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

SS 2022
1. 7. 2022

Functional Data Structures

Exercise Sheet 9

Exercise 9.1 Joining 2-3-Trees (II)

Write a join function for complete 2-3-trees of arbitrary height: The function shall take two 2-3-trees l and r and an element x, and return a new 2-3-tree with the inorder-traversal $l\ x\ r$.

Write two functions, one for the height of l being greater, the other for the height of r being greater. The result should also be a complete tree, with height equal to the greater height of l and r.

height r greater:

fun *joinL* :: "'*a tree23* ⇒ '*a* ⇒ '*a tree23* ⇒ '*a upI*"
lemma *complete_joinL*: "⟦ *complete l*; *complete r*; *height l* < *height r* ⟧
⟹ *complete* (*treeI* (*joinL l x r*)) ∧ *hI* (*joinL l x r*) = *height r*"

lemma *inorder_joinL*: "⟦ *complete l*; *complete r*; *height l* < *height r* ⟧
⟹ *inorder* (*treeI* (*joinL l x r*)) = *inorder l* @*x* # *inorder r*"

height l greater:

fun *joinR* :: "'*a tree23* ⇒ '*a* ⇒ '*a tree23* ⇒ '*a upI*"
lemma *complete_joinR*: "⟦ *complete l*; *complete r*; *height l* > *height r* ⟧ ⟹
complete (*treeI* (*joinR l x r*)) ∧ *hI*(*joinR l x r*) = *height l*"

lemma *inorder_joinR*: "⟦ *complete l*; *complete r*; *height l* > *height r* ⟧ ⟹ *inorder* (*treeI* (*joinR l x r*)) = *inorder l* @*x* # *inorder r*"

Combine both functions.

fun *join* :: "'*a tree23* ⇒ '*a* ⇒ '*a tree23* ⇒ '*a tree23*"
lemma "⟦ *complete l*; *complete r* ⟧ ⟹ *complete* (*join l x r*)"

lemma "⟦ *complete l*; *complete r* ⟧ ⟹ *inorder* (*join l x r*) = *inorder l* @*x* # *inorder r*"

Exercise 9.2 Union Function on Binary Tries

Define a function to merge two tries and show its correctness:

hide_const *Tree23_Set.isin*

fun *union* :: *“trie ⇒ trie ⇒ trie”*
lemma *“isin (union a b) x = Tries_Binary.isin a x ∨ Tries_Binary.isin b x”*

Homework 9.1 Balanced Tree to RBT

Submission until Thursday, 7. 7. 2022, 23:59pm.

A tree is balanced, if its minimum height and its height differ by at most 1.

The following function paints a balanced tree to form a valid red-black tree with the same structure. The task of this homework is to prove this!

```
fun mk_rbt :: "'a tree \<Rightarrow> 'a rbt" where
  "mk_rbt Leaf = Leaf"
| "mk_rbt (Node l a r) = (let
    l'=mk_rbt l;
    r'=mk_rbt r
  in
    if min_height l > min_height r then
      B (paint Red l') a r'
    else if min_height l < min_height r then
      B l' a (paint Red r')
    else
      B l' a r'
  )"
```

Warmup

Show that the left and right subtree of a balanced tree are, again, balanced

lemma *balanced_subt*: *“balanced (Node l a r) ⟹ balanced l ∧ balanced r”*

Show the following alternative characterization of balanced.

Hint: Auxiliary lemma relating *height t* and *Defs.min_height t*

lemma *balanced_alt*:
“balanced t ⟷ height t = min_height t ∨ height t = min_height t + 1”

The Easy Parts

Show that *mk_rbt* does not change the inorder-traversal:

lemma *mk_rbt_inorder*: *"inorder (mk_rbt t) = Tree.inorder t"*

Show that the color of the root node is always black

lemma *mk_rbt_color*: *"color (mk_rbt t) = Black"*

Medium Complex Parts

Show that the black-height of the returned tree is the minimum height of the argument tree.
Hint: Use Isar to have better control when to unfold with *balanced_alt*, and when to use *balanced_subt* (e.g. to discharge the premises of the IH)

lemma *mk_rbt_bheight*: *"balanced t $\Longrightarrow$ bheight (mk_rbt t) = min_height t"*

Show that the returned tree satisfies the height invariant.

lemma *mk_rbt_invh*: *"balanced t $\Longrightarrow$ invh (mk_rbt t)"*

The Hard Part (3 Bonus Points)

For three bonus points, show that the returned tree satisfies the color invariant.
Warning: This requires careful case splitting, via a clever combination of automation and manual proof (Isar, aux-lemmas), in order to deal with the multiple cases without a combinatorial explosion of the proofs.

lemma *mk_rbt_invc*: *"balanced t $\Longrightarrow$ invc (mk_rbt t)"*

Homework 9.2 Linear-Time Repainting

Submission until Thursday, 7. 7. 2022, 23:59pm.
Write a linear-time version of *mk_rbt*, and show that it behaves like *mk_rbt*.
Idea: Compute the min-height during the same recursion as you build the tree.
Note: No formal complexity proof required.

fun *mk_rbt'* :: *"'a tree $\Rightarrow$ 'a rbt $\times$ nat"*

lemma *mk_rbt'_refine*: *"fst (mk_rbt' t) = mk_rbt t"*