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

SS 2020
12. 6. 2020

Functional Data Structures

Exercise Sheet 8

Exercise 8.1 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 *join* :: *"'a tree23 ⇒ 'a tree23 ⇒ 'a upI"*

lemma *join_inorder*:
fixes *t1 t2* :: *"'a tree23"*
assumes *"height t1 = height t2"*
assumes *"complete t1" "complete t2"*
shows *"inorder (treeI (join t1 t2)) = (inorder t1 @ inorder t2)"*

lemma *join_complete*:
fixes *t1 t2* :: *"'a tree23"*
assumes *"height t1 = height t2"*
assumes *"complete t1" "complete t2"*
shows *"complete (treeI (join t1 t2)) ∧ height (join 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.

Exercise 8.2 Bounding the Size of 2-3-Trees

Show that for 2-3-trees, we have:

$log_3\ (\ s(t)\ +\ 1\ )\ \leq\ h(t)\ \leq\ log_2\ (s(t)\ +\ 1)$

Hint: It helps to first raise the two sides of the inequation to the 2nd/3rd power. Use sledgehammer and find-theorems to search for the appropriate lemmas.

```
lemma height_bound_upper: "complete t ⟹ height t ≤ log 2 (size t + 1)"
lemma height_bound_lower: assumes "complete t"
  shows "log 3 (size t + 1) ≤ height t"
```

Homework 8.1 Bounding Fibonacci

Submission until Friday, 19 June, 10:00am.

We start by defining the Fibonacci sequence, and an alternative induction scheme for indexes greater 0:

```
fun fib :: "nat ⇒ nat"
  where
    fib0: "fib 0 = 0"
  | fib1: "fib (Suc 0) = 1"
  | fib2: "fib (Suc (Suc n)) = fib (Suc n) + fib n"

lemma f_alt_induct [consumes 1, case_names 1 2 rec]:
  assumes "n > 0"
      and "P (Suc 0)" "P 2" "⋀n. n > 0 ⟹ P n ⟹ P (Suc n) ⟹ P (Suc (Suc n))"
  shows   "P n"
  using assms(1)
proof (induction n rule: fib.induct)
  case (3 n)
  thus ?case using assms by (cases n) (auto simp: eval_nat_numeral)
qed (auto simp: ‹P (Suc 0)› ‹P 2›)
```

Show that the Fibonacci numbers grow exponentially, i.e., that they are bounded from below by $1.5^n/3$.

Use the alternative induction scheme defined above.

```
lemma fib_lowerbound: "n > 0 ⟹ real (fib n) ≥ 1.5 ^ n / 3"
proof (induction n rule: f_alt_induct)
```

Homework 8.2 AVL Trees

Submission until Friday, 19 June, 10:00am.

AVL trees are binary search trees where, for each node, the heights of its subtrees differ by at most one. In this homework, you are to bound the minimal number of nodes in an AVL tree of a given height.

First, define the AVL invariant on binary trees. Note: In practice, one additionally stores the heights or height difference in the nodes, but this is not required for this exercise.

```
fun avl :: "'a tree ⇒ bool"
```

Show that an AVL tree of height h has at least *fib* $(h+2)$ nodes:

lemma *avl_fib_bound*: *"avl t ⟹ height t = h ⟹ fib (h+2) ≤ size1 t"*

Combine your results to get an exponential lower bound on the number of nodes in an AVL tree.

lemma *avl_lowerbound*:
assumes *"avl t"*
shows *"1.5 ^ (height t + 2) / 3 ≤ real (size1 t)"*

Homework 8.3 Bonus: Delete via Join

Submission until Friday, 19 June, 10:00am. For **5 bonus points** implement and prove correct a delete function based on join instead of *del_min*. Rough idea: To remove an element, join its left and right subtrees.

fun *del'* :: *"'a::linorder ⇒ 'a tree23 ⇒ 'a upD"* **where**
lemma *inorder_del'*: *"⟦ complete t ; sorted(inorder t) ⟧ ⟹*
inorder(treeD (del' x t)) = del_list x (inorder t)"

lemma *bal_treeD_del*: *"complete t ⟹ complete(treeD(del' x t))"*

A few hints:

- Prove auxiliary lemmas on *join* that are suitable to discharge your proof obligations, and disable simplification of join for your main proof (*declare join.simps*[*simp del*]). This will make proof obligations more readable.
- Case splitting by *simp* or *auto* may take quite a long time. Use *split*!: instead of *split*: to make it a bit faster.