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

SS 2021
7. 5. 2021

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 ⇒ 'a ⇒ 'a ⇒ 'a list”*

Show that you list the right set of elements

lemma *“bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}”*

Show that your list is actually in-order

lemma *“bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)”*

Exercise 4.2 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 ⇒ unit tree set”*
lemma *enum_sound*: *“t ∈ enum n ⟹ height t ≤ n”*

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

lemma *enum_complete*: *“height t ≤ n ⟹ t ∈ enum n”*

lemma *enum_correct*: *“enum h = {t. height t ≤ h}”*
  by (*auto simp*: *enum_complete enum_sound*)

Homework 4 Rank Annotated Trees

Submission until Thursday, May 13, 23:59pm.

In this homework, we will develop a binary search tree that additionally stores the rank (= number of nodes) of the left subtree in each node.

With this auxiliary information, it is easy to implement a rank query, i.e., to return the position of a given element in the inorder traversal.

Note that trees with annotations could also be implemented as standard trees with their annotation contained in the node data. That way, some functions such as the inorder traversal don't need to be re-defined – however, it makes proving slightly more complicated.

datatype *'a rtree = Leaf | Node "'a rtree" nat 'a "'a rtree"*

Define a function to count the number of actual nodes in a tree.

fun *num_nodes* :: *"'a rtree ⇒ nat"*

Define a function to check for the invariant: search tree property and the correct rank annotation (number of nodes in left subtree)

fun *rbst* :: *"'a::linorder rtree ⇒ bool"*

Define the insert function. You may assume that the value to be inserted is not contained in the tree. Note: Double-check to correctly update the rank annotation.

fun *rins* :: *"'a::linorder ⇒ 'a rtree ⇒ 'a rtree"*

Show that *rins* actually inserts, and preserves the invariant. Hint: Auxiliary lemma on number of nodes.

lemma *rins_set*: *"set_rtree (rins x t) = insert x (set_rtree t)"*
lemma *rins_invar*: *"x∉set_rtree t ⟹ rbst t ⟹ rbst (rins x t)"*

Define the membership query function and show it correct.

fun *risin* :: *"'a::linorder ⇒ 'a rtree ⇒ bool"*
lemma *risin_set*: *"rbst t ⟹ risin x t ⟷ x∈set_rtree t"*

Define the inorder traversal

fun *inorder* :: *"'a rtree ⇒ 'a list"*

Define a function that returns the rank of an element. Use the rank annotation to avoid unnecessary descents into the tree.

Note: You may assume that the element is contained in the tree.

fun *rank* :: *"'a::linorder ⇒ 'a rtree ⇒ nat"*

The operator (!)::*'a list ⇒ nat ⇒ 'a* indexes a list, i.e., *l*!*n* is the *n*th element of list *l*, or *undefined*, if the index is out of bounds. The following predicate states that index *i* into list *l* contains element *x*

definition *“at_index i l x ≡ i<length l ∧ l!i=x”*

Show your rank function correct. Hint: Auxiliary lemma relating *num_nodes* and *inorder*.

lemma *inorder_index*: *“rbst t ⟹ x∈set_rtree t ⟹ at_index (rank x t) (inorder t) x”*

Define a select function, that returns the *i*th element of the inorder traversal, and prove it correct.

Only recurse over the tree once, following a single path. In particular, *inorder t ! i* is not the desired solution, as it would enumerate all nodes of the tree in a list first, and not exploit the rank annotations at all.

fun *select* :: *“nat ⇒ 'a::linorder rtree ⇒ 'a”*
lemma *select_correct*: *“rbst t ⟹ i<length (inorder t) ⟹ select i t = inorder t ! i”*