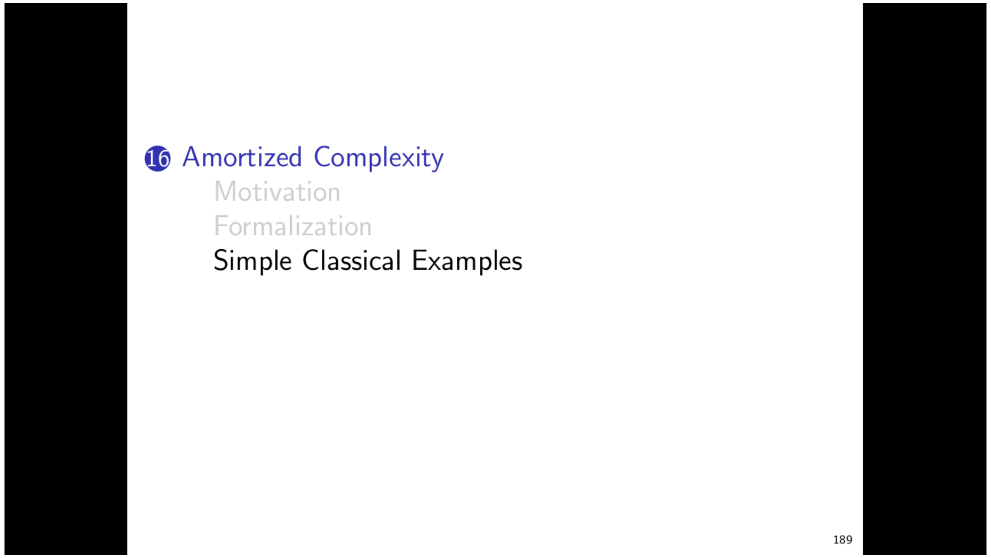
Script generated by TTT

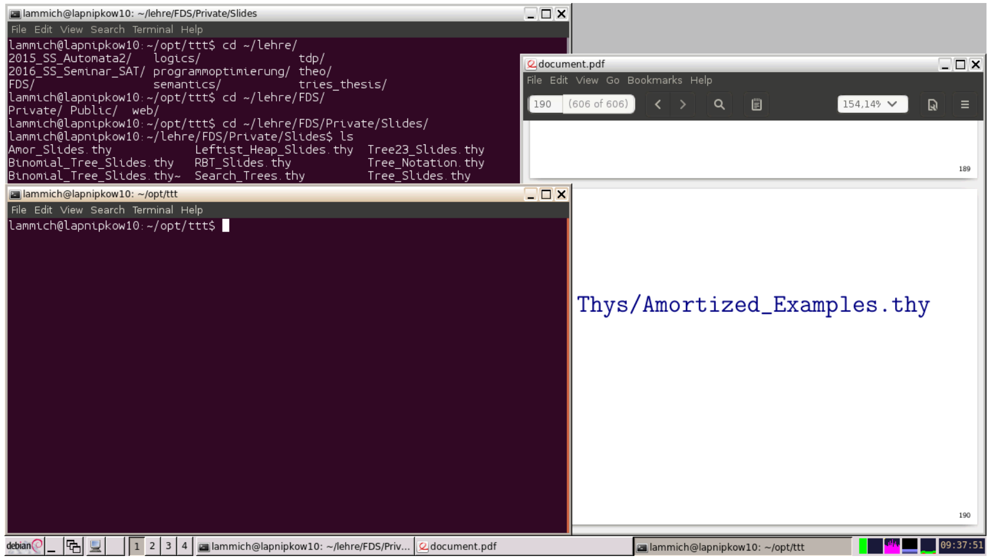
Title: Lammich: FDS Tutorial (14.07.2017)

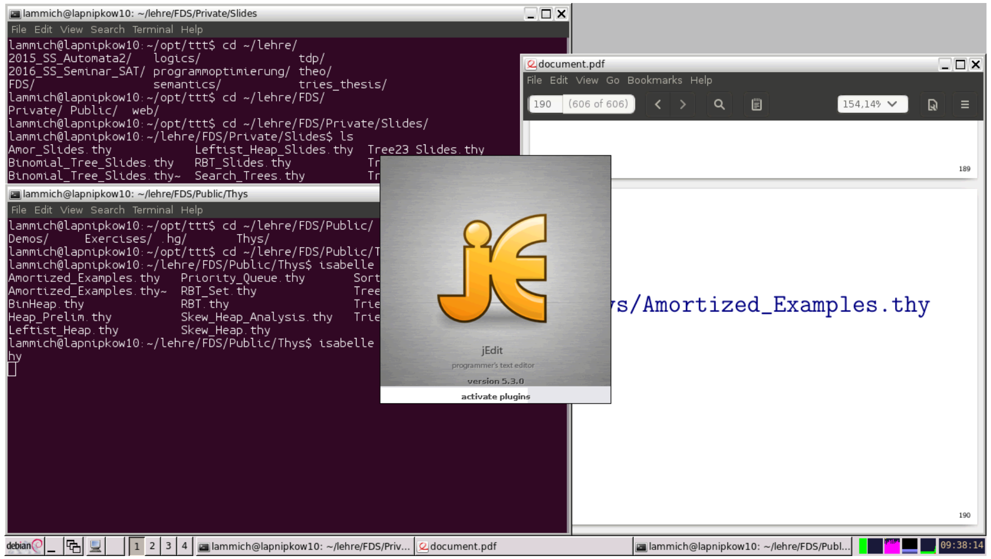
Date: Fri Jul 14 09:37:05 CEST 2017

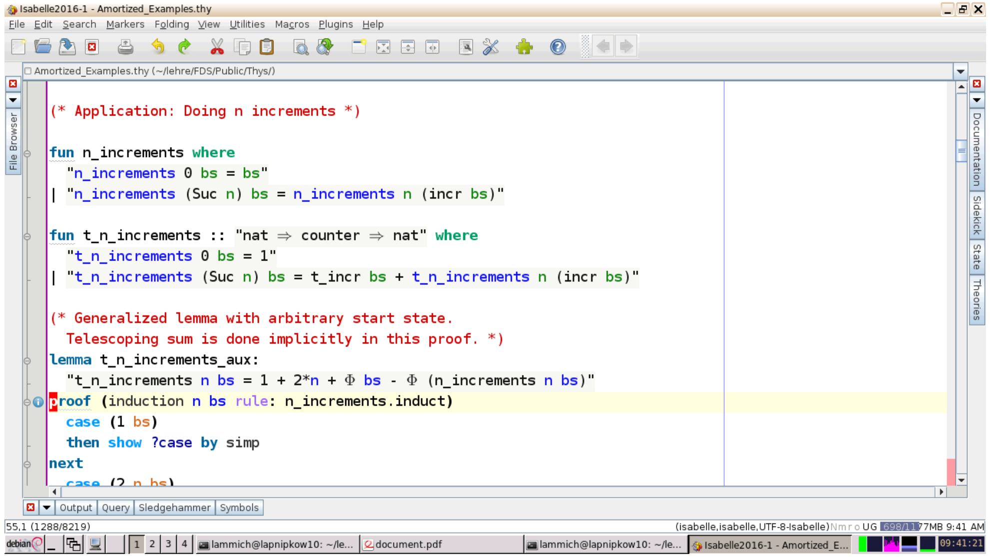
Duration: 25:51 min

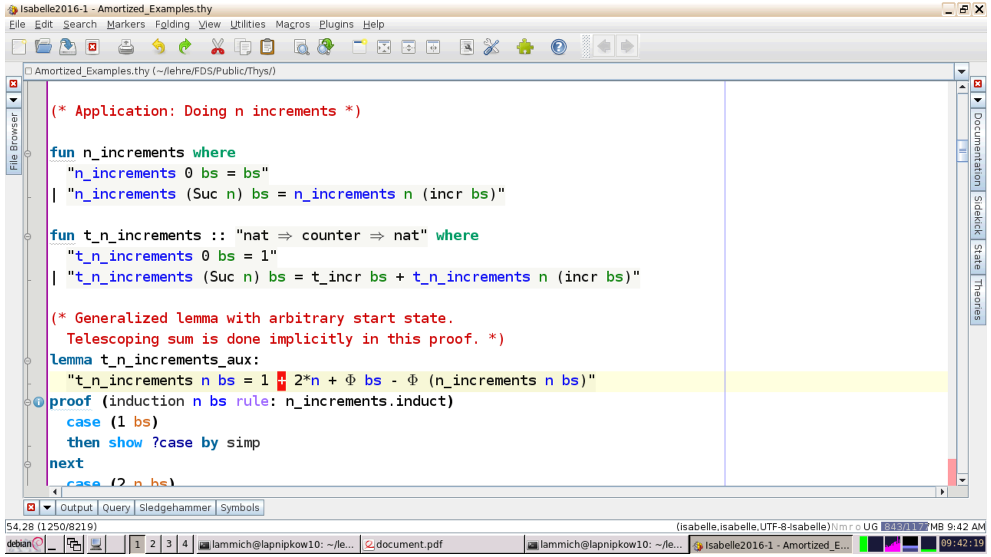
Pages: 23

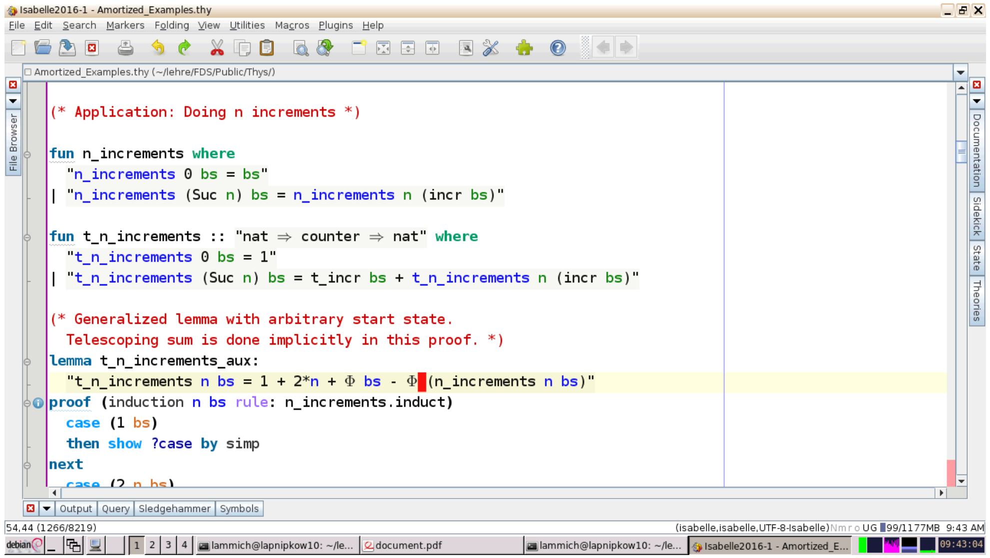


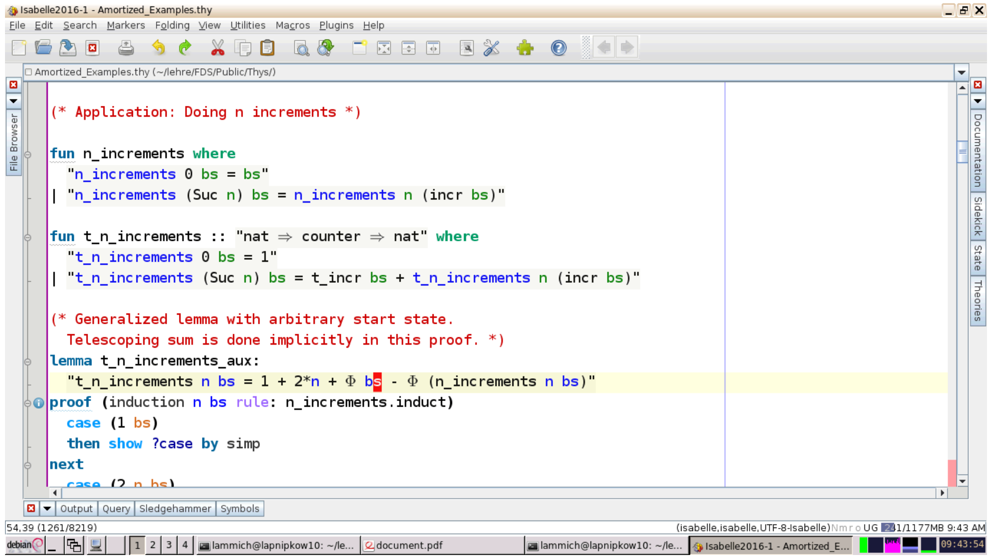


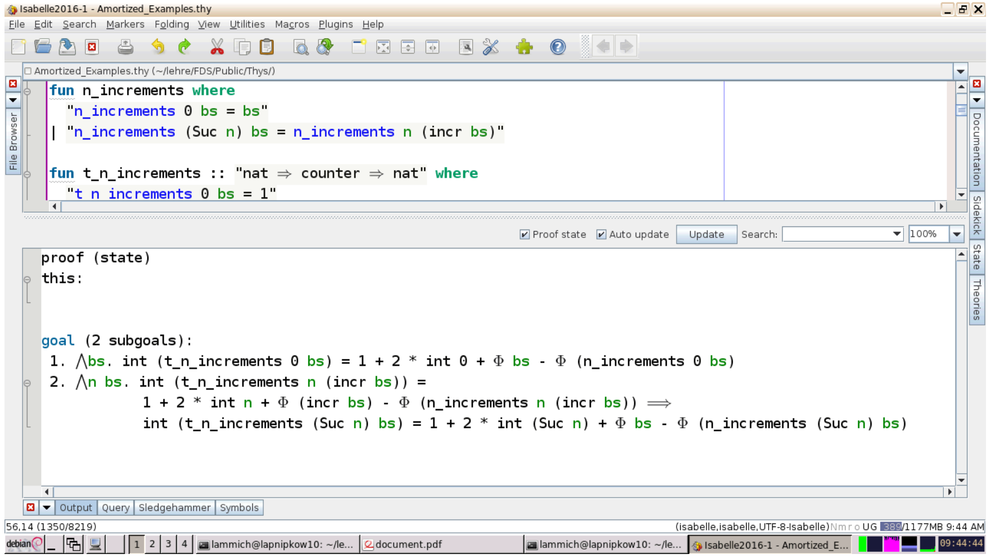



Isabelle2016-1 - Amortized_Examples.thy
(* Application: Doing n increments *)
fun n_increments where
"n_increments 0 bs = bs"
| "n_increments (Suc n) bs = n_increments n (incr bs)"
fun t_n_increments :: "nat ⇒ counter ⇒ nat" where
"t_n_increments 0 bs = 1"
| "t_n_increments (Suc n) bs = t_incr bs + t_n_increments n (incr bs)"
(* Generalized lemma with arbitrary start state.
Telescoping sum is done implicitly in this proof. *)
lemma t_n_increments_aux:
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
Output Query Sledgehammer Symbols
55.1 (1288/8219)

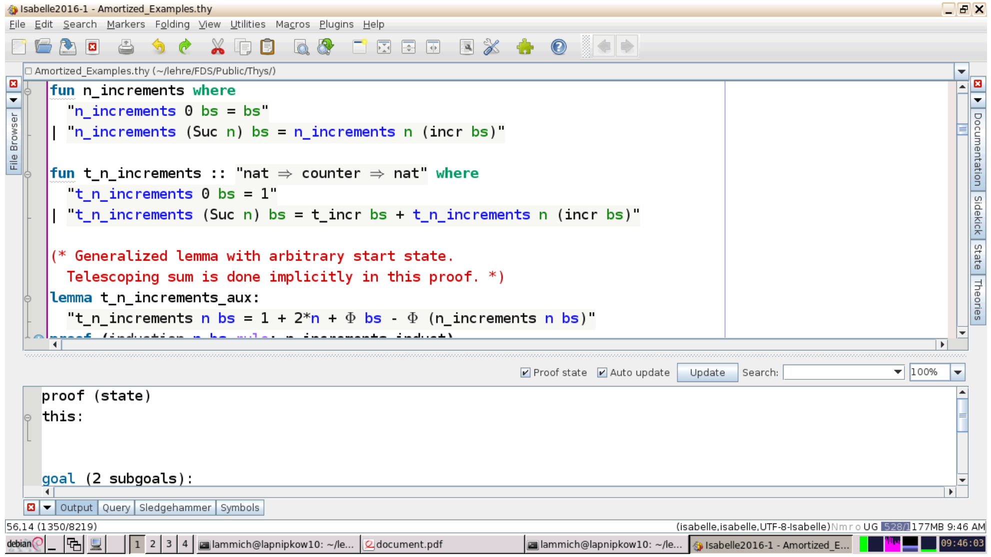

Isabelle2016-1 - Amortized_Examples.thy
(* Application: Doing n increments *)
fun n_increments where
"n_increments 0 bs = bs"
| "n_increments (Suc n) bs = n_increments n (incr bs)"
fun t_n_increments :: "nat ⇒ counter ⇒ nat" where
"t_n_increments 0 bs = 1"
| "t_n_increments (Suc n) bs = t_incr bs + t_n_increments n (incr bs)"
(* Generalized lemma with arbitrary start state.
Telescoping sum is done implicitly in this proof. *)
lemma t_n_increments_aux:
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
Output Query Sledgehammer Symbols
54.28 (1250/8219)


Isabelle2016-1 - Amortized_Examples.thy
(* Application: Doing n increments *)
fun n_increments where
"n_increments 0 bs = bs"
| "n_increments (Suc n) bs = n_increments n (incr bs)"
fun t_n_increments :: "nat ⇒ counter ⇒ nat" where
"t_n_increments 0 bs = 1"
| "t_n_increments (Suc n) bs = t_incr bs + t_n_increments n (incr bs)"
(* Generalized lemma with arbitrary start state.
Telescoping sum is done implicitly in this proof. *)
lemma t_n_increments_aux:
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
Output Query Sledgehammer Symbols
54.44 (1266/8219)

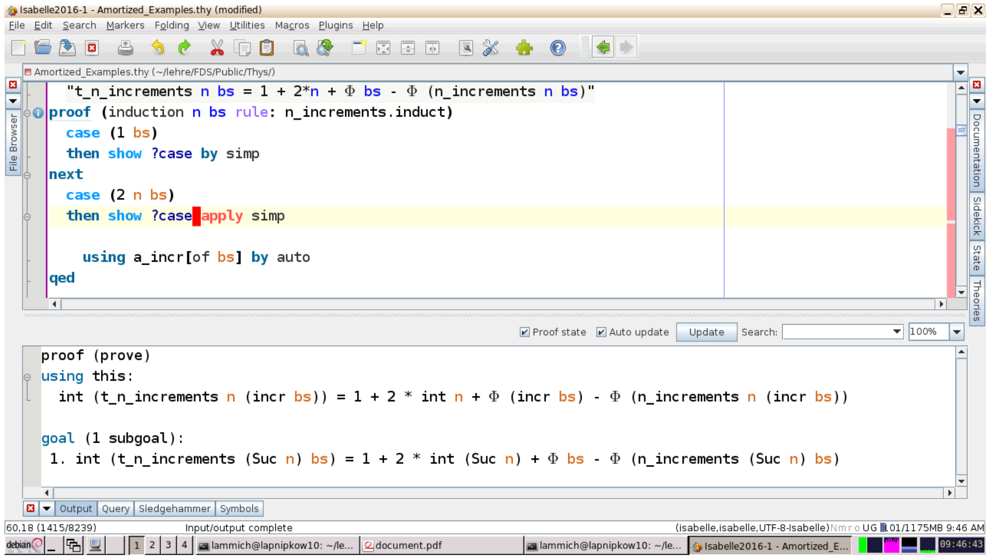

Isabelle2016-1 - Amortized_Examples.thy
(* Application: Doing n increments *)
fun n_increments where
"n_increments 0 bs = bs"
| "n_increments (Suc n) bs = n_increments n (incr bs)"
fun t_n_increments :: "nat ⇒ counter ⇒ nat" where
"t_n_increments 0 bs = 1"
| "t_n_increments (Suc n) bs = t_incr bs + t_n_increments n (incr bs)"
(* Generalized lemma with arbitrary start state.
Telescoping sum is done implicitly in this proof. *)
lemma t_n_increments_aux:
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
Output Query Sledgehammer Symbols
54.39 (1261/8219)



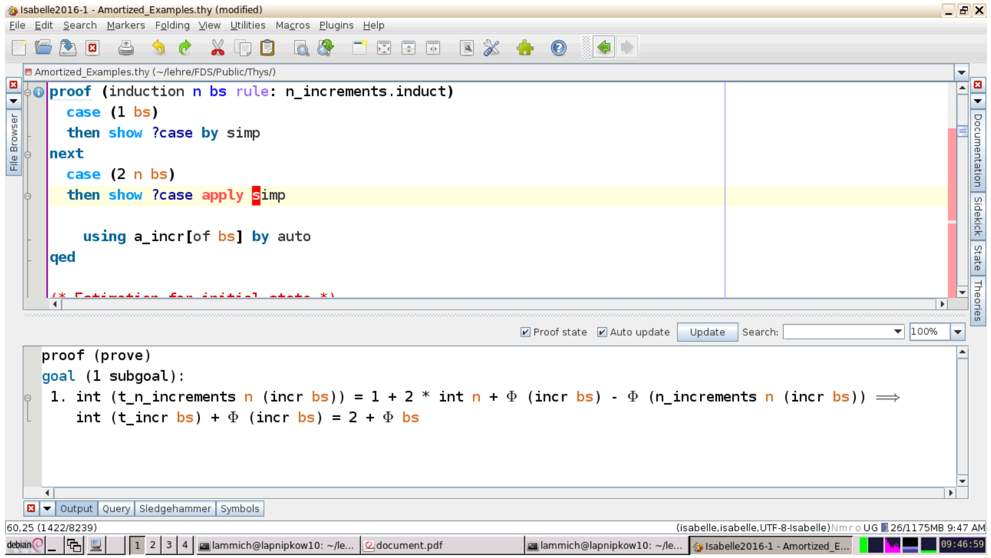
Isabelle2016-1 - Amortized_Examples.thy
fun n_increments where
"n_increments 0 bs = bs"
| "n_increments (Suc n) bs = n_increments n (incr bs)"
fun t_n_increments :: "nat ⇒ counter ⇒ nat" where
"t_n_increments 0 bs = 1"
proof (state)
this:
goal (2 subgoals):
1. ⋀bs. int (t_n_increments 0 bs) = 1 + 2 * int 0 + Φ bs - Φ (n_increments 0 bs)
2. ⋀n bs. int (t_n_increments n (incr bs)) =
1 + 2 * int n + Φ (incr bs) - Φ (n_increments n (incr bs)) ⟹
int (t_n_increments (Suc n) bs) = 1 + 2 * int (Suc n) + Φ bs - Φ (n_increments (Suc n) bs)



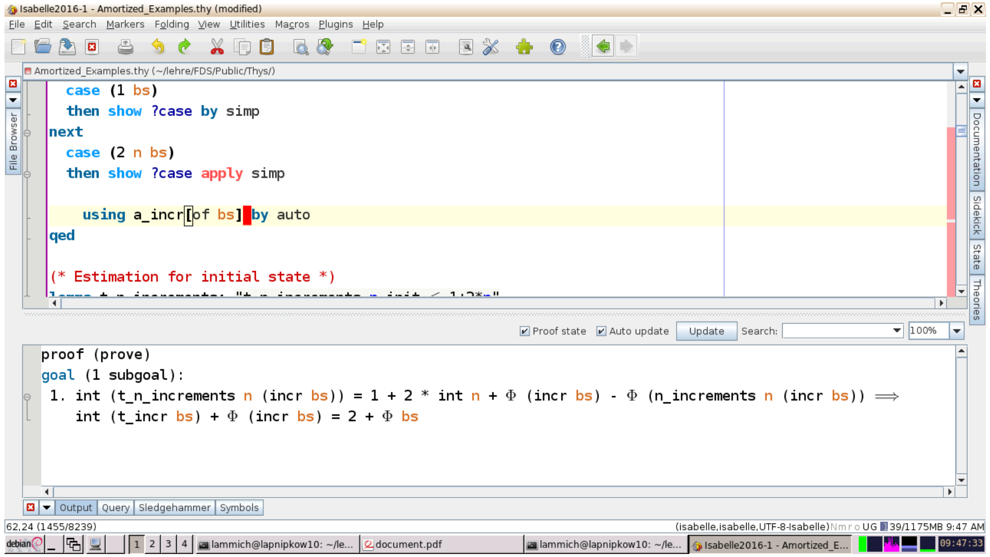
Isabelle2016-1 - Amortized_Examples.thy
fun n_increments where
"n_increments 0 bs = bs"
| "n_increments (Suc n) bs = n_increments n (incr bs)"
fun t_n_increments :: "nat ⇒ counter ⇒ nat" where
"t_n_increments 0 bs = 1"
| "t_n_increments (Suc n) bs = t_incr bs + t_n_increments n (incr bs)"
(* Generalized lemma with arbitrary start state.
Telescoping sum is done implicitly in this proof. *)
lemma t_n_increments_aux:
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (state)
this:
goal (2 subgoals):

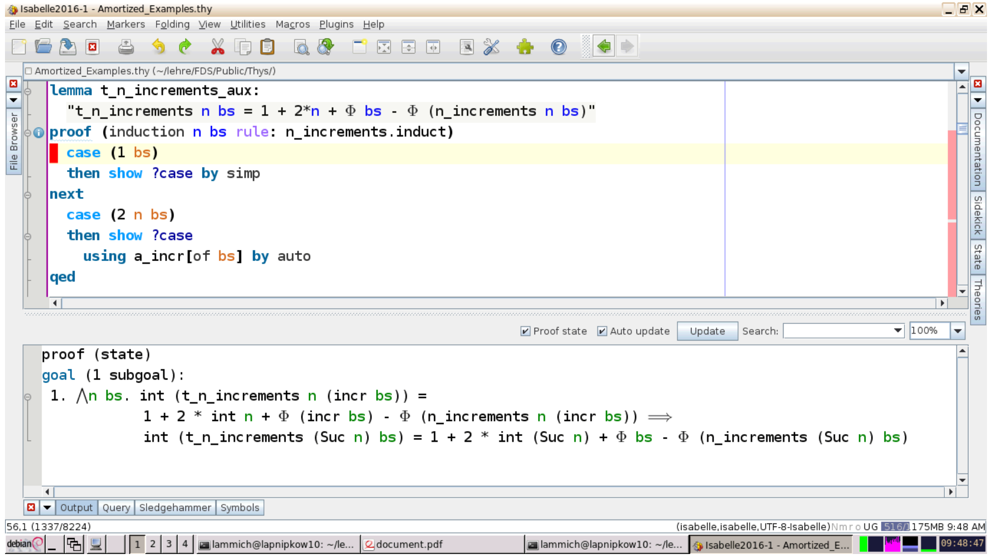


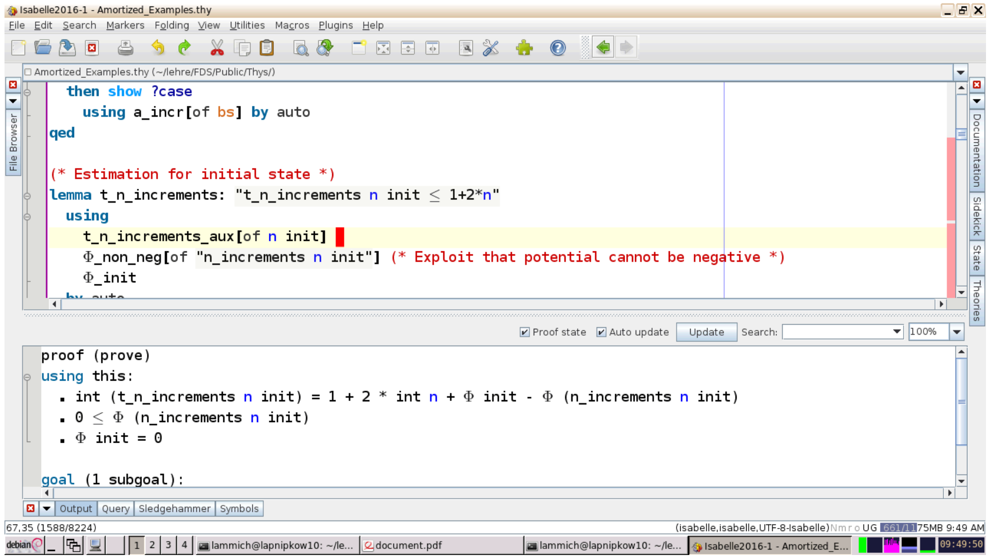
Isabelle2016-1 - Amortized_Examples.thy (modified)
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
then show ?case apply simp
using a_incr[of bs] by auto
qed
proof (prove)
using this:
int (t_n_increments n (incr bs)) = 1 + 2 * int n + Φ (incr bs) - Φ (n_increments n (incr bs))
goal (1 subgoal):
1. int (t_n_increments (Suc n) bs) = 1 + 2 * int (Suc n) + Φ bs - Φ (n_increments (Suc n) bs)

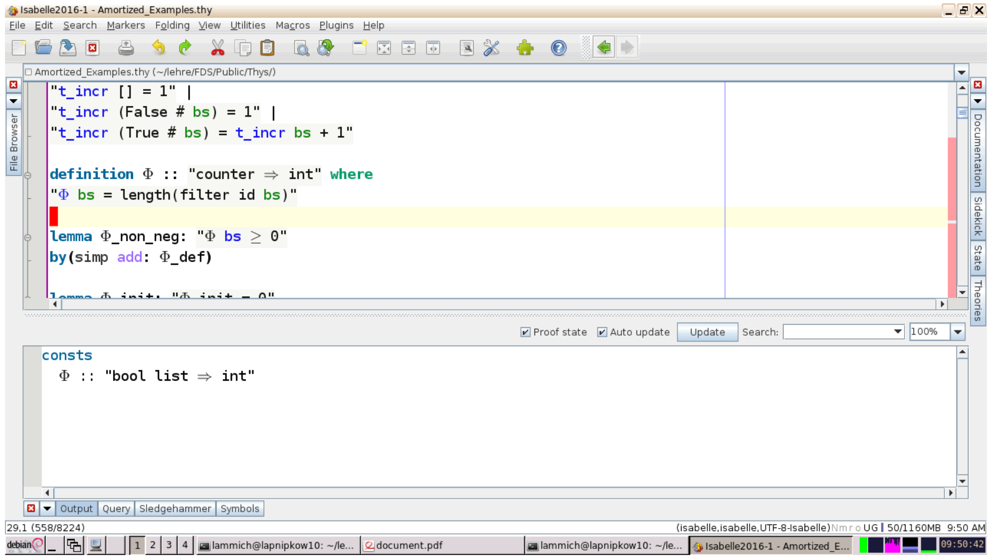


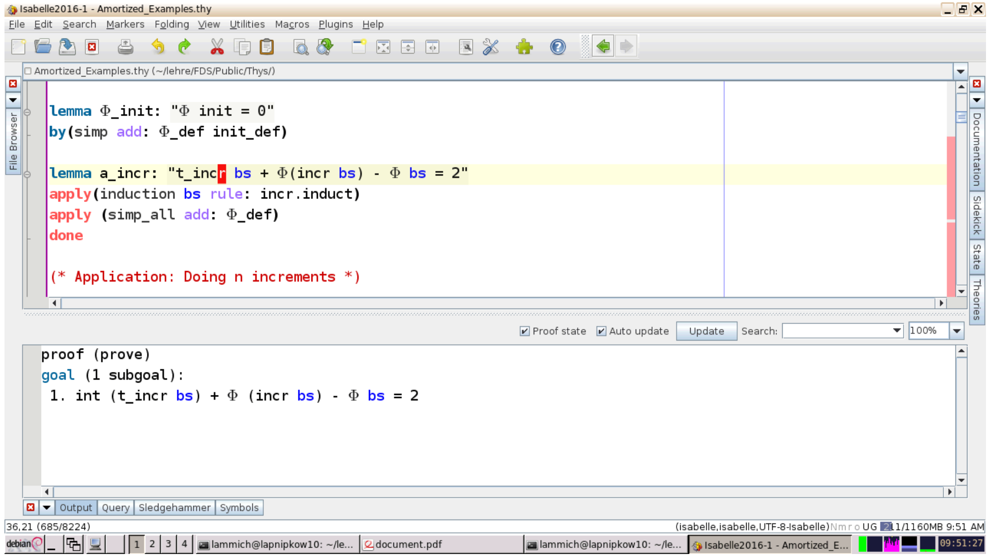
Isabelle2016-1 - Amortized_Examples.thy (modified)
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
then show ?case apply simp
using a_incr[of bs] by auto
qed
proof (prove)
goal (1 subgoal):
1. int (t_n_increments n (incr bs)) = 1 + 2 * int n + Φ (incr bs) - Φ (n_increments n (incr bs)) ⟹
int (t_incr bs) + Φ (incr bs) = 2 + Φ bs

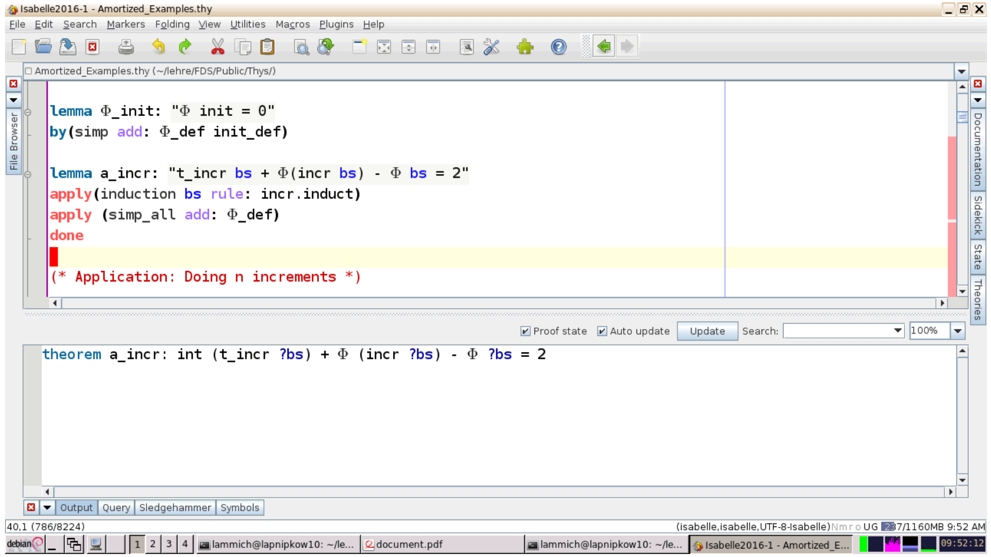

Isabelle2016-1 - Amortized_Examples.thy (modified)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
then show ?case apply simp
using a_incr[of bs] by auto
qed
(* Estimation for initial state *)
proof (prove)
goal (1 subgoal):
1. int (t_n_increments n (incr bs)) = 1 + 2 * int n + Φ (incr bs) - Φ (n_increments n (incr bs)) ⟹
int (t_incr bs) + Φ (incr bs) = 2 + Φ bs

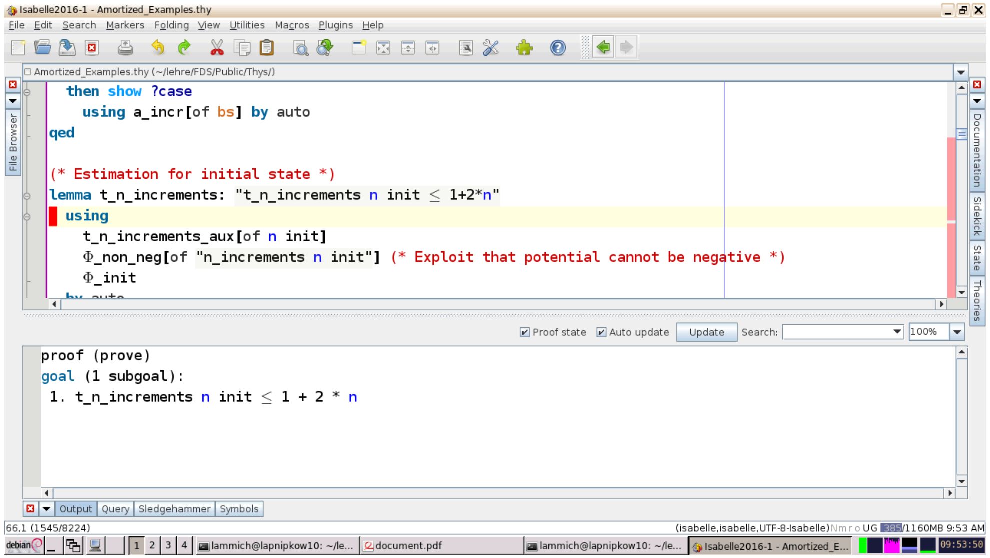

Isabelle2016-1 - Amortized_Examples.thy
lemma t_n_increments_aux:
"t_n_increments n bs = 1 + 2*n + Φ bs - Φ (n_increments n bs)"
proof (induction n bs rule: n_increments.induct)
case (1 bs)
then show ?case by simp
next
case (2 n bs)
then show ?case
using a_incr[of bs] by auto
qed
proof (state)
goal (1 subgoal):
1. ⋀n bs. int (t_n_increments n (incr bs)) =
1 + 2 * int n + Φ (incr bs) - Φ (n_increments n (incr bs)) ⟹
int (t_n_increments (Suc n) bs) = 1 + 2 * int (Suc n) + Φ bs - Φ (n_increments (Suc n) bs)

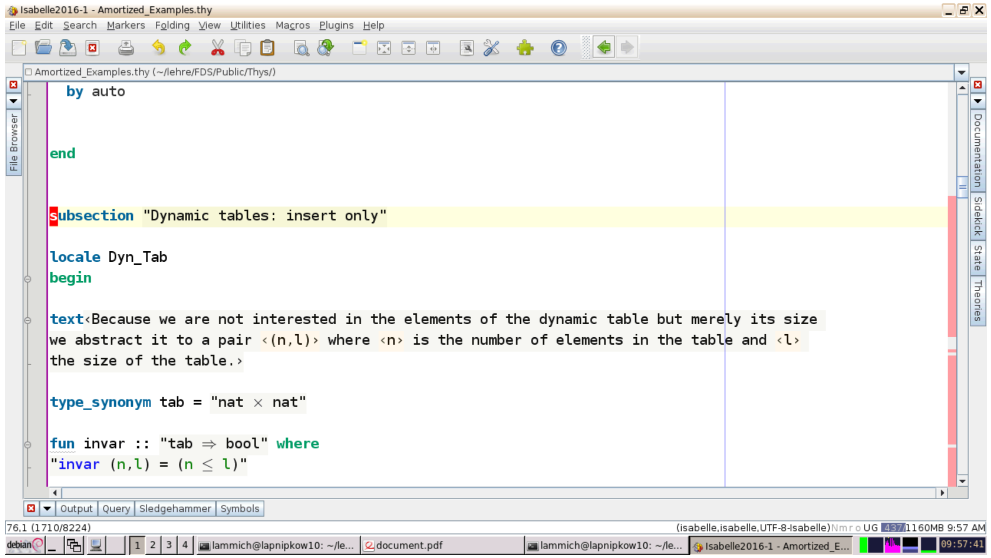

Isabelle2016-1 - Amortized_Examples.thy
then show ?case
using a_incr[of bs] by auto
qed
(* Estimation for initial state *)
lemma t_n_increments: "t_n_increments n init ≤ 1+2*n"
using
t_n_increments_aux[of n init]
Φ_non_neg[of "n_increments n init"] (* Exploit that potential cannot be negative *)
Φ_init
proof (prove)
using this:
int (t_n_increments n init) = 1 + 2 * int n + Φ init - Φ (n_increments n init)
0 ≤ Φ (n_increments n init)
Φ init = 0
goal (1 subgoal):

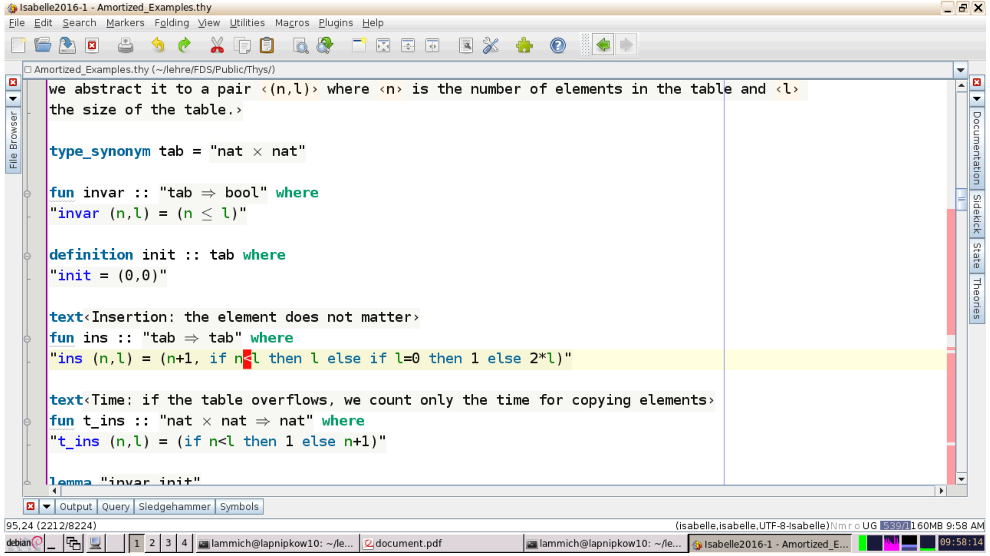

Isabelle2016-1 - Amortized_Examples.thy
"t_incr [] = 1" |
"t_incr (False # bs) = 1" |
"t_incr (True # bs) = t_incr bs + 1"
definition Φ :: "counter ⇒ int" where
"Φ bs = length(filter id bs)"
lemma Φ_non_neg: "Φ bs ≥ 0"
by(simp add: Φ_def)
consts
Φ :: "bool list ⇒ int"

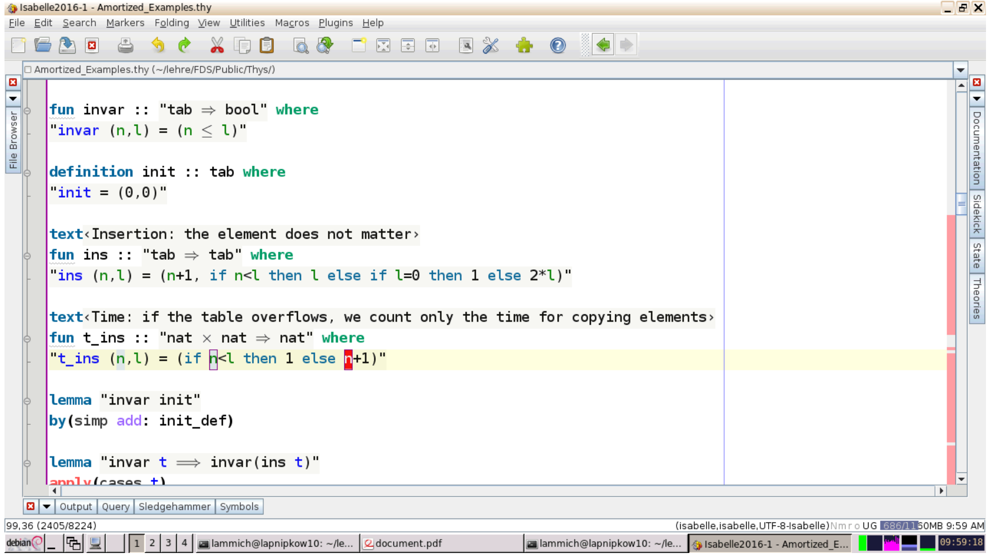

Isabelle2016-1 - Amortized_Examples.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Amortized_Examples.thy (~/lehre/FDS/Public/Thys/)
lemma Φ_init: "Φ init = 0"
by(simp add: Φ_def init_def)
lemma a_incr: "t_incr bs + Φ(incr bs) - Φ bs = 2"
apply(induction bs rule: incr.induct)
apply (simp_all add: Φ_def)
done
(* Application: Doing n increments *)
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. int (t_incr bs) + Φ (incr bs) - Φ bs = 2
Output Query Sledgehammer Symbols
36,21 (685/8224)
(isabelle,isabelle,UTF-8-Isabelle)
09:51:27

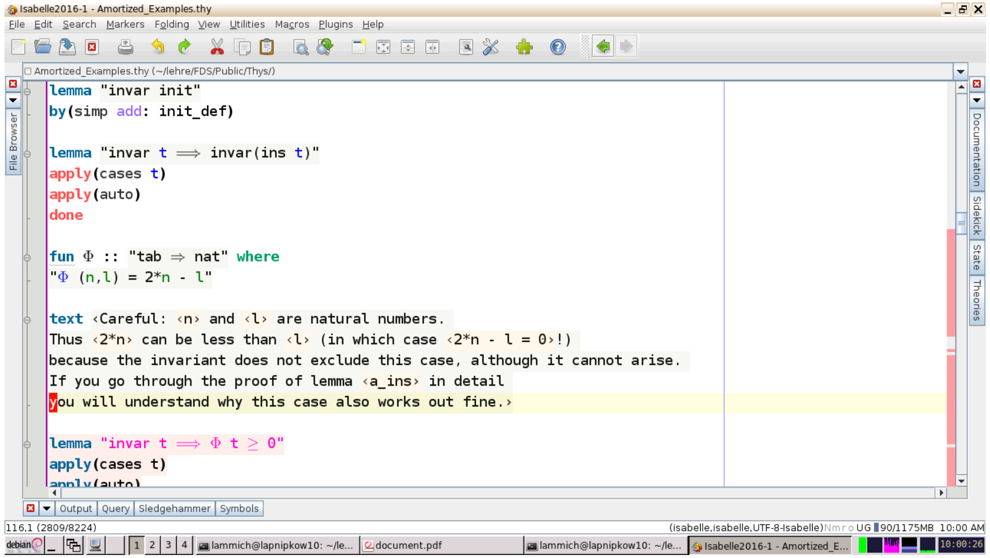

Isabelle2016-1 - Amortized_Examples.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Amortized_Examples.thy (~/lehre/FDS/Public/Thys/)
lemma Φ_init: "Φ init = 0"
by(simp add: Φ_def init_def)
lemma a_incr: "t_incr bs + Φ(incr bs) - Φ bs = 2"
apply(induction bs rule: incr.induct)
apply (simp_all add: Φ_def)
done
(* Application: Doing n increments *)
Proof state Auto update Update Search: 100%
theorem a_incr: int (t_incr ?bs) + Φ (incr ?bs) - Φ ?bs = 2
Output Query Sledgehammer Symbols
40,1 (786/8224)
(isabelle,isabelle,UTF-8-Isabelle)
09:52:12

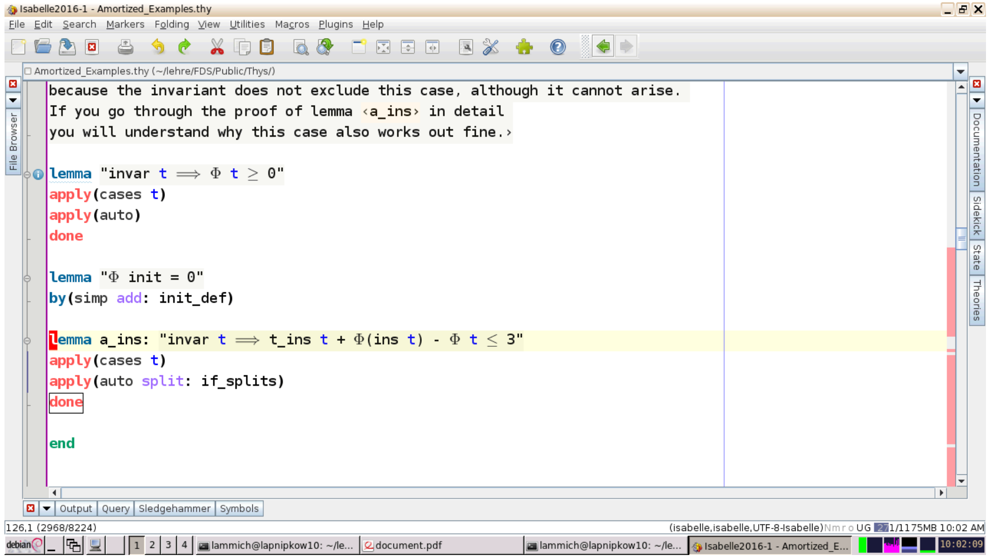

Isabelle2016-1 - Amortized_Examples.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Amortized_Examples.thy (~/lehre/FDS/Public/Thys/)
then show ?case
using a_incr[of bs] by auto
qed
(* Estimation for initial state *)
lemma t_n_increments: "t_n_increments n init ≤ 1+2*n"
using
t_n_increments_aux[of n init]
Φ_non_neg[of "n_increments n init"] (* Exploit that potential cannot be negative *)
Φ_init
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. t_n_increments n init ≤ 1 + 2 * n
Output Query Sledgehammer Symbols
66,1 (1545/8224)
(isabelle,isabelle,UTF-8-Isabelle)
09:53:50


Isabelle2016-1 - Amortized_Examples.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
Amortized_Examples.thy (~/lehre/FDS/Public/Thys/)
by auto
end
subsection "Dynamic tables: insert only"
locale Dyn_Tab
begin
text‹Because we are not interested in the elements of the dynamic table but merely its size
we abstract it to a pair ‹(n,l)› where ‹n› is the number of elements in the table and ‹l›
the size of the table.›
type_synonym tab = "nat × nat"
fun invar :: "tab ⇒ bool" where
"invar (n,l) = (n ≤ l)"
Output Query Sledgehammer Symbols
76,1 (1710/8224)
(isabelle,isabelle,UTF-8-Isabelle)
09:57:41


Isabelle2016-1 - Amortized_Examples.thy
we abstract it to a pair ‹(n,l)› where ‹n› is the number of elements in the table and ‹l›
the size of the table.›
type_synonym tab = "nat × nat"
fun invar :: "tab ⇒ bool" where
"invar (n,l) = (n ≤ l)"
definition init :: tab where
"init = (0,0)"
text‹Insertion: the element does not matter›
fun ins :: "tab ⇒ tab" where
"ins (n,l) = (n+1, if n<l then l else if l=0 then 1 else 2*l)"
text‹Time: if the table overflows, we count only the time for copying elements›
fun t_ins :: "nat × nat ⇒ nat" where
"t_ins (n,l) = (if n<l then 1 else n+1)"


Isabelle2016-1 - Amortized_Examples.thy
fun invar :: "tab ⇒ bool" where
"invar (n,l) = (n ≤ l)"
definition init :: tab where
"init = (0,0)"
text‹Insertion: the element does not matter›
fun ins :: "tab ⇒ tab" where
"ins (n,l) = (n+1, if n<l then l else if l=0 then 1 else 2*l)"
text‹Time: if the table overflows, we count only the time for copying elements›
fun t_ins :: "nat × nat ⇒ nat" where
"t_ins (n,l) = (if n<l then 1 else n+1)"
lemma "invar init"
by(simp add: init_def)
lemma "invar t ⟹ invar(ins t)"


Isabelle2016-1 - Amortized_Examples.thy
lemma "invar init"
by(simp add: init_def)
lemma "invar t ⟹ invar(ins t)"
apply(cases t)
apply(auto)
done
fun Φ :: "tab ⇒ nat" where
"Φ (n,l) = 2*n - l"
text ‹Careful: ‹n› and ‹l› are natural numbers.
Thus ‹2*n› can be less than ‹l› (in which case ‹2*n - l = 0›!)
because the invariant does not exclude this case, although it cannot arise.
If you go through the proof of lemma ‹a_ins› in detail
you will understand why this case also works out fine.›
lemma "invar t ⟹ Φ t ≥ 0"
apply(cases t)


Isabelle2016-1 - Amortized_Examples.thy
because the invariant does not exclude this case, although it cannot arise.
If you go through the proof of lemma ‹a_ins› in detail
you will understand why this case also works out fine.›
lemma "invar t ⟹ Φ t ≥ 0"
apply(cases t)
apply(auto)
done
lemma "Φ init = 0"
by(simp add: init_def)
lemma a_ins: "invar t ⟹ t_ins t + Φ(ins t) - Φ t ≤ 3"
apply(cases t)
apply(auto split: if_splits)
done
end