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

SS 2022
20. 5. 2022

Functional Data Structures

Exercise Sheet 4

Exercise 4.1 List Elements in Interval

Write a function to in-order list all elements of a BST in a given interval. I.e. *in_range t u v* shall list all elements x with $u \le x \le v$. Write a recursive function that does not descend into subtrees that definitely contain no elements in the given range.

fun *in_range* :: "$'a$::*linorder tree* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *list*"

Show that you list the right set of elements

lemma "*bst t* $\Longrightarrow$ *set* (*in_range t u v*) = $\{x \in$ *set_tree t*. $u \le x \land x \le v\}$"

Show that your list is actually in-order

lemma "*bst t* $\Longrightarrow$ *in_range t u v* = *filter* ($\lambda x.\ u \le x \land x \le v$) (*inorder t*)"

Exercise 4.2 Fist Isar Steps

Using Isar, show the following theorem over natural numbers:

theorem
assumes "$x \ge (1 :: nat)$"
shows "$(x + x\hat{}2)\hat{}2 \le 4 * x\hat{}4$"

Hint: When phrasing intermediate goals, check your types. Use *sledgehammer* to fill in simple proof steps.

Exercise 4.3 Enumeration of Trees

Write a function that generates the set of all trees up to a given height. Show that only trees up to the specified height are contained.

fun *enum* :: "*nat* $\Rightarrow$ *unit tree set*"
lemma *enum_sound*: "$t \in$ *enum n* $\Longrightarrow$ *height t* $\le n$"

(Time permitting) Show the other direction, i.e. that all trees of the specified height are contained.

lemma *enum_complete*: "$height\ t \le n \Longrightarrow t \in enum\ n$"

lemma *enum_correct*: "$enum\ h = \{t.\ height\ t \le h\}$"
  by (*auto simp*: *enum_complete enum_sound*)

Homework 4 Popularity Annotated Trees

Submission until Thursday, May 26, 23:59pm.

We define *ptree*s, which are trees that store the popularity of each element, i.e. the number of times it was searched for, as $(nat * 'a)\ tree$.

Define the set of elements of that tree as a recursive function, then show it correct w.r.t. to the normal *set_tree* (‘ is the set *image*):

fun *set_ptree* :: "$('a::linorder)\ ptree \Rightarrow 'a\ set$"
lemma *set_ptree*: "$set\_ptree\ t = snd\ `\ set\_tree\ t$"

Define the binary search tree predicate as well as insert function for those trees (they should be quite similar to the formulations for normal trees).

If a node is already present, overwrite the old popularity value.

fun *pbst* :: "$'a::linorder\ ptree \Rightarrow bool$"
fun *pins* :: "$(nat * 'a::linorder) \Rightarrow 'a\ ptree \Rightarrow 'a\ ptree$"

Show the most interesting property, namely that insert preserves the invariant:

lemma *pins_invar*: "$pbst\ t \Longrightarrow pbst\ (pins\ x\ t)$"

Now define the *isin* function, which should return the updated *ptree* and the number of times it was searched for (i.e., zero for elements not in the tree and at least one for everything in the tree):

fun *pisin* :: "$'a::linorder \Rightarrow 'a\ ptree \Rightarrow ('a\ ptree * nat)$"

Show the correctness of your function:

lemma *pisin_set*: "$pbst\ t \Longrightarrow set\_ptree\ (fst\ (pisin\ x\ t)) = set\_ptree\ t$"
lemma *pisin_invar*: "$pbst\ t \Longrightarrow pbst\ (fst\ (pisin\ x\ t))$"
lemma *pisin_inc*: "$pbst\ t \Longrightarrow (n,x) \in set\_tree\ t \Longrightarrow (Suc\ n,x) \in set\_tree\ (fst\ (pisin\ x\ t))$"

Knowing the popularity of element queries, we can re-order the tree from time to time to optimize query time (assuming that the distribution of searched nodes stays the same).

Implement such a re-ordering — it does not need to be optimal, but the most popular element should be at the root, and the least popular elements should be on the bottom.

Hint: Sorting might be useful. Have a look at the pre-defined *sort* function and its implementation.

term *"sort"*

definition *reorder* :: "$('a::linorder)\ ptree \Rightarrow 'a\ ptree$"

Show that your re-ordering preserves the invariant:

theorem *reorder_pbst*: "$pbst\ t \Longrightarrow pbst\ (reorder\ t)$"

Homework 4 Popularity Annotated Trees (II)

Submission until Thursday, May 26, 23:59pm.

(This is a bonus exercise worth 4 points.)

Show that in the *reorder* function, the set of elements stays unchanged. Start by proving that the *set_ptree* stays unchanged — this should give you an idea how the proof should work.

theorem *reorder_pset*: "$pbst\ t \Longrightarrow set\_ptree\ (reorder\ t) = set\_ptree\ t$"
theorem *reorder_set*: "$pbst\ t \Longrightarrow set\_tree\ (reorder\ t) = set\_tree\ t$"