CSE 341 Assignment 6

February 21, 2006

Due on Thursday, February 23rd 2006 at 11pm. No late assignments will be accepted.

As an introduction to Scheme, implement the following functions. For each of the problems, you should specify all helper functions (if any) in local let bindings rather than in the global environment.

1. Write a function `(foldr f lst a)` that works the same way as `foldr` in SML. As a reminder, `foldr` returns the result of
$$f(x_1, f(x_2, f(x_3, \ldots, f(x_n, a)\ldots)))$$
Your function should not make use of any helper functions to accomplish this. Once you have done this, define a lambda function `SumUp` which, when passed to `foldr`, will sum up the values of all the elements in the list if the expression `(foldr SumUp lst 0)` is evaluated.

2. A tree node can be represented using the list structure (item $(s_1 s_2 s_3 \ldots s_n)$), where item is the element stored in that node of the tree and the list s_1 through s_n is a list of subtrees with this node as their root. Write a function `find n tree` that recursively looks through the tree for a node with value n. If it finds such a node, it should return a list of integers corresponding to the values of the nodes on the path to the goal, and if it cannot locate such a node, it should return `#f`.

3. Recall that the binomial coefficient $C(n,k)$ can be defined as follows
$$\begin{aligned} C(n,0) &= 1 \\ C(n,k) &= 1\,, \quad \text{if } n = k \\ C(n,k) &= C(n-1,k-1) + C(n-1,k)\,, \quad \text{otherwise} \end{aligned}$$
This definition is known as Pascal's Identity, and forms the basis for Pascal's triangle. Write a function `(binomial1 n k)` that computes the binomial coefficient C(n, k) using the above definition. You may not use any helper functions to implement this.

4. Now, implement a function `(binomial2 n k)` that performs the same operation as `binomial1`, but utilizes memoization to make the process more efficient. You should maintain a memo table, and when you need to calculate some C(n, k), if that particular value has already been calculated, you should just look it up in the memo table rather than recalculating it. If the value has not already been calculated, use the recursive definition to determine what it should be, then mutate the memo table to include the value that you have calculated. Your memo table may not be a top level binding. Hint: Association lists are a useful data structure for the memo table — look in the Scheme language definition for details.

5. A non-recursive definition of the binomial coefficient C(n, k) can be defined as follows:
$$C(n,k) = \frac{n!}{k!(n-k)!}$$
Write a function `(binomial3 n k)` that calculates the binomial coefficient using the formula above.

6. MiniMax search is an Artifical Intelligence algorithm used in many game engines that attempts to determine the optimal move to make at some point in a game by attempting to predict the actions that the opponent will take. The goal of the algorithm is to maximize its score, while its opponent

tries to minimize the algorithm's score. This is calculated by utilizing a maximizing function on levels where the algorithm is making a move, and utilizing a minimizing function on levels where the opposing player is making a move. In other words, when the algorithm is making a move, it tries to select the best possible move based on the score of that move, and it assumes that when its opponent makes a move in response, the chosen move will be the one that minimizes the score of the algorithm. This operation can also be visualized as searching a "tree" of possible moves, alternatively looking for the minimum and maximum score at each level. When the algorithm is complete, it picks the move that maximizes its score after the user has tried to minimize it. This process can be performed over an arbitrary number of levels in a tree, but this is generally impractical due to the time needed to compute all possible moves, and the MiniMax search is normally limited to a set number of levels of recursion.

Using the tree node structure described in question 3, write a function `(MiniMax t)` that performs a MiniMax search on a tree and returns a pair consisting of the optimal branch to pick at node `t` and the value of that branch. The MiniMax algorithm can be described as follows:

```
function Max-Value(node) returns (value of branch)
  if (node is a leaf) return value in node
  else
    value = -infinity
    for each successor node s
      value = Max(value, Min-Value(s))
    return v

function Min-Value(node) returns (value of branch)
  if (node is a leaf) return value in node
  else
    value = infinity
    for each successor node s
      value = Min(value, Max-Value(s))
    return v

function MiniMax(tree) returns (optimal node, value of optimal node)
  if (node is a leaf) return (#f, value in node)
  else
    value = -infinity
    for each sucessor node s
      if (Min-Value(s) > value) optimalMove = s, value = Min-Value(s)
    return (optimalMove, value)
```

For this exercise, assume that the algorithm always makes the first move in the tree given to it. As such, the first level is always a Max level. Additionally, you may use the values 1E10 and -1E10 for infinity and negative infinity respectively. In order to implement this function, you should define 2 mutually recursive helper functions.

Turn-in instructions

- Place your solutions to the homework problems in a file called hw6.scm.
- The first line of the file that you turn in should be a comment with your name and section.
- Use the online turn-in form to submit your homework.