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

SS 2018
6. 7. 2018

Functional Data Structures

Exercise Sheet 13

Presentation of Mini-Projects

You are invited, on a voluntary basis, to give a short presentation of your mini-projects in the tutorial on July 13.

Depending on how many presentations we have, the time slots will be 5 to 10 minutes, plus 2 minutes for questions.

If you are interested, please write me a short email until Wednesday, July 11.

The following are old exam questions!

Exercise 13.1 Converting List for Balanced Insert

Recall the standard insertion function for unbalanced binary search trees.

```
fun insert :: "'a::linorder ⇒ 'a tree ⇒ 'a tree" where
"insert x Leaf = Node Leaf x Leaf" |
"insert x (Node l a r) =
  (case cmp x a of
     LT ⇒ Node (insert x l) a r |
     EQ ⇒ Node l a r |
     GT ⇒ Node l a (insert x r))"
```

We define the function *from_list*, which inserts the elements of a list into an initially empty search tree:

```
definition from_list :: "'a::linorder list ⇒ 'a tree" where
  "from_list l = fold insert l Leaf"
```

Your task is to specify a function *preprocess*::*'a list ⇒ 'a list*, that preprocesses the list such that the resulting tree is balanced.

You may assume that the list is sorted, distinct, and has exactly *2^k − 1* elements for some *k*. That is, your *preprocess* function must satisfy:

```
lemma
  assumes "sorted l" and "distinct l" and "length l = 2^k−1"
  shows "set (preprocess l) = set l" and "balanced (from_list (preprocess l))"
```

Note: **No proofs required**, only a specification of the *preprocess* function!

Exercise 13.2 Trees with Same Structure

Question 1

Specify the recursion equations of a function $same::'a\ tree \Rightarrow 'b\ tree \Rightarrow bool$ that returns true if and only if the two trees have the same structure (i.e., ignoring values).

Question 2

Show, by computation induction wrt. *same*, that insertion of arbitrary elements into two Braun heaps with the same structure yields heaps with the same structure again.

same ?t ?t' $\Longrightarrow$
same (*Priority_Queue_Braun.insert ?x ?t*)
(*Priority_Queue_Braun.insert ?y ?t'*)

For your proof, it is enough to cover the (Node,Node) case. If you get analogous subcases, only elaborate one of them!

Hint: Here is the definition of *Priority_Queue_Braun.insert*:

```
fun insert :: "'a::linorder ⇒ 'a tree ⇒ 'a tree" where
"insert a Leaf = Node Leaf a Leaf" |
"insert a (Node l x r) =
 (if a < x then Node (insert x r) a l else Node (insert a r) x l)"
```

Homework 13 Amortized Complexity

Submission until Friday, 13.07.2018, 11:59am.

This is an old exam question, which we have converted to a homework to be done with Isabelle!

A "stack with multipop" is a list with the following two interface functions:

```
fun push :: "'a ⇒ 'a list ⇒ 'a list" where
"push x xs = x # xs"

fun pop :: "nat ⇒ 'a list ⇒ 'a list" where
"pop n xs = drop n xs"
```

You may assume

```
definition t_push :: "'a ⇒ 'a list ⇒ nat" where
"t_push x xs = 1"

definition t_pop :: "nat ⇒ 'a list ⇒ nat" where
```

"t_pop n xs = min n (length xs)"

Use the potential method to show that the amortized complexity of *push* and *pop* is constant.

Provide complete proofs in Isabelle!

Original text here was: *If you need any properties of the auxiliary functions length, drop and min, you should state them but you do not need to prove them.*