CSE1729 - Introduction to Programming October 30, 2018

Laboratory Assignment 9

Objectives

- Work with heaps

Activities

1. For this question, we will use the heap-supporting functions as seen in the lecture slides as building blocks for this assignment to build a new heap implementation. The heap implementation shown in the lecture slides is an example of a min-heap, in which the smallest element is at the root and all elements in child trees are larger than the value at the root. We can also construct max-heap data structures in which the largest element in the heap is at the root and all elements in child trees are smaller than the root.

   The objective is to define SCHEME functions to manipulate a heap which:

   1. maintain a binary tree as a heap,
   2. use a generic (first order) order relation,
   3. provides functions which can determine if a heap is `empty?` as well as `heap-insert`, `heap-remove`, and `combine-heaps`

   (a) Define a SCHEME procedure, named `(heap-insert f x H)`, which adds element `x` to heap `H` using the first-order relation `f` to determine which element belongs at the root of each (sub)tree.
   For instance, if we wanted the same behavior as the heaps in the lecture slides (min-heap), we would use the "less than" function as our first-order relation:

   ```
   (heap-insert < 100 (heap-insert < 10 (list)))
   (10 () (100 () ()))
   ```

   If, instead, we wanted a max-heap implementation, where the largest element is at the root of the heap, we would use the "greater than" function as our first-order relation.

   ```
   (heap-insert > 100 (heap-insert > 10 (list)))
   (100 () (10 () ()))
   ```

   Note, you must use the same first-order relation for all of the heap procedures applied to a particular heap structure.

   (b) Define a SCHEME procedure, named `(heap-insert-list f elements H)` which inserts all of the elements in list `elements` into heap `H` using first-order relation `f`. For example,

   ```
    (heap-insert-list > (list 9 5 7 3) (list))
   (9 (7 () ()) (5 () (3 () ())))
    (heap-insert-list > (list 2 8 4 6) (list))
   (8 (4 () ()) (6 () (2 () ())))
   ```

   (c) Define a SCHEME procedure, named `(combine f Ha Hb)`, which accepts three arguments, `f`, a first-order relation which is used to order the elements in the heap, and two heap structures, `Ha` and `Hb`, which have been constructed using the same first-order relation. For example, for two min-heaps

```
(define Ha (heap-insert-list > (list 9 5 7 3) (list)))
(define Hb (heap-insert-list > (list 2 8 4 6) (list)))
(combine > Ha Hb)
(9 (7 () (5 () (3 () ()))) (8 (4 () ()) (6 () (2 () ()))))
```

(d) Define a SCHEME function, named `(empty? H)` which takes one argument, a heap, and returns a boolean value, true if `H` is the empty heap and false otherwise.

(e) Define a SCHEME function, named `(heap-remove f H)` which takes two arguments, a heap and a first-order relation, a heap containing the elements of `H` with the root value removed. Note, `heap-remove` must take the first-order relation as a parameter to pass to other functions.

```
(heap-remove > (combine > Ha Hb))
(8 (6 (2 () ()) (4 () ())) (7 () (5 () (3 () ()))))
```