185.A05 Advanced Functional Programming SS 2020

Wednesday, 18 March 2020

Assignment 1

on Chapter 1 and Chapter 2

Topics: Streams, Generate/Prune Programming, Memoization

Submission deadline: Monday, 20 April 2020, 12am (at the earliest)
(In case of need, this deadline will be extended (details posted via TISS))

Regarding the deadline for the second submission: Please, refer to „Hinweise zu Organisation und Ablauf der Übung“ available at the homepage of the course.

Important:

1. Carefully read and follow the instructions outlined in the complementary files provided with assignment 1. If you have any questions regarding these instructions, ask your questions in the TISS forum. Following these instructions is paramount to ensure a smooth processing of your submitted file with the test system.

2. Store all functions to be written for this assignment in a top-level file named

   `Assignment1.hs`

   of your group directory. Comment your program meaningfully; use auxiliary functions and constants, where reasonable. The very same file name will be used for the second submission of assignment 1.

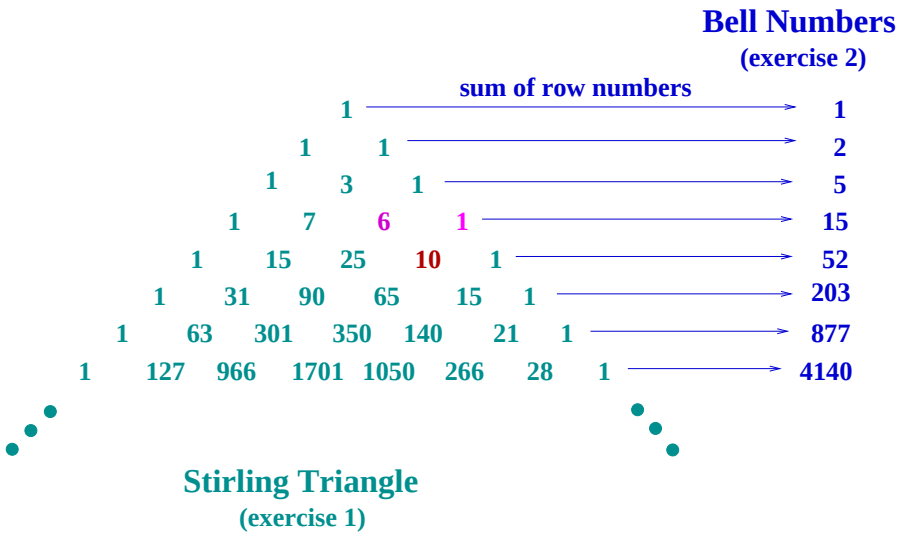
3. *Do not use self-defined modules!* If you want to re-use functions (written for other assignments), copy them to `Assignment1.hs`. Import declarations for self-defined modules will fail: Only `Assignment1.hs` will be copied for the (semi-automatical) evaluation procedure, no other ones.

Programming tasks:

1. The *Stirling numbers* $S(n,k)$ give the number of ways to partition a set S of n elements into k pairwise disjoint non-empty subsets, whose union equals S. E.g., for $S =_{df} \{a,b,c\}$ we get:

   $$\begin{array}{lcll} S(3,1) & = & 1 & // \ \{\,\{a,b,c\}\,\} \\ S(3,2) & = & 3 & // \ \{\,\{\{a,b\},\{c\}\},\{\{a\},\{b,c\}\},\{\{a,c\},\{b\}\}\,\} \\ S(3,3) & = & 1 & // \ \{\,\{a\},\{b\},\{c\}\,\} \end{array}$$

   Stirling numbers can nicely be ordered in triangle shape:



An entry of the Stirling triangle can be computed using the following (informally given) recurrence relation, which applies to rows with index 3 and beyond:

- A number of the triangle in row i, $i \geq 3$, evolves from the two numbers sitting immediately on top of it by adding to the left of these two numbers the product of the right of the two numbers and the column number for the new Stirling number to be computed.

For an example, consider number 10 located at row 5, column 4 of the prefix of the Stirling triangle shown above. On top of this entry in row 4 are sitting the numbers 6 (to the left) and 1 (to the right). The value of $6 + 1 * 4$ equals 10, the value of the entry in row 5, column 4.

Formally, the recurrence relation is given by:

$$S(n+1,k) = S(n,k-1) + k * S(n,k) \quad \text{for } n = 1,2,3,\ldots$$

Implement a corecursive 0-ary Haskell function `st :: [[Integer]]` yielding the stream of rows of the Stirling triangle, i.e., the stream of lists starting with:

`[[1],[1,1],[1,3,1],[1,7,6,1],...`

Examples:

```
take 5 st ->> [[1],[1,1],[1,3,1],[1,7,6,1],[1,15,25,10,1]]
head (drop 3 st) ->> [1,7,6,1]
[rs | rs <- st, mod (length rs) 2 /= 0] ->> [[1],[1,3,1],[1,15,25,10,1],...
```

2. Adding the numbers of the rows of the Stirling triangle, we get the *Bell numbers*. The Bell numbers give the number of ways to partition a set into pairwise disjoint non-empty subsets.

   Implement a corecursive 0-ary Haskell function `bn :: [Integer]` yielding the stream of Bell numbers by transforming the stream of rows of the Stirling triangle, i.e., the stream of numbers starting with:

   `[1,2,5,15,52,...`

   Examples:

```
take 5 bn ->> [1,2,5,15,52]
head (drop 3 bn) ->> 15
[n | n <- bn, mod (length n) 2 /= 0] ->> [5,15,203,877,...
```

   Without submission: Which instance of the generate/prune pattern did you make use of for implementing `bn`?

3. Let map g be defined by:

$$g(x,k) =_{df} x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots = \sum_{n=0}^{k} \frac{x^{2n+1}}{(2n+1)!}$$

   Implement two Haskell functions:

   3.1 `f :: Integer -> Integer -> Double`

   3.2 `f_mt :: Integer -> Integer -> Double`

   computing g with (function `f_mt`) and without memo-tables (function `f`), respectively. Both `f` und `f_mt` may utilize a function implementing map h defined by:

$$h(x,i) =_{df} \frac{x^i}{i!}$$

   which allows to express g as:

$$g(x,k) = \sum_{i=0}^{k} h(x, 2i+1)$$

   Note, `f_mt` may use one or more memo-tables, e.g., a memo-table for the values of argument x, a memo-table for the values of the factorial function, a memo-table storing the images of g for argument pairs x and k, etc.

   Without submission: Compare the run-times of `f` and `f_mt` for arguments of increasing size. Is there a performance difference? Is it a significant one?

4. The binomial coefficients ($0 \leq k \leq n$) are linked by the recurrence relation:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

which suggests the following implementation:

```
binom :: (Integer,Integer) -> Integer
binom (n,k)
  | k==0 || n==k = 1
  | True         = binom (n-1,k) + binom (n-1,k-1)
```

Implement two (more efficient) variants `binom_stpg` and `binom_memo` computing the binomial coefficients using stream programming combined with (a) Münchhausen principle (generalizing the idea underlying the Münchhausen style computation of the stream of Fibonacci numbers to a stream of streams of bionomial coefficients (cf. Chapter 2.3.2)), and (b) memoization (generalizing the idea of using a (1-dimensional) memo list storing computed Fibonacci numbers to a (2-dimensional) memo table storing computed bionomial coefficients (cf. Chapter 2.3.3)), respectively.

(a) Münchhausen style: `binom_stpg :: (Integer,Integer) -> Integer`

(b) memoization: `binom_memo :: (Integer,Integer) -> Integer`

Without submission: Compare pairwise the run-time performances of `binom`, `binom_stpg`, and `binom_memo` for argument pairs of increasing size. Are there performance differences? Are they significant?

Important:

- **Change password:** You should have received your login data for the computer `g0.complang.tuwien.ac.at` by 16 March 2020, which has been sent to your generic e-mail address `e<matrikelnummer>@student.tuwien.ac.at`. Please, log in as soon as possible into your account on `g0` (e.g., via `ssh`) and set your initial password to a new one of your own choice.
- **Submitting assignments:** Your programs will be (semi-automatically) evaluated on `g0` using the there installed GHC interpreter of GHC version 8.65. If you use a different tool or a different version of GHC for program development, please, double-check well in time before the submission deadline that your programs behave on `g0` and the there installed GHC interpreter version as you expect.

Iucundi acti labores.
Getane Arbeiten sind angenehm.

Cicero (106 - 43 v.Chr.)
röm. Staatsmann und Schriftsteller