CS350 2011 Homework 1, v 1.1

August 5, 2011

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.

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.

3. Infinite Series: We can express $e^{(x)}$ using the following Power Series.

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

1. 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^{th} element in the list is the value of the i^{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.

2. 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^x. The `Fold_` function could be used for this.

4. 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

5. Higher Order Programming:

Implement the following functions. Consider a list of elements of type T.

1. Map: Takes two inputs. The first is a function mapping T to T. The second is a list.

   The output should be a list of results obtained when we apply the function to each element in the list.

   That is,

   ```
   {Map f [a b c]} = [{f a} {f b} {f c}]
   ```

2. 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]
   ```

3. FoldL: Implement a left-associative fold function. The `Fold_` in class associated to the right. That is,

   ```
   {Fold_ [a b c d] f Identity} = {f a {f b {f c d}}}
   ```

whereas

```
{FoldL [a b c d] f Identity} = {f {f {f a b} c} d}
```