Functional Programming for Logicians
Homework 5

Péter Mekis
Department of Logic, ELTE Budapest

Deadline: 2019 March 18 17:59 pm

- Solve *any three of the following ten exercises*. Solving more than three is appreciated, but not necessary. Some of the exercises are follow-ups to others; it may be a good idea to choose them together.
- Create *a miniproject of your own*, based on your ideas, preferably inspired by your main field of interest. It should contain more than one function definitions working together on a meaningful task.

1–10 Explain how the functions defined by the ugly one-liners below are evaluated, and what they return. Here's a sample:

```
f0 :: Int -> [Int]
f0 a = take a b where b = (1 : [a * c | c <- b])
{-
  f0 a returns the list of the first a powers of a: eg. f0 2 = [1,2];
  f0 5 = [1,5,25,125,625]. The embedded definition of b contains
  self-reference, but it still works. Let's see how. The head of the list, 1,
  is fixed; then the head is used to evaluate the next item 1 * a = a, which is
  then used to evaluate the third, 1 * a * a, and so on. Note that b is an infinite
  list. The whole construction works because Haskell has lazy evaluation.
  Since take takes only a finite portion of b, only a finite portion of values
  is calculated, each of which is determined by the elements before it, thus
  making the evaluation process finite.
-}
```

And now the exercises (you can find them in `haskell_hw5.hs`):

```
f1 :: (Eq a) => [a] -> Bool
f1 a = minimum (zipWith (==) a (reverse a))

f2 :: [String] -> Char -> String
f2 a b  = foldl1 (c b) a where c b d e = d++b:e

f3 :: Int -> Integer
f3 a = (filter b [1..]) !! a where b c = all d [1..c-1] where d e = gcd c e == 1

f4 :: Integer -> [Integer]
f4 a = if a == 0 then [1] else zipWith (+) (0:b) (b++[0]) where b = f4 (a-1)

f5 :: Int -> Int
f5 a = b !! a where b = 1:[c * b !! (c-1) | c <- [1..]]

f6 :: Int -> Integer
f6 a = b !! (a+1) where b = 1:[sum (take c b) | c <- [1..]]
```

```
f7 :: Eq a => [a] -> [[a]]
f7 a = if a == [] then [[]] else map (head a:) b ++ b where b = f7 (tail a)

f8 :: Int -> [Int]
f8 a = concat (map b [1..a]) where b a = [1..a]

f9 :: Eq a => [[a]] -> [[a]]
f9 a = if head a == [] then [] else map head a : f9 (map tail a)

f10 :: Int -> Integer
f10 a = b !! a where b = 0 : 1 : [x+y | (x,y) <- zip b (tail b)]
```