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

SS 2017
30. 6. 2017

Functional Data Structures

Exercise Sheet 10

Exercise 10.1 Insert for Leftist Heap

- Define a function to directly insert an element into a leftist heap. Do not construct an intermediate heap like insert via meld does!
- Show that your function is correct
- Define a timing function for your insert function, and show that it is linearly bounded by the rank of the tree.

fun *lh_insert* :: "'a::ord ⇒ 'a lheap ⇒ 'a lheap"
lemma *mset_lh_insert*: "mset_tree (lh_insert x t) = mset_tree t + {# x #}"
lemma "heap t ⟹ heap (lh_insert x t)"
lemma "ltree t ⟹ ltree (lh_insert x t)"

fun *t_lh_insert* :: "'a::ord ⇒ 'a lheap ⇒ nat"
lemma "t_lh_insert x t ≤ rank t + 1"

Exercise 10.2 Bootstrapping a Priority Queue

Given a generic priority queue implementation with $O(1)$ *empty*, *is_empty* operations, $O(f_1\ n)$ insert, and $O(f_2\ n)$ *get_min* and *del_min* operations.

Derive an implementation with $O(1)$ *get_min*, and the asymptotic complexities of the other operations unchanged!

Hint: Store the current minimal element! As you know nothing about f_1 and f_2, you must not use *get_min*/*del_min* in your new *insert* operation, and vice versa!

For technical reasons, you have to define the new implementations type outside the locale!

datatype ('a,'s) *bs_pq* =
locale *Bs_Priority_Queue* =
  orig: *Priority_Queue orig_empty orig_is_empty orig_insert orig_get_min orig_del_min orig_invar orig_mset*
  for *orig_empty orig_is_empty orig_insert orig_get_min orig_del_min orig_invar*
  and *orig_mset* :: "'s ⇒ 'a::linorder multiset"
begin

In here, the original implementation is available with the prefix *orig*, e.g.

```
term orig_empty term orig_invar
thm orig.invar_empty

definition empty :: "('a,'s) bs_pq"

fun is_empty :: "('a,'s) bs_pq ⇒ bool"

fun insert :: "'a ⇒ ('a,'s) bs_pq ⇒ ('a,'s) bs_pq"

fun get_min :: "('a,'s) bs_pq ⇒ 'a"

fun del_min :: "('a,'s) bs_pq ⇒ ('a,'s) bs_pq"

fun invar :: "('a,'s) bs_pq ⇒ bool"

fun mset :: "('a,'s) bs_pq ⇒ 'a multiset"
```

Show that your new implementation satisfies the priority queue interface!

```
sublocale Priority_Queue empty is_empty insert get_min del_min invar mset
  apply unfold_locales
proof goal_cases
  case 1
  then show ?case
  next
  case (2 q) — and so on
  qed
end
```

Homework 10 Constructing a Heap from a List of Elements

Submission until Friday, 7. 7. 2017, 11:59am.

The naive solution of starting with the empty heap and inserting the elements one by one can be improved by repeatedly merging heaps of roughly equal size. Start by turning the list of elements into a list of singleton heaps. Now make repeated passes over the list, merging adjacent pairs of heaps in each pass (thus halving the list length) until only a single heap is left. It can be shown that this takes linear time.

Define a function *heap_of_list* :: *'a list ⇒ 'a lheap* and prove its functional correctness.

```
definition heap_of_list :: "'a::ord list ⇒ 'a lheap"
lemma mset_heap_of_list: "mset_tree (heap_of_list xs) = mset xs"
lemma heap_heap_of_list: "heap (heap_of_list xs)"
lemma ltree_ltree_of_list: "ltree (heap_of_list xs)"
```