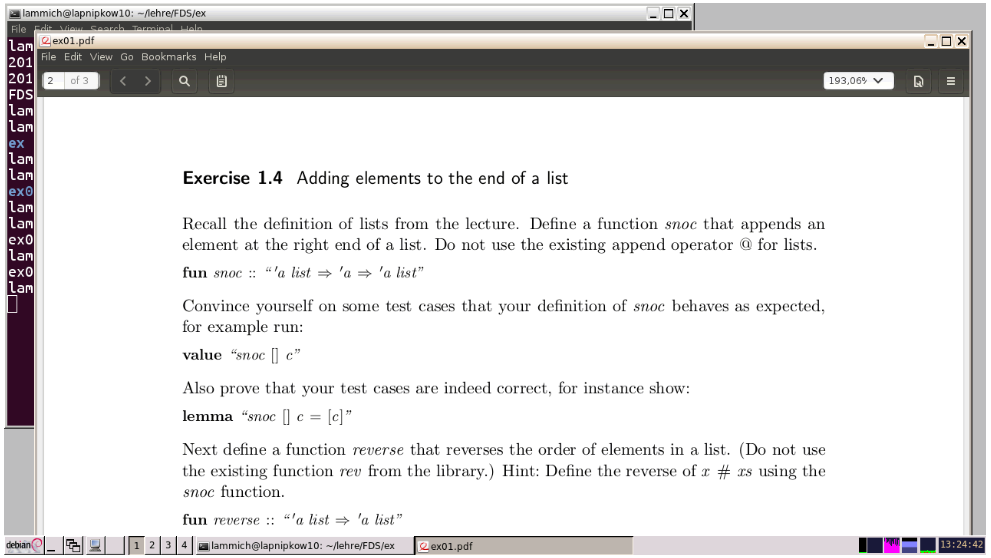
Script **generated by TTT**

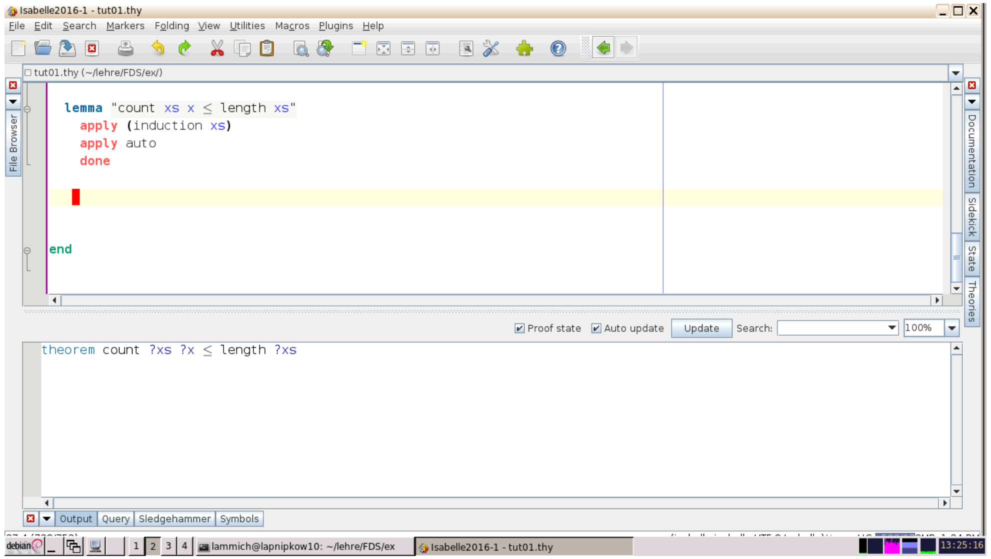
Title: Lammich: FDS Tutorial (28.04.2017)

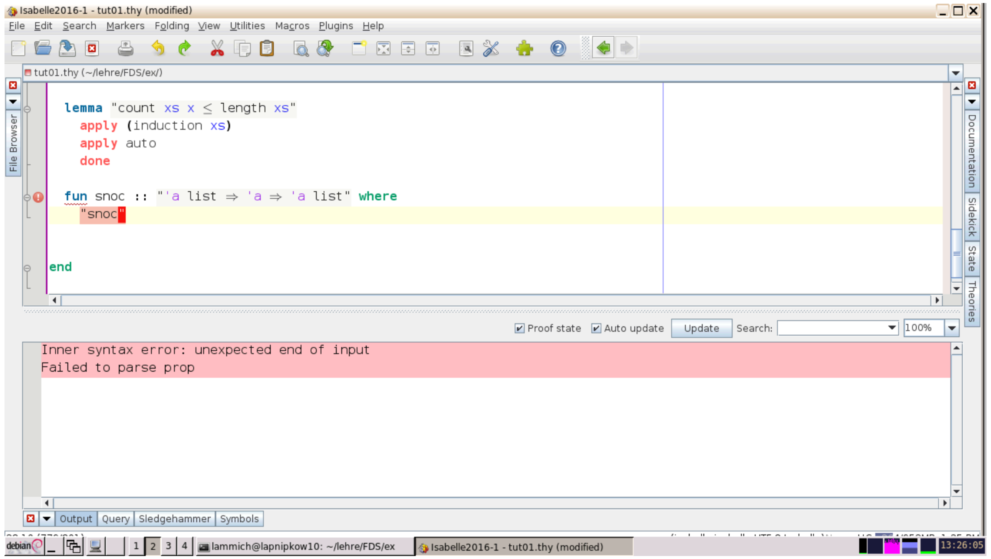
Date: Fri Apr 28 13:24:43 CEST 2017

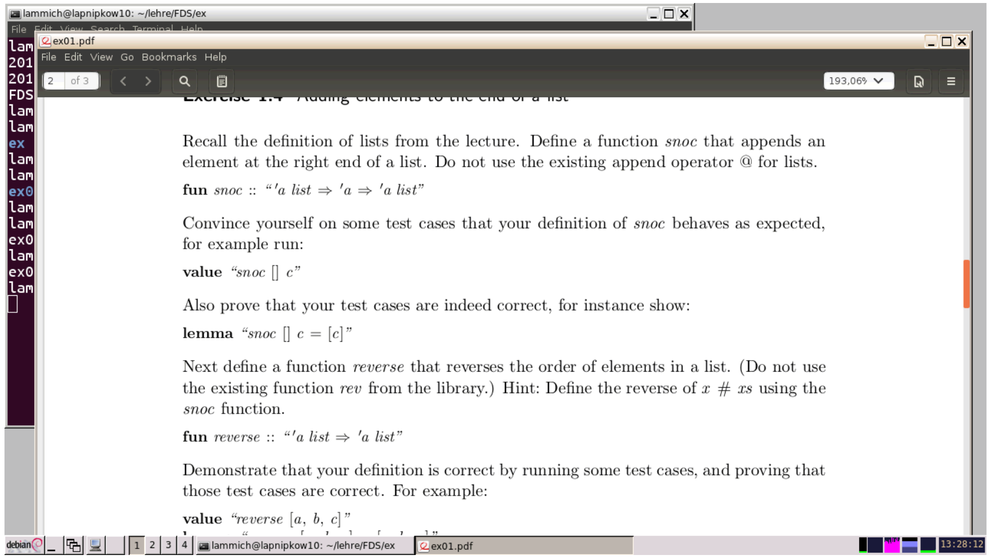
Duration: 20:45 min

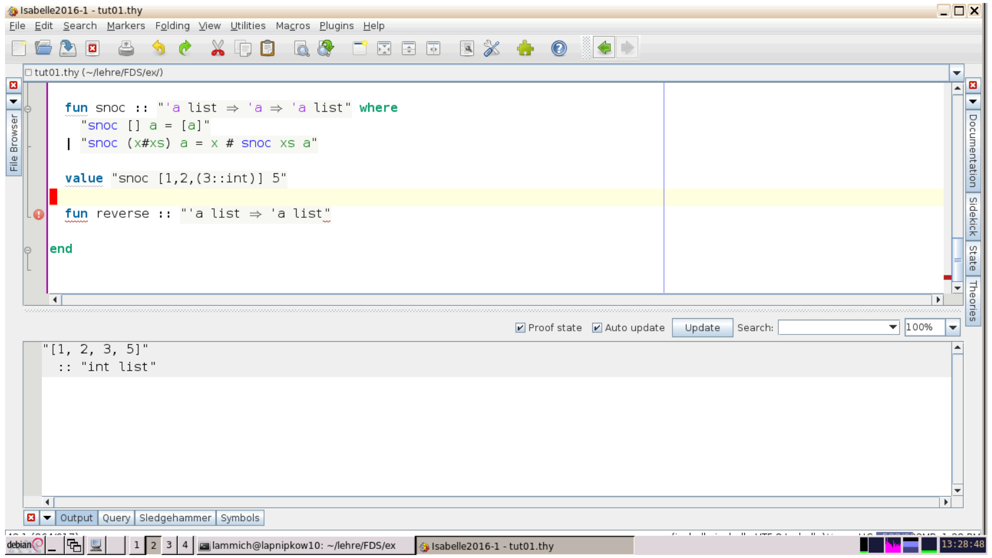
Pages: 19

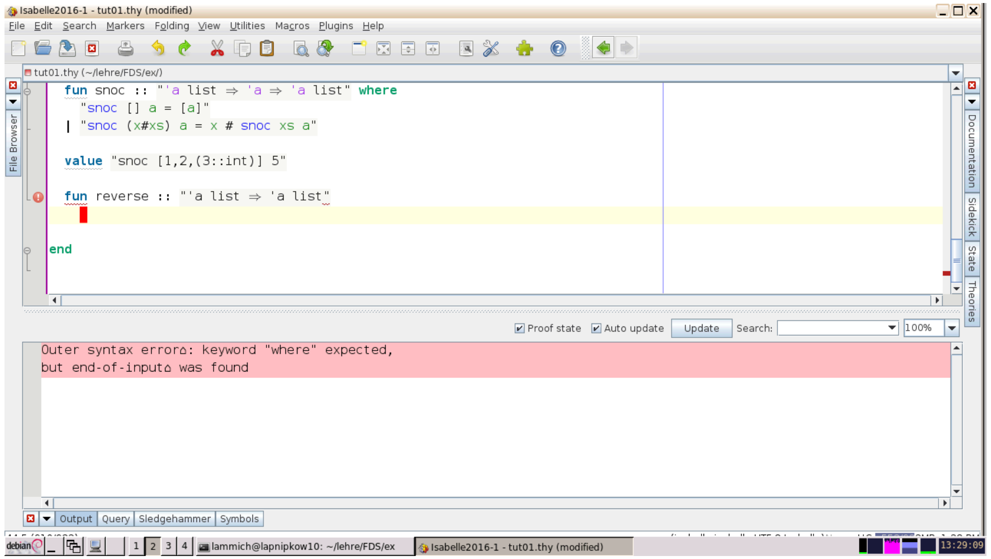


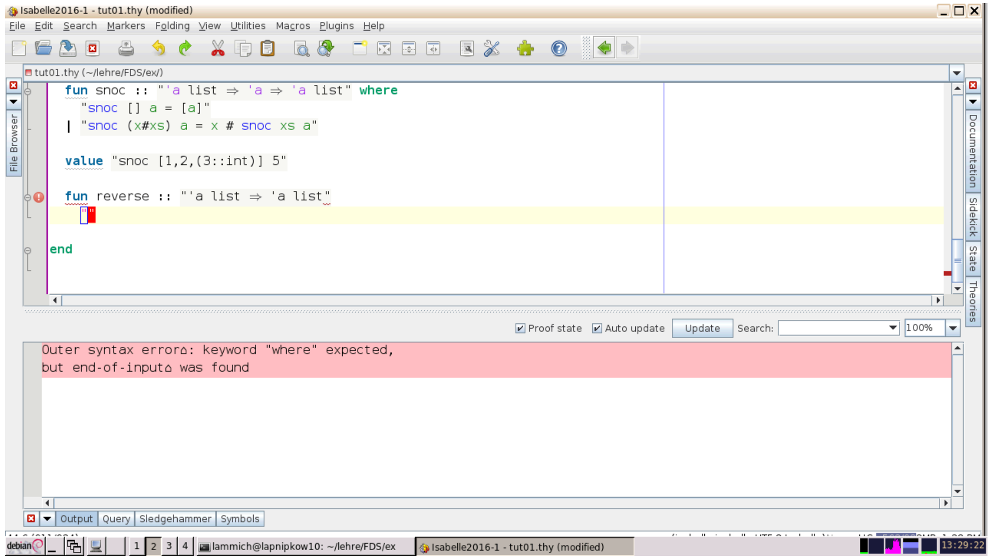


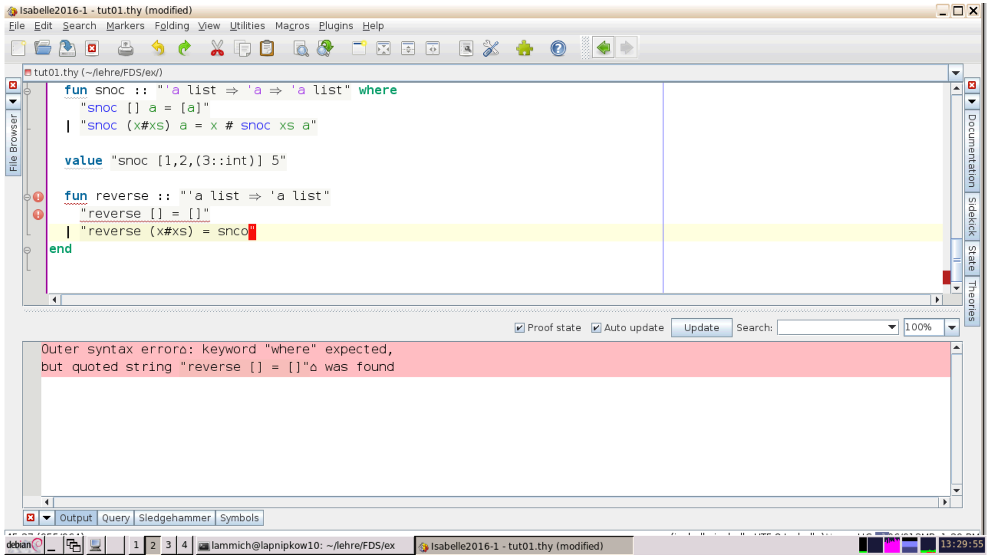


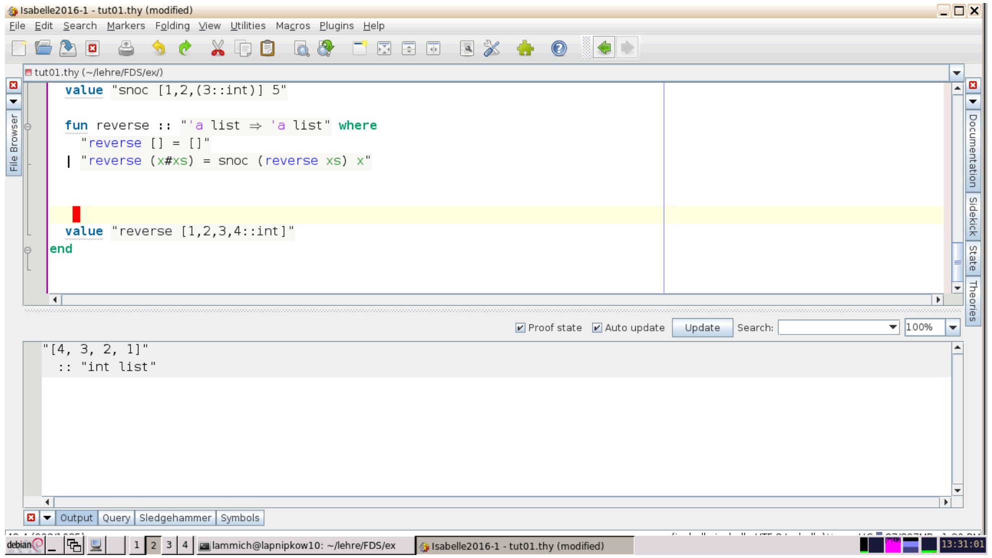

lammich@lapnipkow10: ~/lehre/FDS/ex
ex01.pdf
File Edit View Go Bookmarks Help
2 of 3
193,06%
Exercise 1.4 Adding elements to the end of a list
Recall the definition of lists from the lecture. Define a function snoc that appends an element at the right end of a list. Do not use the existing append operator @ for lists.
fun snoc :: "'a list ⇒ 'a ⇒ 'a list"
Convince yourself on some test cases that your definition of snoc behaves as expected, for example run:
value "snoc [] c"
Also prove that your test cases are indeed correct, for instance show:
lemma "snoc [] c = [c]"
Next define a function reverse that reverses the order of elements in a list. (Do not use the existing function rev from the library.) Hint: Define the reverse of x # xs using the snoc function.
fun reverse :: "'a list ⇒ 'a list"
Demonstrate that your definition is correct by running some test cases, and proving that those test cases are correct. For example:
value "reverse [a, b, c]"
lammich@lapnipkow10: ~/lehre/FDS/ex
ex01.pdf
13:28:12

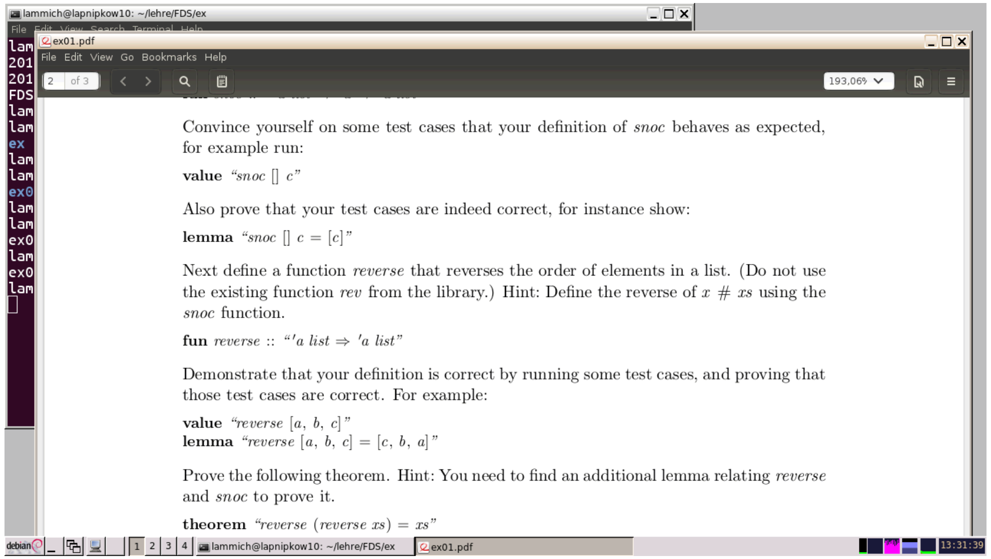

Isabelle2016-1 - tut01.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut01.thy (~/lehre/FDS/ex/)
fun snoc :: "'a list ⇒ 'a ⇒ 'a list" where
"snoc [] a = [a]"
| "snoc (x#xs) a = x # snoc xs a"
value "snoc [1,2,(3::int)] 5"
fun reverse :: "'a list ⇒ 'a list"
end
Proof state
Auto update
Update
Search:
100%
"[1, 2, 3, 5]"
:: "int list"
Output
Query
Sledgehammer
Symbols
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut01.thy
13:28:48

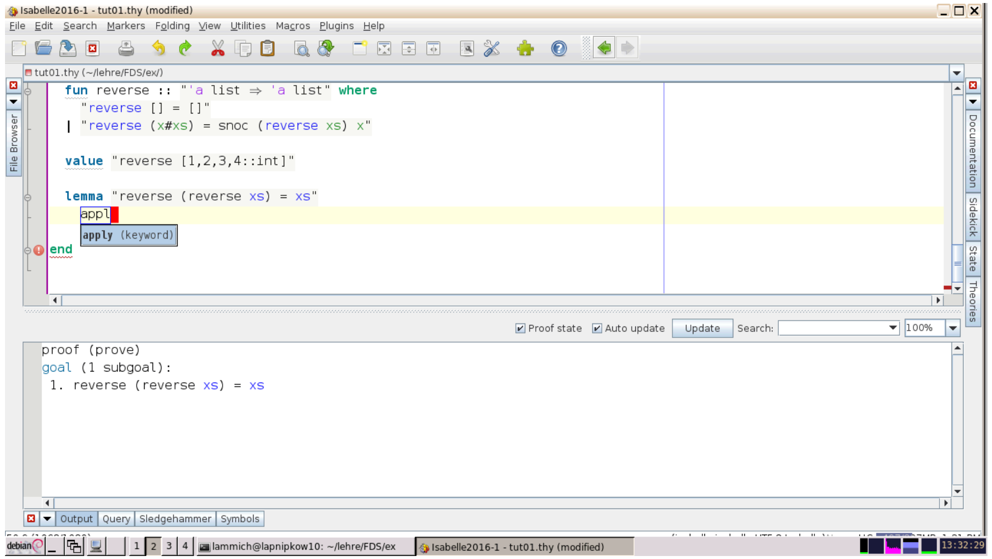

Isabelle2016-1 - tut01.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut01.thy (~/lehre/FDS/ex/)
fun snoc :: "'a list ⇒ 'a ⇒ 'a list" where
"snoc [] a = [a]"
| "snoc (x#xs) a = x # snoc xs a"
value "snoc [1,2,(3::int)] 5"
fun reverse :: "'a list ⇒ 'a list"
end
Proof state
Auto update
Update
Search:
100%
Outer syntax error: keyword "where" expected,
but end-of-input was found
Output
Query
Sledgehammer
Symbols
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut01.thy (modified)
13:29:09

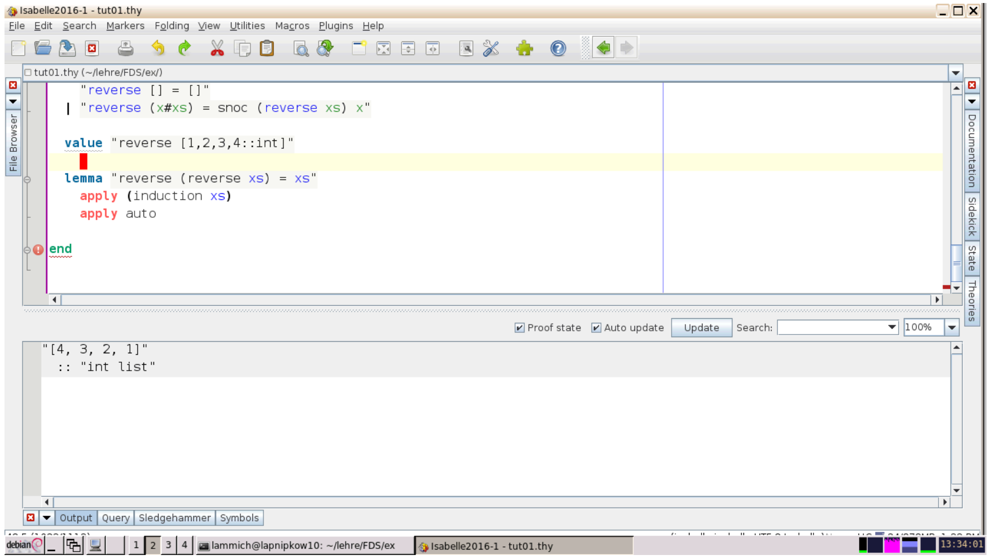

Isabelle2016-1 - tut01.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut01.thy (~/lehre/FDS/ex/)
fun snoc :: "'a list ⇒ 'a ⇒ 'a list" where
"snoc [] a = [a]"
| "snoc (x#xs) a = x # snoc xs a"
value "snoc [1,2,(3::int)] 5"
fun reverse :: "'a list ⇒ 'a list"
end
Proof state
Auto update
Update
Search:
100%
Outer syntax error: keyword "where" expected,
but end-of-input was found
Output
Query
Sledgehammer
Symbols
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut01.thy (modified)
13:29:22

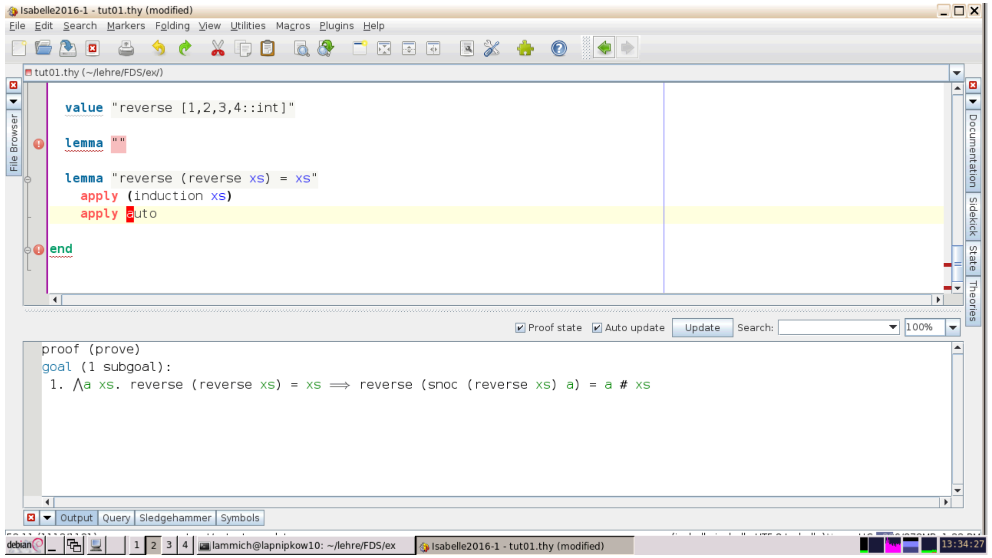

Isabelle2016-1 - tut01.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut01.thy (~/lehre/FDS/ex/)
fun snoc :: "'a list ⇒ 'a ⇒ 'a list" where
"snoc [] a = [a]"
| "snoc (x#xs) a = x # snoc xs a"
value "snoc [1,2,(3::int)] 5"
fun reverse :: "'a list ⇒ 'a list"
"reverse [] = []"
| "reverse (x#xs) = snco"
end
Proof state Auto update Update Search: 100%
Outer syntax error: keyword "where" expected,
but quoted string "reverse [] = []" was found
Output Query Sledgehammer Symbols
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut01.thy (modified)
13:29:55

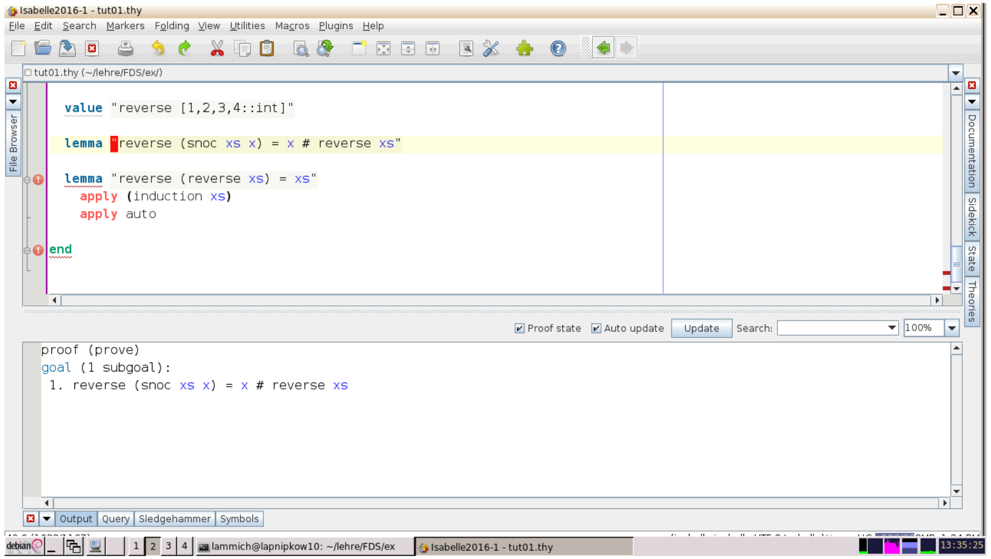

Isabelle2016-1 - tut01.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut01.thy (~/lehre/FDS/ex/)
value "snoc [1,2,(3::int)] 5"
fun reverse :: "'a list ⇒ 'a list" where
"reverse [] = []"
| "reverse (x#xs) = snoc (reverse xs) x"
value "reverse [1,2,3,4::int]"
end
Proof state Auto update Update Search: 100%
"[4, 3, 2, 1]"
:: "int list"
Output Query Sledgehammer Symbols
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut01.thy (modified)
13:31:01

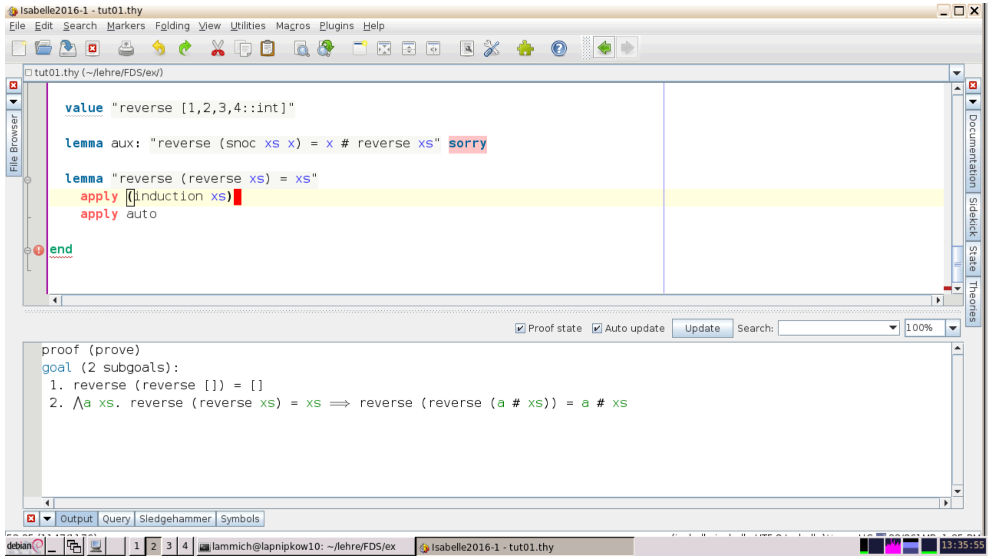

lammich@lapnipkow10: ~/lehre/FDS/ex
ex01.pdf
File Edit View Go Bookmarks Help
2 of 3
193,06%
Convince yourself on some test cases that your definition of snoc behaves as expected, for example run:
value "snoc [] c"
Also prove that your test cases are indeed correct, for instance show:
lemma "snoc [] c = [c]"
Next define a function reverse that reverses the order of elements in a list. (Do not use the existing function rev from the library.) Hint: Define the reverse of x # xs using the snoc function.
fun reverse :: "'a list ⇒ 'a list"
Demonstrate that your definition is correct by running some test cases, and proving that those test cases are correct. For example:
value "reverse [a, b, c]"
lemma "reverse [a, b, c] = [c, b, a]"
Prove the following theorem. Hint: You need to find an additional lemma relating reverse and snoc to prove it.
theorem "reverse (reverse xs) = xs"
lammich@lapnipkow10: ~/lehre/FDS/ex
ex01.pdf
13:31:39

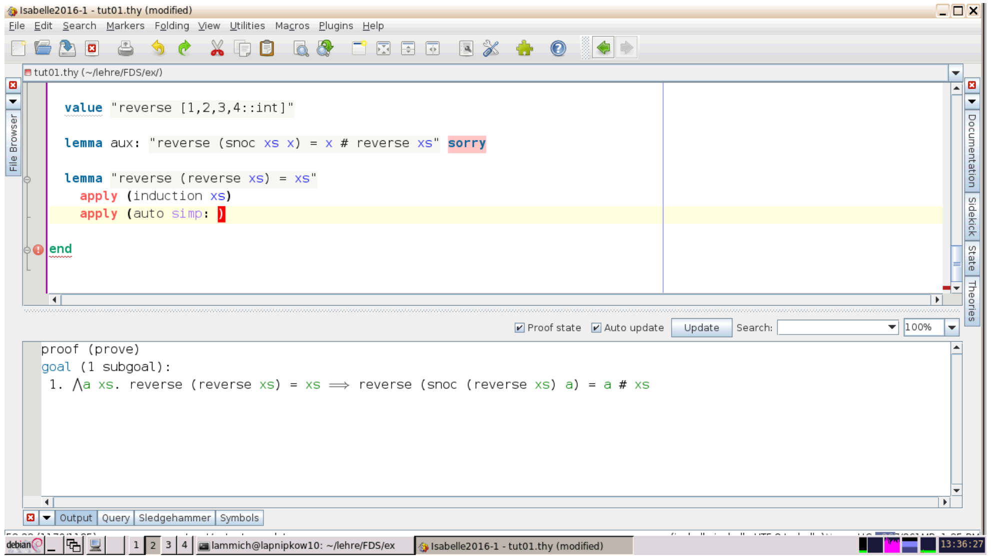

Isabelle2016-1 - tut01.thy (modified)
File Edit Search Markers Folding View Utilities Macros Plugins Help
tut01.thy (~/lehre/FDS/ex/)
fun reverse :: "'a list ⇒ 'a list" where
"reverse [] = []"
| "reverse (x#xs) = snoc (reverse xs) x"
value "reverse [1,2,3,4::int]"
lemma "reverse (reverse xs) = xs"
appl
apply (keyword)
end
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. reverse (reverse xs) = xs
Output Query Sledgehammer Symbols
lammich@lapnipkow10: ~/lehre/FDS/ex
Isabelle2016-1 - tut01.thy (modified)
13:32:29

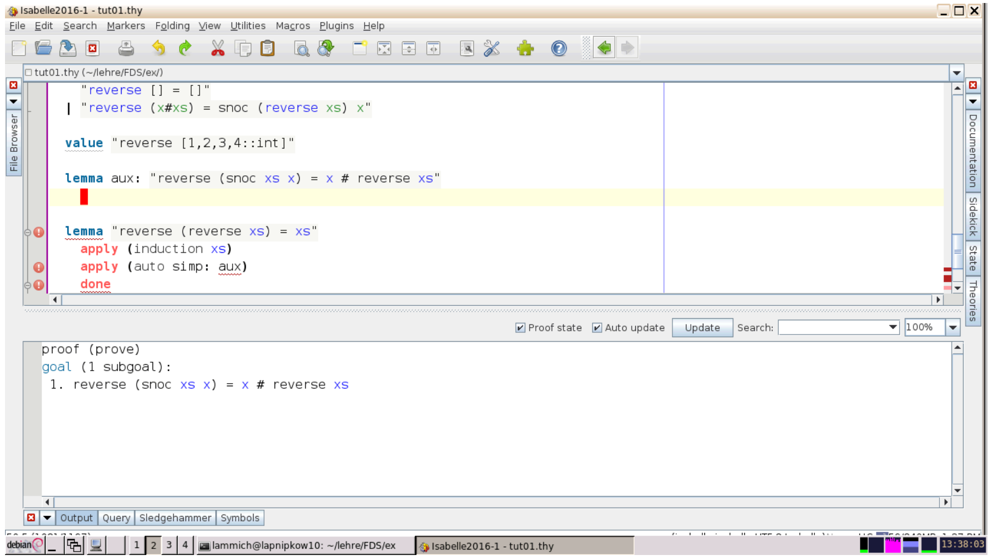

Isabelle2016-1 - tut01.thy
"reverse [] = []"
| "reverse (x#xs) = snoc (reverse xs) x"
value "reverse [1,2,3,4::int]"
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply auto
end
"[4, 3, 2, 1]"
:: "int list"

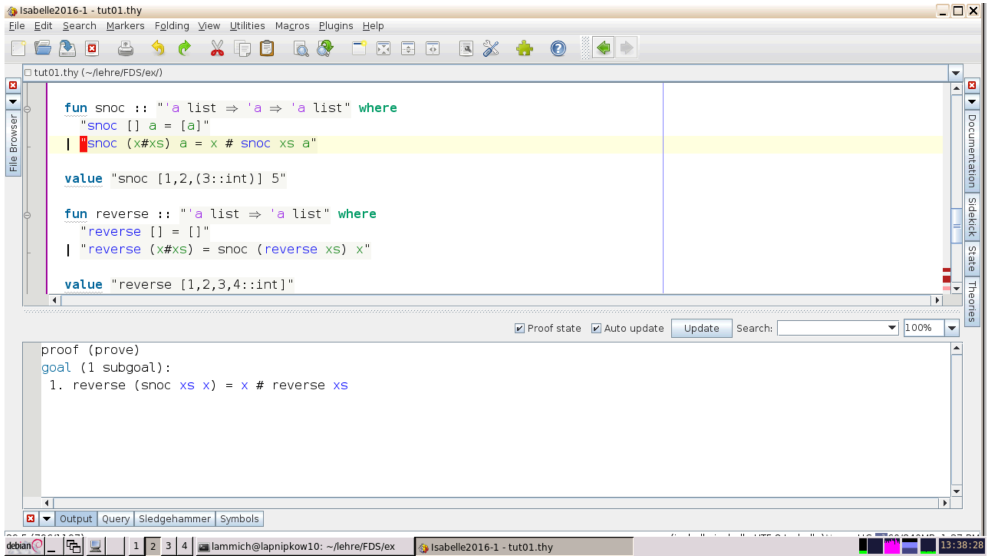

Isabelle2016-1 - tut01.thy (modified)
value "reverse [1,2,3,4::int]"
lemma ""
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply auto
end
proof (prove)
goal (1 subgoal):
1. ⋀a xs. reverse (reverse xs) = xs ⟹ reverse (snoc (reverse xs) a) = a # xs

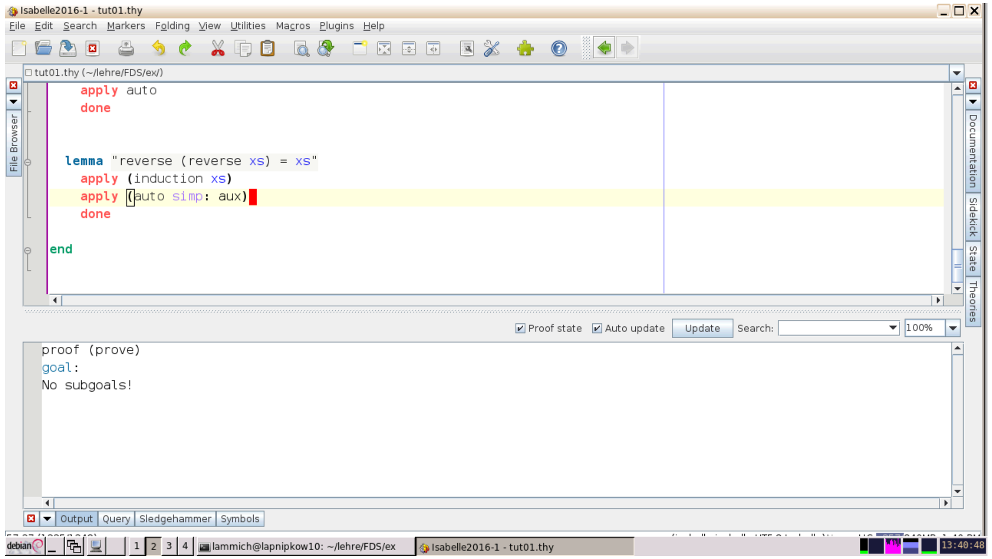

Isabelle2016-1 - tut01.thy
value "reverse [1,2,3,4::int]"
lemma "reverse (snoc xs x) = x # reverse xs"
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply auto
end
proof (prove)
goal (1 subgoal):
1. reverse (snoc xs x) = x # reverse xs


Isabelle2016-1 - tut01.thy
value "reverse [1,2,3,4::int]"
lemma aux: "reverse (snoc xs x) = x # reverse xs" sorry
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply auto
end
proof (prove)
goal (2 subgoals):
1. reverse (reverse []) = []
2. ⋀a xs. reverse (reverse xs) = xs ⟹ reverse (reverse (a # xs)) = a # xs


Isabelle2016-1 - tut01.thy (modified)
value "reverse [1,2,3,4::int]"
lemma aux: "reverse (snoc xs x) = x # reverse xs" sorry
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply (auto simp:)
end
proof (prove)
goal (1 subgoal):
1. ⋀a xs. reverse (reverse xs) = xs ⟹ reverse (snoc (reverse xs) a) = a # xs


Isabelle2016-1 - tut01.thy
"reverse [] = []"
| "reverse (x#xs) = snoc (reverse xs) x"
value "reverse [1,2,3,4::int]"
lemma aux: "reverse (snoc xs x) = x # reverse xs"
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply (auto simp: aux)
done
proof (prove)
goal (1 subgoal):
1. reverse (snoc xs x) = x # reverse xs


Isabelle2016-1 - tut01.thy
fun snoc :: "'a list ⇒ 'a ⇒ 'a list" where
"snoc [] a = [a]"
| "snoc (x#xs) a = x # snoc xs a"
value "snoc [1,2,(3::int)] 5"
fun reverse :: "'a list ⇒ 'a list" where
"reverse [] = []"
| "reverse (x#xs) = snoc (reverse xs) x"
value "reverse [1,2,3,4::int]"
proof (prove)
goal (1 subgoal):
1. reverse (snoc xs x) = x # reverse xs


Isabelle2016-1 - tut01.thy
apply auto
done
lemma "reverse (reverse xs) = xs"
apply (induction xs)
apply (auto simp: aux)
done
end
proof (prove)
goal:
No subgoals!