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

SS 2018
8. 6. 2018

Functional Data Structures

Exercise Sheet 9

Exercise 9.1 Indicate Unchanged by Option

Write an insert function for red-black trees that either inserts the element and returns a new tree, or returns None if the element was already in the tree

fun *ins'* :: "$'a$::*linorder* $\Rightarrow$ $'a$ *rbt* $\Rightarrow$ $'a$ *rbt option*"
lemma "*invc t* $\Longrightarrow$ *case ins' x t of None* $\Rightarrow$ *ins x t* $=$ *t* $|$ *Some t'* $\Rightarrow$ *ins x t* $=$ *t'*"

Exercise 9.2 Joining 2-3-Trees

Write a join function for 2-3-trees: 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.

height r greater

fun *joinL* :: "$'a$ *tree23* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *tree23* $\Rightarrow$ $'a$ up_i"
lemma *bal_joinL*: "$[\![$ *bal l*; *bal r*; *height l* $\leq$ *height r* $]\!]$ $\Longrightarrow$
bal (*tree*$_i$ (*joinL l x r*)) $\wedge$ *height*(*joinL l x r*) $=$ *height r*"
lemma *inorder_joinL*: "$[\![$ *bal l*; *bal r*; *height l* $\leq$ *height r* $]\!]$ $\Longrightarrow$ *inorder* (*tree*$_i$ (*joinL l x r*)) $=$ *inorder l* @*x* # *inorder r*"

height l greater

fun *joinR* :: "$'a$ *tree23* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *tree23* $\Rightarrow$ $'a$ up_i"
lemma *bal_joinR*: "$[\![$ *bal l*; *bal r*; *height l* $\geq$ *height r* $]\!]$ $\Longrightarrow$
bal (*tree*$_i$ (*joinR l x r*)) $\wedge$ *height*(*joinR l x r*) $=$ *height l*"

Note the generalization: We augmented the lemma with a statement about the height of the result.

lemma *inorder_joinR*: "$[\![$ *bal l*; *bal r*; *height l* $\geq$ *height r* $]\!]$ $\Longrightarrow$ *inorder* (*tree*$_i$ (*joinR l x r*)) $=$ *inorder l* @*x* # *inorder r*"

Combine both functions

fun *join* :: "$'a$ *tree23* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *tree23* $\Rightarrow$ $'a$ *tree23*"

lemma "$\llbracket$ *bal l*; *bal r* $\rrbracket \Longrightarrow$ *bal* (*join l x r*)"
lemma "$\llbracket$ *bal l*; *bal r* $\rrbracket \Longrightarrow$ *inorder* (*join l x r*) = *inorder l* @*x* # *inorder r*"

Homework 9.1 Balanced Tree to RBT

Submission until Friday, 15. 6. 2018, 11:59am.

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

```
fun min_height :: "('a,'b) tree ⇒ nat" where
"min_height Leaf = 0" |
"min_height (Node _ l _ r) = min (min_height l) (min_height r) + 1"

definition "balanced t ≡ height t − min_height t ≤ 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,unit) tree ⇒ '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 c l a r*) $\Longrightarrow$ *balanced l* $\wedge$ *balanced r*"

Show the following alternative characterization of balanced:

lemma *balanced_alt*:
"*balanced t* $\longleftrightarrow$ *height t* = *min_height t* $\vee$ *height t* = *min_height t* + *1*"

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

The Easy Parts

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

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

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

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)

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 Friday, 15. 6. 2018, 11:59am.

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,unit) tree* $\Rightarrow$ *'a rbt* $\times$ *nat"* — Returns the RBT and the min-height of the argument

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