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

SS 2022
22. 7. 2022

Functional Data Structures

Exercise Sheet 12

Presentation of Mini-Projects:
You are invited, on a voluntary basis, to give a short presentation (5-10 minutes) of your mini-projects in the tutorial on July 29.
If you are interested, please write me a short email until Thursday.

The following are old exam questions!

Exercise 12.1 Amortized Complexity

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.
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.

Exercise 12.2 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*, that preprocesses the list such that the resulting tree is almost complete.

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:

```
fun preprocess :: "'a list ⇒ 'a list"

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

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

Exercise 12.3 Trees with Same Structure

Question 1

Specify the recursion equations of a function *same* that returns true if and only if the two trees have the same structure (i.e., ignoring values).

```
fun same :: "'a tree ⇒ 'a tree ⇒ bool"
```

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.

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 *Tree_Set.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)"

lemma *same_insert*: *"same t t' ⟹ same (insert x t) (insert y t')"*

Homework 12.1 DYI double-ended queue

Submission until Thursday, 28. 7. 2022, 23:59pm.

Think up and show correct a double-ended queue (i.e., one where elements can be enqueued/dequeued from both sides) where every operation has at worst logarithmic complexity (in the size of the queue).

Fill your definitions into the interpretation below and discharge the proof obligations.

Briefly state why your queue has the desired complexity properties (as measured in a canoncial timing function). No formal definition or proof is required.

Hint: Re-use concepts you've learned in the lecture. You may import any theory from *HOL−Data_Structures*, but keep the *Defs* as first import. Show all the properties as individual lemmas first, and make small steps!

global_interpretation *Q*: *DYI_queue*
  where *empty = undefined*
    and *push_front = undefined*
    and *push_back = undefined*
    and *pop_front = undefined*
    and *pop_back = undefined*
    and *invar = undefined*
    and *α = undefined*
proof (*standard*, *goal_cases*)
  oops