Programming Paradigms Fall 2022 — Problem Sets

by Nikolai Kudasov

September 29, 2022

1 Problem set №5

1. Implement the following functions over a list of binary digits in Haskell using **explicit recursion**. Each function should be implemented independently.

   (a) Convert into a decimal number:

   ```
   binaryToDecimal [1,0,1,1,0]      -- 22
   ```

   (b) Count zeros in a binary string (not counting leading zeros):

   ```
   countZeros [0,0,0,1,0,1,1,0]   -- 2
   ```

   (c) Encode a binary string by removing leading zeros and replacing each consecutive substring of digits with its length. For example, `[0,0,0,1,1,0,1,1,1,0,0]` has some leading zeros,then 2 ones,then 1 zero,then 3 ones,then 2 zeros,so it should be encoded as `[2,1,3,2]`):

   ```
   encodeWithLengths [0,0,0,1,1,0,1,1,1,0,0]    -- [2,1,3,2]
   ```

   (d) Check whether a given binary string represents an odd number:

   ```
   binaryOdd [1,0,1,1,0]    -- False
   binaryOdd [1,0,1,1,1]    -- True
   ```

   (e) Decrement a binary number. Decrementing zero should produce zero:

   ```
   decrement [1,0,1,1,0]    -- [1,0,1,0,1]
   decrement [1,0,0,0,0]    -- [1,1,1,1]
   decrement [0]            -- [0]
   ```

   (f) Implement function `propagate :: (Bool, [Int]) -> [(Bool, Int)]` that pairs a given boolean value with every integer in the list:

   ```
   propagate (False, [1, 2, 3])    -- [(False,1),(False,2),(False,3)]
   propagate (True, [1, 1])        -- [(True,1),(True,1)]
   ```

2. Implement in Haskell a function `alternatingSum` that computes a sum of a list of numbers, multiplying every second number by `-1`.
   For example, `alternatingSum [6,2,4,1,3,9]` should compute $6 - 2 + 4 - 1 + 3 - 9 = 1$.

   (a) Implement `alternatingSum` using explicit recursion.

   (b) Use equational reasoning to analyze evaluation of `alternatingSum [1,2,3,4,5]`.

3. Consider the following definitions:

   ```
   data Radians = Radians Double
   data Degrees = Degrees Double

   pi :: Double
   pi = 3.14159
   ```

   Implement the following functions that convert between degrees and radians:

   ```
   toDegrees :: Radians -> Degrees
   fromDegrees :: Degrees -> Radians
   ```