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

SS 2021
25. 6. 2021

Functional Data Structures

Exercise Sheet 11

Exercise 11.1 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 merge 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 *set_lh_insert*: *"set_tree (lh_insert x t) = set_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 *"ltree t ⟹ t_lh_insert x t ≤ min_height t + 1"*

Exercise 11.2 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* **where**
    empty = *orig_empty* **and**

is_empty = *orig_is_empty* **and**
insert = *orig_insert* **and**
get_min = *orig_get_min* **and**
del_min = *orig_del_min* **and**
invar = *orig_invar* **and**
mset = *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"*
lemmas [*simp*] = *orig.is_empty orig.mset_get_min orig.mset_del_min*
orig.mset_insert orig.mset_empty
orig.invar_empty orig.invar_insert orig.invar_del_min

Show that your new implementation satisfies the priority queue interface!

sublocale *Priority_Queue*
where *empty* = *empty*
and *is_empty* = *is_empty*
and *insert* = *insert*
and *get_min* = *get_min*
and *del_min* = *del_min*
and *invar* = *invar*
and *mset* = *mset*
apply *unfold_locales*
proof *goal_cases*

Homework 11.1 Converting a binary tree into a heap

Submission until Thursday, 1. 7. 2021, 23:59pm.

The following predicate describes the heap property for a binary tree.

fun *heap*::*"'a::linorder tree ⇒ bool"* **where**
"heap Leaf = True"
| *"heap (Node l x r) = ((∀ y∈set_tree l. x ≤ y) ∧ (∀ y∈set_tree r. x ≤ y) ∧ heap l ∧ heap r)"*

Recall the function *sift_down* from the AFP entry *Priority_Queue_Braun*. Define an equivalent function for sifting the root of a binary tree. Hint: that function will need to include extra cases that account for the fact that, unlike a Braun tree, a binary tree is not necessarily balanced.

fun *siftdown*:: "'*a*::*linorder tree* ⇒ '*a*::*linorder tree*"

Define a function *heapify* which, given a binary tree, reorders the elements of the binary tree into a heap. That function has to use the function *sift_down*. Show that the function indeed creates a heap and that it preserves the elements in the given binary tree.

fun *heapify* :: "'*a*::*linorder tree* ⇒ '*a*::*linorder tree*"

theorem *heapify_heap*: "*heap* (*heapify t*)"
theorem *heapify_mset*: "*mset* (*inorder* (*heapify t*)) = *mset* (*inorder t*)"

Homework 11.2 Be Original!

Submission until Thursday, July 8, 23:59pm.

Develop a nice Isabelle formalisation yourself!

- This homework goes in parallel to other homeworks for most of the remaining lecture period. We will reduce regular homework load (the sheets are half the size/points), such that you have a time-frame of 3 weeks with reduced regular homework load. You should have formalized key concepts of your topic until next week.
- The homework will yield 15 points (for minimal solutions). Additionally, up to 15 bonus points may be awarded for particularly nice/original/etc solutions.
- You may develop a formalisation from all areas, not only functional data structures.
- Document your solution, such that it is clear what you have formalised and what your main theorems state!
- Set yourself a time frame and some intermediate/minimal goals. Your formalisation needs not be universal and complete after 3 weeks.
- Should you still need inspiration to find a project: Sparse matrices, skew binary numbers, arbitrary precision arithmetic (on lists of bits), interval data structures (e.g. interval lists), spatial data structures (quad-trees, oct-trees), Fibonacci heaps, prefix tries/arrays and BWT, etc. You can also ask the tutor for possible ideas, and you are encouraged to discuss the realisability of your project with us!