Introduction to Functional Programming

Lent 2006

Exercises on lists and trees

1. **Binary trees**.

   (a) Write a function `reverse` that creates the mirror image of a binary tree. That is, if `t` is a binary tree, then `reverse t` is a binary tree in which, at every node, left and right branches are interchanged.

   (b) Write a function `ToTree` of type `'a list -> 'a tree` which takes a sorted list and creates a balanced binary tree with the same elements.

2. **Red/Black trees**.

   (a) What is the result of

   `foldl (fn(x,t) => RBinsert x t) O [0,1,2,3,4,5,6,7,8,9]` ?

   (b) Does red/black-tree insertion preserve the inorder listing? And the preorder and postorder ones? If not, how would you modify the function so that it does in each case?

   (c) [Exercise 3.9 of *Purely functional data structures* by Chris Okasaki. CUP, 1998.] Write a function `fromOrdList` of type `int list -> int RBtree` that converts a sorted list with no duplicates into a red-black tree. Your function should run in linear time.

3. **Queues**.

   (a) The functional queue implementation of Lecture V can be extended to support a *double-ended queue*, or simply *dequeue*, that allows read and writes at both ends of the queue. The invariant is updated to be symmetric in its treatment of the pairs of lists implementing the front and rear of the queue in that both are required to be non-empty whenever the dequeue contains two or more elements. When one list becomes empty, the other list should be split in half, and an implementation of the dequeue satisfying the invariant should be given.

   Implement this version of dequeues, including `cons`, `head`, and `tail` functions that respectively insert, inspect, and remove the front element; and `snoc`, `last`, and `init` functions that respectively insert, inspect, and remove the rear element. Use exceptions to handle errors.

   References

   – Rob R. Hoogerwoord. A symmetric set of efficient list operations. *Journal of Functional Programming*, 2(4):505–513, 1992.

   – Exercise 5.1 of *Purely functional data structures* by Chris Okasaki. CUP, 1998.

   – Chris Okasaki. Purely functional data structures. Ph.D. thesis, CMU, 1996. (Available on-line from ⟨`http://www.cs.cmu.edu/~rwh/theses/okasaki.pdf`⟩.)

   (b) Write a function `'a tree -> 'a list` that lists the nodes of a binary tree in its breadth-first traversal using queues. Decompose your function into two functions, respectively of type `'a tree -> 'a Queue.t` and `'a Queue.t -> 'a list`.

   Write a function `'a tree -> int tree` that given a binary tree produces one of exactly the same shape, but with the nodes replaced by their corresponding number in breadth-first traversal.

   Reference

   – Chris Okasaki. Breadth-first numbering: Lessons from a small exercise in algorithm design. ICFP 2000. (Available on-line from ⟨`http://www.eecs.usma.edu/Personnel/okasaki/pubs.html`⟩.)