Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Fabian Huch

SS 23
6. 6. 2023

Functional Data Structures

Exercise Sheet 7

Exercise 7.1 Round wrt. Binary Search Tree

The distance between two integers x and y is $|x - y|$.

1. Define a function *round* :: *int tree* ⇒ *int* ⇒ *int option*, such that *round t x* returns an element of a binary search tree t with minimum distance to x, and *None* if and only if t is empty.

   Define your function such that it does no unnecessary recursions into branches of the tree that are known to not contain a minimum distance element.

2. Specify and prove that your function is correct. Note: You are required to phrase the correctness properties yourself!

   Hint: Specify 3 properties:

   - None is returned only for the empty tree.
   - Only elements of the tree are returned.
   - The returned element has minimum distance.

3. Estimate the time of your round function to be linear in the height of the tree

fun *round* :: *"int tree ⇒ int ⇒ int option"*
fun *T_round* :: *"int tree ⇒ int ⇒ nat"*

Exercise 7.2 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* $\Rightarrow$ *intervals* $\Rightarrow$ *bool"*
definition *inv* **where** *"inv* $\equiv$ inv' *0"*

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

fun *set_of* :: *"intervals* $\Rightarrow$ *nat set"*

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

fun *add* :: *"nat* $\Rightarrow$ *intervals* $\Rightarrow$ *intervals"*
lemma *add_correct_1*:
"inv is $\Longrightarrow$ *inv (add x is)"*
lemma *add_correct_2*:
"inv is $\Longrightarrow$ *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* $x'\#xs' \Rightarrow \ldots\ f\ (x'\#xs')$ $\ldots$ combined with *split*: *list.splits* this will make the simplifier loop!

Homework 7.1 Deletion from Interval Lists

Submission until Monday, 12 June, 23:59pm.

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., $inv'\ m\ is \Longrightarrow inv'\ m'\ is$ 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* $\Rightarrow$ *intervals* $\Rightarrow$ *intervals"*

lemma *del_correct_1*: *"inv is* $\Longrightarrow$ *inv (del x is)"*
lemma *del_correct_2*: *"inv is* $\Longrightarrow$ *set_of (del x is) = (set_of is)* $-$ *{x}"*

Homework 7.2 Addition of Interval to Interval List

Submission until Monday, 12 June, 23:59pm. 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 itself, e.g., iterating over the interval and adding the elements separately is not allowed!

fun *addi* :: *"nat ⇒ nat ⇒ intervals ⇒ intervals"*

lemma *addi_correct_1*: *"inv is ⟹ i≤j ⟹ inv (addi i j is)"*
lemma *addi_correct_2*:
"inv is ⟹ i≤j ⟹ set_of (addi i j is) = {i..j} ∪ (set_of is)"