Functional Programming (Sheet #3)

Marius Gavrilescu

1. The type is `Int -> Int -> Int`. (a) is false, (b) is true, (c) is false.

2. ```
   uncurry :: (a -> b -> c) -> (a,b) -> c
   uncurry f (x,y) = f x y
   ```

3. ```
   const    :: a -> b -> a
   subst    :: (a -> b -> c) -> (a -> b) -> a -> c
   apply    :: (a -> b) -> a -> b
   flip     :: (a -> b -> c) -> b -> a -> c
   strange  :: ((a -> b) -> a) -> (a -> b) -> b
   -- stranger does not make sense as it would require an infinite type
   square   :: Num a => a -> a
   ```

4. ```
   type Distance = Float
   (===) :: Distance -> Distance -> Bool
   (===) a b
     | abs(a - b) <= 10 = True
     | otherwise        = False
   ```

   Calling it `(==)` is ambiguous as the name conflicts with that of `Prelude.(==)`.

5. ```
   instance (Ord a) => Ord [a] where
   compare [] [] = EQ
   compare a  [] = GT
   compare [] a  = LT
   compare (a:as) (b:bs)
     | compare a b == EQ = compare (as) (bs)
     | otherwise = compare a b
   ```

6. `map (map square) [[1,2],[3,4,5]]=[map square [1,2], map square [3,4,5]]=[[1,4],[9,16,`

7. `map f` $\bot$ `=` $\bot$ and `map` $\bot$ `[] = []`

8. `map map :: [a -> b] -> [[a] -> [b]]`

9. ```
   filter p = concat . map box
     where box x = if p x then [x] else []
   ```

10. (a) is true, because $\sum_{i \in I} 2 * x_i = 2 \sum_{i \in I} x_i$

    (b) is true, because $\sum_{i \in I} \sum_{j \in J_i} x_i = \sum_{i \in I, j \in J_i} x_i$

    (c) is true, because addition is commutative.
   ```