CSE 341 Assignment 3

January 24, 2006

This homework is mostly short problems involving functions using accumulators and simple uses of higher-order functions.

Due: Thursay, January 26, at 11 pm. No late assignments will be accepted. Submit your assignment using the online turnin link on the course web.

Implement SML functions to solve the following problems.

1. Write a function **lreduce** that takes a two-parameter function F and a list $[a_1, a_2, \ldots, a_n]$ and produces
$$F(\ldots F(F(a_1, a_2), a_3) \ldots a_n)$$
The function works the same way as **reduce** (see ML book p.162), but it groups the elements of the list from the beginning of the list instead of the end.

2. Write a function $foldl$ that takes a list of functions $[f_1, f_2, \ldots, f_n]$ and applies it to an argument i in the following way:
$$f_n(\ldots(f_2(f_1(i))))$$
Be aware that this is **NOT** the same $foldl$ as is provided with sml.

3. Part of a Pascal's triangle is depicted below. Each row is obtained in a very simple way from the previous row (involving additions of neighbouring pairs):
```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```
Define a function *pascal* that takes a single integer n as parameter and returns a *list* of *int lists* that forms the first n rows of such a triangle.

4. Define a function *innerproduct* that take two int lists and return the innerproduct:
$$innerproduct([x_1, x_2, \ldots x_n], [y_1, y_2, \ldots y_n]) = x_1 * y_1 + x_2 * y_2 + \ldots + x_n * y_n$$
If the two int lists that you get are not of the same size, you should throw a BadLists exception. To implement this function, you should use one or more high-order helper functions and have your *innerproduct* function call it. This function should *not* use tail recursion (i.e., an accumulator).

   Note: You **MUST** use high-order functions to solve this problem.

   (*hint*: one way of doing this is to use a high-order helper function that takes two functions and two lists as parameters such that when given the parameters **op *, op +, X, Y** produces the innerproduct result.)

5. Solve the previous problem again with a function *innerproductTail* that uses tail recursion.
6. **Quicksort Redux:** The SML language contains a function *partition* that takes a list and a boolean higher order function f that is called on each element in the list. If f returns true for a particular element, it is placed in a list l. Otherwise, it is placed in a second list m. When finished, the *partition* function returns a pair of lists consisting of (l, m). You are to create an implementation of this function and provide a higher order function *lessThan* that will, given a pivot point p, place all elements of the list that are less than or equal to p in the list l, and all other elements in m. For obvious reasons, you may **not** make use of the built in partition function.
7. Define an integer Tree datatype with constructors EmptyT and Tree. The Tree constructor contains an int, a left Tree, and a right Tree.
   - Define a function *sumTree* that recursively sums up the values of all the nodes in the tree and returns this sum. If the tree is empty, this function should return 0.
   - Define a function *countNodes* that returns the number of nodes in a tree. An empty tree has 0 nodes.
   - Define a function *findMax* that looks through the tree, and returns the value of the maximum node. If the tree is empty, it should return NONE.
   - Reimplement the previous parts of the question using a higher-order function that traverses a tree and applies a function that is its parameter to produce the results: sum of the nodes in the tree, number of nodes in the tree, and maximum value in the tree. You will need to also define three functions that are used as parameters to the higher-order function to compute the specific results. In addition to writing the high-order function *traverse*, you should also provide examples on how to pass it the correct functions to calculate sum, node count, and max value.

Type Summary.

The functions you implement should have the following types:

val lreduce = fn : ('a * 'a → 'a) * 'a list → 'a
val foldl = fn : ('a → 'a) list * 'a → 'a
val pascal = fn : int → int list list
val innerproduct = fn : int list * int list → int
val innerproductTail = fn : int list * int list → int
val partition = fn : fn: 'a list * ('a → bool) → 'a list * 'a list
val lessThan = fn: int → int → bool
val sumTree = fn : int Tree → int (*recursive version*)
val countNodes = fn : 'a Tree → int (*recursive version*)
val findMax = fn : int Tree → int option (*recursive version*)

(high-order *traverse* and usage omitted)

Sample Log Execution.

- lreduce(op mod, [139, 25, 10, 5]);
val it = 4 : int
*- foldl ([fn x ⇒ x * 2, fn y ⇒ y + 3], 1;*
val it = 5 : int
- pascal(5);
val it = [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]] : int list list
- innerproduct([1,3,5],[2,4,6])
val it = 44 : int

```
- partition([1,11,2,8,4], lessThan 5);
val it = ([1,2,4],[11,8]) : int list * int list
- val myTree = Tree(5, Tree(4, EmptyT, Tree(7, EmptyT, EmptyT)), Tree(8, Tree(11, EmptyT, EmptyT),
EmptyT) );
val myTree = Tree (5,Tree (4,EmptyT,Tree #),Tree (8,Tree #,EmptyT)) : int Tree
- sumTree(myTree);
val it = 35 : int
- countNodes(myTree);
val it = 5 : int
- findMax(myTree);
val it = SOME 11 : int option
```