Exercise 7.1 Interval Lists

Sets of natural numbers can be implemented as lists of intervals, where an interval is simply a pair of numbers. For example the set \{2, 3, 5, 7, 8, 9\} can be represented by the list \[[2, 3), (5, 8), (7, 9)]\). A typical application is the list of free blocks of dynamically allocated memory.

We introduce the type

\[
\text{type-synonym intervals} = "\text{nat} \times \text{nat} \text{ list}"
\]

Next, define an invariant that characterizes valid interval lists: For efficiency reasons intervals should be sorted in ascending order, the lower bound of each interval should be less than or equal to the upper bound, and the intervals should be chosen as large as possible, i.e. no two adjacent intervals should overlap or even touch each other. It turns out to be convenient to define `inv` in terms of a more general function such that the additional argument is a lower bound for the intervals in the list:

\[
\text{fun inv :: } \text{nat} \Rightarrow \text{intervals} \Rightarrow \text{bool}
\]

To relate intervals back to sets define an abstraction function

\[
\text{fun set_of :: } \text{intervals} \Rightarrow \text{nat set}
\]

Define a function to add a single element to the interval list, and show its correctness

\[
\text{fun add :: } \text{nat} \Rightarrow \text{intervals} \Rightarrow \text{intervals}
\]

\text{lemma add_correct:}
For efficiency reasons intervals should be sorted in ascending order, the lower bound of each interval should be less than or equal to the upper bound, and the intervals should be chosen as large as possible, i.e., no two adjacent intervals should overlap or even touch each other. It turns out to be convenient to define $\text{inv}$ in terms of a more general function such that the additional argument is a lower bound for the intervals in the list:

```ml
fun inv : "nat ⇒ intervals ⇒ bool" where
  "inv" n [] = True
| "inv" n ((a, b)#vs) = n ≤ a ∧ a ≤ b ∧ "inv" (b+2) vs
```

**Definition**: $\text{inv}$ where $\text{inv} = \text{inv} \ 0$

---

**Constr**

```ml
fun set_of : "(nat ⇒ nat) list ⇒ nat set" where
  "set_of" [] = undefined

fun set_of : "(nat ⇒ nat) list ⇒ nat set" where
  "set_of" [] = undefined

fun set_of : "(nat ⇒ nat) list ⇒ nat set" where
  "set_of" [] = undefined

test { To relate intervals back to sets define an $\text{elem}(\text{abstraction function})$)

fun set_of : "intervals ⇒ nat set" where
  "set_of" = undefined

test { Define a function to add a single element to the interval list.
  and show its correctness:

  constr
  "set_of" = (nat ⇒ nat) list ⇒ nat set
  Found termination order: "{E}"
```
definition inv where "inv = inv'" 0

fun set_of :: "intervals => nat set"
  where
    "set_of [1] = {1}"
    "set_of ((a,b)::ivs) = {a..b}"

fun add :: "nat => intervals => intervals"
  where
    "add i [x] = undefined"

lemma add_correct:
  assumes "inv is inv'"
  shows "inv (add x is) set_of (add x is) = insert x (set_of is)"
fun add :: nat => intervals => intervals
  where
  | add (i :: [] = [(i, i)])
  | add ((a, b) i :: s) = (if i > a then (i, a) # s else else if i > b then (b, a) # s else if i = b then (b, b) # s else if i < a then (i, b) # s
  else if i = a then case s of
  | (c, d) i :: s => if i + 1 = c then (a, d) # s else else if i < a then (a, c) # s
  else if i = a then case s of
  | (c, d) i :: s => if (b, c) # s = s then (a, d) # s else else if (b, c) # s = s then (a, d) # s
  else if (b, c) # s = s then (a, d) # s
  else (a, d) # s
  else (a, d) # s

lemma add_correct:

const: add :: nat => nat list => nat list

Fun terminated with: "(sp, size_list ((sp, size (snd p))) (snd p)) 

lemma size_correct:
fun add :: "nat ⇒ intervals ⇒ intervals"
where
  "add i [] = [(i,i)]"
  "add i [(c,d)#ivs] = (if i ≤ c then [(i,c)#ivs] else [(i,c)#ivs]#ivs)"
  "add i [(a,b)#ivs] = (if i ≤ a then [(i,a)#ivs] else [(i,a)#ivs]#ivs)"
lemma add_correct:
```
lemma add_correct:
  assumes "inv is" 
  shows "inv (add x is)" "set_of (add x is) = \{x\} \cup set_of is"
 oops

text "HINTS:
  * Sketch the different cases (position of element relative to the first interval of the list)"

proof (prove)
  goal (1 subgoal):
  1. tut07.inv (add x is) \&\& set_of (add x is) = \{x\} \cup set_of is

proof (prove)
  goal (1 subgoal):
  1. tut07.inv (add x is) \&\& set_of (add x is) = \{x\} \cup set_of is
```
lemma add_pres_inv: "inv is --> inv (add x is)"

lemma add_correct:
  assumes "inv is"
  shows "inv (add x is)" = set_of (add x is) = (x) \cup set_of is
eqns

test -Hints:
  1. Sketch the different cases (position of element relative to the first interval of the list)
  2. On paper first

proof (prove)
proof (prove)

goal (1 subgoal):
1. inv is --> inv (add x is)

proof (prove)

proof (prove)

lemma add_pres_inv: "inv is --> inv (add x is)"

lemma add_correct:
  assumes "inv is"
  shows "inv (add x is)" = set_of (add x is) = (x) \cup set_of is
eqns

test -Hints:
  1. Sketch the different cases (position of element relative to the first interval of the list)
  2. On paper first

proof (prove)
proof (prove)

goal (1 subgoal):
1. inv is --> inv (add x is)
proof (prove): goal (1 subgoal):
  1. \[ n \leq x : \text{inv'} n is \Rightarrow \text{inv'} n (\text{add x is}) \]

proof (prove): goal (2 subgoals):
  1. \[ n \leq x : \text{inv'} n is \Rightarrow \text{inv'} n (\text{add x is}) \]
  2. \[ n \leq x : \text{inv'} n is \Rightarrow \text{inv'} n (\text{add x is}) \]
Theorem set_of_add:
[\forall n \leq 7: \forall x. n + x \in \mathbb{N}] \implies \text{set_of}\ (\text{add}\ x\ \mathbb{N}) = \{7x\} \cup \text{set_of}\ \mathbb{N}

proof (prove):
goal: No subgoals!
proof (prove)
goal:
No subgoals!

lemma add_correct:
  assumes "inv_is"
  shows "set_of (add x ia) = (x) ∪ set_of ia"
  using add_pres_inv assms tut07.inv_def apply fastforce
  sledgehammer

Sledgehammering...
Proof Found...
"e": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (100 ms)
"e": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (100 ms)
"e": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (100 ms)
"e": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (100 ms)
Sledgehammering...

Proof state: 0% proof state

Text - Hints:

proof (prove)

1. set_of (add x is) = {x} \cup set_of is

theorem add_correct:

assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} \cup set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce

Text - Hints:

* Sketch the different cases (position of element relative to the first interval of the list)
  on paper first.
proof (prove)
goal 1 subgoal:
  1. insert a (b ∪ d) = c ∪ d

lemma A: "a ∪ b = c" sorry

lemma B: "a ∪ b = c" sorry

apply [simp add: A]
using A apply simp

proof (prove)
goal 1 subgoal:
  1. a ∪ b ∪ c = a ∪ d

lemma C: "a ∪ b = c" sorry

lemma D: "a ∪ b = c" sorry

apply [simp add: A]
using A apply simp

proof (prove)
goal 1 subgoal:
  1. a ∪ b ∪ c = a ∪ d

lemma E: "a ∪ b = c" sorry

lemma F: "a ∪ b = c" sorry

apply [simp add: A]
using A apply simp
proof (prove)
goal (1 subgoal):
  1. \(\text{inv} \; \text{add} \; x \; \text{is} = \{ x \} \cup \text{set_of} \; \text{is}\)

consts
const \(\text{sort} : \text{a list} \rightarrow \text{a list}\)
Found termination order: \(\text{length} < \text{lex} = \{\}\)

proof (prove)
goal (1 subgoal):
  1. \(n = \text{length} \; \text{xs} \quad \text{sort} \; 2 \; n \; \text{xs} = \text{sort} \; \text{xs}\)
fun mort2 :: "nat ⇒ 'a:listorder list ⇒ 'a list" where
mort2 n xs = \{ if n ≤ 1 then xs
else merge (mort2 ... (take (n div 2) xs)) (mort2 ... (drop (n div 2) xs))\}

lemma "n = length xs ⇒ mort2 n xs = mort xs"

proof (prove)

proof (prove)

fun mort2 :: "nat ⇒ 'a:listorder list ⇒ 'a list" where
mort2 n xs = \{ if n ≤ 1 then xs
else merge (mort2 ... (take (n div 2) xs)) (mort2 ... (drop (n div 2) xs))\}

lemma "n = length xs ⇒ mort2 n xs = mort xs"

proof (prove)

proof (prove)
Proof (prove)

goal: No subgoals!

fun add1 :: "nat ⇒ intervals ⇒ intervals"

where

add1 i j x = undefined

lemma add1_correct:

assumes "inv (add1 i j x) = i .. j + (set of i .. x)"

shows "inv (add1 i j x) = i .. j + (set of x)"

let x = x

have [simp]: "sort ([..x] @ x @ ..x) = ..x @ x @ ..x"

using partition_correct [symmetric]

text = "Hint: use the lemma [[the source partition_correct] and another auxiliary lemma here:]" by (auto simp: aux12)

note [simp del] = "sort_key_simps"
proving "drei findet" suchen "manchmal" einer sehr lange set. Beispielsweise aber nicht immer"

```plaintext
proof
have \( |\lambda x . y \cdot x + y | + |\lambda x . y \cdot y + x | < x \)
proof (state)
this:
\( |\lambda x . y \cdot x + y | + |\lambda x . y \cdot y + x | = k \)

goal (1 goal):
1. \( |\lambda x . y \cdot x + y | < k \implies \text{quickselect} (x \# (\# x)) k = \text{sort} (x \# (\# x)) \leq k \)
```
let \( ?x1 \) = "[\( y < x \), \( y = x \)]"

let \( ?x2 \) = "[\( y < x \), \( y > x \)]"

have [simp]: "sort (\( ?x1 \)) = sort ?x1 # x # sort ?x2"
  using partition_correct[of "\( ?x1 \)""]

text ⟨Hint: Use the lemma ⟨(the [source] partition_correct) and another auxiliary lemma here.⟩

by (auto simp aux2)

note [simp del] = sort_key_simps

consider (L) "\( \text{length } ?x1 \)" | (E) "\( \text{length } ?x2 \)" | (G) "\( \text{length } ?x2 \)"
  using nat_neq_iff by blast

then show !case proof_cases