School of Computing and Information Systems
COMP30026 Models of Computation Tutorial Week 2

31 July to 4 August 2017

Plan

Friday's lecture went over a non-trivial Haskell program. The aim was to show how useful Haskell can be at certain tasks, in particular tasks that involve symbolic manipulation. Many language details would have been unclear, but you are hopefully convinced that, with Haskell, it is possible to build useful tools with little effort. Now your job is to learn some Haskell details—although we won't need all Haskell features.

The main exercises this week are 1 and 2. If there is time to do more, that's great, but consider Exercise 5 optional; it is there for those who just can't stop.

However, first introduce yourself to your tutorial classmates and tell them a bit about yourself. Check out what your neighbours' favourite programming languages are. If they say Haskell, be extra friendly. You may also want to check who (if any) are currently at the same stage as you in the Haskell Grok modules (for Exercise 1).

The exercises

1. Get together with one or two other students, making sure one of you has a laptop. Visit Grok and work through modules "Functions and Recursion" and "Lists".

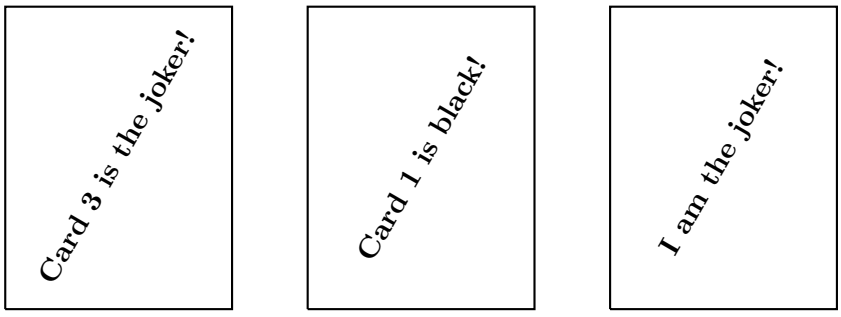
2. Still on Grok, work through the Week 2 worksheet.

3. Consider an n by m grid. We want to find out how many different paths there are that one can travel, starting from the bottom left and ending in the top right cell, given that *one can only move up or right*. Write a Haskell function

   ```
   paths :: Integer -> Integer -> Integer
   ```

   so that `paths m n` returns the number of such paths. (We use `Integer`s as the numbers can get quite large.)

4. Three playing cards lie face down on a table. One is red, one is black, and one is the joker. On the back of each card is written a sentence:

   Card 1 Card 2 Card 3

   The red card has a true sentence written on its back and the black card has a false sentence. Which card is red, which is black, and which is the joker?

5. Optionally, write a Haskell function `a2r` to translate a positive natural number to the corresponding roman numeral, using the symbols M, D, C, L, X, V, and I. The meaning of these: M=1000, D=500, C=100, L=50, X=10, V=5, and I=1. For example, `a2r 2009` should yield the string `"MMIX"`. If you find more time, write a function to translate the other way too. For example, `r2a "DXXVI"` should yield 526. If you have way too much time on your hand, add a function which checks that a string is a well-formed roman numeral.

   Why we still use Roman numerals is a mystery to me—they have no redeeming features. The value of a string of symbols is the sum of the symbols' values, except when a smaller value appears before a larger, the contribution of that pair is the difference between the larger and the smaller. The rules for well-formedness are rather cumbersome:

   (a) I can precede V and X, but not L, C, D, or M.

   (b) X can precede L or C, but not D or M.

   (c) C can precede D or M.

   (d) V, L, or D can't be followed by a numeral of greater or equal value.

   (e) A symbol with two consecutive occurrences can't be followed by a symbol of larger value.