CS350 2012 Homework 1, v 1.0

August 14, 2012

Due: August 24, 2012

Instructions

1. Comment your code sensibly - neither too much nor too little - so that it is easy for the grader to understand your code.

2. Copying the code verbatim from any source is forbidden, but discussion is encouraged. Cite all your sources, including other groups that you discuss with.

3. If you spot any errors or omissions in the problem set, please let us know.

4. For all the following, use only the **declarative** style of programming we discussed in class.

Problems

1. Counting Change: Imagine that you are in an era where the following denominations of coins are still in vogue: 1 paisa, 5 paise, 10 paisa, 25 paise, and 50 paise. We are interested to know how many ways we can represent a given amount.

   Write a recursive procedure, which, given N paise, computes the number of ways to represent it using the given denominations. You can assume that amounts are given in paise alone - Rs. 1.10 is given as 110 paise, for example.

   Source: H. Abelson and G. J. Sussman, "The Structure and Interpretation of Computer Programs", M.I.T. Press 1996. (15 points)

2. Recall our discussion in class that a tail recursive call is one where we do not do any computation in the calling function after the recursive call returns: For example,

```
declare
fun   {SumUpTo N}
  if N==0
  then 1
  else N+{SumUpTo N-1}
end
```

is *not* tail-recursive, since we perform an addition in the calling function after the recursion returns.

Write a tail-recursive version of the Fibonacci sequence. In detail, write a tail-recursive function, which on input N, returns a list of the first N Fibonacci numbers. (10 points)

3. Programming on Lists.

   (a) Define a function `Equals`, which takes a nested list, whose leaf elements are literals, and checks for structural equality according to the following rules.

   1. Two literals are equal if they are the same. e.g. `nil` equlas `nil`, `a` equals `a`.
   2. Two lists are equal if every element in them are equal (note that this is a recursive condition.)

   (10 points)

   (b) Write a function to delete multiple consecutive occurrences of a literal from a list of literals, replacing it with a single one.

   For example,

   ```
   [the the springs are '\n' '\n' loose]
   ```

   should be converted into

   ```
   [the springs are '\n' loose]
   ```

   (15 points)

   (c) Define a function which takes two finite sorted lists of numbers as inputs and outputs their merged list. (10 points)

4. Infinite Series: We can express e^x using the following Power Series.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}.$$

   (a) Write a lazy function to implement the above series. That is, the function takes an input x, and computes an infinite list of terms. The i$^{\text{th}}$ element in the list is the value of the i$^{\text{th}}$ term in the above series.

   If `N` is an integer, then {`IntToFloat N`} converts `N` to the nearest floating point number. '/' without the quotes is the floating point division operator. (15 points)

   (b) Using the above function, write an approximation function which does the following: On input X and a number N, it computes the sum of the first N terms of the Taylor series of e^N. The `FoldR` function could be used for this. (10 points)

   (c) Write a function `Within` which takes two arguments - first, an infinite list and second, an ϵ. It should examine the elements of the list until it finds the first consecutive two elements in the list within epsilon of each other. It should then return the second value. Using `Within`, write a function which takes a variable `Epsilon` and evaluates the exponential series until consecutive terms are within `Epsilon` of each other. (20 points)

5. Higher Order Programming: Implement the following functions. Consider a list of elements of type T.

   (a) Filter: Takes two arguments. The first is a list of elements of type T. The second is a predicate, which maps T to either true or false.

   The result is a subsequence of elements in the list which satisfy the predicate.

   e.g.

```
{Filter fun X if X => 0 return true else false end
        [-1 0 1 -2 2]} = [0 1 2]
```

(10 points)

(b) FoldL: Implement a left-associative fold function. The `FoldR` in class associated to the right. That is,

```
{FoldR [a b c] F Identity} = {F a {F b {F c Identity}}}
```

whereas

```
{FoldL [a b c] F Identity} = {F {F {F Identity a} b} c}
```

(10 points)

6. Programming using Map and FoldR/FoldL.

(a) Redefine `Map` using `FoldR`. (10 points)

(b) We define a "feed" to be a list of lists of literals similar to the following.

```
[[spiderman rocks] [gow is so-so] [batman takes a hit] [colorless
green ideas sleep furiously]]
```

We are interested in which movies are generating the greatest buzz. [1] We are interested in a fixed list of keywords, such as

```
[spiderman batman wasseypur]
```

Define a function, using Map and FoldR/FoldL, to count the number of occurrences of the keywords in the "feed". The inputs to the function are the "feed" and the list of keywords. Ignore any list that has more than 4 occurrences of a single keyword. For the above feed, the output should be of the following form:

```
[[spiderman 1] [batman 1] [wasseypur 0]]
```

(20 points)

7. A binary tree can be represented using a record. A binary tree is either `nil` or is a record of the following form:

```
tree(key:X left:Y right:Z)
```

where `Y` and `Z` are binary trees.

Write a function `FoldTree` which folds over a tree, similar to `FoldR` over a list. The inputs are a tree, a binary function and the identity element of the binary function. (15 points)

8. Threads: Write a function taking an input `Count`. The function should run with two threads: One thread producing an infinite list of random bits, and another computing an average of the first `Count` number of random bits.

To generate random bits, you can use the function {`OS.rand`}. It produces a random integer. So to get a random bit, you can use {OS.rand} mod 2 (10 points)

[1] "There's no bad publicity except an obituary." - Brendan Behan