Functional Programming for BDA - List 3

User-defined types and classes; trees.

Marcin Michalski, DCS FFPT WUST 2020/2021

Deadline: 05.12.2020 $(0:00-\varepsilon)$

Treat Exercises as a warm up. Submit Tasks 1 Task 2 and (optional) Bonus 1 only. Bonus 1, as the name suggest is optional and is worth extra 5 points. Your solutions should be as elementary and effective as practically possible, e.g. avoid using functions and syntactic aids that were not (yet) introduced like notation `do`, bind `>>=`, etc.

Exercise 1. Use lambda calculus to implement the following functions

a) $a(x,y)=x+y$;

b) $\text{const}_x(y)=x$;

c) $\pi_2(x,y)=y$;

d) $\text{id}(x)=x$

Exercise 2. Express `map` via `foldr` using lambda expressions.

Exercise 3. Let `f :: b -> c` and `g :: a -> b`. Express `f.g` (composition) using lambda expressions.

Exercise 4. The expression `let x=y in z` is a "syntactic sugar" for a certain expression in lambda calculus. Desugar it.

Exercise 5. Let π_1 =`\xy->x` and π_2 =`\xy->y`. Calculate $\pi_1.\pi_2$ and $\pi_2.\pi_1$.

Exercise 6. Come up with a lambda expression that cannot be β-reduced to a form that does not allow further β-reduction.

Exercise 7. Let us define

```
data Point = Point Float Float
data Shape = Circle Point Float | Rectangle Point Point
```

where `Circle x y` models a planar circle with the center x and the radius y and `Rectangle x y` models a rectangle with top-left corner x and bottom-right corner y. Implement a function that calculates the surface of a given shape.

Exercise 8. Let us define

```
data Vector3D a = Vector a a a
```

that models 3D vectors. Define addition, multiplication by a scalar and scalar multiplication for your vectors. Make it an instance of class `Show`.

Exercise 9. Consider the following IntOrString type

```
data IntOrString = Word String | Number Int
```

Make it an instance of classes `Eq` and `Show`.

Task 1. Let us recall the definition of a binary tree structure

```
data Tree a = Leaf a | Node (Tree a) a (Tree a)
```

a) Make it an instance of the class `Show` so tree are printed nicely.

b) Make it an instance of the class `Foldable` by implementing `foldr` that starts with the right-most branch.

c) Implement functions that: count numbers of roots and leafs, determine whether a given x is an element of a tree, calculate the height of a tree. Fold may be useful here.

Task 2. Define trees with roots that may have any number of children (*Hint:* *`[Tree a]`*). Then

a) Make it a nice looking instance of the class `Show`.

b) Make it an instance of the class `Functor`, i.e. define `fmap` for your trees.

c) Make it a partial instance of the class `Foldable`, i.e. define some kind of fold.

d) Repeat c) of the previous Task.

Bonus 1. Use the following tagged tree type

```
data Tagged_tree a =Empty |
                    Leaf Integer a |
                    Node Integer (Tagged_tree a) (Tagged_tree a)
```

to transform lists to trees so that the k-th element of the list is the k-th leftmost leaf of the tree and you can access elements of the list in logarithmic time. The function

```
list_to_tree :: [a] -> Tagged_tree a
```

should transform a given list into a balanced tree in $O(n\log(n))$ time. The function

get_elem :: Integer -> Tagged_tree a$->a$

should return the k-th element from the tagged tree in $O(\log(n))$ time.