Title: Lammich: FDS Tutorial (14.07.2017)
Date: Fri Jul 14 09:37:05 CEST 2017
Duration: 25:51 min
Pages: 23
(* 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 \to \nat \to \nat where
  "t_n_increments 0 bs = 1"
| "t_n_increments (Suc n) bs = t_n_incr bs + \sum n_increments n (incr bs)"

(* General lemma with arbitrary start state. Telescoping sum is done implicitly in this proof. *)

lemma t_n_increments_aux: 
  "t_n_incréments n bs = 1 + \sum n_increments n bs"
proof (induction n bs rule: n_increments.induct)
  case (1 bs)
  then show \false by simp
next
  case \_ \_ bs
proof (induction n bs rule: n_increments.induct)
  case (1 bs)
  then show \false by simp
next
  case \_ \_ \_ bs

(* General lemma with arbitrary start state. Telescoping sum is done implicitly in this proof. *)

lemma t_n_increments_aux: 
  "t_n_incréments n bs = 1 + \sum n_increments n bs"
proof (induction n bs rule: n_increments.induct)
  case (1 bs)
  then show \false by simp
next
  case \_ \_ bs
proof (induction n bs rule: n_increments.induct)
  case (1 bs)
  then show \false by simp
next
  case \_ \_ \_ bs
proof (state)
this:

goal (2 subgoals):  
1. (∀n. nat → counter → nat) = 1 * 2 * int 0 * φ bs - φ (n increments 0 bs) 
2. (∀n bs. Int (t_n_increments n 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)

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)
proof (prove):

goal (1 subgoal):
1. \[ \text{int } (\text{t_n_increments } n \text{ incr bs}) = 1 + 2 ^ n \text{ int } n + 0 \text{ (incr bs)} - 0 \text{ (n_increments } n \text{ incr bs)} \]

then show \text{?case} by simp
using \text{a_incr[of bs]} by auto

qed

(" Estimation for initial state ")

lemma \text{t_n_increments}:
\[ \text{t_n_increments } n \text{ init} \leq 1 + 2 ^ n \]
using \text{t_n_increments_aux[of init]}
\[ 0 \text{ _non_neg[of (t_n_increments } n \text{ init)}] \]
\[ \text{t_init} \]
(" Explicit that potential cannot be negative ")

proof (prove):

using this:
\[ \text{int } (\text{t_n_increments } n \text{ init}) = 1 + 2 ^ n \text{ int } n + 0 \text{ (n_increments } n \text{ init)} \]
\[ 0 \leq 0 \text{ (n_increments } n \text{ init)} \]
\[ \text{init} = 0 \]

goal (1 subgoal):

lemma \text{t_n_increments}:
\[ \text{t_n_increments } n \text{ init} \leq 1 + 2 ^ n \]
using \text{t_n_increments_aux[of init]}
\[ 0 \text{ _non_neg[of (t_n_increments } n \text{ init)}] \]
"Explicit that potential cannot be negative"
\[ \text{t_init} \]

proof (prove):

using this:
\[ \text{int } (\text{t_n_increments } n \text{ init}) = 1 + 2 ^ n \text{ int } n + 0 \text{ (n_increments } n \text{ init)} \]
\[ 0 \leq 0 \text{ (n_increments } n \text{ init)} \]
\[ \text{init} = 0 \]

goal (1 subgoal):

lemma \text{t_n_increments} aux:
\[ \text{t_n_increments } n \text{ bs} = 1 + 2 ^ n \text{ int } n + 0 \text{ (n_increments } n \text{ bs)} \]

proof (induction \text{ n bs } rule: \text{n_increments.induct})

then show \text{?case} by simp
next
case (2 \text{ n bs})
then show \text{?case} apply simp
using \text{a_incr[of bs]} by auto

qed

(" Estimation for initial state ")

proof (prove):

goal (1 subgoal):

\[ \text{int } (\text{t_n_increments } n \text{ incr bs}) = 1 + 2 ^ n \text{ int } n + 0 \text{ (incr bs)} - 0 \text{ (n_increments } n \text{ incr bs)} \]

then show \text{?case} by simp
using \text{a_incr[of bs]} by auto

qed

(" Estimation for initial state ")
proof (prove)

goal (1 subgoal):
1.  int (t_incr bs) + 0 (incr bs) - 0 bs = 2

then show ?case
using a_incr[of bs] by auto
qed

(* Case that potential cannot be negative *)

lemma t_n_increments: "t_n_increments n init < 1+2^n"
using
  t_n_increments_le[of n init]
  0_non_neg[of "t_n_increments n init"] (* Exploit that potential cannot be negative *)
  0_init

proof (prove)

goal (1 subgoal):
1.  t_n_increments n init <= 1+2^n

proof (prove)

goal (1 subgoal):
1.  t_incr bs + 0 (incr bs) - 0 bs = 2

apply (induction bs rule: incr.induct)
apply (simp_all add: 0_def)
done

(* Application: Doing n increments *)

theorem a_incr: int (t_incr bs) + 0 (incr bs) - 0 bs = 2

apply (induction bs rule: incr.induct)
apply (simp_all add: 0_def)
done

(* Application: Doing n increments *)

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,1), where \( n \) is the number of elements in the table and \( 1 \) the size of the table.)

type symtab = nat x nat

fun invar :: "tab \Rightarrow bool" where
"invar (n,1) = (n = 1)"
we abstract it to a pair \((n, 1)\) where \(n\) is the number of elements in the table and \(1\)
the size of the table.

type synonym tab = nat \times nat

fun invar :: "tab \Rightarrow bool" where
  "invar \((n, 1)\) = \((n \leq 1)\)"

definition init :: tab where
  "init = \((0, 0)\)"

text: Insertion: the element does not matter:
fun ins :: "tab \Rightarrow tab" where
  "ins \((n, 1)\) = \(\langle n+1, \text{if } \text{ncl then } \text{l else if } \text{l0 then } 1 \text{ else } 2\rangle\)"

text: Time: if the table overflows, we count only the time for copying elements:
fun t_ins :: "nat \times nat \Rightarrow nat" where
  "t_ins \((n, 1)\) = \(\text{if } \text{ncl then } 1 \text{ else } n\)"

lemma "invar init"
by (simpr add: init_def)

lemma "invar t \Rightarrow invar(ins t)"
apply (cases t)
apply (auto)
done

fun \phi :: "tab \Rightarrow nat" where
  "\phi \((n, 1)\) = 2n - 1"

text: (Careful: \text{n} and \text{l} are natural numbers.
Thus \(2n\) can be less than \(\text{l}\) [in which case \(2n - 1 = 0\)].
because the invariant does not exclude this case, although it cannot arise.
If you go through the proof of lemma \(\alpha\_\text{ins}\) in detail
you will understand why this case also works out fine.)

lemma "invar t \Rightarrow \phi t \geq 0"
apply (cases t)
apply (auto)
done

lemma "\phi init = 0"
by (simp add: init_def)

lemma \(\alpha\_\text{ins} \Rightarrow \alpha\_\text{ins} t + \phi (\alpha\_\text{ins} t) \times \phi t \leq \text{zl}"
apply (cases t)
apply (auto split: if_splits)
done

end