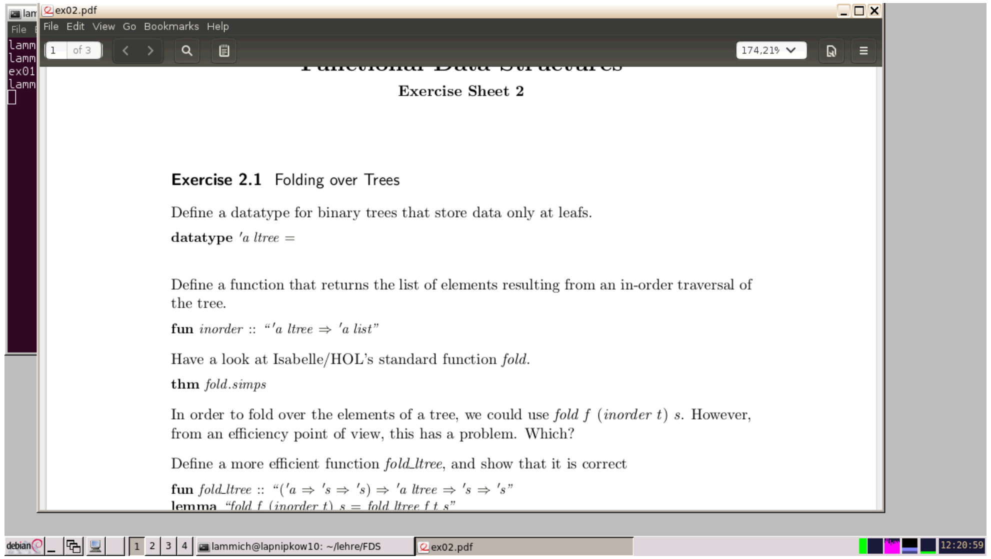
Title: Lammich: FDS Tutorial (05.05.2017)

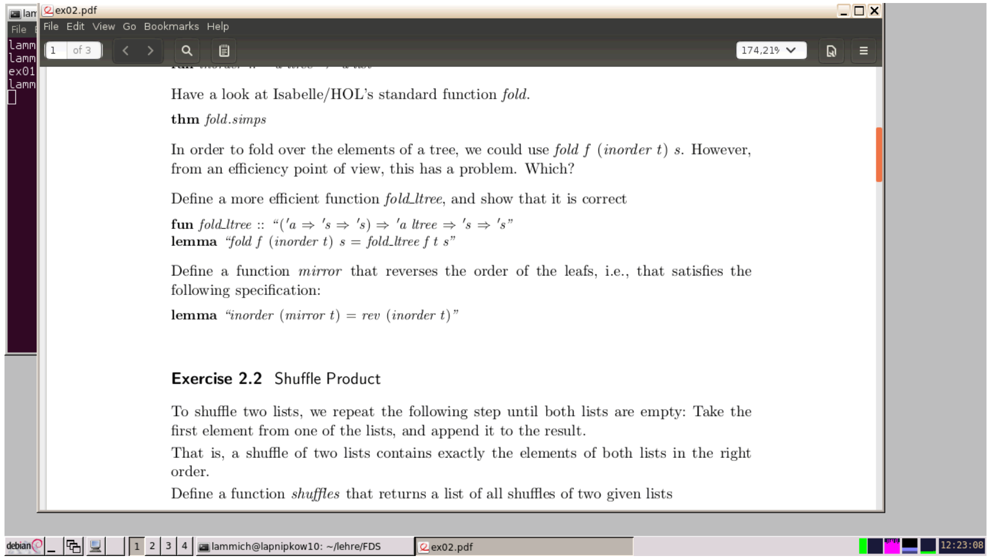
Date: Fri May 05 12:20:39 CEST 2017

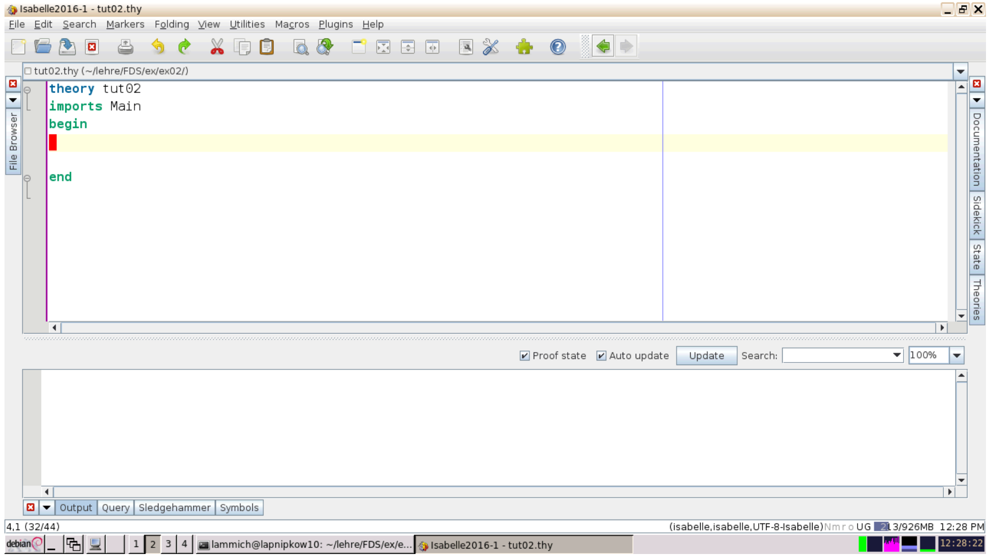
Duration: 95:50 min

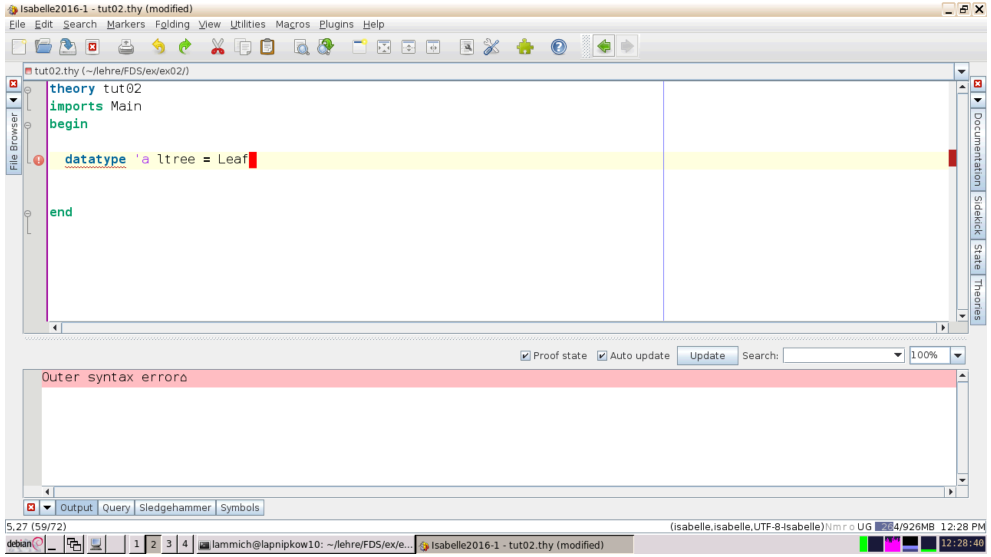
Pages: 91

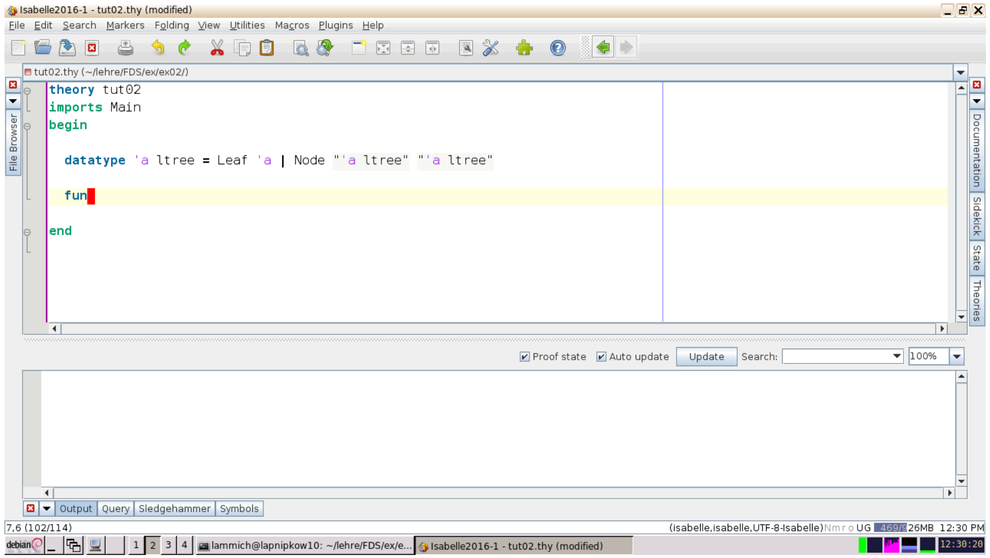


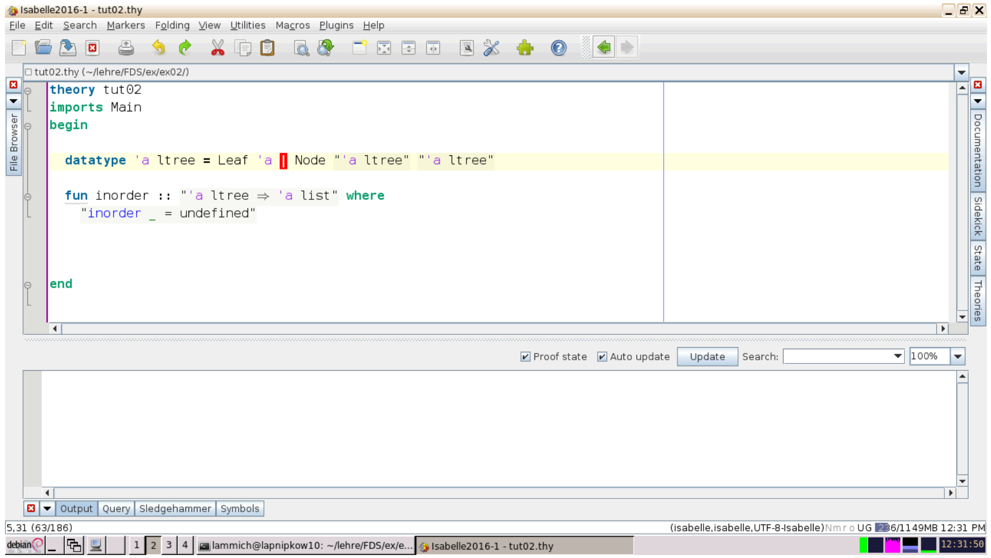


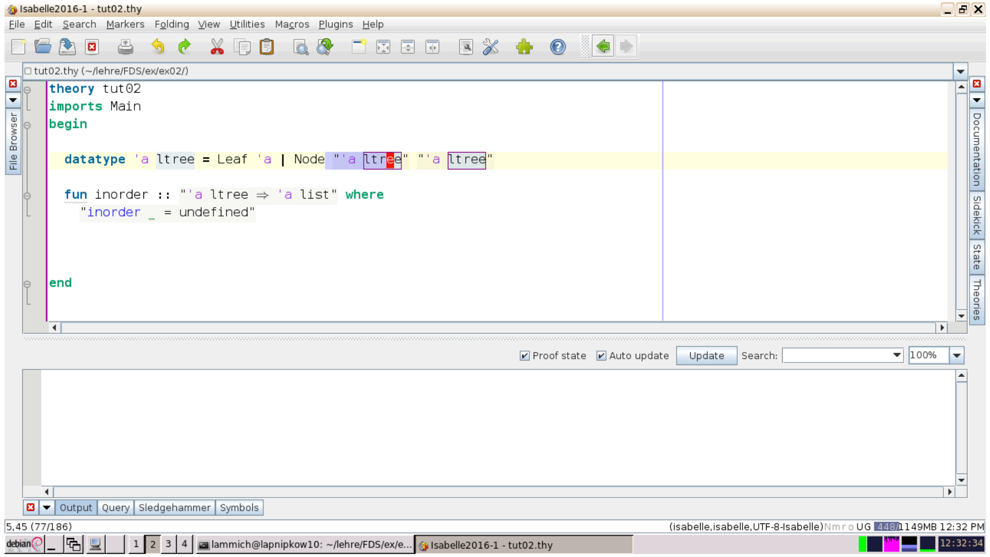


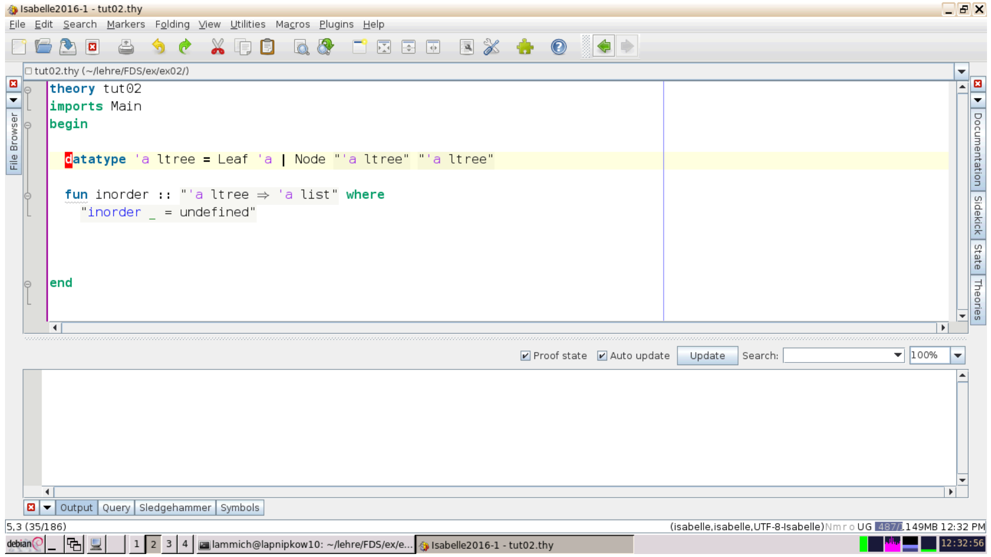

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
4,1 (32/44)
12:28 PM

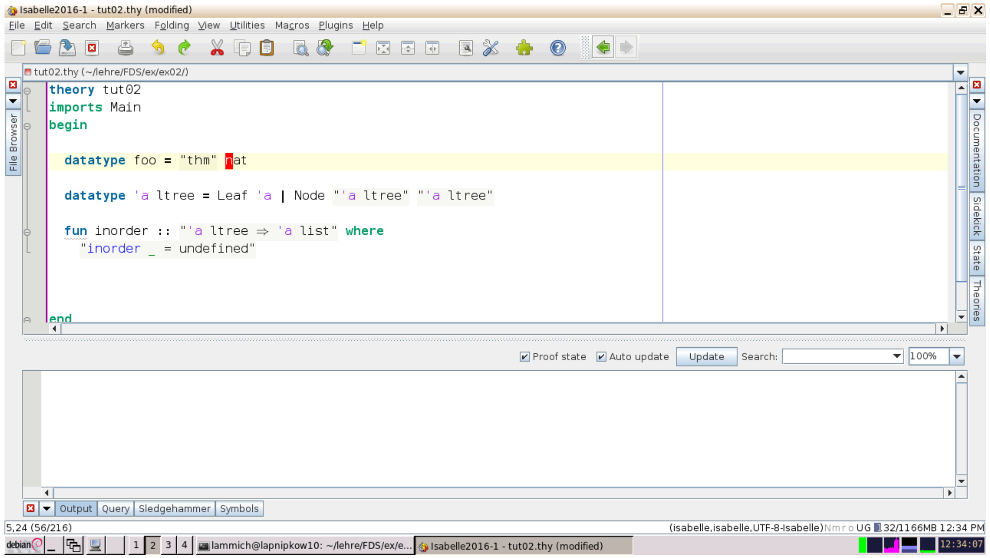

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf
end
Outer syntax error⌂
Output
Query
Sledgehammer
Symbols
5,27 (59/72)
12:28 PM

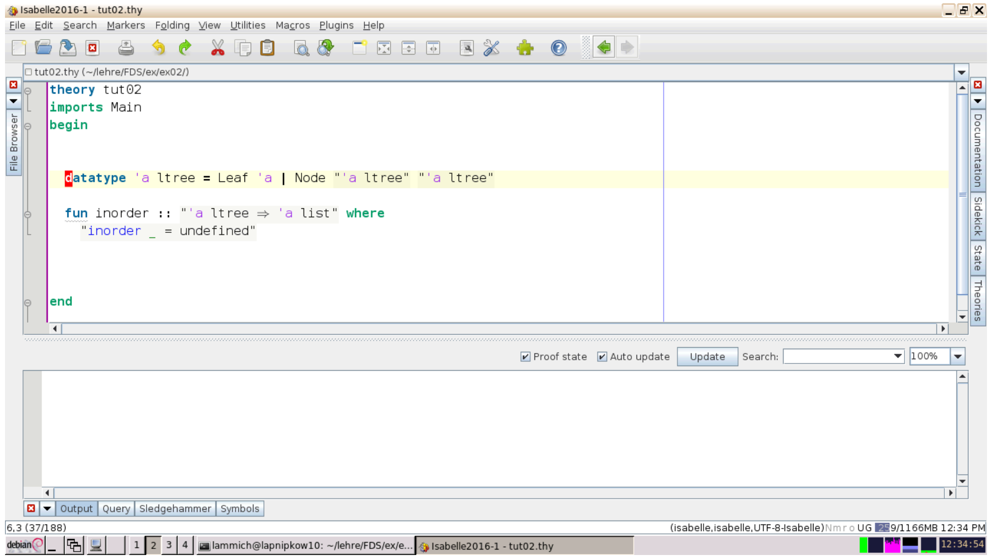

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf 'a | Node "'a ltree" "'a ltree"
fun
end
Output
Query
Sledgehammer
Symbols
7,6 (102/114)
12:30 PM

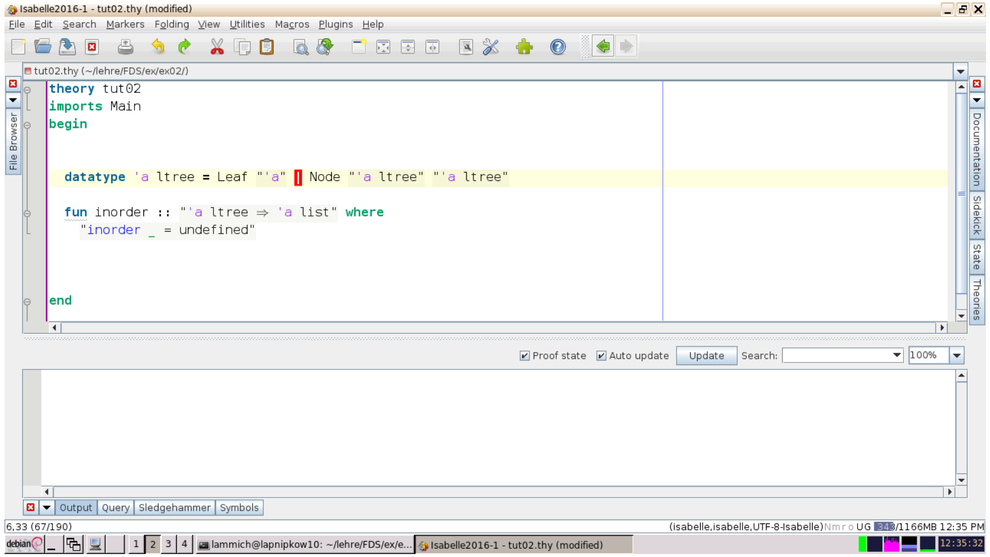

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf 'a | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Output
Query
Sledgehammer
Symbols
5,31 (63/186)
12:31 PM

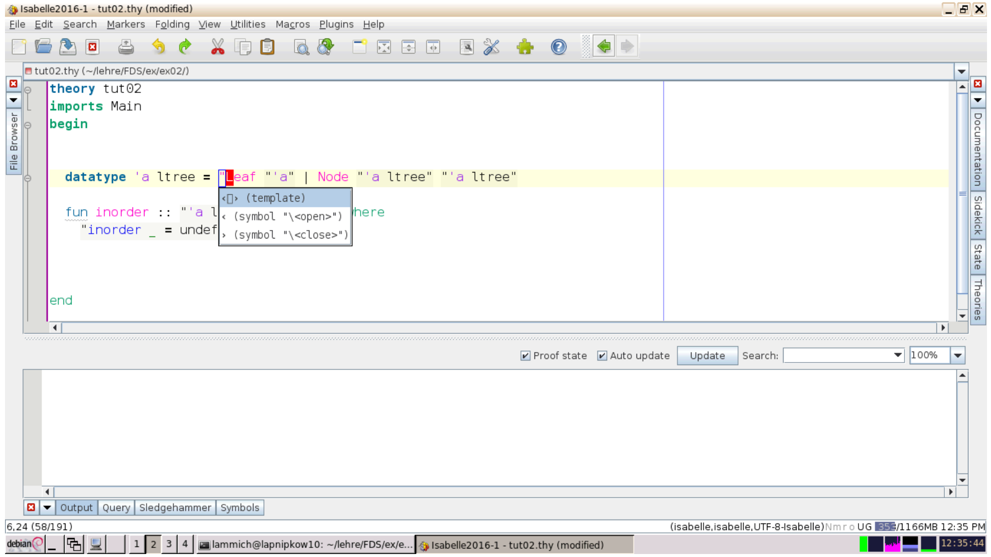

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf 'a | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
5,45 (77/186)
12:32 PM

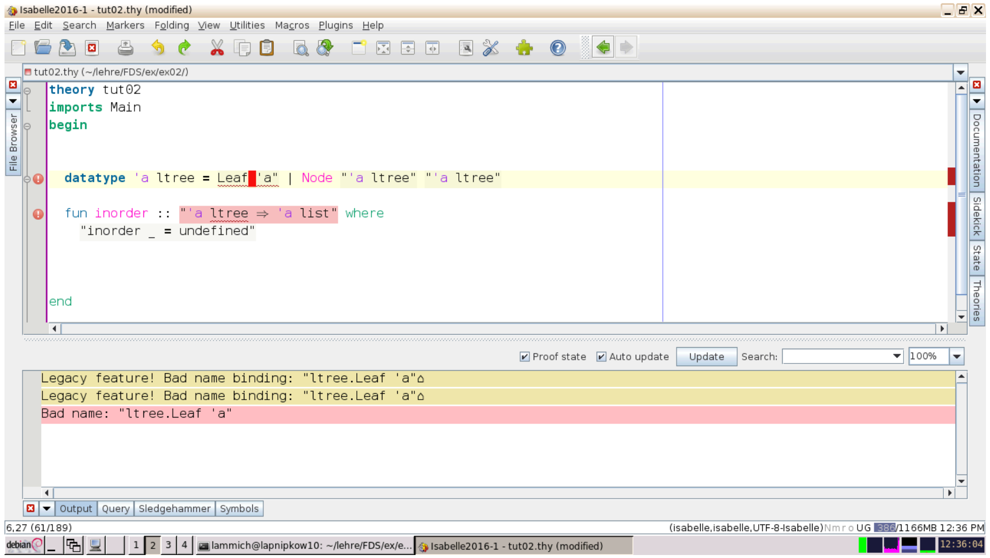

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf 'a | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
5,3 (35/186)
12:32 PM

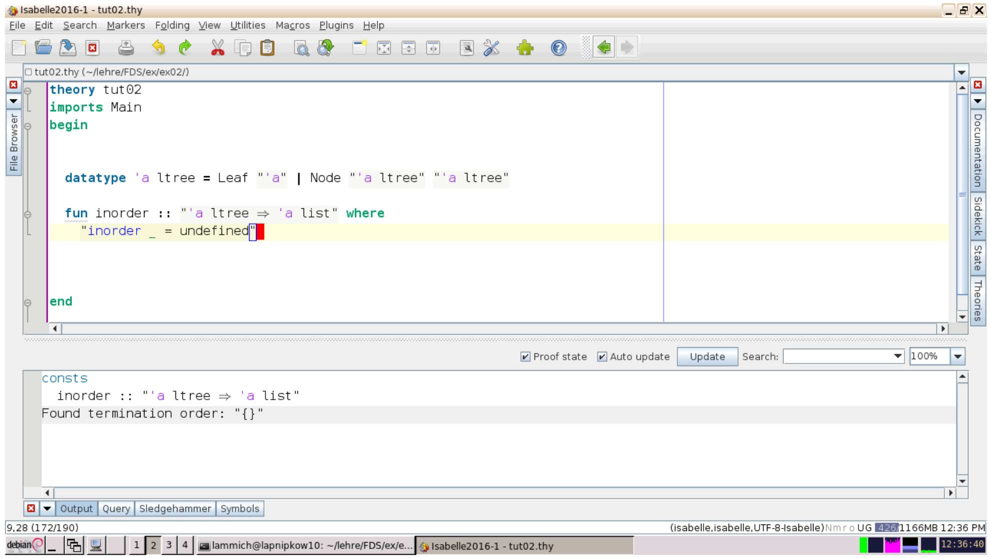

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype foo = "thm" nat
datatype 'a ltree = Leaf 'a | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
5,24 (56/216)
12:34 PM

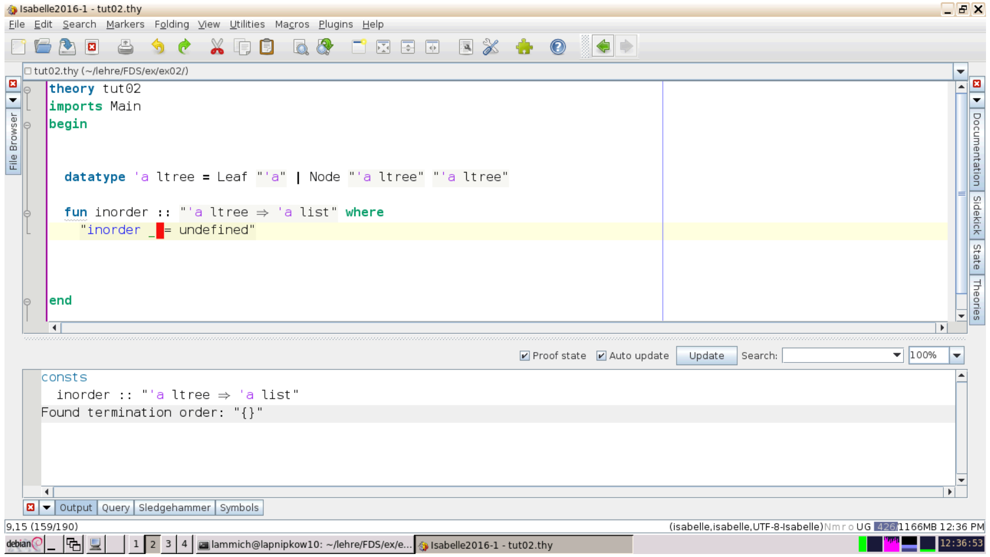

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf 'a | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
6,3 (37/188)
12:34 PM

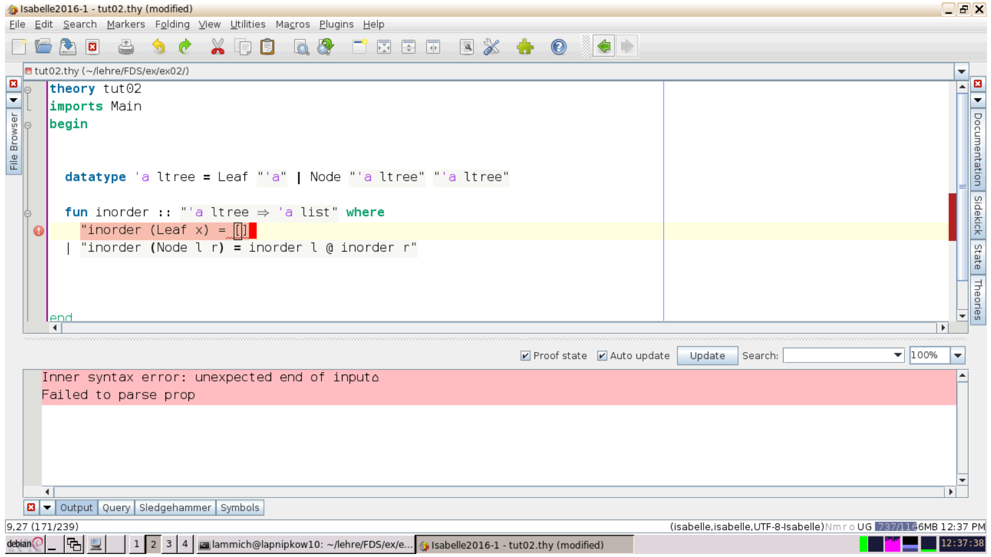

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf "'a" | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
6,33 (67/190)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/ex/e...
Isabelle2016-1 - tut02.thy (modified)
12:35:32

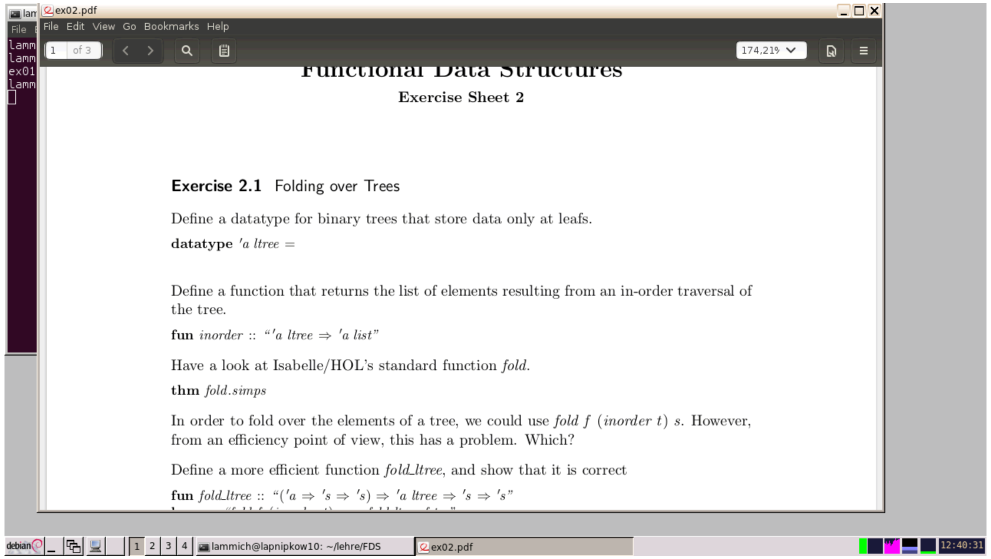

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = "Leaf "'a" | Node "'a ltree" "'a ltree"
(template)
(symbol "\<open>")
(symbol "\<close>")
fun inorder
"inorder _ = undef
here
end
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
6,24 (58/191)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/ex/e...
Isabelle2016-1 - tut02.thy (modified)
12:35:44

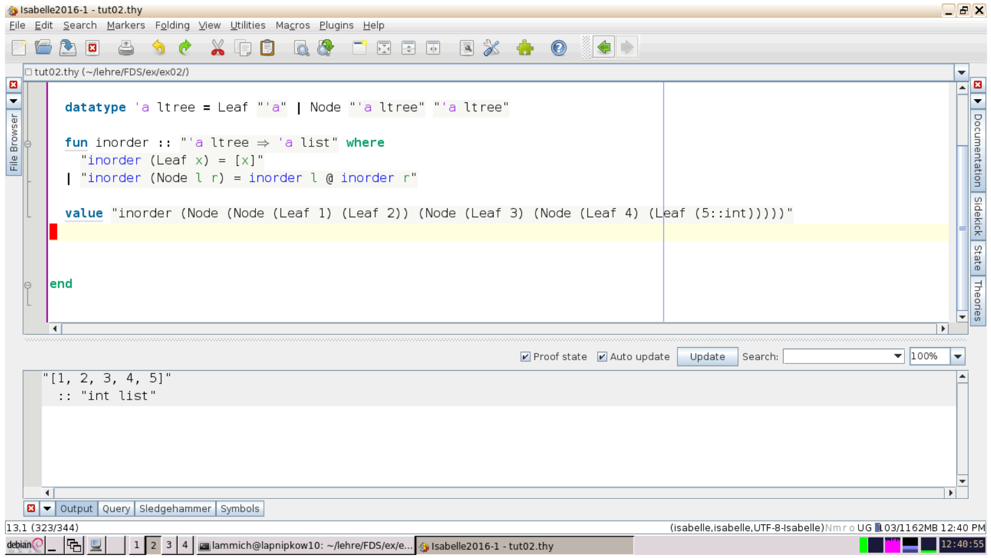

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf"'a" | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
Legacy feature! Bad name binding: "ltree.Leaf 'a"
Legacy feature! Bad name binding: "ltree.Leaf 'a"
Bad name: "ltree.Leaf 'a"
Output
Query
Sledgehammer
Symbols
6,27 (61/189)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/ex/e...
Isabelle2016-1 - tut02.thy (modified)
12:36:04

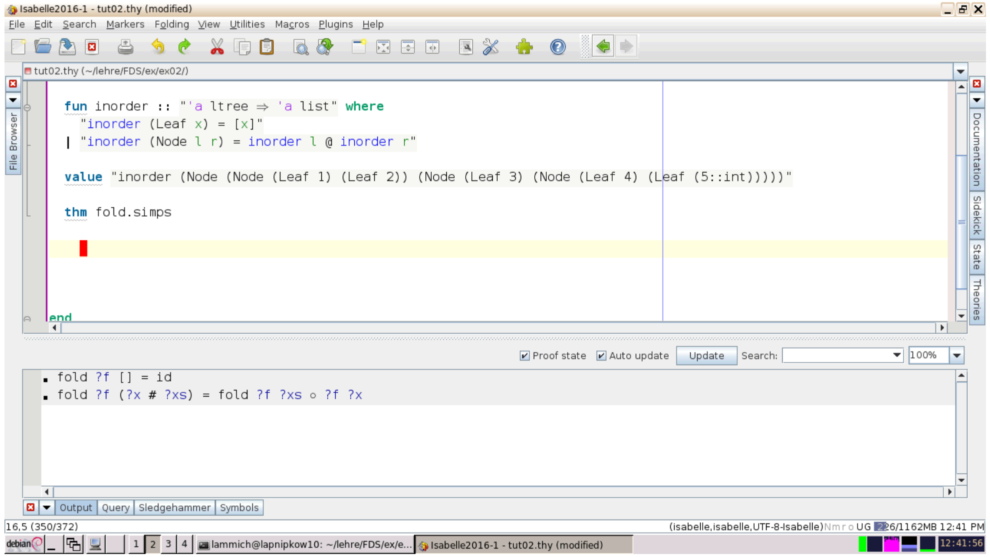

Isabelle2016-1 - tut02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf "'a" | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
Proof state
Auto update
Update
Search:
100%
consts
inorder :: "'a ltree ⇒ 'a list"
Found termination order: "{}"
Output
Query
Sledgehammer
Symbols
9,28 (172/190)
(isabelle,isabelle,UTF-8-Isabelle)
lammich@lapnipkow10: ~/lehre/FDS/ex/e...
Isabelle2016-1 - tut02.thy
12:36:40

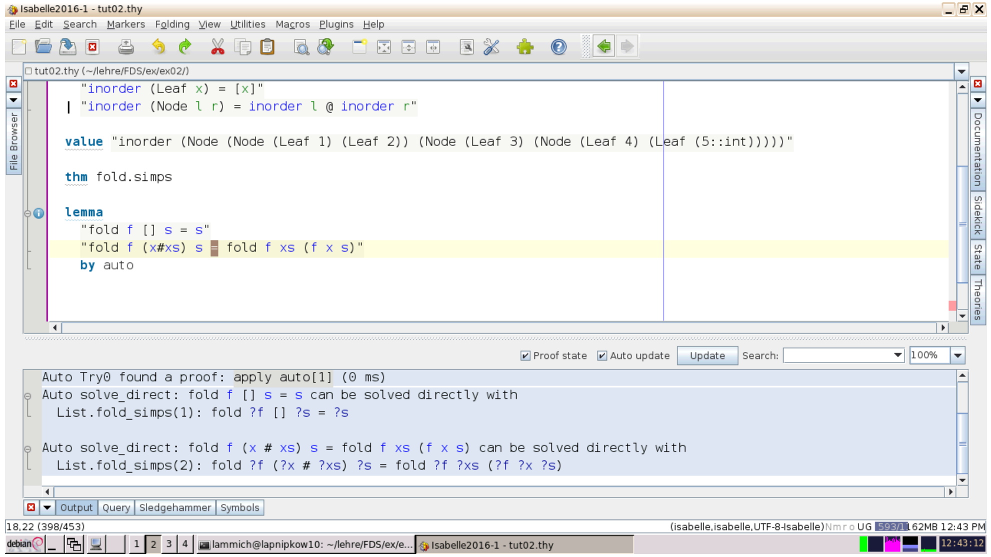

Isabelle2016-1 - tut02.thy
theory tut02
imports Main
begin
datatype 'a ltree = Leaf "'a" | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder _ = undefined"
end
consts
inorder :: "'a ltree ⇒ 'a list"
Found termination order: "{}"

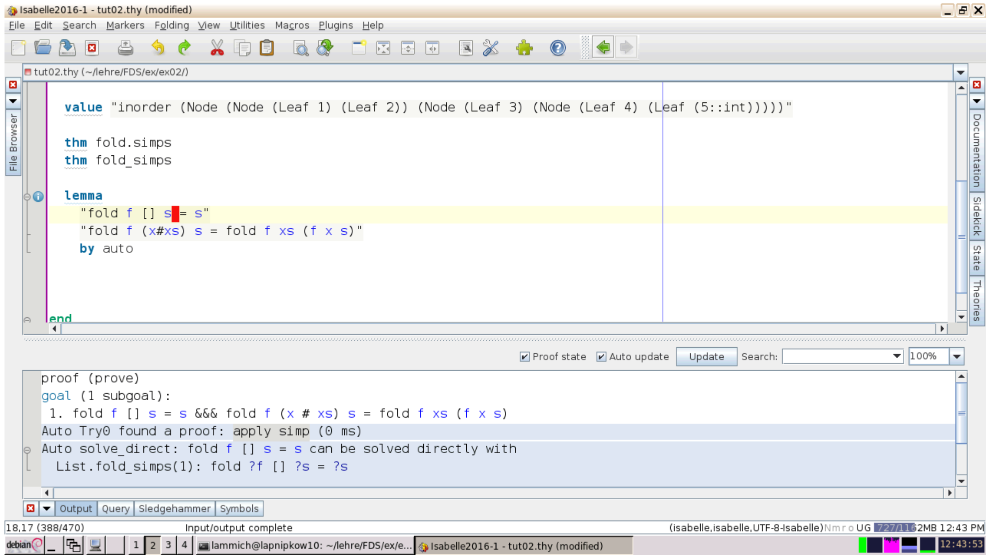

Isabelle2016-1 - tut02.thy (modified)
theory tut02
imports Main
begin
datatype 'a ltree = Leaf "'a" | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder (Leaf x) = []"
| "inorder (Node l r) = inorder l @ inorder r"
Inner syntax error: unexpected end of input
Failed to parse prop

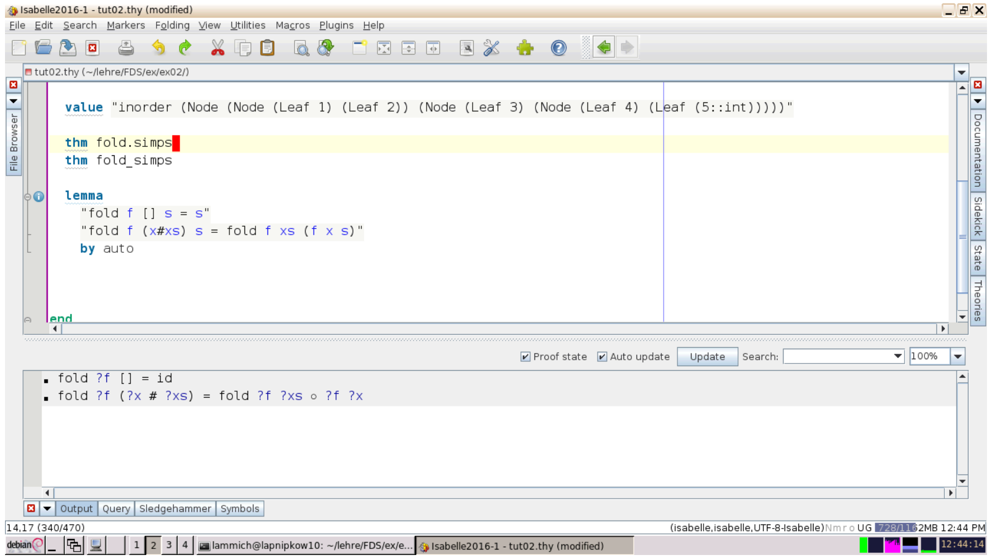

ex02.pdf
Functional Data Structures
Exercise Sheet 2
Exercise 2.1 Folding over Trees
Define a datatype for binary trees that store data only at leafs.
datatype 'a ltree =
Define a function that returns the list of elements resulting from an in-order traversal of the tree.
fun inorder :: "'a ltree ⇒ 'a list"
Have a look at Isabelle/HOL's standard function fold.
thm fold.simps
In order to fold over the elements of a tree, we could use fold f (inorder t) s. However, from an efficiency point of view, this has a problem. Which?
Define a more efficient function fold_ltree, and show that it is correct
fun fold_ltree :: "('a ⇒ 's ⇒ 's) ⇒ 'a ltree ⇒ 's ⇒ 's"

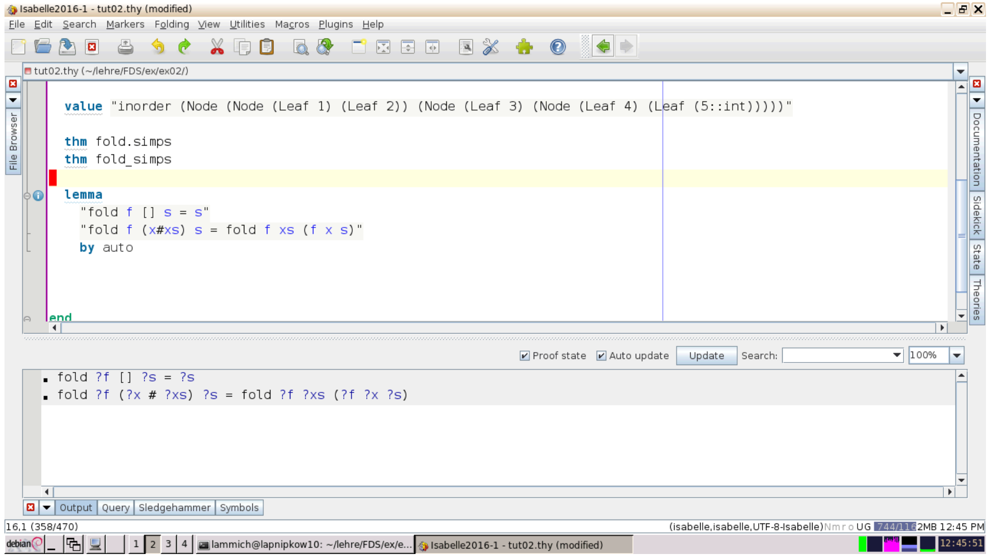

Isabelle2016-1 - tut02.thy
datatype 'a ltree = Leaf "'a" | Node "'a ltree" "'a ltree"
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder (Leaf x) = [x]"
| "inorder (Node l r) = inorder l @ inorder r"
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
end
"[1, 2, 3, 4, 5]"
:: "int list"

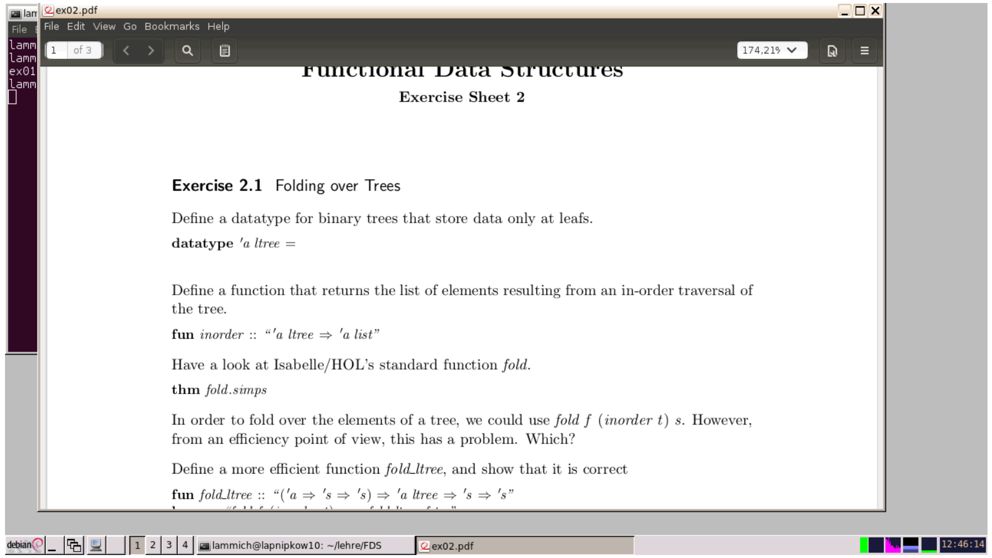

Isabelle2016-1 - tut02.thy (modified)
fun inorder :: "'a ltree ⇒ 'a list" where
"inorder (Leaf x) = [x]"
| "inorder (Node l r) = inorder l @ inorder r"
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
thm fold.simps
end
fold ?f [] = id
fold ?f (?x # ?xs) = fold ?f ?xs ∘ ?f ?x

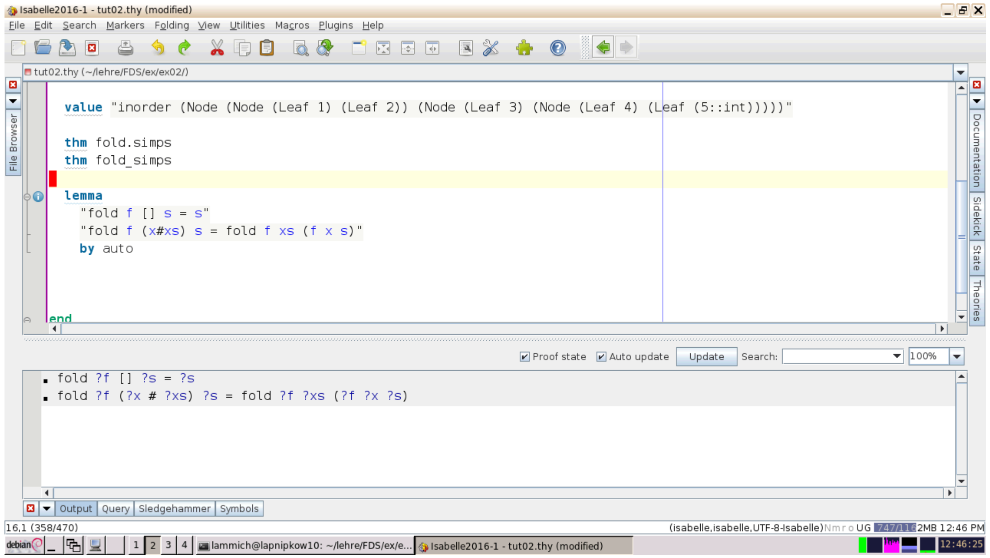

Isabelle2016-1 - tut02.thy
"inorder (Leaf x) = [x]"
| "inorder (Node l r) = inorder l @ inorder r"
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
thm fold.simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
Auto Try0 found a proof: apply auto[1] (0 ms)
Auto solve_direct: fold f [] s = s can be solved directly with
List.fold_simps(1): fold ?f [] ?s = ?s
Auto solve_direct: fold f (x # xs) s = fold f xs (f x s) can be solved directly with
List.fold_simps(2): fold ?f (?x # ?xs) ?s = fold ?f ?xs (?f ?x ?s)

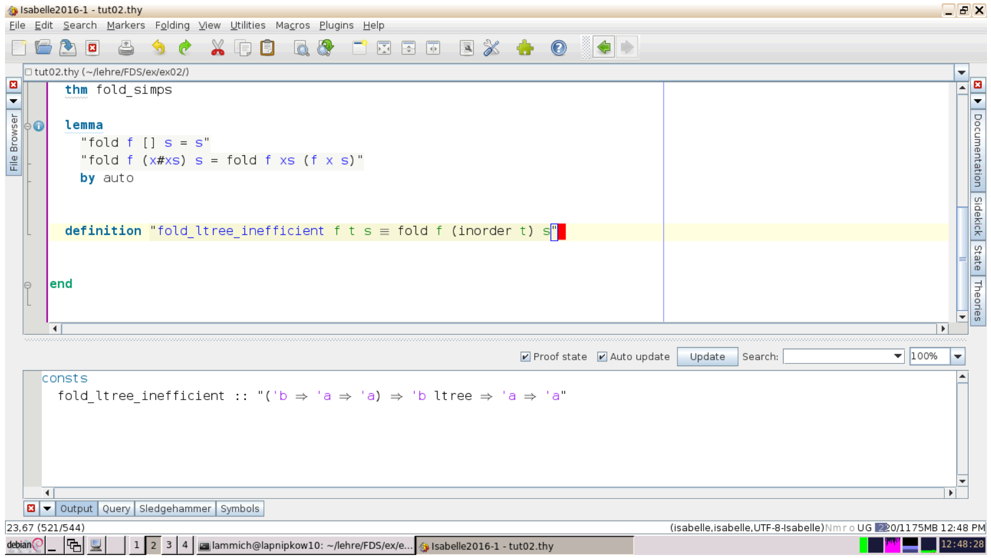

Isabelle2016-1 - tut02.thy (modified)
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
thm fold.simps
thm fold_simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
end
proof (prove)
goal (1 subgoal):
1. fold f [] s = s &&& fold f (x # xs) s = fold f xs (f x s)
Auto Try0 found a proof: apply simp (0 ms)
Auto solve_direct: fold f [] s = s can be solved directly with
List.fold_simps(1): fold ?f [] ?s = ?s

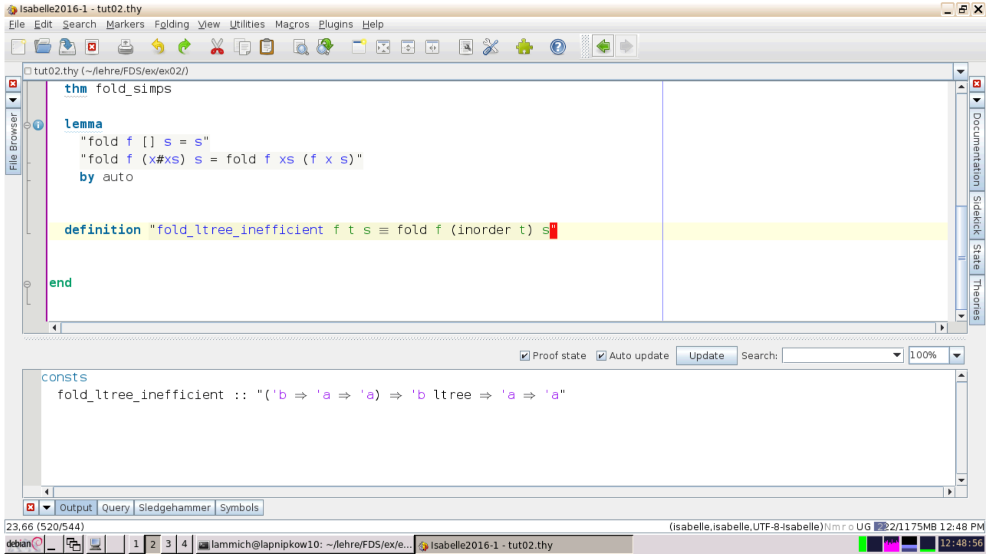

Isabelle2016-1 - tut02.thy (modified)
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
thm fold.simps
thm fold_simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
end
fold ?f [] = id
fold ?f (?x # ?xs) = fold ?f ?xs ∘ ?f ?x

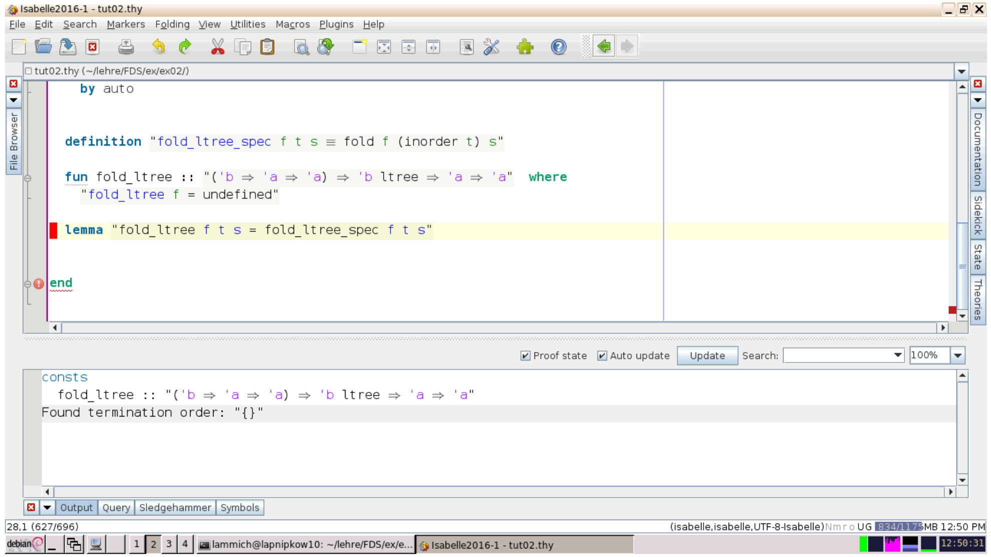

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
thm fold.simps
thm fold_simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
end
Proof state
Auto update
Update
Search:
100%
fold ?f [] ?s = ?s
fold ?f (?x # ?xs) ?s = fold ?f ?xs (?f ?x ?s)
Output
Query
Sledgehammer
Symbols
16,1 (358/470)
12:45 PM

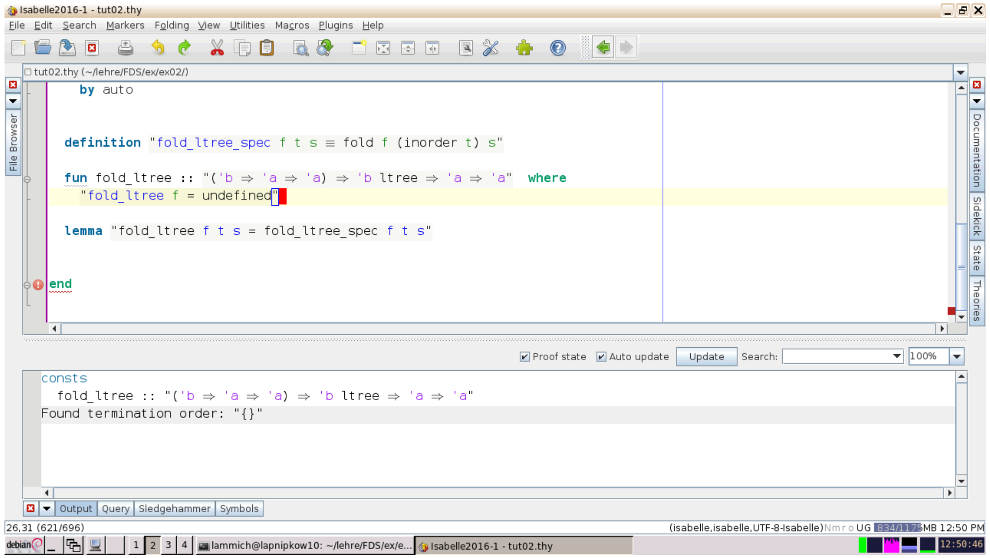

ex02.pdf
Functional Data Structures
Exercise Sheet 2
Exercise 2.1 Folding over Trees
Define a datatype for binary trees that store data only at leafs.
datatype 'a ltree =
Define a function that returns the list of elements resulting from an in-order traversal of the tree.
fun inorder :: "'a ltree ⇒ 'a list"
Have a look at Isabelle/HOL's standard function fold.
thm fold.simps
In order to fold over the elements of a tree, we could use fold f (inorder t) s. However, from an efficiency point of view, this has a problem. Which?
Define a more efficient function fold_ltree, and show that it is correct
fun fold_ltree :: "('a ⇒ 's ⇒ 's) ⇒ 'a ltree ⇒ 's ⇒ 's"
12:46:14

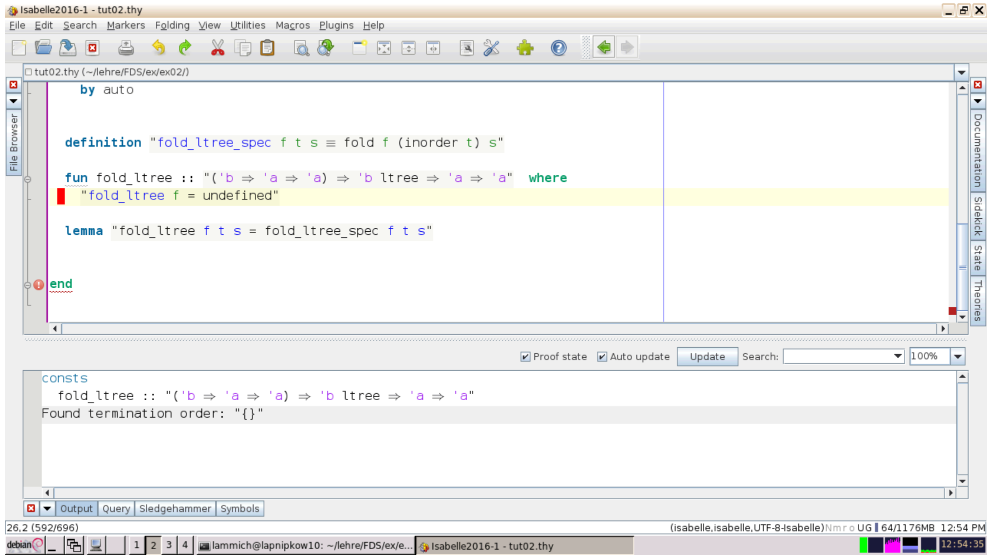

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
value "inorder (Node (Node (Leaf 1) (Leaf 2)) (Node (Leaf 3) (Node (Leaf 4) (Leaf (5::int)))))"
thm fold.simps
thm fold_simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
end
fold ?f [] ?s = ?s
fold ?f (?x # ?xs) ?s = fold ?f ?xs (?f ?x ?s)
16,1 (358/470)
12:46 PM

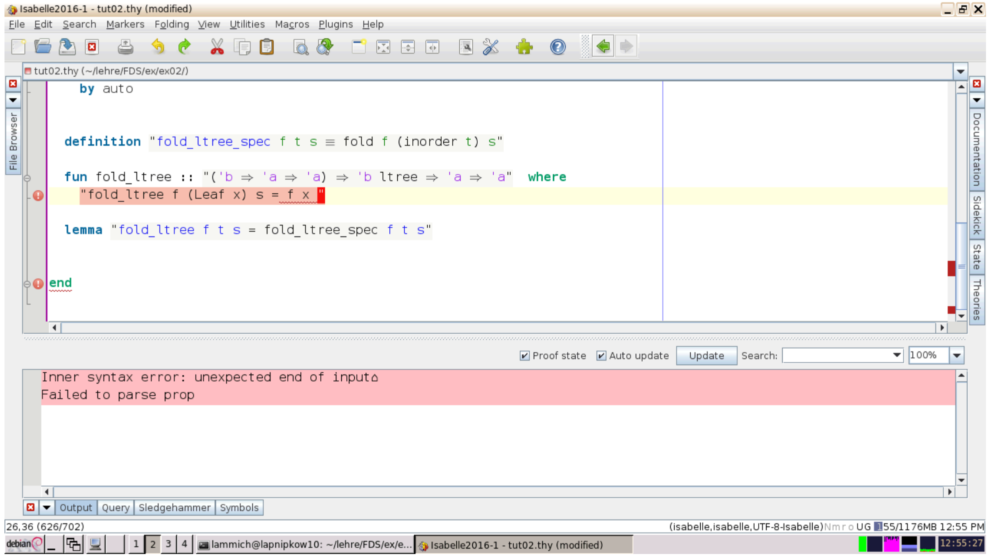

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
thm fold_simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
definition "fold_ltree_inefficient f t s ≡ fold f (inorder t) s"
end
consts
fold_ltree_inefficient :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a"
23,67 (521/544)
12:48 PM

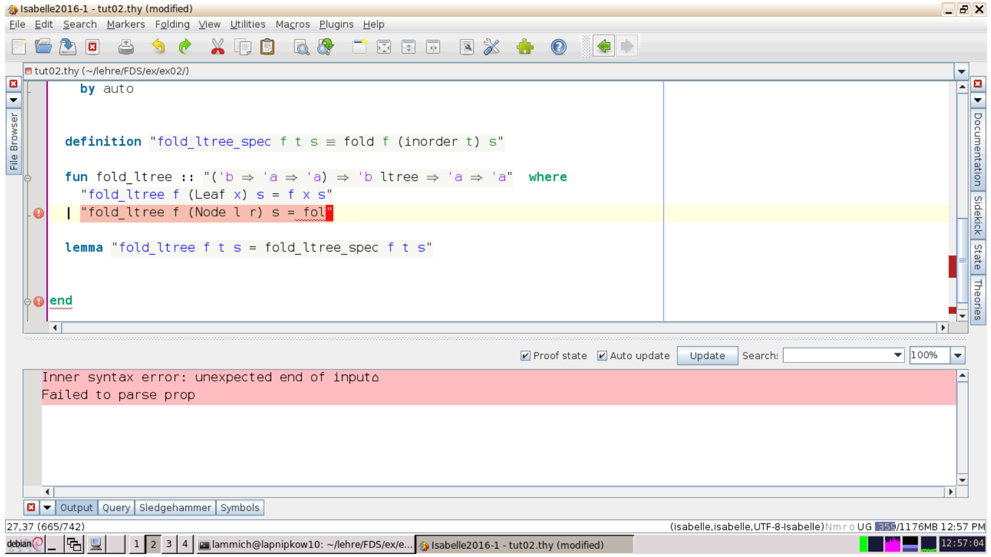

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
thm fold_simps
lemma
"fold f [] s = s"
"fold f (x#xs) s = fold f xs (f x s)"
by auto
definition "fold_ltree_inefficient f t s ≡ fold f (inorder t) s"
end
Proof state
Auto update
Update
Search:
100%
consts
fold_ltree_inefficient :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a"
Output
Query
Sledgehammer
Symbols
23.66 (520/544)
(isabelle,isabelle,UTF-8-Isabelle)
12:48 PM

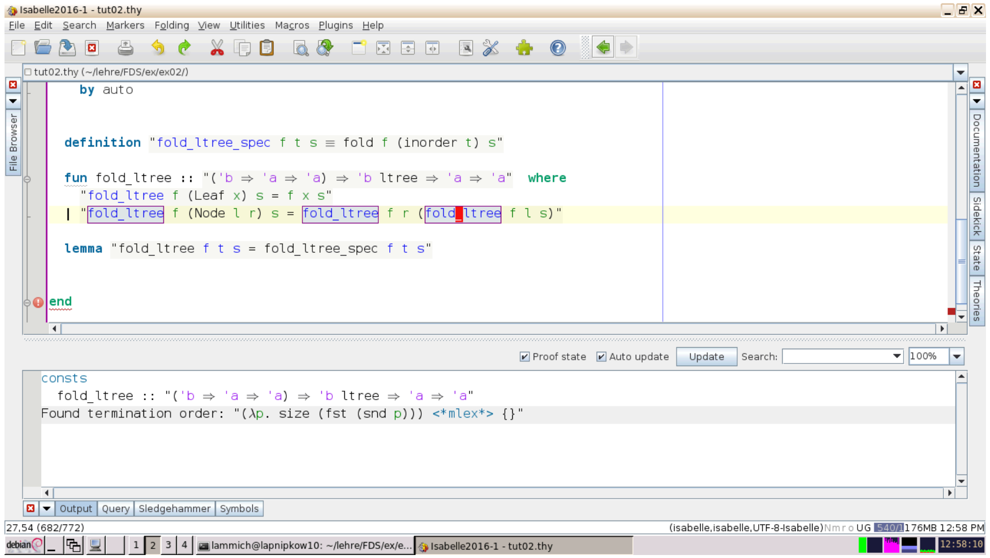

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
by auto
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f = undefined"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
end
Proof state
Auto update
Update
Search:
100%
consts
fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a"
Found termination order: "{}"
Output
Query
Sledgehammer
Symbols
28.1 (627/696)
(isabelle,isabelle,UTF-8-Isabelle)
12:50 PM

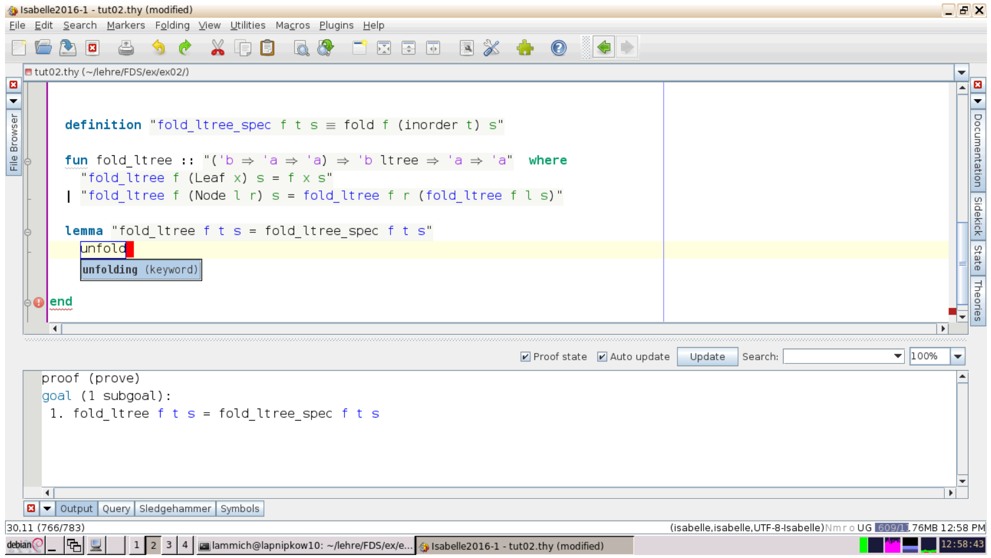

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
by auto
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f = undefined"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
end
Proof state
Auto update
Update
Search:
100%
consts
fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a"
Found termination order: "{}"
Output
Query
Sledgehammer
Symbols
26.31 (621/696)
(isabelle,isabelle,UTF-8-Isabelle)
12:50 PM

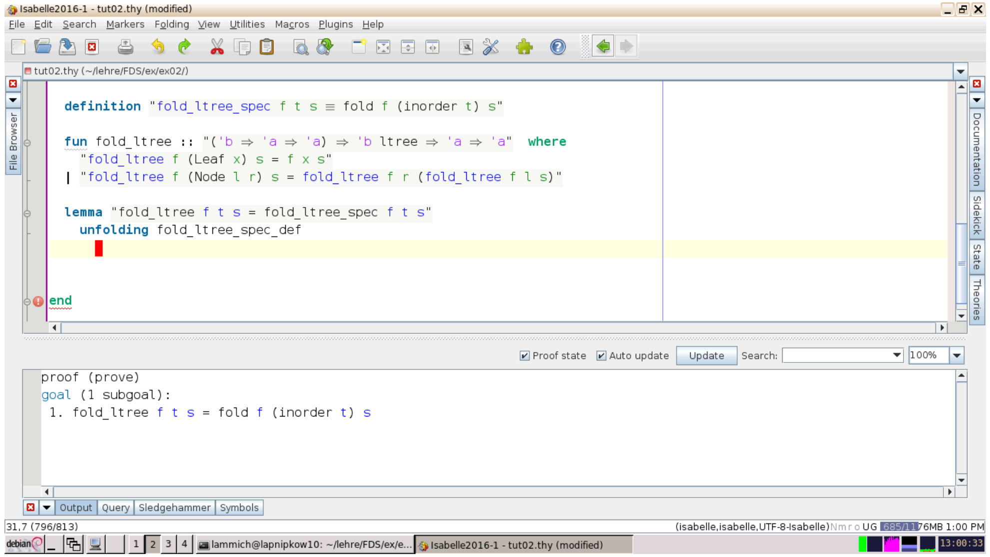

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
by auto
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f = undefined"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
end
Proof state
Auto update
Update
Search:
100%
consts
fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a"
Found termination order: "{}"
Output
Query
Sledgehammer
Symbols
26.2 (592/696)
(isabelle,isabelle,UTF-8-Isabelle)
12:54 PM

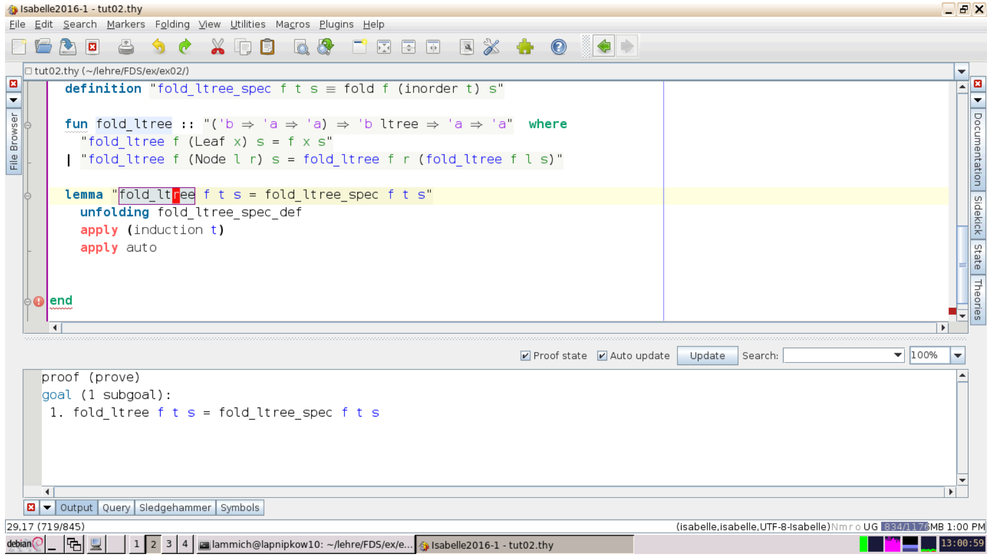

Isabelle2016-1 - tut02.thy (modified)
by auto
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x "
lemma "fold_ltree f t s = fold_ltree_spec f t s"
end
Inner syntax error: unexpected end of input
Failed to parse prop

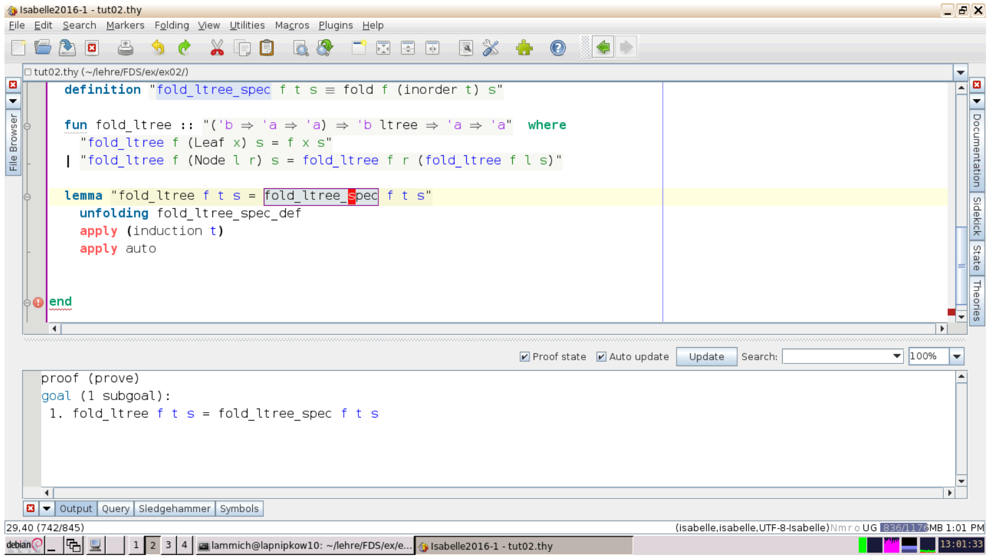

Isabelle2016-1 - tut02.thy (modified)
by auto
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fol"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
end
Inner syntax error: unexpected end of input
Failed to parse prop

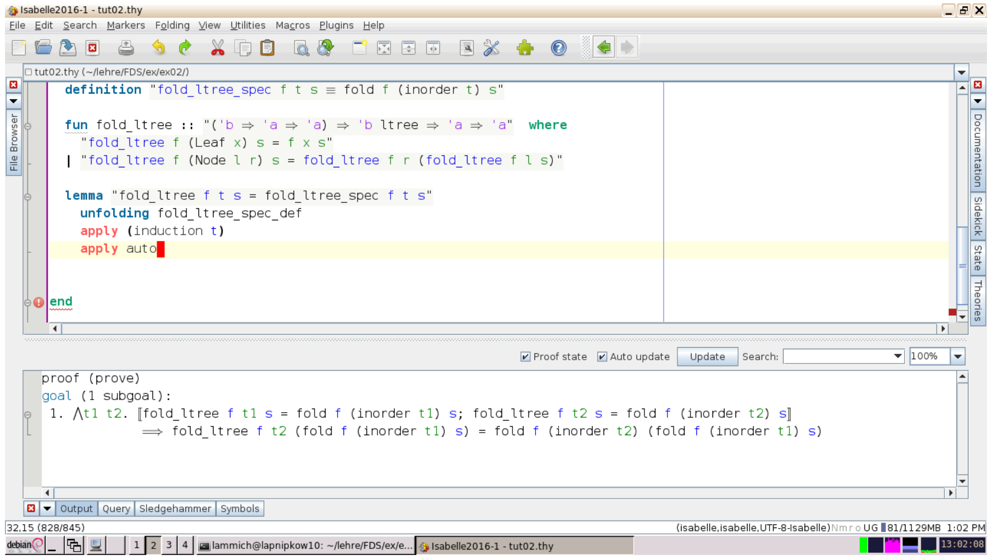

Isabelle2016-1 - tut02.thy
by auto
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
end
consts
fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a"
Found termination order: "(λp. size (fst (snd p))) <*mlex*> {}"

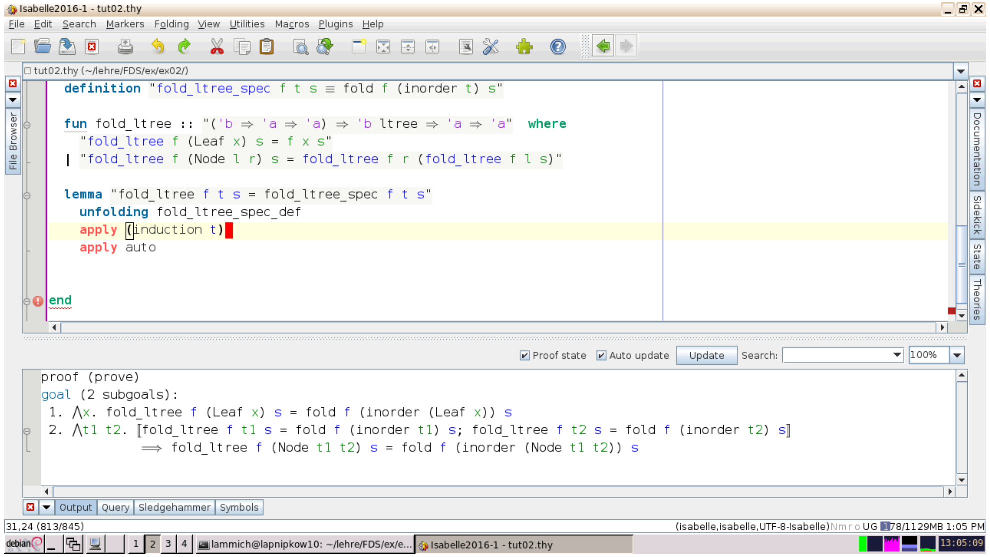

Isabelle2016-1 - tut02.thy (modified)
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfold
unfolding (keyword)
end
proof (prove)
goal (1 subgoal):
1. fold_ltree f t s = fold_ltree_spec f t s

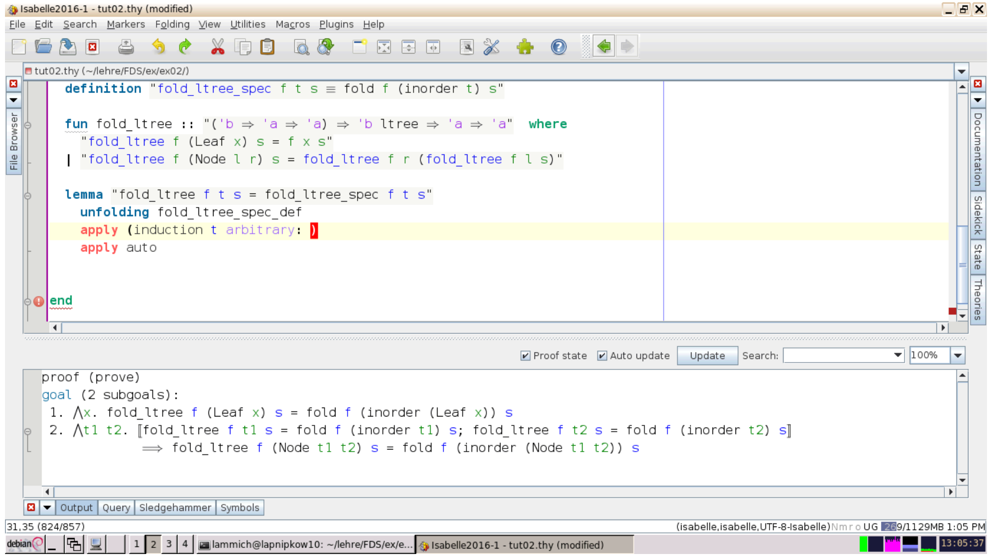

Isabelle2016-1 - tut02.thy (modified)
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
end
proof (prove)
goal (1 subgoal):
1. fold_ltree f t s = fold f (inorder t) s

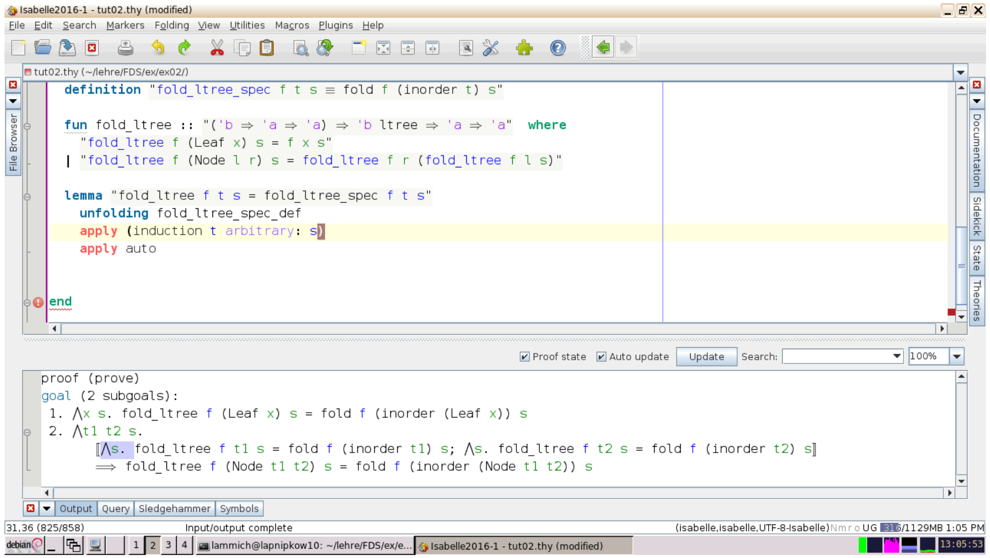

Isabelle2016-1 - tut02.thy
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t)
apply auto
end
proof (prove)
goal (1 subgoal):
1. fold_ltree f t s = fold_ltree_spec f t s

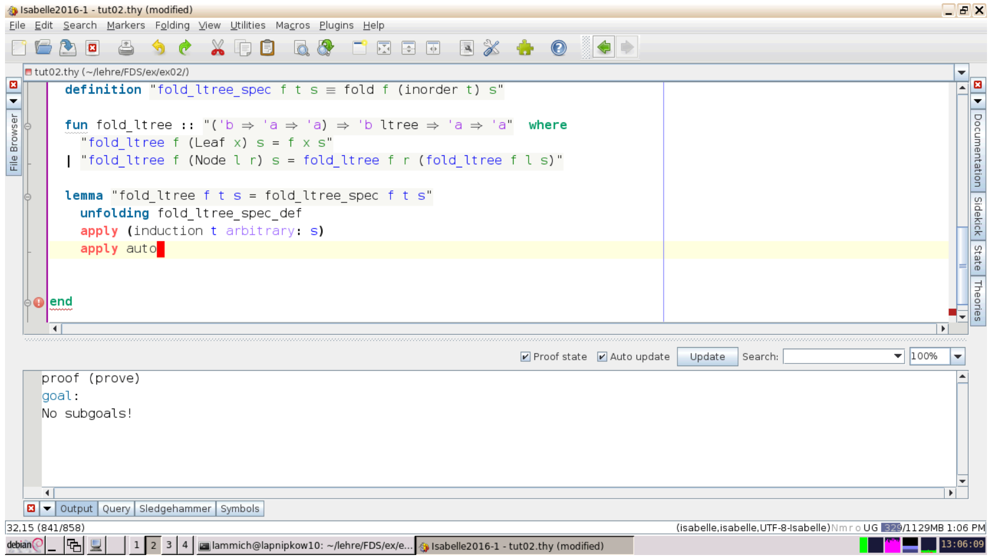

Isabelle2016-1 - tut02.thy
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t)
apply auto
end
proof (prove)
goal (1 subgoal):
1. fold_ltree f t s = fold_ltree_spec f t s

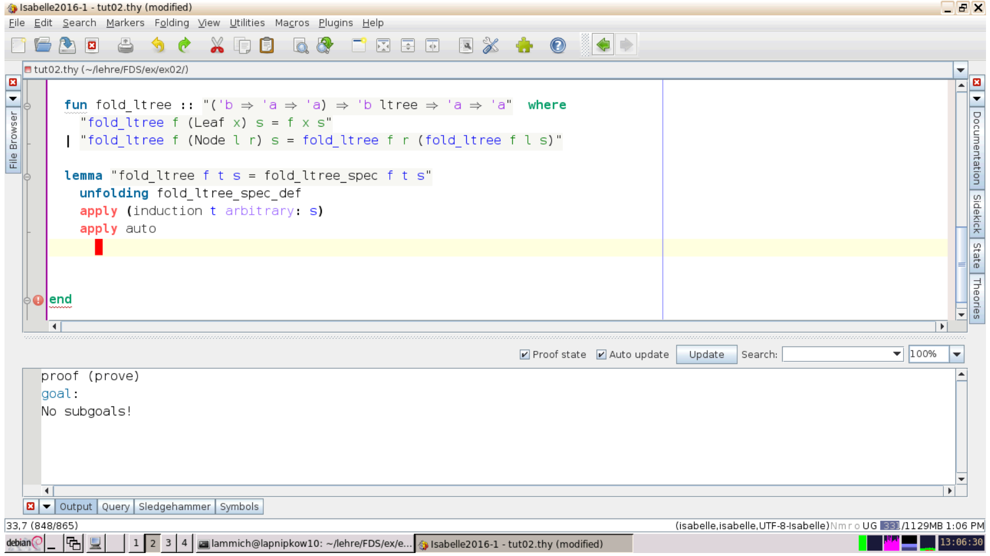

Isabelle2016-1 - tut02.thy
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t)
apply auto
end
proof (prove)
goal (1 subgoal):
1. ⋀t1 t2. ⟦fold_ltree f t1 s = fold f (inorder t1) s; fold_ltree f t2 s = fold f (inorder t2) s⟧
⟹ fold_ltree f t2 (fold f (inorder t1) s) = fold f (inorder t2) (fold f (inorder t1) s)

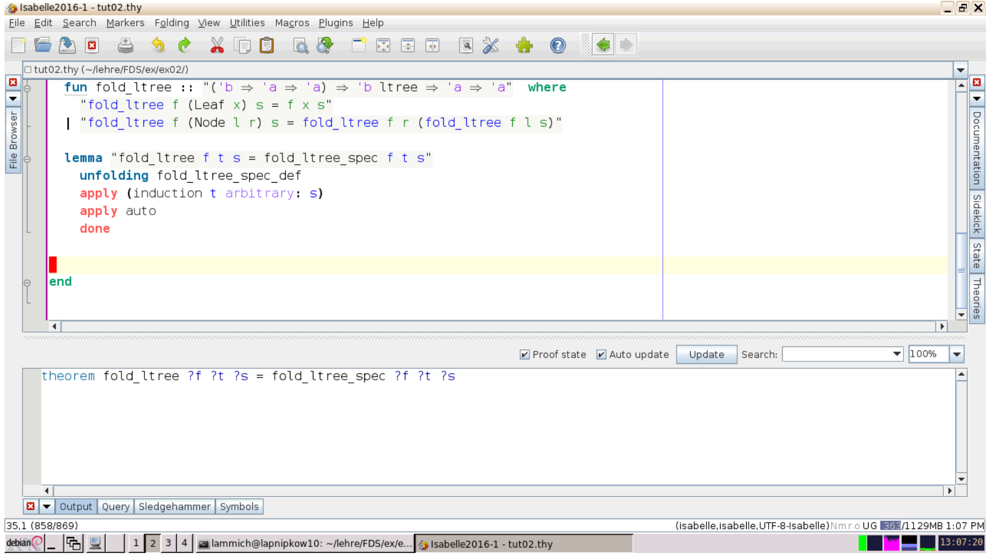

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t)
apply auto
end
proof (prove)
goal (2 subgoals):
1. ⋀x. fold_ltree f (Leaf x) s = fold f (inorder (Leaf x)) s
2. ⋀t1 t2. ⟦fold_ltree f t1 s = fold f (inorder t1) s; fold_ltree f t2 s = fold f (inorder t2) s⟧
⟹ fold_ltree f (Node t1 t2) s = fold f (inorder (Node t1 t2)) s

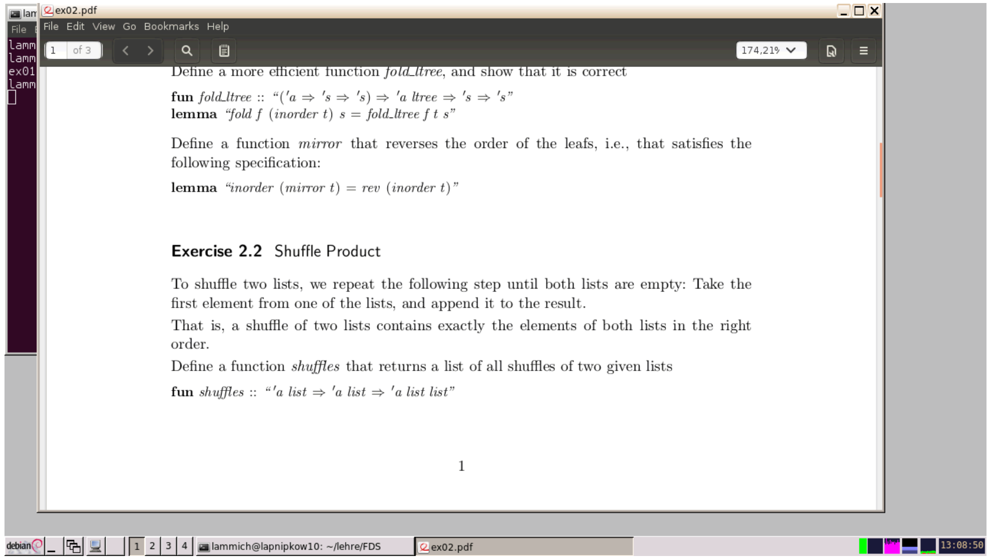

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary:)
apply auto
end
proof (prove)
goal (2 subgoals):
1. ⋀x. fold_ltree f (Leaf x) s = fold f (inorder (Leaf x)) s
2. ⋀t1 t2. ⟦fold_ltree f t1 s = fold f (inorder t1) s; fold_ltree f t2 s = fold f (inorder t2) s⟧
⟹ fold_ltree f (Node t1 t2) s = fold f (inorder (Node t1 t2)) s

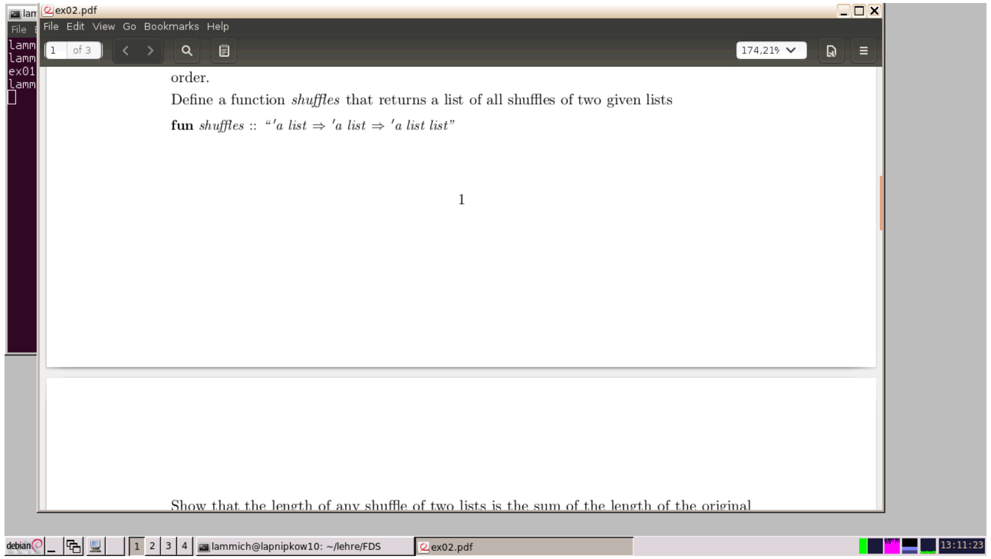

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary: s)
apply auto
end
proof (prove)
goal (2 subgoals):
1. ⋀x s. fold_ltree f (Leaf x) s = fold f (inorder (Leaf x)) s
2. ⋀t1 t2 s.
⟦⋀s. fold_ltree f t1 s = fold f (inorder t1) s; ⋀s. fold_ltree f t2 s = fold f (inorder t2) s⟧
⟹ fold_ltree f (Node t1 t2) s = fold f (inorder (Node t1 t2)) s
Input/output complete

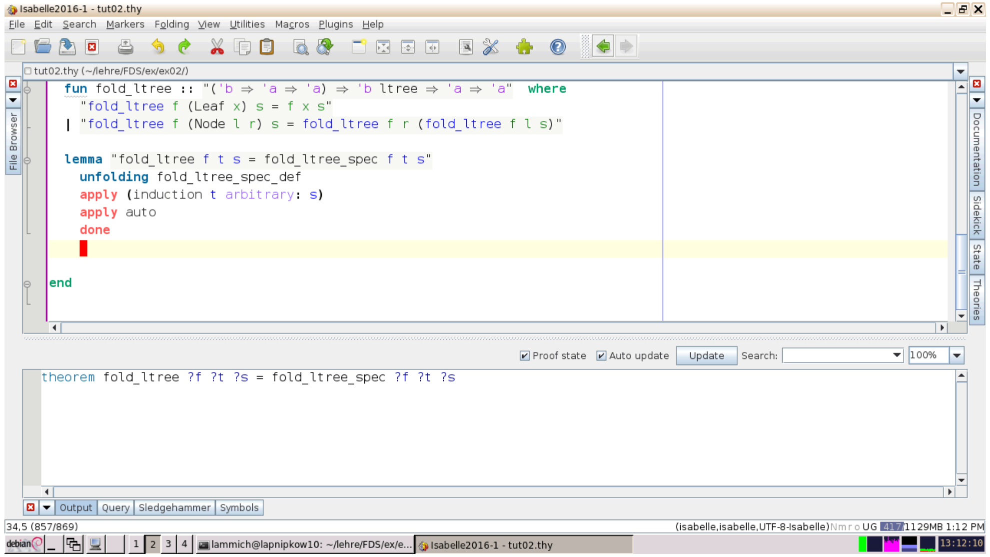

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
definition "fold_ltree_spec f t s ≡ fold f (inorder t) s"
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary: s)
apply auto
end
proof (prove)
goal:
No subgoals!

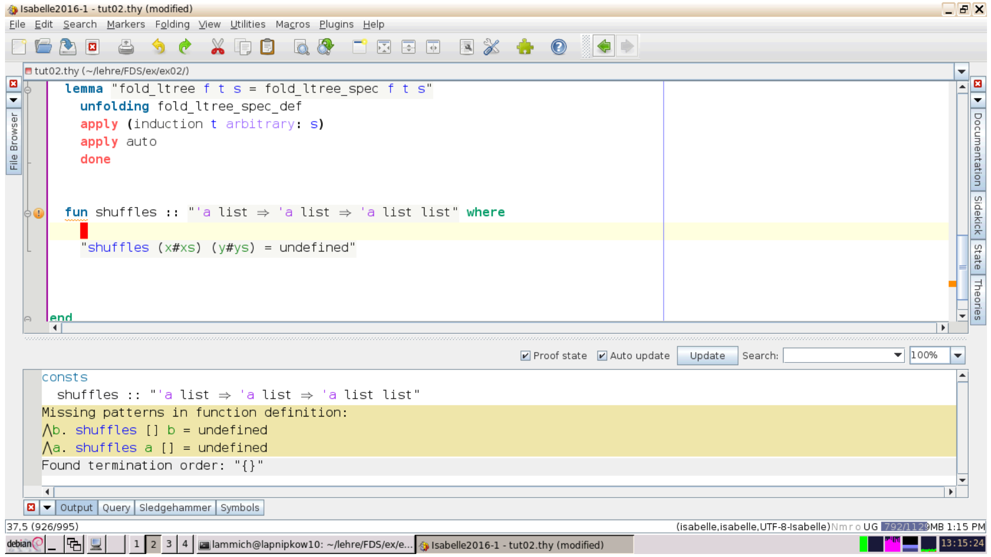

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary: s)
apply auto
end
Proof state Auto update Update Search: 100%
proof (prove)
goal:
No subgoals!
Output Query Sledgehammer Symbols
33,7 (848/865)
(isabelle,isabelle,UTF-8-Isabelle) 1129MB 1:06 PM

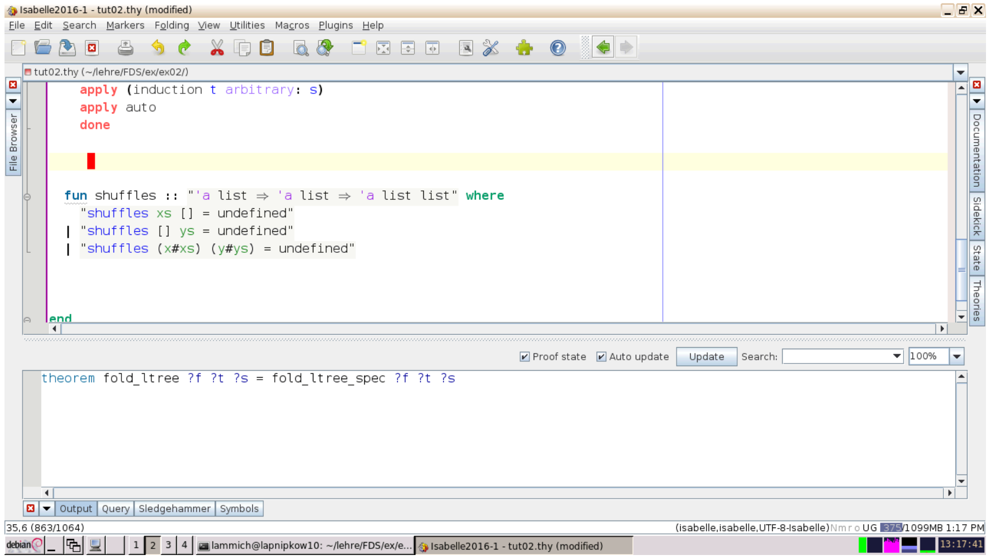

Isabelle2016-1 - tut02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary: s)
apply auto
done
end
Proof state Auto update Update Search: 100%
theorem fold_ltree ?f ?t ?s = fold_ltree_spec ?f ?t ?s
Output Query Sledgehammer Symbols
35,1 (858/869)
(isabelle,isabelle,UTF-8-Isabelle) 1129MB 1:07 PM

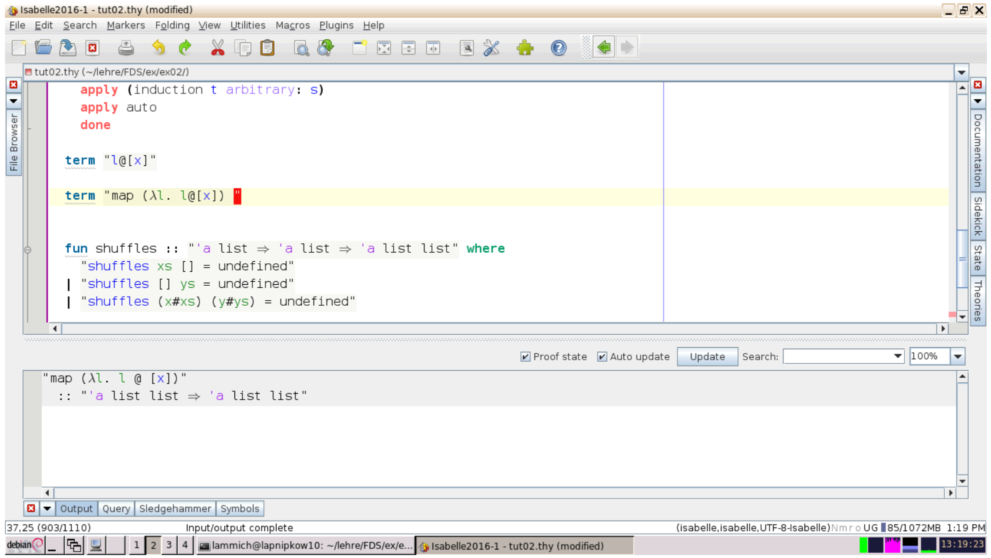

ex02.pdf
File Edit View Go Bookmarks Help
1 of 3
174,21%
Define a more efficient function fold_ltree, and show that it is correct
fun fold_ltree :: "('a ⇒ 's ⇒ 's) ⇒ 'a ltree ⇒ 's ⇒ 's"
lemma "fold f (inorder t) s = fold_ltree f t s"
Define a function mirror that reverses the order of the leafs, i.e., that satisfies the following specification:
lemma "inorder (mirror t) = rev (inorder t)"
Exercise 2.2 Shuffle Product
To shuffle two lists, we repeat the following step until both lists are empty: Take the first element from one of the lists, and append it to the result.
That is, a shuffle of two lists contains exactly the elements of both lists in the right order.
Define a function shuffles that returns a list of all shuffles of two given lists
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
1

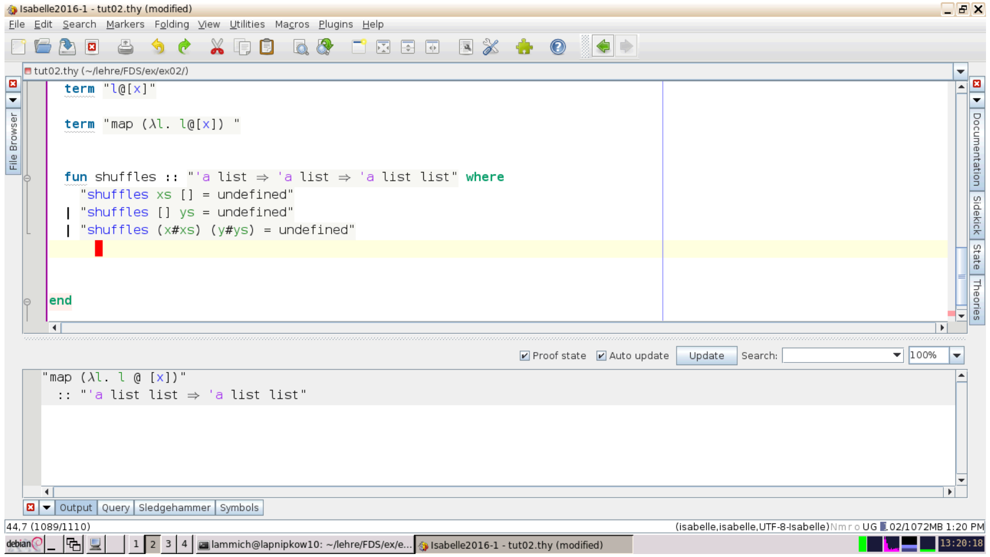

ex02.pdf
File Edit View Go Bookmarks Help
1 of 3
174,21%
order.
Define a function shuffles that returns a list of all shuffles of two given lists
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
1
Show that the length of any shuffle of two lists is the sum of the length of the original

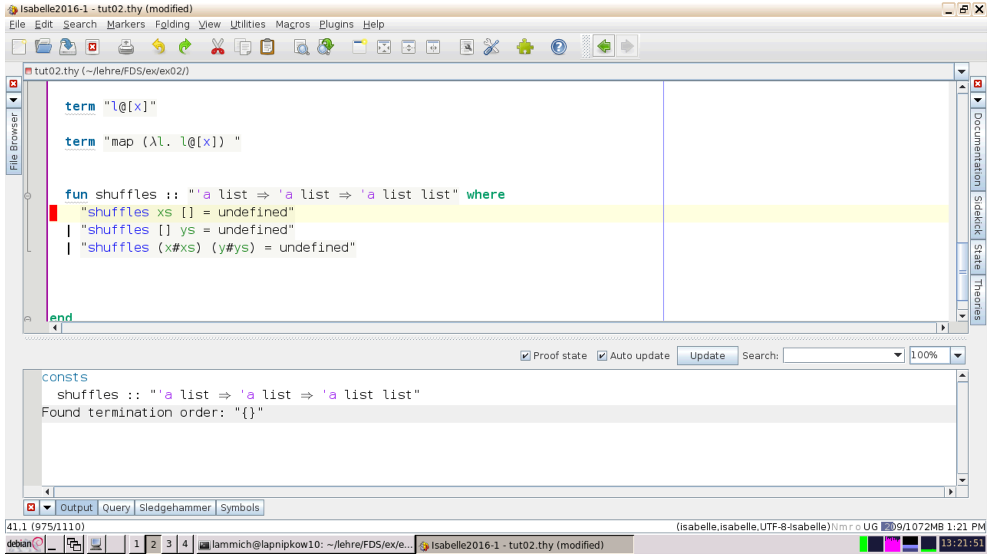

Isabelle2016-1 - tut02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
fun fold_ltree :: "('b ⇒ 'a ⇒ 'a) ⇒ 'b ltree ⇒ 'a ⇒ 'a" where
"fold_ltree f (Leaf x) s = f x s"
| "fold_ltree f (Node l r) s = fold_ltree f r (fold_ltree f l s)"
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary: s)
apply auto
done
end
Proof state Auto update Update Search: 100%
theorem fold_ltree ?f ?t ?s = fold_ltree_spec ?f ?t ?s
Output Query Sledgehammer Symbols
34,5 (857/869)
(isabelle,isabelle,UTF-8-Isabelle) UG 1129MB 1:12 PM

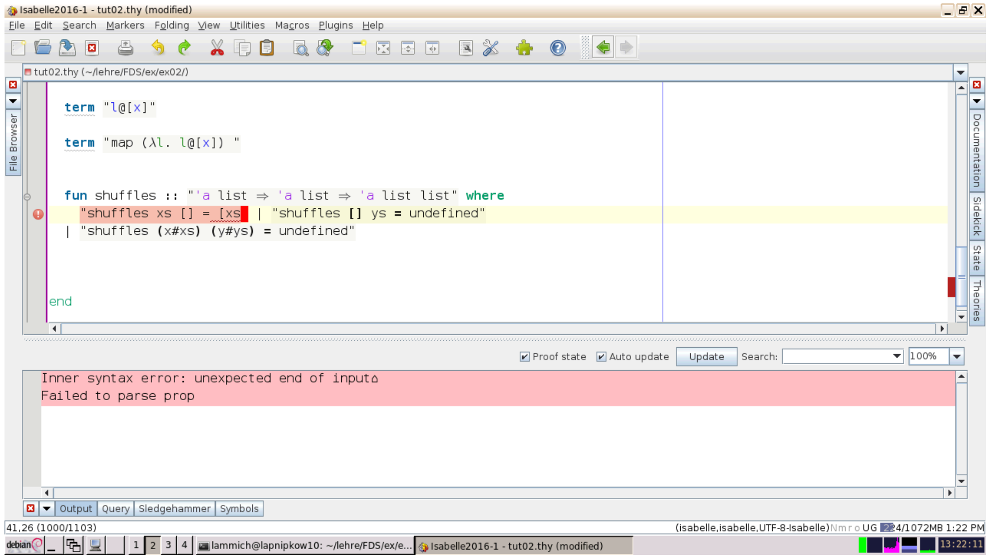

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
lemma "fold_ltree f t s = fold_ltree_spec f t s"
unfolding fold_ltree_spec_def
apply (induction t arbitrary: s)
apply auto
done
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles (x#xs) (y#ys) = undefined"
end
Proof state Auto update Update Search: 100%
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Missing patterns in function definition:
⋀b. shuffles [] b = undefined
⋀a. shuffles a [] = undefined
Found termination order: "{}"
Output Query Sledgehammer Symbols
37,5 (926/995)
(isabelle,isabelle,UTF-8-Isabelle) UG 1:15 PM

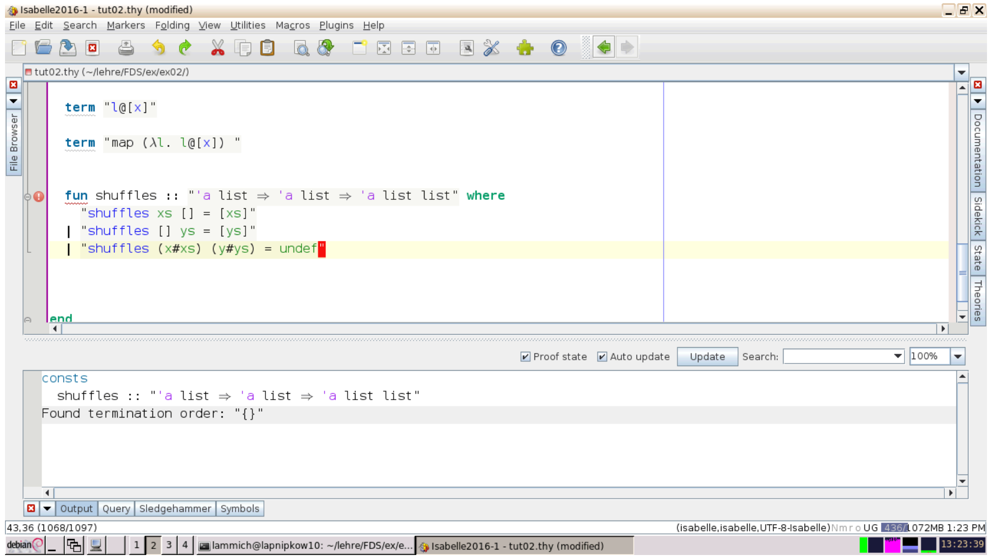

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
apply (induction t arbitrary: s)
apply auto
done
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = undefined"
| "shuffles [] ys = undefined"
| "shuffles (x#xs) (y#ys) = undefined"
end
Proof state Auto update Update Search: 100%
theorem fold_ltree ?f ?t ?s = fold_ltree_spec ?f ?t ?s
Output Query Sledgehammer Symbols
35,6 (863/1064)
(isabelle,isabelle,UTF-8-Isabelle) UG 1099MB 1:17 PM

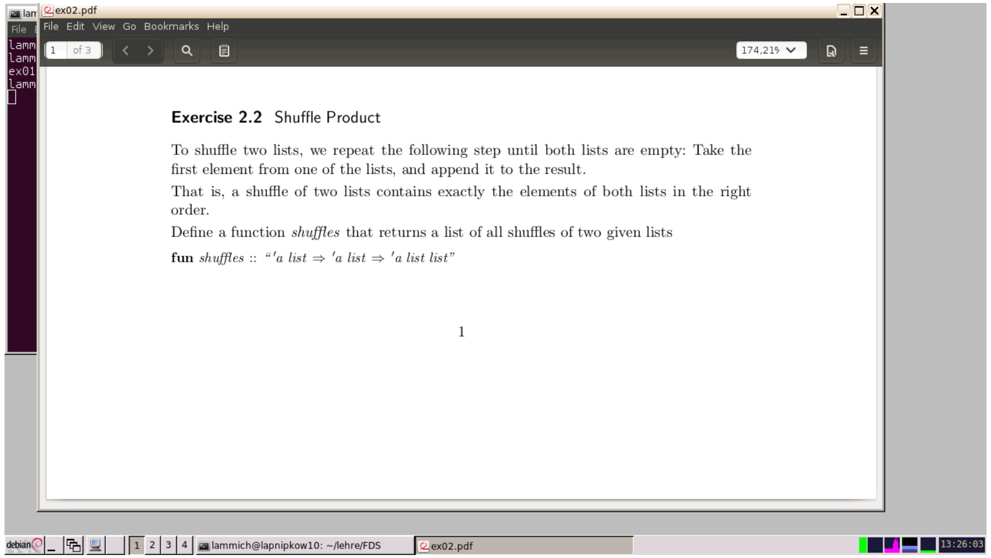

Isabelle2016-1 - tut02.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
apply (induction t arbitrary: s)
apply auto
done
term "l@[x]"
term "map (λl. l@[x])"
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = undefined"
| "shuffles [] ys = undefined"
| "shuffles (x#xs) (y#ys) = undefined"
Proof state Auto update Update Search: 100%
"map (λl. l @ [x])"
:: "'a list list ⇒ 'a list list"
Output Query Sledgehammer Symbols
37,25 (903/1110) Input/output complete
(isabelle,isabelle,UTF-8-Isabelle) UG 85/1072MB 1:19 PM

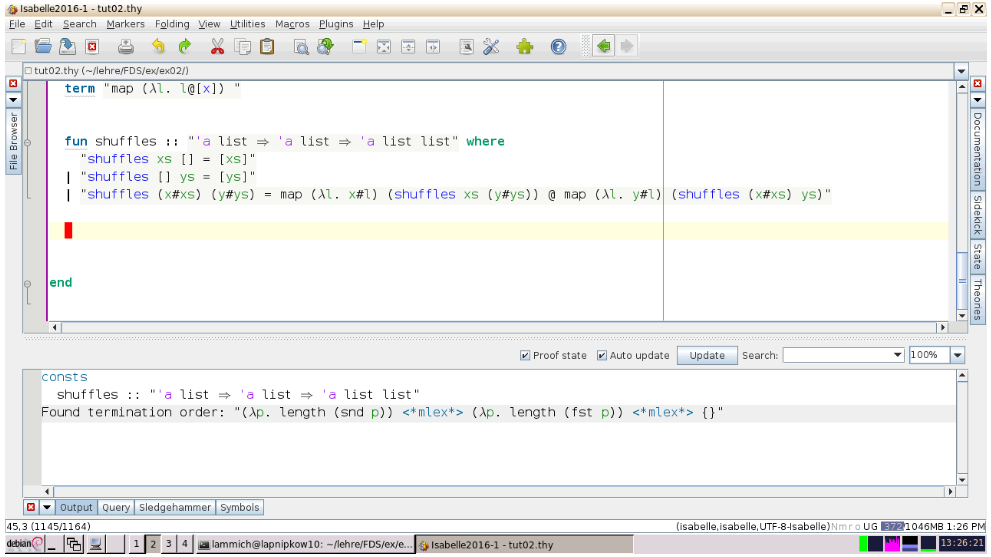

Isabelle2016-1 - tut02.thy (modified)
term "l@[x]"
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = undefined"
| "shuffles [] ys = undefined"
| "shuffles (x#xs) (y#ys) = undefined"
end
"map (λl. l @ [x])"
:: "'a list list ⇒ 'a list list"

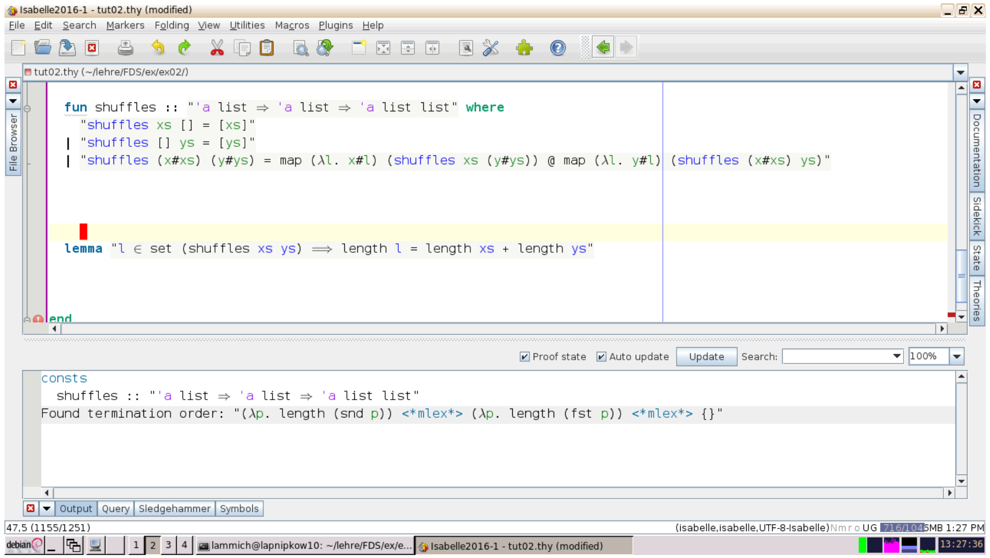

Isabelle2016-1 - tut02.thy (modified)
term "l@[x]"
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = undefined"
| "shuffles [] ys = undefined"
| "shuffles (x#xs) (y#ys) = undefined"
end
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "{}"

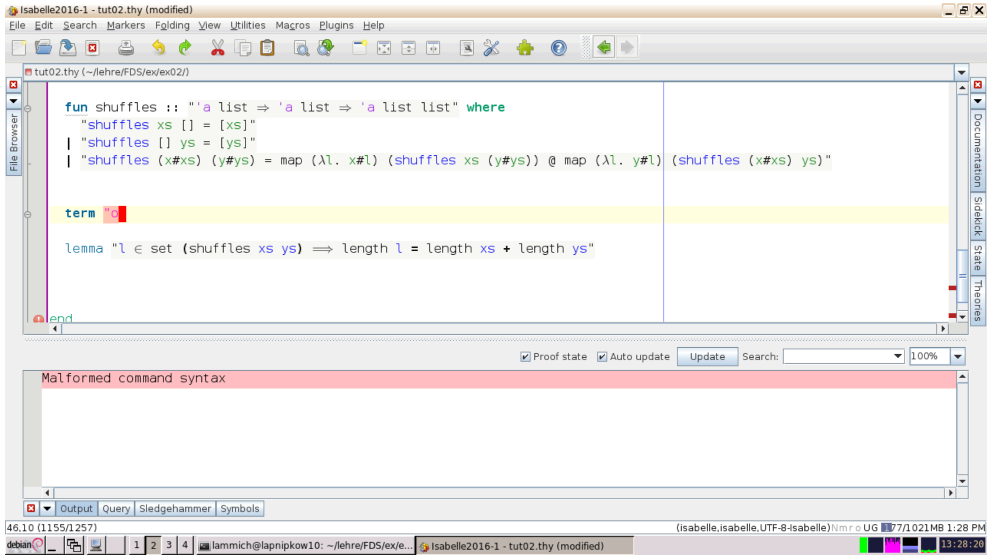

Isabelle2016-1 - tut02.thy (modified)
term "l@[x]"
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs | "shuffles [] ys = undefined"
| "shuffles (x#xs) (y#ys) = undefined"
end
Inner syntax error: unexpected end of input
Failed to parse prop

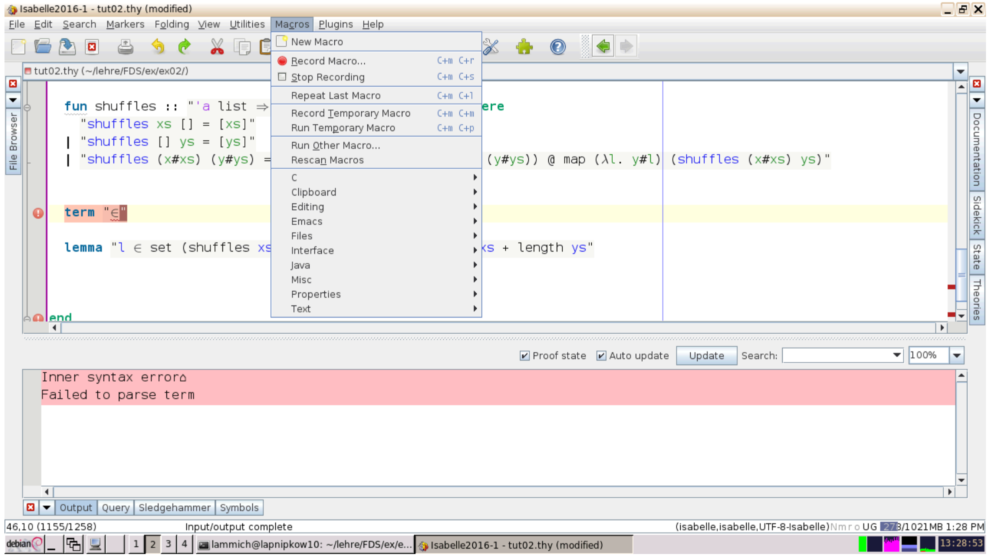

Isabelle2016-1 - tut02.thy (modified)
term "l@[x]"
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = undef
end
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "{}"

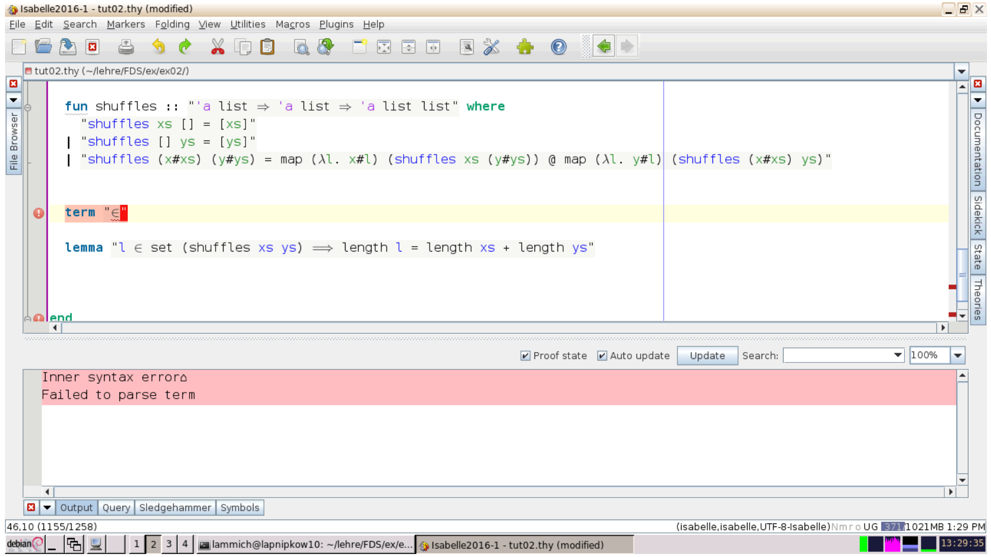

ex02.pdf
Exercise 2.2 Shuffle Product
To shuffle two lists, we repeat the following step until both lists are empty: Take the first element from one of the lists, and append it to the result.
That is, a shuffle of two lists contains exactly the elements of both lists in the right order.
Define a function shuffles that returns a list of all shuffles of two given lists
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
1
lammich@lapnipkow10: ~/lehre/FDS
13:26:03

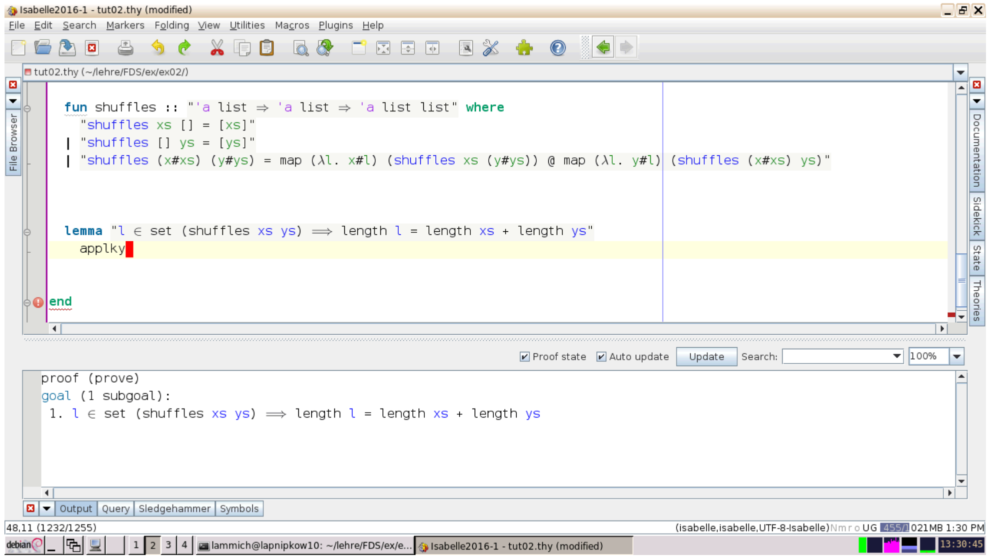

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
end
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "(λp. length (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"
Output Query Sledgehammer Symbols
13:26:21

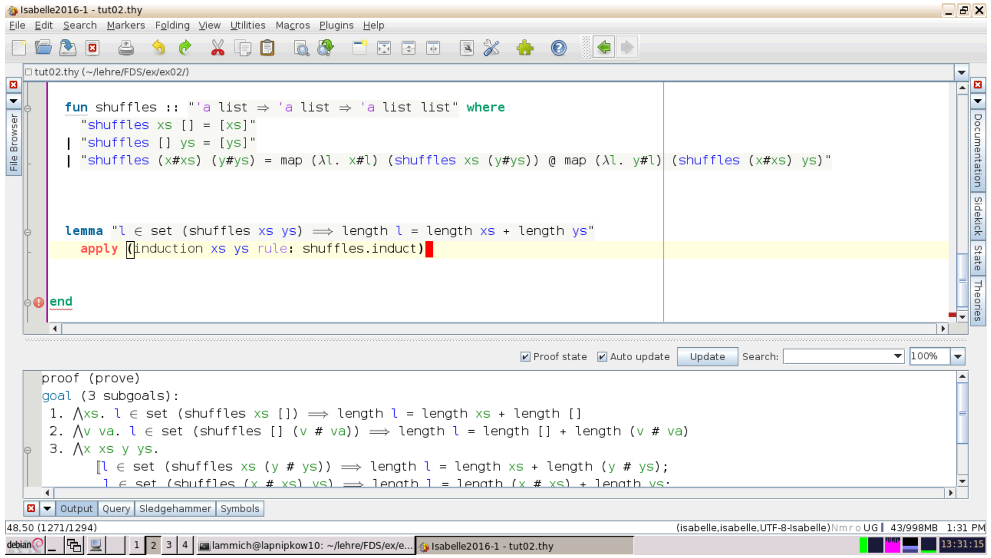

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
end
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "(λp. length (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"
Output Query Sledgehammer Symbols
13:27:36

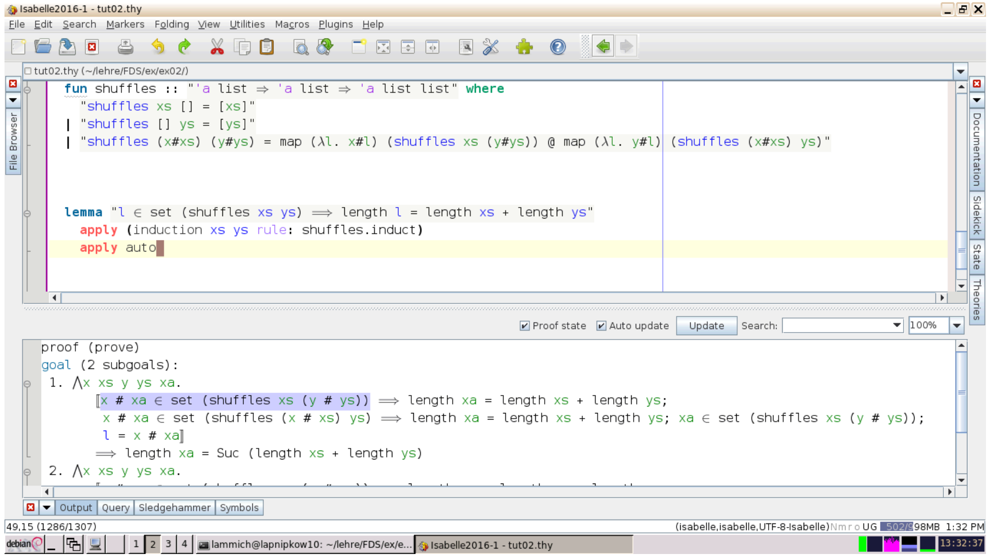
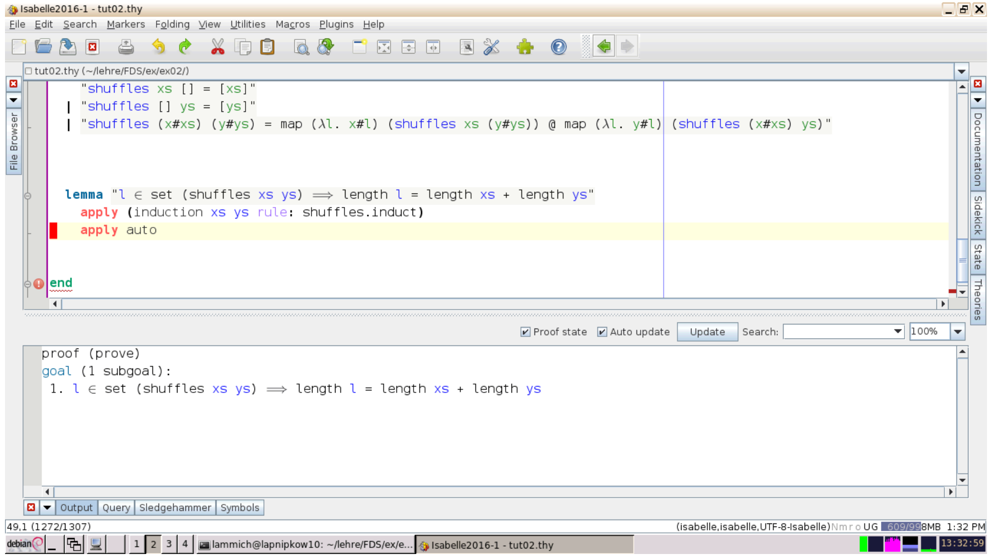
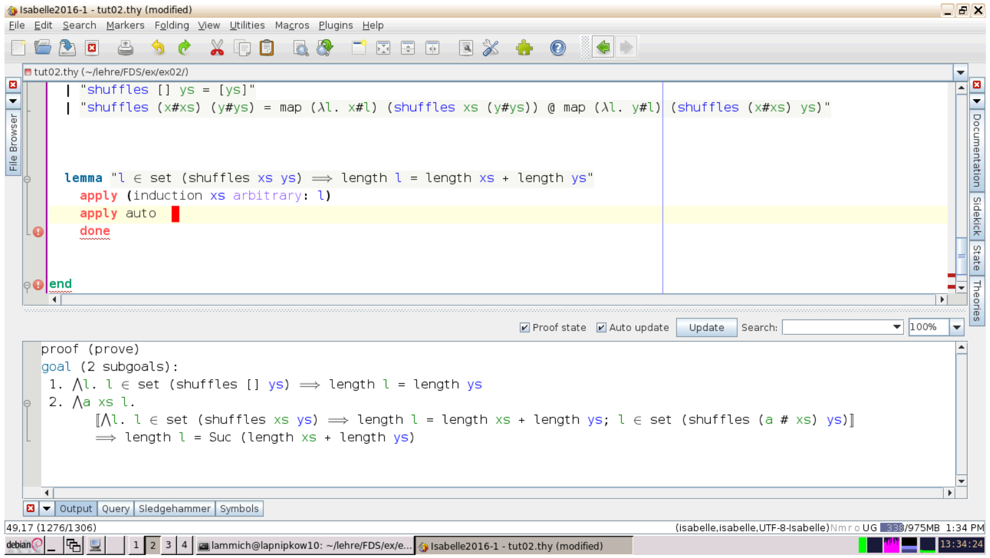
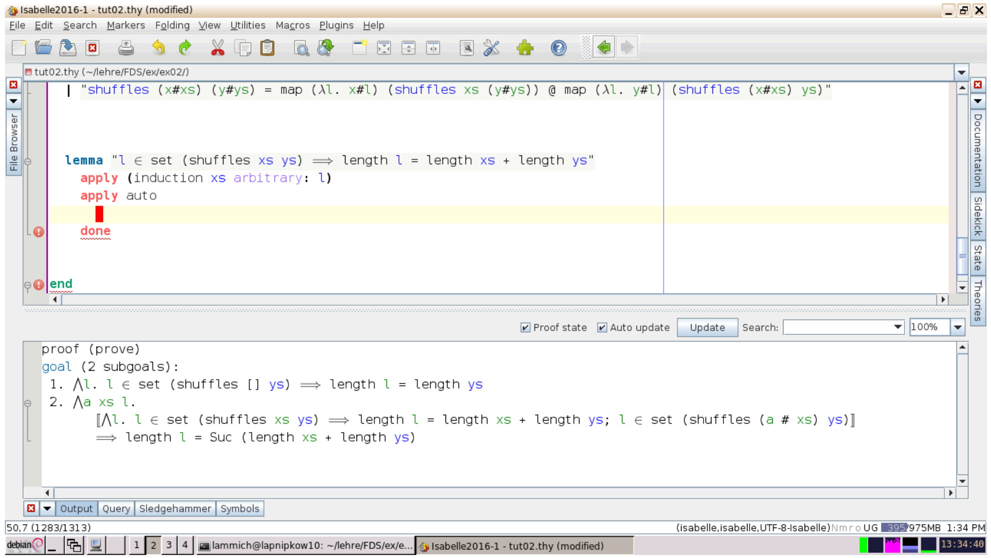

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
term "o
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
end
Malformed command syntax
Output Query Sledgehammer Symbols
13:28:20

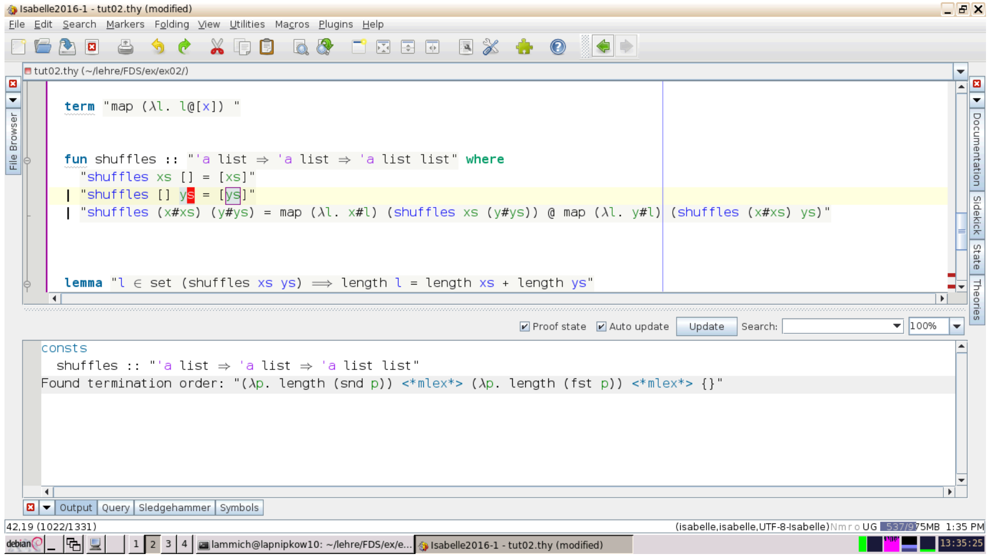

Isabelle2016-1 - tut02.thy (modified)
New Macro
Record Macro...
Stop Recording
Repeat Last Macro
Record Temporary Macro
Run Temporary Macro
Run Other Macro...
Rescan Macros
fun shuffles :: "'a list ⇒
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) =
term "∈"
lemma "l ∈ set (shuffles xs
end
Inner syntax error
Failed to parse term

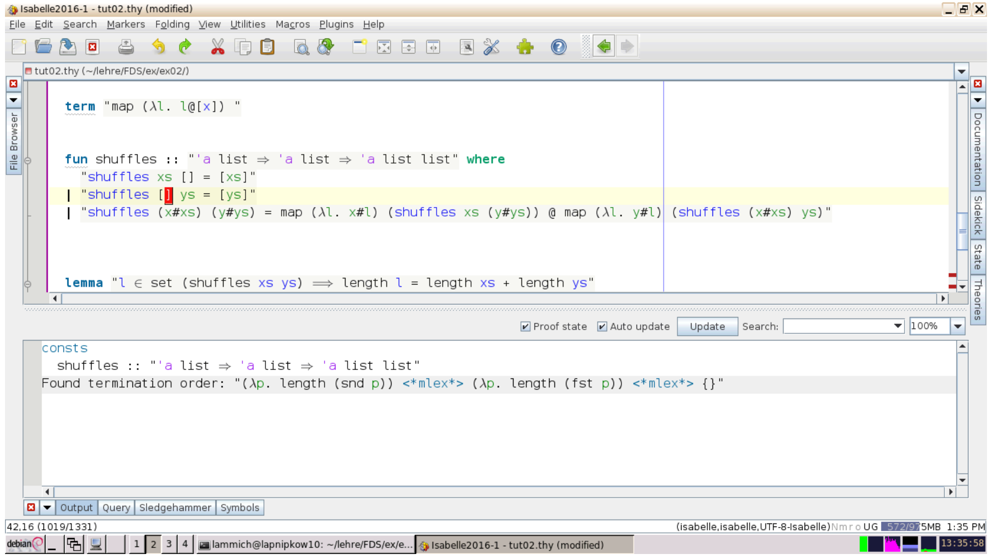

Isabelle2016-1 - tut02.thy (modified)
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
term "∈"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
end
Inner syntax error
Failed to parse term

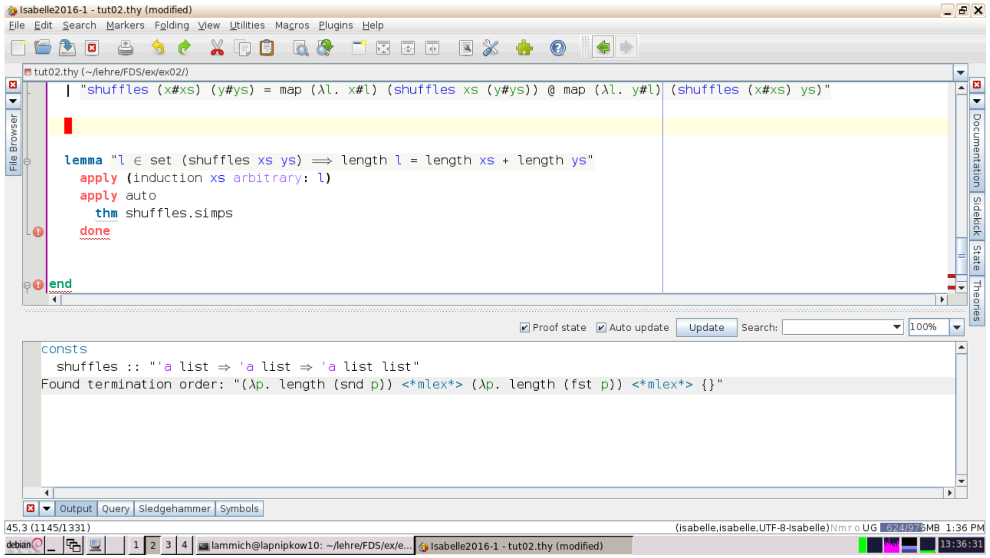

Isabelle2016-1 - tut02.thy (modified)
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
applky
end
proof (prove)
goal (1 subgoal):
1. l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys

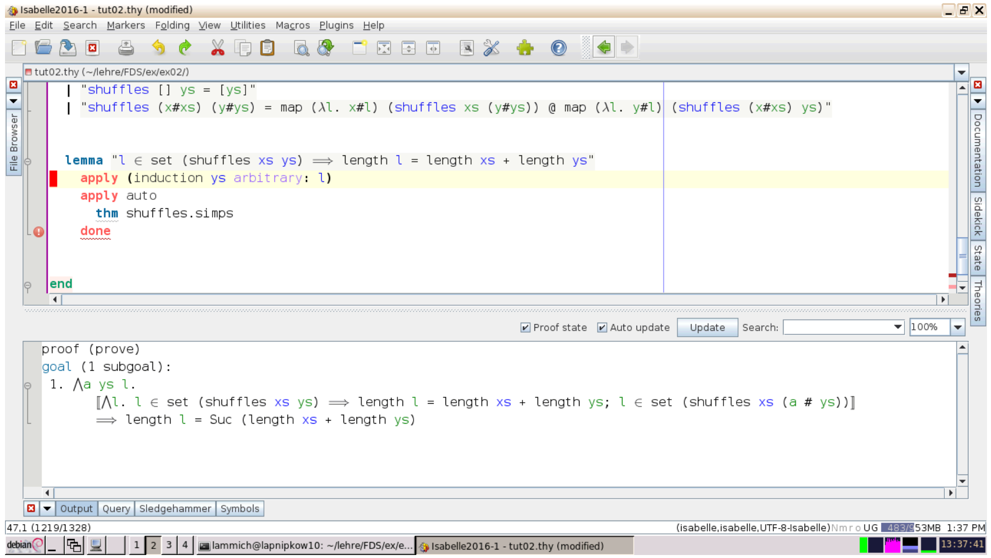

Isabelle2016-1 - tut02.thy
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs ys rule: shuffles.induct)
end
proof (prove)
goal (3 subgoals):
1. ⋀xs. l ∈ set (shuffles xs []) ⟹ length l = length xs + length []
2. ⋀v va. l ∈ set (shuffles [] (v # va)) ⟹ length l = length [] + length (v # va)
3. ⋀x xs y ys.
⟦l ∈ set (shuffles xs (y # ys)) ⟹ length l = length xs + length (y # ys);

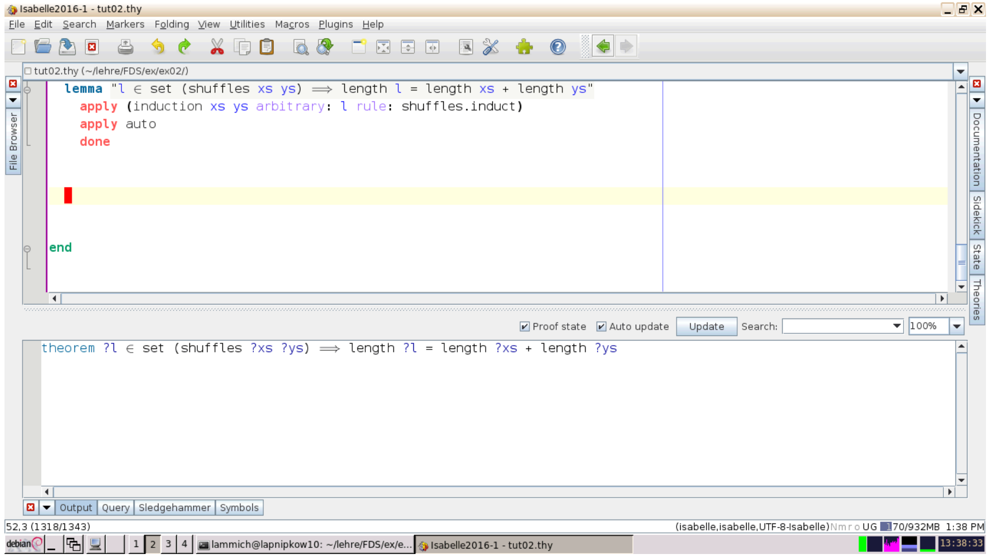

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs ys rule: shuffles.induct)
apply auto
proof (prove)
goal (2 subgoals):
1. ⋀x xs y ys xa.
⟦x # xa ∈ set (shuffles xs (y # ys)) ⟹ length xa = length xs + length ys;
x # xa ∈ set (shuffles (x # xs) ys) ⟹ length xa = length xs + length ys; xa ∈ set (shuffles xs (y # ys));
l = x # xa⟧
⟹ length xa = Suc (length xs + length ys)
2. ⋀x xs y ys xa.

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs ys rule: shuffles.induct)
apply auto
end
proof (prove)
goal (1 subgoal):
1. l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs arbitrary: l)
apply auto
done
end
proof (prove)
goal (2 subgoals):
1. ⋀l. l ∈ set (shuffles [] ys) ⟹ length l = length ys
2. ⋀a xs l.
⟦⋀l. l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys; l ∈ set (shuffles (a # xs) ys)⟧
⟹ length l = Suc (length xs + length ys)

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs arbitrary: l)
apply auto
done
end
proof (prove)
goal (2 subgoals):
1. ⋀l. l ∈ set (shuffles [] ys) ⟹ length l = length ys
2. ⋀a xs l.
⟦⋀l. l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys; l ∈ set (shuffles (a # xs) ys)⟧
⟹ length l = Suc (length xs + length ys)

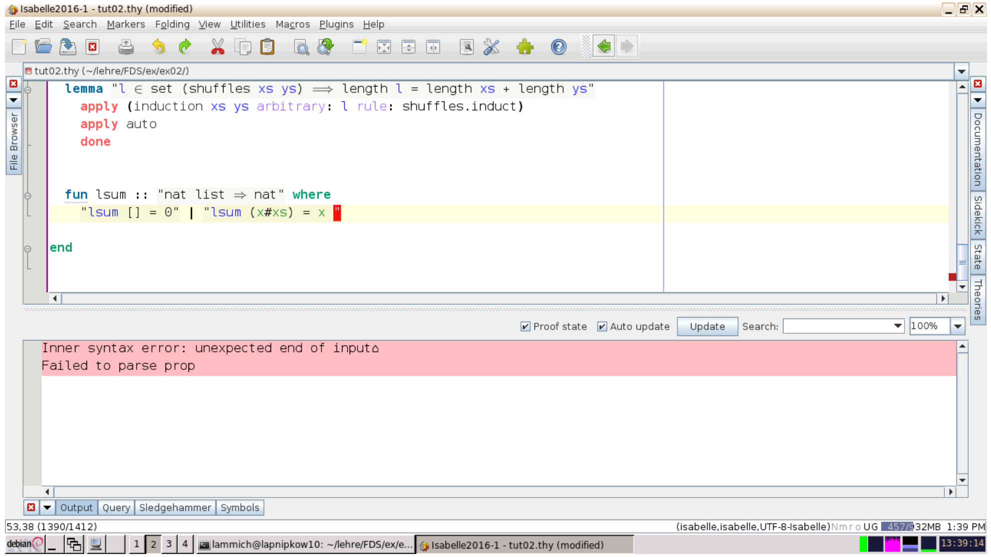

Isabelle2016-1 - tut02.thy (modified)
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "(λp. length (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"

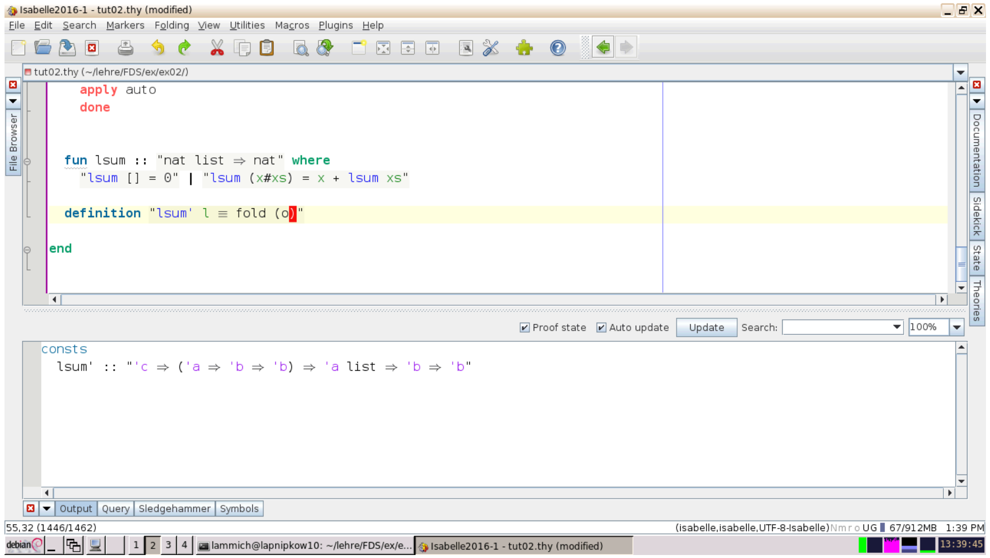

Isabelle2016-1 - tut02.thy (modified)
term "map (λl. l@[x]) "
fun shuffles :: "'a list ⇒ 'a list ⇒ 'a list list" where
"shuffles xs [] = [xs]"
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "(λp. length (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"

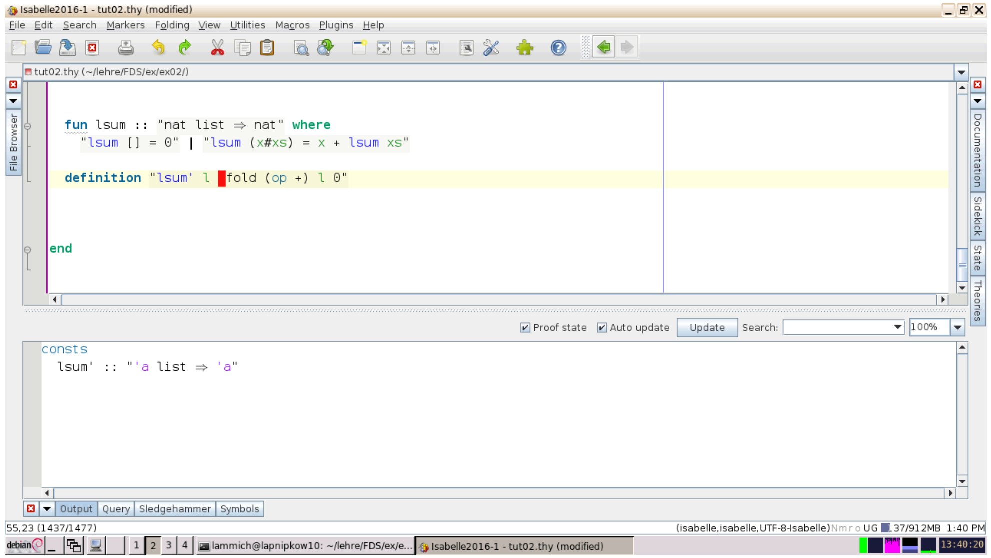

Isabelle2016-1 - tut02.thy (modified)
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs arbitrary: l)
apply auto
thm shuffles.simps
done
end
consts
shuffles :: "'a list ⇒ 'a list ⇒ 'a list list"
Found termination order: "(λp. length (snd p)) <*mlex*> (λp. length (fst p)) <*mlex*> {}"

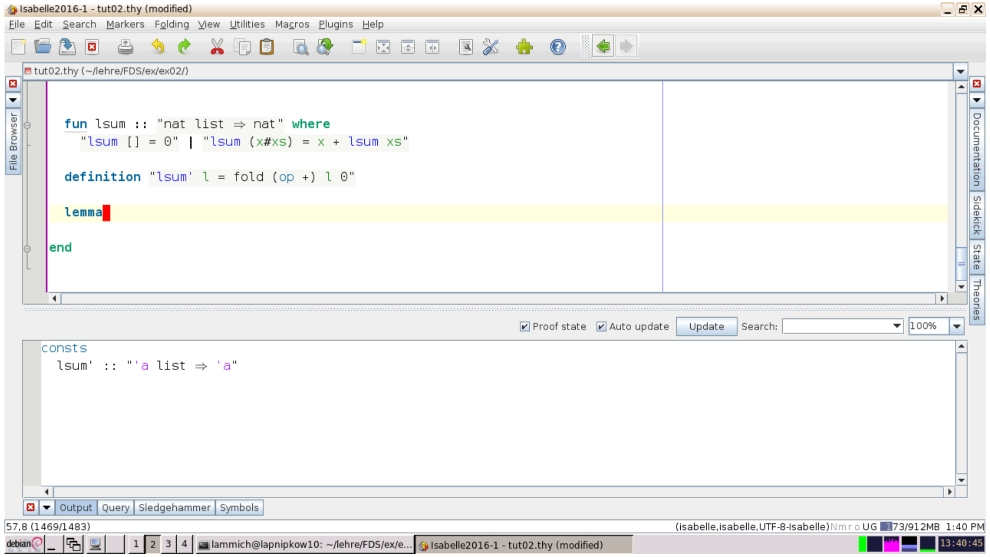

Isabelle2016-1 - tut02.thy (modified)
| "shuffles [] ys = [ys]"
| "shuffles (x#xs) (y#ys) = map (λl. x#l) (shuffles xs (y#ys)) @ map (λl. y#l) (shuffles (x#xs) ys)"
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction ys arbitrary: l)
apply auto
thm shuffles.simps
done
end
proof (prove)
goal (1 subgoal):
1. ⋀a ys l.
⟦⋀l. l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys; l ∈ set (shuffles xs (a # ys))⟧
⟹ length l = Suc (length xs + length ys)

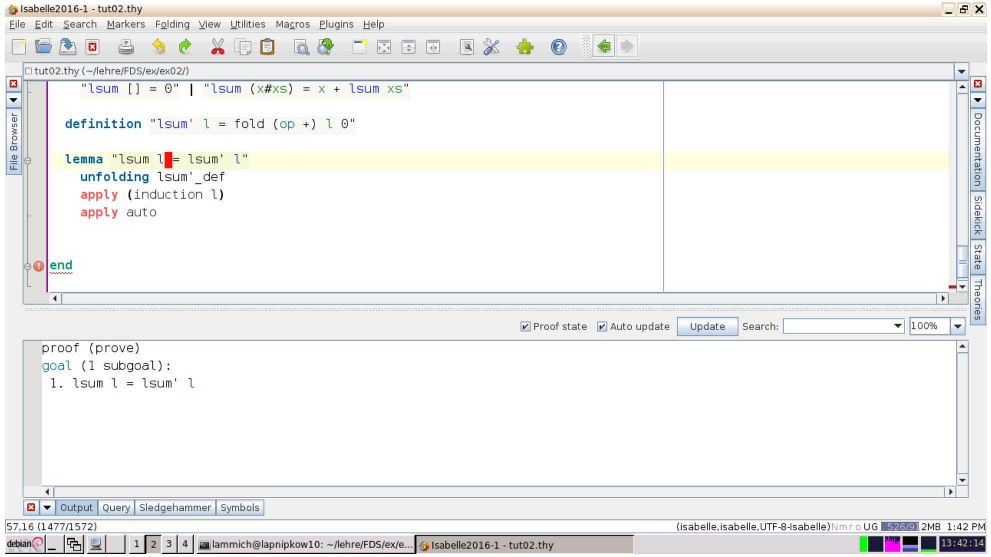

Isabelle2016-1 - tut02.thy
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs ys arbitrary: l rule: shuffles.induct)
apply auto
done
end
theorem ?l ∈ set (shuffles ?xs ?ys) ⟹ length ?l = length ?xs + length ?ys

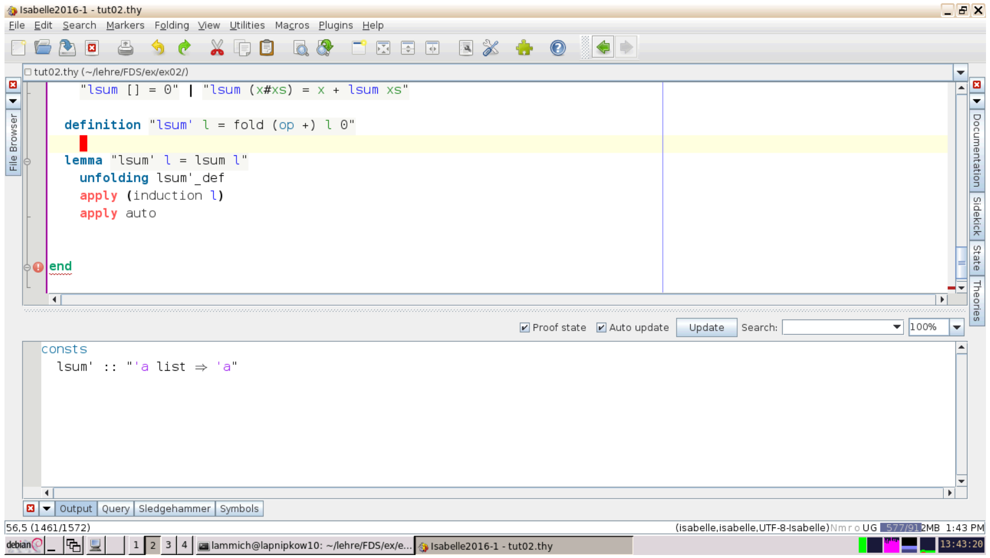

Isabelle2016-1 - tut02.thy (modified)
lemma "l ∈ set (shuffles xs ys) ⟹ length l = length xs + length ys"
apply (induction xs ys arbitrary: l rule: shuffles.induct)
apply auto
done
fun lsum :: "nat list ⇒ nat" where
"lsum [] = 0" | "lsum (x#xs) = x "
end
Inner syntax error: unexpected end of input
Failed to parse prop

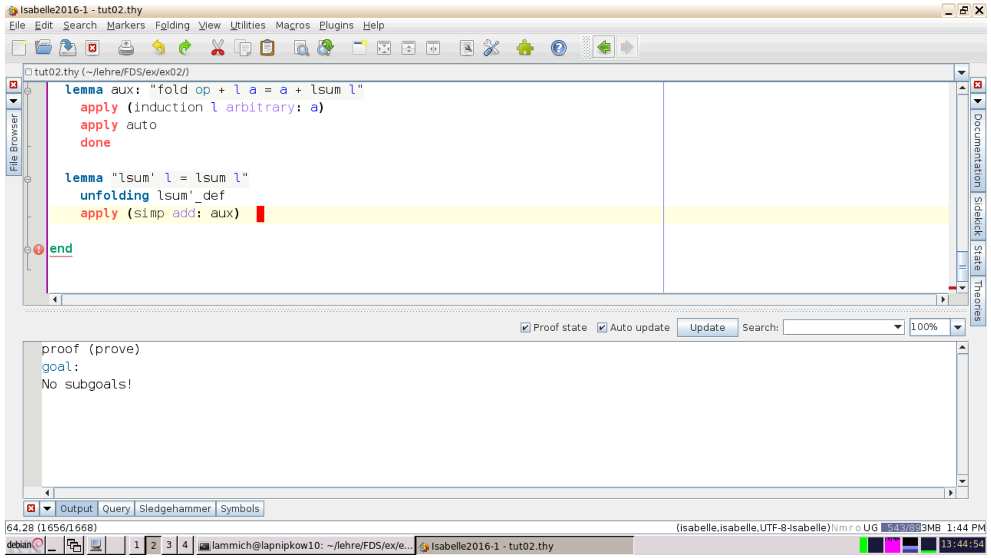

Isabelle2016-1 - tut02.thy (modified)
apply auto
done
fun lsum :: "nat list ⇒ nat" where
"lsum [] = 0" | "lsum (x#xs) = x + lsum xs"
definition "lsum' l ≡ fold (o)"
end
consts
lsum' :: "'c ⇒ ('a ⇒ 'b ⇒ 'b) ⇒ 'a list ⇒ 'b ⇒ 'b"

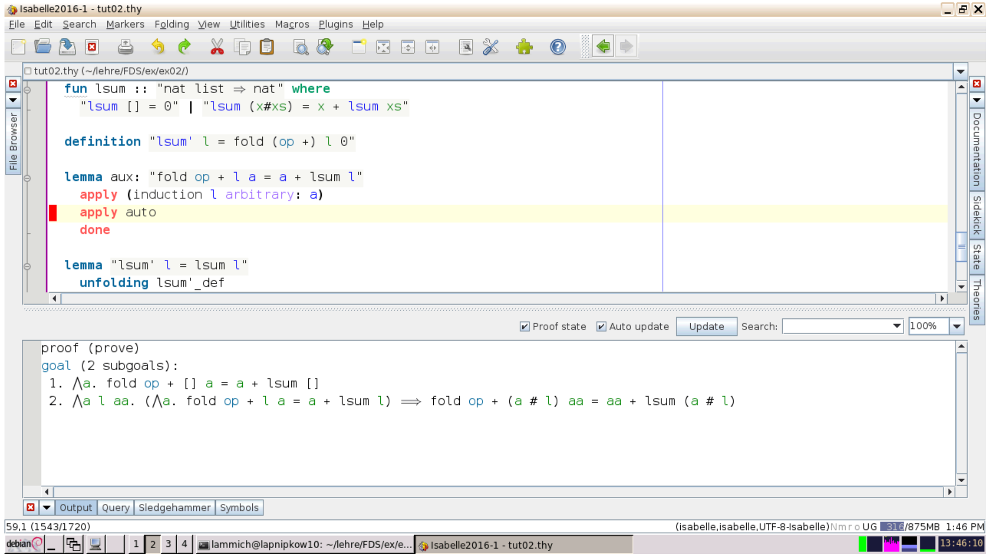

Isabelle2016-1 - tut02.thy (modified)
fun lsum :: "nat list ⇒ nat" where
"lsum [] = 0" | "lsum (x#xs) = x + lsum xs"
definition "lsum' l fold (op +) l 0"
end
consts
lsum' :: "'a list ⇒ 'a"

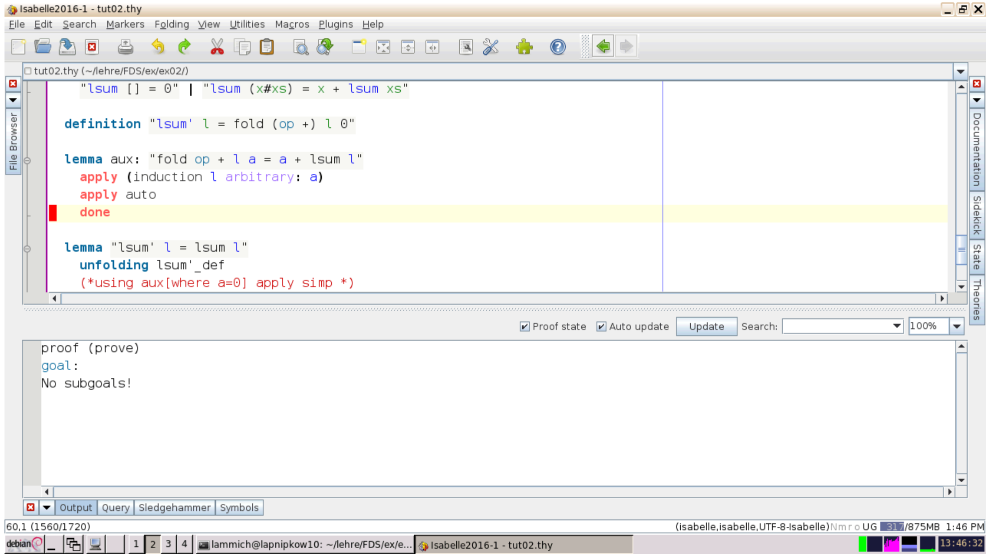

Isabelle2016-1 - tut02.thy (modified)
tut02.thy (~/lehre/FDS/ex/ex02/)
fun lsum :: "nat list ⇒ nat" where
"lsum [] = 0" | "lsum (x#xs) = x + lsum xs"
definition "lsum' l = fold (op +) l 0"
lemma
end
consts
lsum' :: "'a list ⇒ 'a"

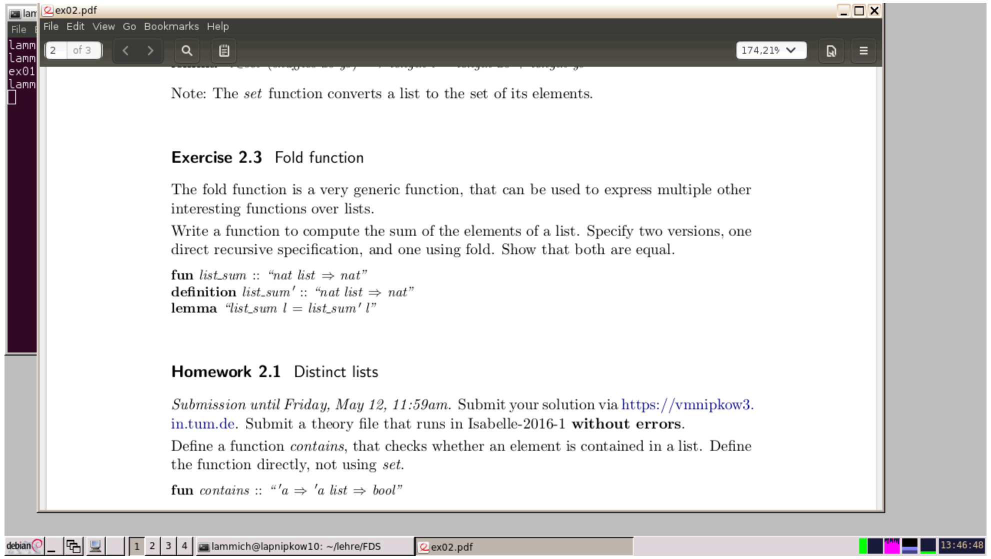

Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
"lsum [] = 0" | "lsum (x#xs) = x + lsum xs"
definition "lsum' l = fold (op +) l 0"
lemma "lsum l = lsum' l"
unfolding lsum'_def
apply (induction l)
apply auto
end
proof (prove)
goal (1 subgoal):
1. lsum l = lsum' l


Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
"lsum [] = 0" | "lsum (x#xs) = x + lsum xs"
definition "lsum' l = fold (op +) l 0"
lemma "lsum' l = lsum l"
unfolding lsum'_def
apply (induction l)
apply auto
end
consts
lsum' :: "'a list ⇒ 'a"


Isabelle2016-1 - tut02.thy
tut02.thy (~/lehre/FDS/ex/ex02/)
lemma aux: "fold op + l a = a + lsum l"
apply (induction l arbitrary: a)
apply auto
done
lemma "lsum' l = lsum l"
unfolding lsum'_def
apply (simp add: aux)
end
proof (prove)
goal:
No subgoals!







Note: The *set* function converts a list to the set of its elements.

Exercise 2.3 Fold function

The fold function is a very generic function, that can be used to express multiple other interesting functions over lists.

Write a function to compute the sum of the elements of a list. Specify two versions, one direct recursive specification, and one using fold. Show that both are equal.

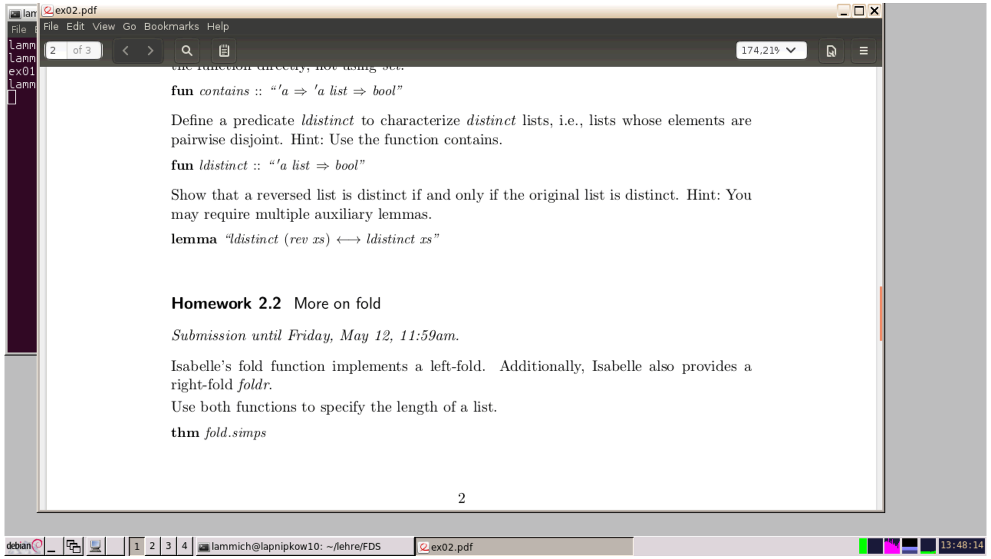
fun *list_sum* :: *"nat list ⇒ nat"*
definition *list_sum'* :: *"nat list ⇒ nat"*
lemma *"list_sum l = list_sum' l"*

Homework 2.1 Distinct lists

Submission until Friday, May 12, 11:59am. Submit your solution via https://vmnipkow3.in.tum.de. Submit a theory file that runs in Isabelle-2016-1 **without errors**.

Define a function *contains*, that checks whether an element is contained in a list. Define the function directly, not using *set*.

fun *contains* :: *"'a ⇒ 'a list ⇒ bool"*



fun *contains* :: *"'a ⇒ 'a list ⇒ bool"*

Define a predicate *ldistinct* to characterize *distinct* lists, i.e., lists whose elements are pairwise disjoint. Hint: Use the function contains.

fun *ldistinct* :: *"'a list ⇒ bool"*

Show that a reversed list is distinct if and only if the original list is distinct. Hint: You may require multiple auxiliary lemmas.

lemma *"ldistinct (rev xs) ⟷ ldistinct xs"*

Homework 2.2 More on fold

Submission until Friday, May 12, 11:59am.

Isabelle's fold function implements a left-fold. Additionally, Isabelle also provides a right-fold *foldr*.

Use both functions to specify the length of a list.

thm *fold.simps*

2

thm *foldr.simps*

definition *length_fold* :: "'a list ⇒ nat"

definition *length_foldr* :: "'a list ⇒ nat"

lemma "*length_fold l = length l*"
lemma "*length_foldr l = length l*"

Homework 2.3 List Slices

Submission until Friday, May 12, 11:59am. Specify a function *slice xs s l*, that, for a list $xs=[x_0,...,x_n]$ returns the slice starting at s with length l, i.e., $[x_s,...,x_{s+len}]$.

If *s* or *len* is out of range, return a shorter (or the empty) list.

fun *slice* :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where

Hint: Use pattern matching instead of *if*-expressions. For example, instead of writing $f\ x = (\textit{if } x{>}0 \textit{ then } \dots \textit{ else } \dots)$ you should define two equations $f\ 0 = \dots$ and $f\ (Suc\ n) = \dots$.

Some test cases, which should all hold, i.e., yield *True*

If *s* or *len* is out of range, return a shorter (or the empty) list.

fun *slice* :: "'a list ⇒ nat ⇒ nat ⇒ 'a list"
where

Hint: Use pattern matching instead of *if*-expressions. For example, instead of writing $f\ x = (\textit{if } x{>}0 \textit{ then } \dots \textit{ else } \dots)$ you should define two equations $f\ 0 = \dots$ and $f\ (Suc\ n) = \dots$.

Some test cases, which should all hold, i.e., yield *True*

value "*slice [0,1,2,3,4,5,6::int] 2 3 = [2,3,4]*" — In range
value "*slice [0,1,2,3,4,5,6::int] 2 10 = [2,3,4,5,6]*" — Length out of range
value "*slice [0,1,2,3,4,5,6::int] 10 10 = []*" — Start index out of range

Show that concatenation of two adjacent slices can be expressed as a single slice:

lemma "*slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)*"

Show that a slice of a distinct list is distinct.

lemma "*ldistinct xs ⟹ ldistinct (slice xs s l)*"

Show that a slice of a distinct list is distinct.

lemma "*ldistinct xs ⟹ ldistinct (slice xs s l)*"

fun *list_sum* :: "nat list ⇒ nat"
definition *list_sum'* :: "nat list ⇒ nat"
lemma "*list_sum l = list_sum' l*"

Homework 2.1 Distinct lists

Submission until Friday, May 12, 11:59am. Submit your solution via https://vmnipkow3.in.tum.de. Submit a theory file that runs in Isabelle-2016-1 **without errors**.

Define a function *contains*, that checks whether an element is contained in a list. Define the function directly, not using *set*.

fun *contains* :: "'a ⇒ 'a list ⇒ bool"

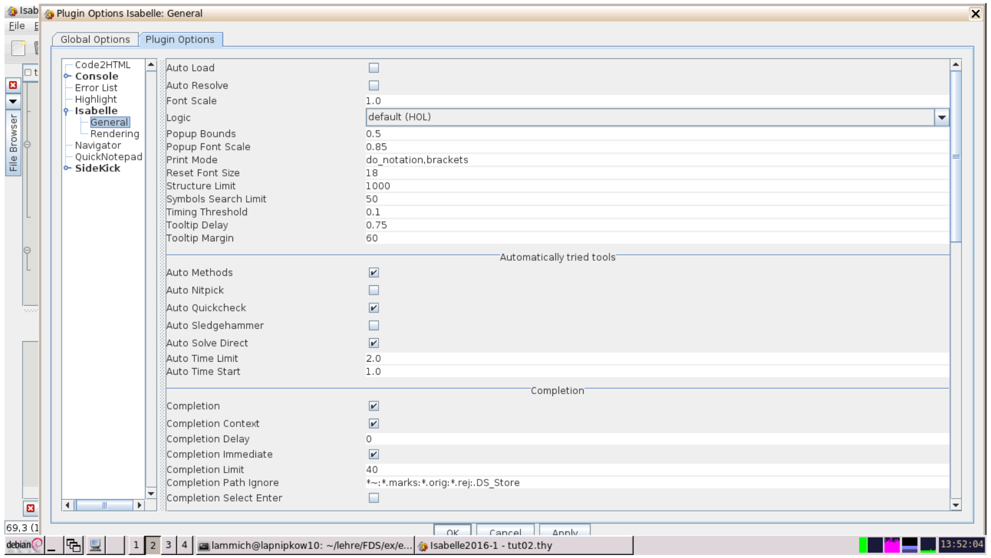
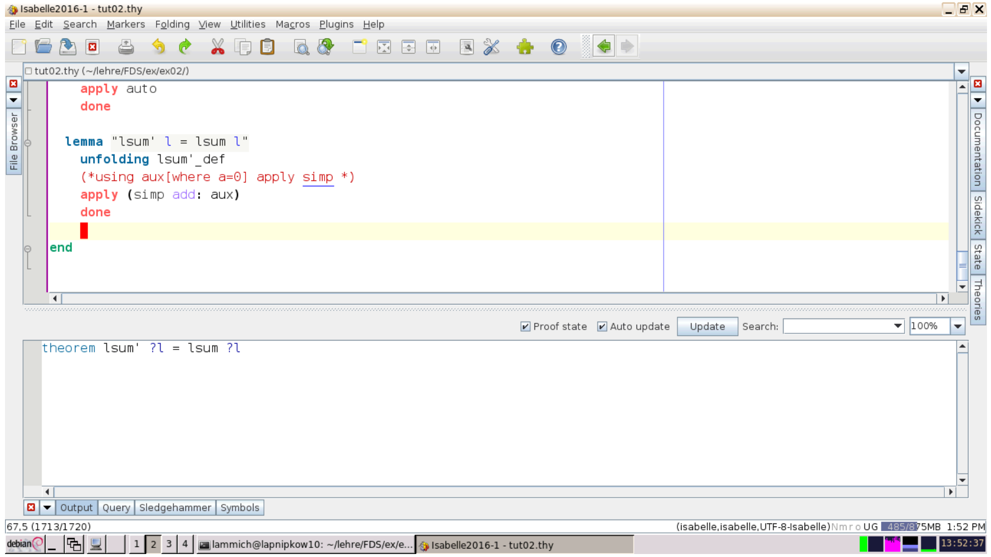
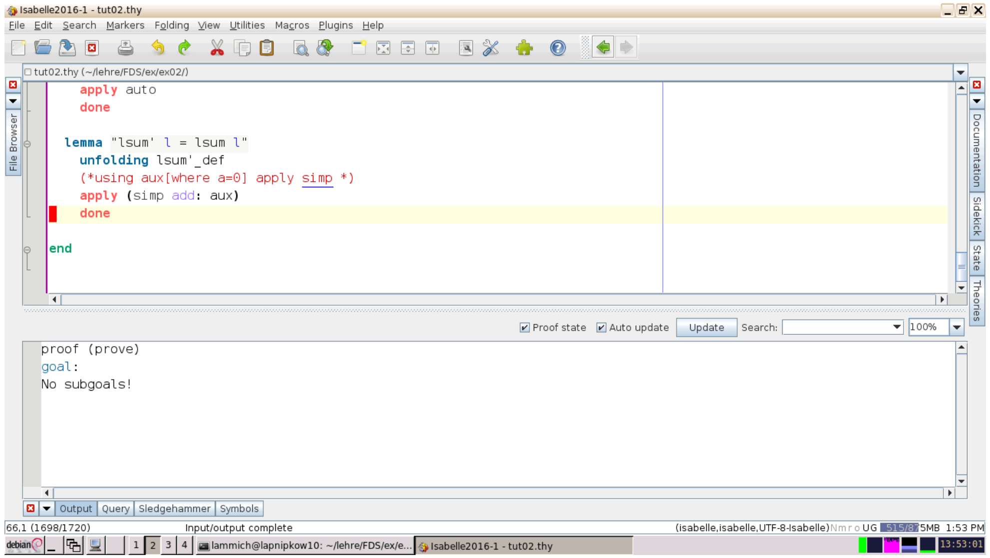
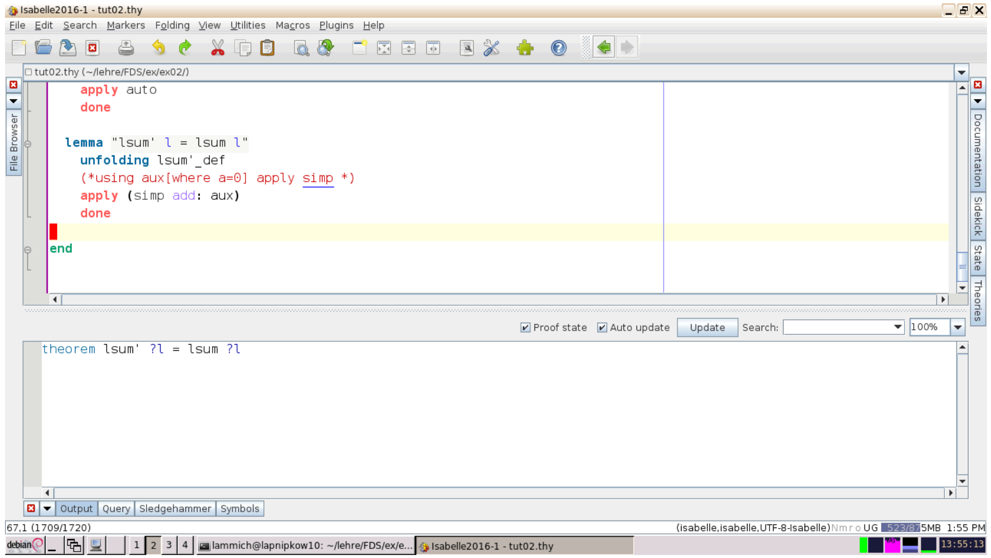
Define a predicate *ldistinct* to characterize *distinct* lists, i.e., lists whose elements are pairwise disjoint. Hint: Use the function contains.

fun *ldistinct* :: "'a list ⇒ bool"

Show that a reversed list is distinct if and only if the original list is distinct. Hint: You may require multiple auxiliary lemmas.

lemma "*ldistinct (rev xs) ⟷ ldistinct xs*"

Homework 2.2 More on fold


Plugin Options Isabelle: General
Global Options
Plugin Options
Code2HTML
Console
Error List
Highlight
Isabelle
General
Rendering
Navigator
QuickNotepad
SideKick
Auto Load
Auto Resolve
Font Scale 1.0
Logic default (HOL)
Popup Bounds 0.5
Popup Font Scale 0.85
Print Mode do_notation,brackets
Reset Font Size 18
Structure Limit 1000
Symbols Search Limit 50
Timing Threshold 0.1
Tooltip Delay 0.75
Tooltip Margin 60
Automatically tried tools
Auto Methods
Auto Nitpick
Auto Quickcheck
Auto Sledgehammer
Auto Solve Direct
Auto Time Limit 2.0
Auto Time Start 1.0
Completion
Completion
Completion Context
Completion Delay 0
Completion Immediate
Completion Limit 40
Completion Path Ignore *~:*.marks:*.orig:*.rej:.DS_Store
Completion Select Enter
OK
Cancel
Apply

Isabelle2016-1 - tut02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
apply auto
done
lemma "lsum' l = lsum l"
unfolding lsum'_def
(*using aux[where a=0] apply simp *)
apply (simp add: aux)
done
end
Proof state
Auto update
Update
Search:
100%
theorem lsum' ?l = lsum ?l
Output
Query
Sledgehammer
Symbols
67,5 (1713/1720)
(isabelle,isabelle,UTF-8-Isabelle)
1:52 PM

Isabelle2016-1 - tut02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
apply auto
done
lemma "lsum' l = lsum l"
unfolding lsum'_def
(*using aux[where a=0] apply simp *)
apply (simp add: aux)
done
end
Proof state
Auto update
Update
Search:
100%
proof (prove)
goal:
No subgoals!
Output
Query
Sledgehammer
Symbols
66,1 (1698/1720)
Input/output complete
(isabelle,isabelle,UTF-8-Isabelle)
1:53 PM

Isabelle2016-1 - tut02.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut02.thy (~/lehre/FDS/ex/ex02/)
apply auto
done
lemma "lsum' l = lsum l"
unfolding lsum'_def
(*using aux[where a=0] apply simp *)
apply (simp add: aux)
done
end
Proof state
Auto update
Update
Search:
100%
theorem lsum' ?l = lsum ?l
Output
Query
Sledgehammer
Symbols
67,1 (1709/1720)
(isabelle,isabelle,UTF-8-Isabelle)
1:55 PM