Homework 3 — assigned 22 March — due Monday 2 April

General instructions

A skeleton Haskell source file homework3.hs will be provided on the Slack channel #homework3 with the type signatures of the functions you are to write. Please edit that file and submit.

The skeleton file will start with the `import` declarations for the Haskell library `Data.List`, which we expect you to find useful. You may not add any other `import` declarations.

3.1 Lists and trees (10pts)

Do exercise 4 on page 109 of the textbook.

3.2 Simple functions on numbers (10pts)

The Goldbach conjecture states that any even number greater than two can be written as the sum of two prime numbers. Using list comprehensions, write a function

```
goldbach :: Int -> [(Int,Int)]
```

which, when given an even number n, returns a list of all pairs of primes which sum to n. Note: You will have to write a function which tests an integer for primality and this should be written as a list comprehension also. For example, `goldbach 6` should evaluate to `[(3,3)]`. When the two primes in the pair are unequal, report them only once, smaller prime first. Report the pairs in lexicographically sorted order. Thus, `goldbach 20` should evaluate to `[(3,17),(7,13)]`.

3.3 Higher-order functions (10pts)

The function `church :: Int -> (c -> c) -> c -> c` takes an integer n as its argument and returns a function which composes any unary function n times.

For example, `(church 4) tail "ABCDEFGH"` evaluates to `"EFGH"`. Write `church` using `foldr`.

3.4 Recursive functions over lists (10pts)

Let us use the Haskell type `[Int]` to represent sets of integers. The representation invariants are that there are no duplicates in the list, and that the order of the list elements is increasing. Write a Haskell function `powerset :: [Int] -> [[Int]]` that takes a set S and returns its powerset 2^S. (The powerset 2^S of a set S (sometimes written $P(S)$) is the set of all subsets of S.) Note that the result uses the Haskell type `[[Int]]` to represent sets of sets of integers. Here the representation invariant is that there are no duplicates in the list; the order of the sublists is immaterial.

3.5 Lists and strings (10pts)

In this exercise, we develop a simple tool for drawing. A drawing is just a line drawing consisting of some number of polygons. A polygon is given as a list of vertices, and a vertex is simply a pair of real numbers for the x and y coordinates. For instance,

```
[[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
 [(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

is an internal representation in Haskell of a drawing consisting of one triangle and one square.

Your task is to convert such a representation of a drawing into a simple page description in the PostScript language. Specifically, you are to write a Haskell function

```
makeCommand :: [[(Double,Double)]] -> String
```

The result returned by `makeCommand` is a Haskell value of type String, which must contain valid PostScript commands for drawing the given polygons.

For instance, the expression

```
makeCommand [[(100.0,100.0),(100.0,200.0),(200.0,100.0)],
             [(150.0,150.0),(150.0,200.0),(200.0,200.0),(200.0,150.0)]]
```

should evaluate to the text:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 100.0 100.0 200.0 200.0

100.0 100.0 moveto
100.0 200.0 lineto
200.0 100.0 lineto
closepath
stroke

150.0 150.0 moveto
150.0 200.0 lineto
200.0 200.0 lineto
200.0 150.0 lineto
closepath
stroke

showpage
%%EOF
```

which would be printed by a PostScript printer as in Figure 1.

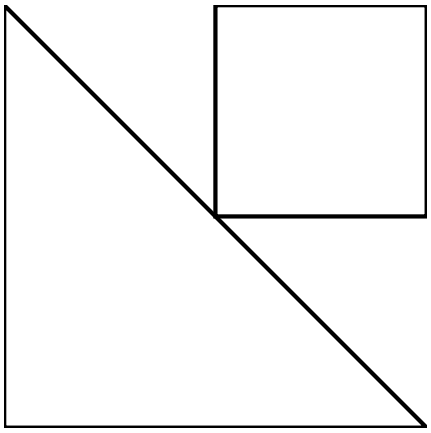

Figure 1: A triangle and a square.

Note that the bounding box is the smallest upright rectangle such that no points of the drawing lie outside it; it is specified by giving the coordinates of its lower left and upper right corners, in our example $(100.0, 100.0)$ and $(200.0, 200.0)$.

3.6 Trees (25pts)

We can define binary trees without any interesting content as follows:

```
data T = Leaf | Node T T
```

A path from the root to any subtree consists of a series of instructions to go left or right, which can be represented using another datatype:

```
data P = GoLeft P | GoRight P | This
```

where the path `This` denotes the whole tree. Given some tree, we would like to find all paths, i.e., the list of all paths from the root of the given tree to each of its subtrees. Write a function `allpaths :: T -> [P]` to do so.

For instance, `allpaths (Node Leaf (Node Leaf Leaf))` should evaluate to `[This,GoLeft This,GoRight This,GoRight (GoLeft This),GoRight (GoRight This)]` (but the ordering of the paths is immaterial).

3.7 Logic (25pts)

We can use the following type `Expr` to represent Boolean formulas in conjunctive normal form succinctly:

```
type Expr  =  [[Int]]
```

In this representation, for instance, `[[-1, 2, 4], [-2, -3]]` stands for the more conventional $(\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_3)$.

Write a function `eval :: (Int -> Bool) -> Expr -> Bool` to compute the Boolean value of a formula under a given assignment of Boolean values to the variables that appear in the formula; here the first argument is a function that describes the assignment.

Write a function `satisfiable :: Expr -> Bool`, which determines if the given formula is satisfiable, i.e., true for some assignment of Boolean values to the variables that appear in the formula.

How to turn in

Use the UNM Learn facility as follows: Navigate to `https://learn.unm.edu/` and log in. Then click on CS-357L-000 (Spring 2018). Now click on Assignments in the left side navigation menu. After that, click on the appropriate homework assignment link. Now attach your `.hs` file(s) and click submit. You are allowed to submit as many times as you like but only the latest submission will be graded.