Programming Paradigms Fall 2022 — Problem Sets

by Nikolai Kudasov

October 20, 2022

1 Problem set №8

1. Implement the following functions over lists. You can use explicit recursion or higher-order functions that we covered so far. Make sure that your functions are **lazy** and work properly on all provided examples:

   (a) A function that checks whether a given list is a singleton (contains exactly one element):

   ```
   isSingleton :: [a] -> Bool
   ```

   ```
   >>> isSingleton [1]
   True
   >>> isSingleton [1..]
   False
   >>> isSingleton [[1..]]
   True
   ```

   (b) A function that inserts a number into a sorted list of numbers:

   ```
   insert :: Int -> [Int] -> [Int]
   ```

   ```
   >>> insert 3 [1,2,5,7]
   [1,2,3,5,7]
   >>> insert 3 [0,1,1]
   [0,1,1,3]
   >>> take 5 (insert 3 [1..])
   [1,2,3,3,4]
   ```

   (c) A function that puts a separator between every two consecutive elements:

   ```
   separateBy :: a -> [a] -> [a]
   ```

   ```
   >>> separateBy ',' "hello"
   "h,e,l,l,o"

   >>> take 5 (separateBy 0 [1..])
   [1,0,2,0,3]
   ```

   (d) Split a list into a maximal prefix where all elements satisfy the predicate and suffix (all leftover elements):

   ```
   splitWhenNot :: (a -> Bool) -> [a] -> ([a], [a])
   ```

   ```
   >>> splitWhenNot (/= ' ') "Hello, world!"
   ("Hello,"," world!")

   >>> take 10 (fst (splitWhenNot (< 100) [1..]))
   [1,2,3,4,5,6,7,8,9,10]

   >>> take 10 (snd (splitWhenNot (< 100) [1..]))
   [100,101,102,103,104,105,106,107,108,109]

   >>> take 10 (fst (splitWhenNot (> 0) [1..]))
   [1,2,3,4,5,6,7,8,9,10]
   ```

(e) A function that groups elements, removing separators (elements that satisfy a given predicate):

```
groupsSeparatedBy :: (a -> Bool) -> [a] -> [[a]]
```

```
>>> groupsSeparatedBy (== ' ') "Here are some words!"
["Here","are","some","words!"]

>>> take 3 (groupsSeparatedBy (\n -> n `mod` 4 == 0) [1..])
[[1,2,3],[5,6,7],[9,10,11]]
```

(f) A function that replicates each element x_i of the list i times:

```
replicateWithPos :: [a] -> [a]
```

```
>>> replicateWithPos [1..3]
[1,2,2,3,3,3]

>>> replicateWithPos "Hello"
"Heellllllooooo"

>>> take 10 (replicateWithPos [1..])
[1,2,2,3,3,3,4,4,4,4]
```

2. Define the following infinite lists:

(a) A sequence of Lucas numbers: 2, 1, 3, 4, 7, ... The sequence starts with 2, 1, and each of the next elements x_n is defined as the sum of the two previous elements: $x_n = x_{n-1} + x_{n-2}$:

```
lucas :: [Int]
```

```
>>> take 10 lucas
[2,1,3,4,7,11,18,29,47,76]
```

(b) A sequence of approximations of the square root of 2. Any given approximation x can be improved into a better approximation x' using the formula $x' = x - \frac{x}{2} + \frac{1}{x}$.

```
approximationsOfRoot2 :: Double -> [Double]
```

```
>>> take 5 (approximationsOfRoot2 1)
[1.0,1.5,1.4166666666666665,1.4142156862745097,1.4142135623746899]
```