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

SS 2017
12. 5. 2017

Functional Data Structures

Exercise Sheet 3

Exercise 3.1 Insert with Less Comparisons

- Define a function *ins2* that inserts into a binary search tree, using only one comparison per node. Use the same idea as *isin2*.
- Show that your function is equal to *ins* on binary search trees. Hint: You may need an auxiliary lemma of the form:
  $[\![\ bst\ t;\ \ \forall x \in set\_tree\ t.\ y < x\ ]\!] \Longrightarrow ins2\ x\ (Some\ y)\ t = \ldots$

fun *ins2* :: "'a::linorder ⇒ 'a option ⇒ 'a tree ⇒ 'a tree"

lemma *ins2_None*: "bst t ⟹ ins2 x None t = ins x t"

Exercise 3.2 Height-Preserving In-Order Join

Write a function that joins two binary trees such that

- The in-order traversal of the new tree is the concatenation of the in-order traversals of the original tree
- The new tree is at most one higher than the highest original tree

  Hint: Once you got the function right, proofs are easy!

fun *join* :: "'a tree ⇒ 'a tree ⇒ 'a tree"
lemma "inorder(join t1 t2) = inorder t1 @ inorder t2"
lemma "height(join t1 t2) ≤ max (height t1) (height t2) + 1"

Exercise 3.3 Sorting with BSTs

- Define a function to create a binary search tree from a list. Hint: Use *fold*.
- Show that your function returns a binary search tree with the correct elements
- We define *bst_sort* as the inorder traversal of the tree created from a list. Show that it contains the right elements, is distinct, and sorted.

definition *mk_tree* :: "'a::linorder list ⇒ 'a tree"
lemma *bst_mk_tree*: "bst (mk_tree l)"
lemma *set_mk_tree*: "set_tree (mk_tree l) = set l"

definition "bst_sort l = inorder (mk_tree l)"

lemma *bst_sort_set*: "set (bst_sort l) = set l"
lemma *bst_sort_sorted*: "sorted (bst_sort l)"
lemma *bst_sort_distinct*: "distinct (bst_sort l)"

Homework 3 BSTs with Duplicates

Submission until Friday, May 19, 11:59am.

- Have a look at *bst_eq* in *~~/src/HOL/Library/Tree*, which defines BSTs with duplicate elements.
- Warmup: Show that *isin* and *ins* are also correct for *bst_eq*.

lemma "bst_eq t ⟹ isin t x = (x ∈ set_tree t)"
lemma *bst_eq_ins*: "bst_eq t ⟹ bst_eq (ins x t)"

- Define a function *ins_eq* to insert into a BST with duplicates.

fun *ins_eq* :: "'a::linorder ⇒ 'a tree ⇒ 'a tree"

- Show that *ins_eq* preserves the invariant *bst_eq*

lemma *bst_eq_ins_eq*: "bst_eq t ⟹ bst_eq (ins_eq x t)"

- Define a function *count_tree* to count how often a given element occurs in a tree

fun *count_tree* :: "'a ⇒ 'a tree ⇒ nat"

- Show that the *ins_eq* function inserts the desired element, and does not affect other elements.

lemma "count_tree x (ins_eq x t) = Suc (count_tree x t)"
lemma "x≠y ⟹ count_tree y (ins_eq x t) = count_tree y t"

The next exercise is a bonus exercise, yielding bonus points. Bonus points count as achieved points, but not for the maximum achievable points, when computing the percentage of the achieved homework points.

- Bonus (5p): Use BSTs with duplicates to sort a list (cf. Exercise 3). Prove that the resulted list is sorted, and contains exactly the same number of each element as the original list. Hint: Use a *count* function for lists, and relate it with the *count_tree*-function for trees.

definition *bst_eq_sort* :: "$'a$::*linorder list* $\Rightarrow$ $'a$ *list*"
theorem *count_bst_eq_sort*: "*count x* (*bst_eq_sort l*) = *count x l*"
theorem *sorted_bst_eq_sort*: "*sorted* (*bst_eq_sort l*)"