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

SS 23
9. 5. 2023

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 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 Rotating Chains

Submission until Monday, 15 May, 23:59pm.

A *right-linear chain* is a binary tree that only descends to the right, i.e., all the values form a list along the right spine. Define a recursive function to characterize such trees:

fun *rlc* :: *"'a tree ⇒ bool"*

Some examples:

value *"rlc ⟨⟨⟩,1::nat,⟨⟨⟩,2,⟨⟨⟩,3,⟨⟩⟩⟩⟩"*
value *"¬rlc ⟨⟨⟩,1::nat,⟨⟨⟨⟩,3,⟨⟩⟩,2,⟨⟩⟩⟩"*

The task is now to transform any binary search tree into a right-linear chain, using only tree rotations. The given *rotate* function rotates a single node.
Show its correctness:

lemma *bst_rotate*[*simp*]: *"bst t ⟹ bst (rotate t)"*

lemma *set_rotate*[*simp*]: *"set_tree (rotate t) = set_tree t"*

Now, define a function to traverse a tree and perform the first available rotation:

fun *rotate1* :: *"'a tree ⇒ 'a tree"*

We want to prove that at most *size t* rotations are necessary for the transformation.
To prove that, we need to define a potential function that decreases in every rotation, and reaches zero for a right-linear chain:

fun *pot* :: *"'a tree ⇒ nat"*

Prove these two properties:

lemma *pot_0*: *"rlc t ⟷ pot t = 0"*
lemma *pot_rotate_n*[*simp*]: *"pot ((rotate1 ^^ n) t) = pot t − n"*

Put everything together, and show the final statement:

theorem *rlc_rotate*: *"∃ n ≤ size t. rlc ((rotate1 ^^ n) t)"*

Hint: Use straightforward definitions and check if they are correct first. Create auxiliary lemmas where appropriate. All proofs should only take a few lines.