Technical University of Munich
Chair for Logic and Verification
Prof. Tobias Nipkow, Ph.D.
J. Rädle, L. Stevens, K. Kappelmann, MC M. Eberl

WS 2019/20
01.11.2019
Deadline: 11.11.2019, 23:59

Functional Programming and Verification
Sheet 3

Tutorial Exercises

Exercise T3.1 Matrices

1. Write a function `dimensions :: [[a]] -> (Int,Int)` that determines the dimensions of its input matrix encoded as a list. For example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

   will be encoded as `[[1,2,3,4][5,6,7,8][9,10,11,12]]`. Calling `dimensions` on this matrix should return `(3,4)`. If the input is not a valid matrix, e.g. if one row contains fewer elements than the other rows, the function should return `(-1,-1)`.

2. Define a predicate `isSquare :: [[a]] -> Bool` that returns true iff its input is a square matrix. Also define predicates

```
canAdd :: [[a]] -> [[a]] -> Bool
canMult :: [[a]] -> [[a]] -> Bool
```

   that determine whether their input matrices have the right dimensions to be added or multiplied together.

3. Write a function `diagonal :: [[a]] -> [a]` that returns the diagonal of a square matrix encoded as a list. For example, the diagonal line of the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

   above matrix is `[1,5,9]`.

   Can you extend this to a function that takes the diagonal line of a cube?

4. Define functions

```
matrixAdd :: [[Integer]] -> [[Integer]] -> [[Integer]]
matrixMult :: [[Integer]] -> [[Integer]] -> [[Integer]]
```

   which add/multiply two matrices.

   Hint: for multiplication, you may want to use the `transpose` function from the `List` library.

Exercise T3.2 Merge Sort

In the lecture you have seen a Haskell implementation of Quicksort. In this assignment you will have to implement *Merge Sort* in Haskell.

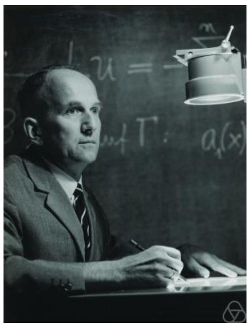
Recall: Merge Sort is based on the divide-and-conquer principle. First, it splits a list in two halves and sorts these lists separately. In the conquer step, it merges the two sorted lists. Note that this can be done recursively by comparing the two heads of the lists.

- Implement a Haskell function `mergeSort :: [Integer] -> [Integer]` that sorts an integer list in ascending order by using Merge Sort. To split the list, you can use the functions `take` and `drop`.
- Implement a function `adjacentPairs :: [a] -> [(a,a)]` that generates all adjacent pairs of elements from a given list.
- Test your sorting function by checking whether all adjacent pairs of its result are indeed in the correct order.

Exercise T3.3 Collatz Conjecture

We define the following function $f : \mathbb{N} \to \mathbb{N}$

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{otherwise} \end{cases}$$


Lothar Collatz

and, for every $n \in \mathbb{N}_+$, the sequence

$$c_0 = n, \quad c_{i+1} = f(c_i).$$

The longstanding Collatz conjecture states that for every $n \in \mathbb{N}_+$, the sequence $(c_i)_{i\in\mathbb{N}}$ stabilises to 1.

1. Write a function `collatz :: Integer -> [Integer]` such that `collatz n` computes the sequence $(c_i)_{i\in\mathbb{N}}$ until it stabilises at 1. For example,

   ```
   collatz 12 = [12,6,3,10,5,16,8,4,2,1].
   ```

2. Write a quickCheck test that checks whether the collatz conjecture holds for $1 \leq n \leq 100$.

Homework

You need to collect 5 out of 6 points (P) to pass this sheet.

Exercise H3.1 Decomposition [1+2: 1P, 3+4: 1P]

1. Write a function `decomposition :: Int -> [Int]` such that `decomposition n` returns the list of all prime divisors of $n \in \mathbb{N}_+$ in ascending order. For example,

   ```
   decomposition 180 = [2,2,3,3,5].
   ```

   Use this to write a function

   ```
   decomposition2 :: Int -> [(Int, Int)]
   ```

   such that `decomposition2 n` returns the list of all prime divisors of `n` including their multiplicity. For example,

   ```
   decomposition2 180 = [(2,2),(3,2),(5,1)].
   ```

2. Write two functions

   ```
   isPrime :: Int -> Bool
   primes :: Int -> [Int]
   ```

   such that `isPrime n` returns true if and only if `n` is prime and `primes n` returns the list of all prime numbers up to `n`. Use the functions defined in task 1.

3. Implement a function

   ```
   takes :: [Int] -> [a] -> [[a]]
   ```

   such that for every list of positive natural numbers `ns` and list `xs`, the function creates a list of sublists of `xs` whose lengths are specified by `ns`. For example,

   ```
   takes [1..3] [0..5]  = [[0],[1,2],[3,4,5]]
   takes [1..5] [0..11] = [[0],[1,2],[3,4,5],[6,7,8,9],[10],[11]]
   takes [1..3] [0..11] = [[0],[1,2],[3,4,5]]
   ```

   As you can see in the second example, if the next length specified by `ns` is greater than the remaining size of `xs`, the function inserts the remaining elements as singletons. Moreover, if the sum of `ns` is smaller than the length of `xs`, the process just stops earlier as you can see in the third example.

4. Write a function

   ```
   takePrimes :: [a] -> [[a]]
   ```

   that creates a list of sublists of a given list whose lengths are consecutive prime numbers (starting from 2). For example,

   ```
   takePrimes [1..7]  = [[1,2],[3,4,5],[6],[7]]
   takePrimes [1..12] = [[1,2],[3,4,5],[6,7,8,9,10],[11],[12]]
   ```

Exercise H3.2 Laurent Polynomials [1: 1P, 2+3: 1P]

In this exercise, we create yet another mathematical library. This time: Laurent polynomials. Laurent polynomials, named after Pierre Alphonse Laurent, are just like regular polynomials except that they may also contain negative exponents. We consider Laurent polynomials with coefficients in $\mathbb{Z}$, that is expressions of the form $\sum_{i=m}^{n} c_i x^i$ with $c_i, m, n \in \mathbb{Z}$. For example, $x^{-2} - 2x^{-1} + 5 + x^2$ and $-x^{-1} + 9$ are Laurent polynomials.

We encode Laurent polynomials as a list of coefficients and exponents of type `[(Integer,Integer)]`. We always keep our polynomials sorted in ascending order by their exponents. Moreover, as we want to save precious bytes on our computer, we only store entries whose coefficients are non-zero. For example, the polynomial $x^{-2}-2x^{-1}+5+x^2$ will be encoded as `[(1,-2),(-2,-1),(5,0),(1,2)]`.

1. Write a function

   ```
   add :: [(Integer,Integer)] -> [(Integer,Integer)] ->
          [(Integer,Integer)]
   ```

   that adds together two Laurent polynomials.

2. Write a function

   ```
   derivative :: [(Integer,Integer)] -> [(Integer,Integer)]
   ```

   that takes the formal derivative of a Laurent polynomial:

   $$\text{derivative}\left(\sum_{i=m}^{n} c_i x^i\right) = \sum_{i=m}^{n} i c_i x^{i-1}$$

3. Write a function

   ```
   flipNegExp :: [(Integer,Integer)] -> [(Integer,Integer)]
   ```

   that transforms a Laurent polynomial to a regular polynomial by flipping the sign of its negative exponents. For example,

   ```
   flipNegExp [(1,-2),(-2,-1),(5,0),(1,2)] = [(5,0),(-2,1),(2,2)]
   ```

Exercise H3.3 Penguin Tribalism [1+2: 1P, 3+4: 1P]

Penguins, as we all know, are pack animals, that is they like to pack things into binary format. To prevent overpopulation, the well-known species of J.K. penguins invented complex rules written down in binary in the their book of life that governs their society. They decide based on their local neighbours whether they want to reproduce or fight to death as written in the holy book. Sadly, just recently on Halloween, they got "verHEXt" by a evil witch that transformed all their binary rules into hexadecimal.

1. Help the J.K. penguins to break the witch's spell and retranslate the hexcodes back to binary. More precisely, write a counterspell

```
unspell :: String -> [Int]
```

such that `unspell s` converts `s` to its binary representation. Moreover, the output list's length must be a multiple of four. For example,

```
unspell  "7" = [0,1,1,1]
unspell  "b" = [1,0,1,1]
unspell "f3" = [1,1,1,1,0,0,1,1]
```

2. Thanks to you, the penguins have access to their much needed rules again. Each rule consists of a triple (l, m, r) and an outcome o with $l, m, r, o \in \{0, 1\}$. A triple (l, m, r) represents three neighbouring positions (left,middle,right) that are either occupied by a penguin (1) or not (0). The outcome o of the rule indicates whether a penguin will be born (1) or have to say goodbye (0).

   There exists exactly one rule for each configuration of l, m, r. The binary loving penguins hence sophisticatedly packed the rules into a list of length 8 in the following way: the outcome of rule (l, m, r) will be saved at position $(lmr)_2$ in the list, where $(lmr)_2$ is the decimal number obtained by interpreting lmr as a binary number. For example, the outcome of rule $(1, 1, 0)$ will be saved at index 6.

   Write a function `index :: Int -> Int -> Int -> Int` such that `index l m r` computes the index of rule (l, m, r) as described.

3. Every full moon, the penguins can now again conduct their anti-overpopulation ritual. For this, they all form a line with possible gaps and apply the rules as written down in their book – and decoded by you – to their society.

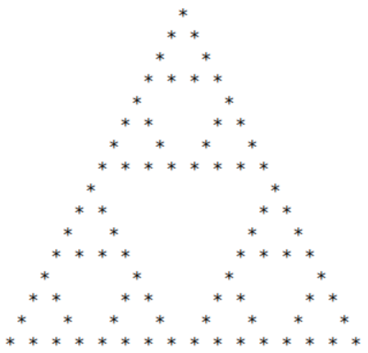
   For each position p of the line, the penguins apply the rule encoded by p and its two neighbours to p. For example, if there is a penguin at position p accompanied by a left penguin neighbour but an empty space to its right, the rule $(1, 1, 0)$ applies to p. Position p will consequently be replaced by outcome o of rule $(1, 1, 0)$ as saved at index 6 in the list of rules. Just last full moon, for example, the J.K. penguins formed a line `[0,1,0,1]` and used the rules `[0,1,0,1,0,1,0,0]` resulting in a new penguin colony `[1,0,1,0]`.

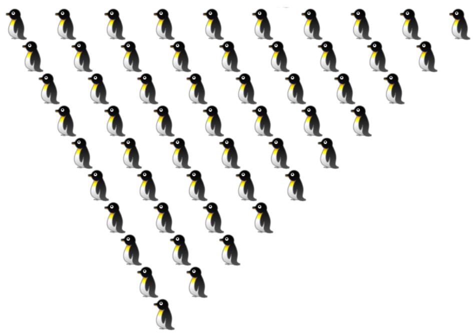
   Write a function `ritual :: [Int] -> [Int] -> [Int]` that takes the list of rules and line of penguins and returns the outcome of the ritual.

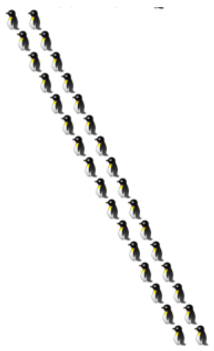
4. The famous penguin and Linux mascot Tux decided that he wants to simulate the destiny of his penguin fellows and obtain, when printed to his favourite GNU terminal, a visually appealing sequence of the penguin population.

   Write a function `simulate :: [Int] -> [Int] -> Int -> [[Int]]` such that `simulate r p n` returns the list of penguin populations obtained by running the ritual following the rules `r` starting with population `p` for `n` times. For example,

```
  simulate [0,1,0,1,0,1,0,0] [0,1,0,1] 2
= [[0,1,0,1],[1,0,1,0],[0,1,0,0]]
```


Sierpiński triangle


Death of the penguins


Penguin couple

Some of the MC Jr's splendid penguin simulations.

As Tux is a strong adherent of open-source software, he shares his visualisation function `showPenguins` with you that takes a list of penguin populations (`[[Int]]`) and an appealing penguin visualisation (`Char`) and prints the results of the ritual to your terminal.

The MC Jr. created some stunning images of penguin colonies using the following snippets

```
showPenguins (simulate (unspell "5a")
  (take 60 (repeat 0) ++ [1] ++ take 60 (repeat 0)) 31) '*'

showPenguins (simulate (unspell "16")
  (concat $ take 60 (repeat [0,1])) 60) '*'

showPenguins (simulate (unspell "2f")
  ([1,1] ++ take 30 (repeat 0)) 15) '*'
```

Be creative and submit your most tasteful colonies on Piazza.

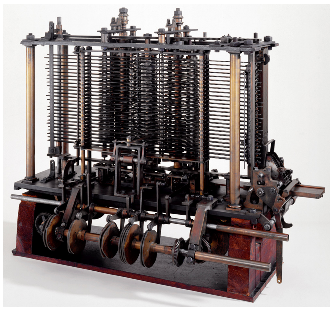
Continue on next page!

It's ~~turtles~~ penguins all the way down.

— original author unknown

Ada Lovelace

Charles Babbage


Part of the *Analytical Engine*

A replica of the Zuse Z3 in the *Deutsches Museum*

Exercise H3.4 (**Competition only**) Bernoulli returns [0P]

Attention! This exercise does *not* give you any homework points. It is only intended for those who want to compete in the competition.

The MC Sr and his minions could not be bothered to come up with a new competition problem, so they decided to simply pose the same problem as last week again: Implement a function

```
bernoulli :: Integer -> Rational
```

that, given a non-negative integer n, computes the n-th Bernoulli number B_n.

However, to mix things up a little bit, this week the MC wants you to optimise your solutions for *performance*, not for tokens. Since the simple approach suggested on the last exercise sheet becomes very slow very quickly, you will probably have to find another method. The MC's implementation can compute B_{6000} (whose numerator has over 15000 digits) in under 10 seconds. Can you beat him?

Note: Again, no cheating! The MC Sr will not look kindly upon attempts to use look-up tables for the first 10000 values or to query *WolframAlpha* via HTTP or anything like that.

Trivia

Did you know? In 1843, Augusta Ada King, Countess of Lovelace, wrote what is widely considered to be the first computer program ever – and its purpose was to compute Bernoulli numbers!

It was designed to run on Charles Babbage's *Analytical Engine*, a mechanical general-purpose computer far ahead of its time. Sadly, it was never actually built (partly because it was deemed too expensive and its value was not sufficiently recognised).

It then took more than 100 years until the first 'proper' general-purpose computer was realised: Konrad Zuse's Z3. There is a working replica in the *Deutsches Museum* here in Munich. The MC suggests you have a look at it some time!

Important: If you submit a competition exercise, you agree that we are allowed to publish your name as part of the competition on our website. If you just want to submit a competition exercise as part of your homework without taking part in the competition, you can just remove the `{-WETT-}` ... `{-TTEW-}` comments of your submission.