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

SS 24
21. 6. 2024

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 ⇒ 'a rbt ⇒ 'a rbt option"*
lemma *"invc t ⟹ case ins' x t of None ⇒ ins x t = t | Some t' ⇒ ins x t = t'"*

Exercise 9.2 Joining 2-3-Trees

Implement and prove correct a function to combine two 2-3-trees of equal height, such that the inorder traversal of the resulting tree is the concatenation of the inorder traversal of the arguments, and the height of the result is either the height of the arguments, or has increased by one. Use *'a upI* to return the result, similar to *Tree23_Set.ins*:

fun *joinS* :: *"'a tree23 ⇒ 'a tree23 ⇒ 'a upI"*
lemma *joinS_inorder*:
fixes *t1 t2* :: *"'a tree23"*
assumes *"height t1 = height t2"*
assumes *"complete t1" "complete t2"*
shows *"inorder (treeI (joinS t1 t2)) = (inorder t1 @ inorder t2)"*

lemma *joinS_complete*:
fixes *t1 t2* :: *"'a tree23"*
assumes *"height t1 = height t2"*
assumes *"complete t1" "complete t2"*
shows *"complete (treeI (joinS t1 t2)) ∧ hI (joinS t1 t2) = height t2"*

Hints:

- Try to use automatic case splitting (*auto split*: ...) instead of explicit case splitting via Isar (There will be dozens of cases).
- To find bugs in your join function, or isolate the case where your automatic proof does not (yet) work, use Isar to perform the induction proof case by case.

(Time permitting, similar ideas, do not use *joinS*)

Write a join function for complete 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. 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 *joinA* :: *"'a tree23 ⇒ 'a ⇒ 'a tree23 ⇒ 'a tree23"*
lemma *complete_joinA*: *"⟦ complete l; complete r ⟧ ⟹ complete (joinA l x r)"*

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

Homework 9.1 Insertion 1-2 Trees

Submission until Thursday, June 27, 23:59pm.

Similar to a 2-3 tree, we can construct search trees that consist of 1- and and 2-nodes and maintains completeness. To achieve a logarithmic height, 1-nodes may not be chained, i.e. the full invariant is::

invar $\langle\rangle$ = *True*

invar $\langle t\rangle$ = (*case t of* $\langle\rangle$ ⇒ *True* | $\langle x\rangle$ ⇒ *False* | $\langle l, x, r\rangle$ ⇒ *height l = height r ∧ invar l ∧ invar r*)

invar $\langle l, uu, r\rangle$ = (*height l = height r ∧ invar l ∧ invar r*)

Define an insert function, similar to 2-3 trees (start by copying that function). Instead of the three-node constructor, use an auxiliary *merge* function, which may be recursive

again. Your *merge* function should retain the invariant and preserve the inorder traversal of its arguments.

fun *merge* :: "'*a tree12* ⇒ '*a* ⇒ '*a tree12* ⇒ '*a* ⇒ '*a tree12* ⇒ '*a upI*"
fun *ins* :: "'*a*::*linorder* ⇒ '*a tree12* ⇒ '*a upI*"

Show that *merge* retains the inorder:

lemma *inorder_merge*[*simp*]:
"*inorder*(*treeI*(*merge l a m b r*)) = (*inorder l*) @ *a* # (*inorder m*) @ *b* # (*inorder r*)"

Show that *insert* retains the invariant:

theorem *invar_ins*: "*invar t* ⟹ *invar* (*treeI*(*ins x t*)) ∧ *hI* (*ins x t*) = *height t*"

Hint: Suitable lemmas about *merge* (invariant, height,...) are probably needed.

Homework 9.2 List to RBT

Submission until Thursday, June 27, 23:59pm.

In this task you are to define a function *list_to_rbt* which constructs a red-black tree that contains the members of a given list.

Hint:

This function could be constructed by composing two functions. The first is a function that constructs an almost complete binary tree from a list (see the function *balance_list* in *HOL−Data_Structures.Balance*) – a tree is almost complete if its minimum height and its height differ by at most 1 (see *acomplete* in the file *HOL−Library.Tree*)

The second function, which is *mk_rbt*, constructs the equivalent red-black tree to a given almost complete binary tree:

```
fun mk_rbt :: "'a tree ⇒ 'a rbt" where
  "mk_rbt ⟨⟩ = ⟨⟩"
| "mk_rbt ⟨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'
  )"
```

fun *list_to_rbt* :: "'*a list* ⇒ '*a rbt*"

Hint: If you follow the hint above and construct the function *list_to_rbt* by composing the functions *mk_rbt* and *balance_list*, then a good idea to prove the theorems required

below is to prove lemmas about *mk_rbt* applied to almost complete trees, and then leverage the results to get the theorems about *list_to_rbt*

Warmup

Show the following alternative characterization of almost complete:

lemma *acomplete_alt*:
"acomplete t $\longleftrightarrow$ *height t = min_height t* $\vee$ *height t = min_height t + 1"*

The Easy Parts

Show that the inorder traversal of the tree constructed by *list_to_rbt* is the same as the given list:

lemma *mk_rbt_inorder*: *"Tree2.inorder (list_to_rbt xs) = xs"*

Show that the color of the root node is always black:

lemma *mk_rbt_color*: *"color (list_to_rbt xs) = Black"*

Medium Complex Parts

Show that the returned tree satisfies the height invariant.

lemma *mk_rbt_invh*: *"invh (list_to_rbt xs)"*

Hint: Use Isar to have better control on when to unfold with *acomplete_alt*, and when to use (e.g. to discharge the premises of the IH). Also, a useful lemma to prove is *acomplete ?t* $\Longrightarrow$ *bheight (mk_rbt ?t) = min_height ?t*.

The Hard Part (Bonus, 3 points)

Show that the returned tree satisfies the color invariant.

lemma *mk_rbt_invc*: *"invc (list_to_rbt t)"*

Hint: A useful lemma is *acomplete ?t* $\Longrightarrow$ *invc (mk_rbt ?t)*. To prove it, combine case splitting, automation and manual proof (Isar, aux-lemmas), in order to deal with the multiple cases without a combinatorial explosion of the proofs.