Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Roßkopf

SS 24
3. 5. 2024

Functional Data Structures

Exercise Sheet 3

Exercise 3.1 Membership Test with Less Comparisons

In the worst case, the *isin* function (see template) performs two comparisons per node. In this exercise, we want to reduce this to one comparison per node. The idea is that we never test for >, but always goes right if not <. However, one remembers the value where one should have tested for =, and performs the comparison when a leaf is reached:

fun *isin2* :: *"('a::linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"*

The second parameter of the function should store the value for the deferred comparison.

Show that your function is correct.

Hint: Auxiliary lemma for *isin2 t (Some y) x*!

lemma *isin2_None*:
  "bst t ⟹ isin2 t None x = isin t x"

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 trees
- 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 *join_inorder[simp]*: *"inorder(join t1 t2) = inorder t1 @ inorder t2"*
lemma *"height(join t1 t2) ≤ max (height t1) (height t2) + 1"*

Exercise 3.3 Implement Delete

Implement delete using the *join* function from last exercise.

Note: At this point, we are not interested in the implementation details of join any more, but just in its properties, i.e. what it does to trees. Thus, as first step, we declare its equations to not being automatically unfolded.

declare *join.simps*[*simp del*]

Both *set_tree* and *bst* can be expressed by the inorder traversal over trees:

thm *set_inorder*[*symmetric*] *bst_iff_sorted_wrt_less*

Note that *set_inorder* is declared as simp. Be careful not to have both directions of the lemma in the simpset at the same time, otherwise the simplifier is likely to loop.

You can use *simp del*: *set_inorder add*: *set_inorder*[*symmetric*] to temporarily remove the first direction of the lemma from the simpset.

Alternatively, you can write *declare set_inorder*[*simp del*] to remove it once and forall.

For *bst*, you might want to delete the *bst_wrt* simps, and use the append lemma:

thm *bst_wrt.simps*
thm *sorted_wrt_append*

Show that join preserves the set of entries

lemma *join_set*[*simp*]: *"set_tree (join t1 t2) = set_tree t1 ∪ set_tree t2"*

Show that joining the left and right child of a BST is again a BST:

lemma *bst_pres*[*simp*]: *"bst (Node l (x::_::linorder) r) ⟹ bst (join l r)"*

Implement a delete function using the idea contained in the lemmas above.

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

Prove it correct! Note: You'll need the first lemma to prove the second one!

lemma *bst_set_delete*[*simp*]: *"bst t ⟹ set_tree (delete x t) = (set_tree t) − {x}"*

lemma *bst_del_pres*: *"bst t ⟹ bst (delete x t)"*

Homework 3.1 Remdups

Submission until Thursday, May 09, 23:59pm.

Your task is to write a function that removes duplicates from a list, using a BST to efficiently store the set of already encountered elements.

You may want to start with an auxiliary function, that takes the BST with the elements seen so far as additional argument, and then define the actual function.

fun *bst_remdups_aux* :: *"'a::linorder tree ⇒ 'a list ⇒ 'a list"*
definition *"bst_remdups xs ≡ bst_remdups_aux Leaf xs"*

Show that your function preserves the set of elements, and returns a list with no duplicates (predicate *distinct* in Isabelle). Hint: Generalization!

theorem *remdups_set*: *"set (bst_remdups xs) = set xs"*

theorem *remdups_distinct*: *"distinct (bst_remdups xs)"*

A list *xs* is a sublist of *ys*, if *xs* can be produced from *ys* by deleting an arbitrary number of elements.

Define a function *sublist xs ys* to check whether *xs* is a sublist of *ys*.

fun *sublist* :: *"'a list ⇒ 'a list ⇒ bool"*

Show that your remdups function produces a sublist of the original list!

Hint: Generalization. Auxiliary lemma required.

theorem *remdups_sub*: *"sublist (bst_remdups xs) xs"*

Homework 3.2 Tree Addressing

Submission until Thursday, May 09, 23:59pm.

A position in a tree can be given as a list of navigation instructions from the root, i.e., whether to go to the left or right subtree. We call such a list a path.

datatype *direction = L | R*
type_synonym *path = "direction list"*

Specify a function to return the subtree addressed by a given path:

fun *get* :: *"'a tree ⇒ path ⇒ 'a tree"*

Specify a function *put t p s*, that returns *t*, with the subtree at *p* replaced by *s*.

fun *put* :: *"'a tree ⇒ path ⇒ 'a tree ⇒ 'a tree"*

How you define those functions for invalid paths is up to you.

Next, specify when a path is valid:

fun *valid* :: "'*a tree* ⇒ *path* ⇒ *bool*"

Write a function *find t s*, that returns the set of all paths which address the subtree *s* in *t*.

fun *find* :: "'*a tree* ⇒ '*a tree* ⇒ *path set*"

Prove the following algebraic laws on *put* and *get*.

lemma *get_put*: "$valid\ t\ p \Longrightarrow put\ t\ p\ (get\ t\ p) = t$"

lemma *put_get*: "$valid\ t\ p \Longrightarrow get\ (put\ t\ p\ s)\ p = s$"

Prove the the correctness of *find* with respect to *put* and *get*:

lemma *find_get*: "$p \in find\ t\ s \Longrightarrow get\ t\ p = s$"

lemma *put_find*: "$valid\ t\ p \Longrightarrow p \in find\ (put\ t\ p\ s)\ s$"