Practice Problem Set – 2

Higher Order Functions and Lists

1. Suppose we wanted to write the following functions, explained through examples of how they may be called:

   (a) `(f0 1 100)`: Starting with 1, the sum of every 13th number in the interval 1 to 100.

   (b) `(f1 2 100)`: The sum of squares of all odd numbers between 2 and 100.

   (c) `(f2 1 100)`: The product of the factorials of multiples of 3 between 1 and 100.

   (d) `(f3 10 200)`: The sum of the squares of the prime numbers in the interval 10 to 200.

   (e) `(f4 50)`: The product of all positive integers less than `50` that are relatively prime to 50.

   Write a higher-order function called `filtered-accumulate` so that the functions described above are instances of `filtered-accumulate`. Write the functions `f0, f1, f2, f3` and `f4` in terms of `filtered-accumulate`. You can assume that the arguments to `f0, f1, f2, f3` and `f4` are positive.

2. We want to implement sets. Since the most important thing you can do with a set is to test membership of a given element, we want to represent sets as functions from some type to boolean. This function associated with a set is called the *charateristic function of the set* in set-theoretic jargon. Under such a representation, the empty set is represented as `(lambda (x) #f)`. This is because the empty set returns false when tested for membership with any element. Similarly the complement of the set represented by `f` would be `(lambda (x) (not (f x))`.

   Under such a representation, write definitions for the following set-theoretic operations:

   - `insert` (inserts an element in the set)
   - `member` (tests for membership)
   - `union`
   - `intersection`
   - `difference`

3. Let us now represent sets as lists. Write a function called `powerset`, which will take a set as input and return the powerset (set of all subsets) of the set. powerset should be written using foldr.

   ```
   powerset '(1 2 3) = (() (1) (2) (3) (1 2) (1 3) (1 2 3))
   ```

4. (a) Using foldr, write a function called inits all the initial segments of a list: Example:

```
(inits ()) = (())
(inits '(1 2 3 4))  = (() (1) (1 2) (1 2 3) (1 2 3 4))
```

(b) Using `foldr`, write a function called `tails` which will find all the tail segments of a list: Example:

```
(tails ())  = (())
(tails '(1 2 3 4)) = (() (4) (3 4) (2 3 4) (1 2 3 4))
```

(c) Now, using `inits`, `tails` and `map`, define a function called `sublists` which will find all the sublists of a list:

```
(sublist ()) = (())
(sublist '(1 2 3 4)) = (() (1) (1 2) (1 2 3) (1 2 3 4)
                       (2) (2 3) (2 3 4) (3) (3 4) (4))
```

5. Assume that matrices are represented as lists of lists. For example, the matrix

```
| 1 2 3 |
| 4 5 6 |
| 7 8 9 |
```

can be represented as `((1 2 3) (4 5 6) (7 8 9))`.

Write the following functions:

(a) `(transpose m)` - computes the transpose of a matrix `m`

(b) `(matmult m1 m2)` - multiplies two matrices `m1` and `m2` of appropriate sizes.

6. Consider a representation of polynomials in a single variable as a list of coefficients:

$x^2 + 2x$ is represented as `(0 2 1)`
$4x^3 + 7$ is represented as `(7 0 0 4)`
5 is represented as `(5)`

(a) Write a function `(evaluate valuex poly)` which will evaluate the polynomial `poly` when the value of x is `valuex`

Example: `(evaluate 0.3 ‘(7 0 0 4)) => 7.108`

(b) Write a function `(add poly1 poly2)` which will add the two polynomials `poly1` and `poly2`.

(c) Write a function `(split n)` which behaves as follows:

`(split 5) => ((0 5) (1 4) (2 3) (3 2) (4 1) (5 0))`

(c) Using `split`, write a function `(power n poly1 poly2)` which will return the coefficient of x^n in the product of the polynomials `poly1 and poly2`.

(d) Using `power`, write a function `(multiply poly1 poly2)` which will multiply the polynomials `poly1` and `poly2`.

`(multiply ‘(5 3 2) ‘(0 2 0 3 2)) = (0 10 6 19 19 12 4)`

7. Run-length encoding (RLE) is a very simple form of data compression in which runs of data (that is, sequences of the same data value) are stored as a single data value and count. As an example:

   `(rle ‘(1 1 1 3 3 2 5 5 5 4 4)) = ((1 3) (3 2) (2 1) (5 3) (4 2))`

8. Suppose we have two list of numbers, and we want to convert the first list to the second by doing one of three operations. We can insert a number into the first list in any position, we can delete an occurrence of a number from the first list, or we can replace one occurrence of a number with another number. Write a function `make-equal` that computes the minimal number of changes we need to make to convert the first list to the second?

9. Define a function `(flatten l)` which flattens out its argument list `l`. Here are examples of `flatten`:

   ```
   (flatten '((1 2 3) 4 5) = (1 2 3 4 5)}
   (flatten '((1 2 3) (4 5)) = (1 2 3 4 5)}
   (flatten '(((1 2 3) 4 5)) = (1 2 3 4 5)
   (flatten '(()(())) = ()}
   ```

10. n-bit Gray code is a sequence of n-bit strings constructed according to certain rules. For example,

    ```
    (gc 1) =  ((0) (1))
    (gc 2) =  ((0 0) (0 1) (1 1) (1 0))
    (gc 3) =  ((0 0 0) (0 0 1) (0 1 1) (0 1 0)
                (1 1 0) (1 1 1) (1 0 1) (1 0 0))
    ```

    Observe in the examples how `(gc n)` is constructed using `(gc (- n 1))` and then define `gc`. Assume that `n` $>$ `0`.

11. The solution of a Sudoku puzzle is correct if (a) Each of its rows is filled with digits 1 to 9 with each digit occurring once (b) Each of its columns is filled with digits 1 to 9 with each digit occurring once, and (c) each of its 3×3 squares is filled with digits 1 to 9 with each digit occurring once.

    Assume that a solution of a Sudoku puzzle is represented as list of lists, where each of the inner lists is a row of the solution. Write a function `check` that returns `#t` if the solution is correct with respect to conditions above and `#f` otherwise.

12. Write a function called shuffle, which will produce all interleavings of two lists `xs1` and `xs2`. As an example:

    ```
    (shuffle '(1 2) '(3 5 4)) =
        ((1 2 3 5 4) (1 3 2 5 4) (1 3 5 2 4) (1 3 5 4 2) (3 1 2 5 4)
         (3 1 5 2 4) (3 1 5 4 2) (3 5 1 2 4) (3 5 1 4 2) (3 5 4 1 2))
    ```

13. The elements of shuffled list need not be in the same order as in the example. 3. Given a list `xs = (x1 x2 x3...xn)` of numbers, the sequence of successive maxima, `(ssm xs)`, is the longest subsequence `(xj1 xj2 ... xjm)` such that `j1 = 1` and `xji < xjk` for `ji < jk`. For example, `(ssm '(3 1 3 4 9 2 10 7)) = (3 4 9 10)`. Define `ssm`

14. Write a function called `summands` in Haskell which takes as input a positive integer `n`, and produces a list containing all ways of writing `n` as a sum of positive integers. Example:

```
(summands 1)  = ((1))
(summands 2)  = ((1 1) (2))
(summands 3)  = ((1 1 1) (1 2) (2 1) (3))
(summands 4)  = ((1 1 1 1) (1 1 2) (1 2 1) (1 3) (2 1 1) (2 2) (3 1) (4))
(summands 5)  = ((1 1 1 1 1) (1 1 1 2) (1 1 2 1) (1 1 3) (1 2 1 1)
                   (1 2 2) (1 3 1) (1 4) (2 1 1 1) (2 1 2) (2 2 1)
                   (2 3) (3 1 1) (3 2) (4 1) (5))
```

15. The function `cprod` takes a list of lists and computes its Cartesian product:

```
(cprod ((1 2 3) (4) (5 6))) = ((1 4 5) (1 4 6) (2 4 5) (2 4 6) (3 4 5) (3 4 6))
```

Define `cprod`.

16. Imagine the following game: You are given a path that consists of white and black squares. You start on the leftmost square (which we'll call square 1) and your goal is to move off the right end of the path in the least number of moves. However, the rules stipulate that:

- If you are on a white square, you can move either 1 or 2 squares to the right.
- If you are on a black square, you can move either 1 or 4 squares to the right.

Write a function `fewest_moves` that takes a path representated as a list of 0's and 1's (1 is white and 0 is black) and computes the minimum number of moves. As an example:

`(fewest-moves '(1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1 1))` returns 6 and is obtained by stepping on the squares at positions 1, 2, 6, 10, 11 and 15.

17. Write a function `lcs` to find a longest common subsequence of two lists. For example the following call to `lcs`:

`(lcs '(1 2 3 2 4 1) '(2 4 3 1 2 1))` returns `(2 3 2 1)`.

18. Write a function `(perm n r)` which will behave as follows:

```
perm 5 1 = ((1) (2) (3) (4) (5))
perm 5 2 = ((1 2) (1 3) (1 4) (1 5) (2 1) (2 3) (2 4) (2 5)
            (3 1) (3 2) (3 4) (3 5) (4 1) (4 2) (4 3) (4 5)
            (5 1) (5 2) (5 3) (5 4)
           )
```

What is an appropriate value of `(perm n 0)`. You can assume that perm is always called with `n >= r`.

19. Define a function (choose n m) which will return all possible ways of choosing m numbers from the numbers 1 to n. Explain with examples only, the role of each function used in defining choose. (12 Marks)

    Examples: (choose 5 0) = (()) (choose 5 1) = ((1) (2) ... (5)) (choose 5 3) = ((1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5) (2 3 4) (2 3 5) (2 4 5) (3 4 5)) (choose 5 4) = ((1 2 3 4) (1 2 3 5) (1 2 4 5) (1 3 4 5) (2 3 4 5))

20. Pascal's triangle is a sequence of rows of numbers that go like this:

```
         1
       1   1
     1   2   1
   1   3   3   1
 -  -   -   -   -
```

   Suppose that Pascal's triangle were represented as a list of lists. Write a function `(pascal k)` which will generate `k` rows of Pascal's triangle.

21. A segment of a list is a group of contiguous elements of the list. Given a list containing positive and negative numbers, find the maximum sum of any segment in the list. As an example, if the list is:

```
2 -5 7 8 -6 12 -2 -7 4 -16 8
     ---------
```

   Then the underlined segment has the maximum sum, which is 21.

   Write a function named `mss` using a single `foldr`, which will compute the maximum segment sum of a given list. The function should not use any recursive function other than foldr.

To be continued...