Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

SS 2018
25. 5. 2018

Functional Data Structures

Exercise Sheet 7

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 $[(2,\ 3),\ (5,\ 5),\ (7,\ 9)]$. A typical application is the list of free blocks of dynamically allocated memory.

We introduce the type

type_synonym *intervals* = *"(nat∗nat) 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:

fun *inv'* :: *"nat ⇒ intervals ⇒ bool"* **where**
definition *inv* **where** *"inv ≡ inv' 0"*

To relate intervals back to sets define an *abstraction function*

fun *set_of* :: *"intervals => nat set"*

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

fun *add* :: *"nat ⇒ intervals ⇒ intervals"*
lemma *add_correct*:
  assumes *"inv is"*
  shows *"inv (add x is)" "set_of (add x is) = insert x (set_of is)"*

Hints:

- Sketch the different cases (position of element relative to the first interval of the list) on paper first
- In one case, you will also need information about the second interval of the list. Do this case split via an auxiliary function! Otherwise, you may end up with a recursion equation of the form $f\ (x\#xs) = \ldots\ case\ xs\ of\ x'\#xs' \Rightarrow \ldots\ f\ (x'\#xs')$ ... combined with *split*: *list.splits* this will make the simplifier loop!

Exercise 7.2 Optimized Mergesort

Import *Sorting* for this exercise. The *msort* function recomputes the length of the list in each iteration. Implement an optimized version that has an additional parameter keeping track of the length, and show that it is equal to the original *msort*.

fun *msort2* :: *"nat ⇒ 'a::linorder list ⇒ 'a list"*
lemma *"n = length xs ⟹ msort2 n xs = msort xs"*

Hint: Use *msort.simps* only when instantiated to a particular *xs* (*msort.simps* [*of xs*]), otherwise the simplifier will loop!

Homework 7.1 Deletion from Interval Lists

Submission until Friday, June 1, 11:59am.

Implement and prove correct a delete function.

Hints:

- The correctness lemma is analogous to the one for add.
- A monotonicity property on *inv'* may be useful, i.e., *inv' m is* ⟹ *inv' m' is* if $m' \leq m$
- A bounding lemma, relating m and the elements of *set_of is* if *inv' m is*, may be useful.

fun *del* :: *"nat ⇒ intervals ⇒ intervals"*
lemma *del_correct*: *"Come up with a meaningful spec yourself"*

Homework 7.2 (Bonus) Addition of Interval to Interval List

Submission until Friday, June 1, 11:59am. For 3 **bonus points**, implement and prove correct a function to add a whole interval to an interval list. The runtime must not depend on the size of the interval, e.g., iterating over the interval and adding the elements separately is not allowed!

fun *addi* :: *"nat ⇒ nat ⇒ intervals ⇒ intervals"*
lemma *addi_correct*:
  assumes *"inv is"* *"i≤j"*
  shows *"inv (addi i j is)"* *"set_of (addi i j is) = {i..j} ∪ (set_of is)"*