Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Mohammad Abdulaziz, Ph.D.

SS 2020
8. 5. 2020

Functional Data Structures

Exercise Sheet 3

Exercise 3.1 Membership Test with Less Comparisons

In the worst case, the *isin* function 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 stores 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 definition, 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*], or *auto simp del*: *set_inorder simp*: *set_inorder*[*symmetric*] to temporarily remove the lemma from the simpset.
Alternatively, you can write *declare set_inorder*[*simp del*] to remove it once and forall.

For the *sorted_wrt* predicate, you might want to use these lemmas as simp:

thm *sorted_wrt_append sorted_wrt.simps(2)*

Show that join preserves the set of entries

lemma [*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:

thm *bst_iff_sorted_wrt_less*
thm *sorted_wrt_append*

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

lemma *"bst t ⟹ bst (delete x t)"*

Homework 3 BSTs with Duplicates

Submission until Friday, 15 May, 10:00am.

- Have a look at *bst* in ~~/*src*/*HOL*/*Library*/*Tree*, which defines BSTs.
- Define a new function *bst_eq* that is like *bst* but allows duplicate in the tree.
- Show that *isin* and *ins* are also correct for *bst_eq*.

abbreviation *bst_eq* :: *"('a::linorder) tree ⇒ bool"* **where**
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* $\Longrightarrow$ *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* $\Rightarrow$ *'a tree* $\Rightarrow$ *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*$\neq$*y* $\Longrightarrow$ *count_tree y* (*ins_eq x t*) = *count_tree y t*"