The University of Melbourne
School of Computing and Information Systems
COMP30026 Models of Computation

Sample Answers to Tutorial Exercises, Week 2

3. We need to move from the lower left corner to the upper right. It does not really matter whether we are moving between grid points or we move from square to square—the two variants are isomorphic. Here is a recursive solution:

```
paths_rec :: Integer -> Integer -> Integer
paths_rec 1 _ = 1
paths_rec _ 1 = 1
paths_rec n m = (paths_rec (n-1) m) + (paths_rec n (m-1))
```

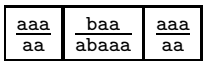
   This is quite nice but unfortunately runs in exponential time. We are better off analysing the problem first. Maybe we can find a closed form solution. Thinking ...

   The paths that we are enumerating all have $(m-1)+(n-1)$ steps, of which $m-1$ are horizontal steps and $n-1$ are vertical. So it is really a matter of deciding which of the $m+n-2$ steps are the horizontal ones. Think of the steps being numbered from 1 to $m+n-2$; we are asking how many ways there are to choose $m-1$ of those numbers. Well, there are K_{m-1}^{m+n-2} ways. We can easily write a function `k` for "n choose m", that is, K_m^n:

```
paths :: Integer -> Integer -> Integer
paths n m = k (n+m-2) (m-1)

k n m = numerator `div` denominator
  where
    numerator   = product [(n-m+1)..n]
    denominator = product [1..m]
```

4. (a) Here is a solution:



   (b) No solution is possible. Why?

   (c) No solution is possible. Why?

   Anyway, that was great: we decided each of the three problem instances. Here's something to ponder: What does a decision procedure look like, for this problem? More later ...

5. (a) `g x y (f x)`

   (b) `curry2 f x y = f (x,y)`; this says that `curry2 f` is a function which behaves like `f`, except it takes arguments `x` and `y` one at a time.

   (c) `uncurry2 f (x,y) = f x y`; this says that `uncurry2 f` is a function which behaves like `f`, except it receives `x` and `y` in one go, in the form of a pair.

   (d) There is a (logic-functional) programming language called Curry. There can't be many who have had *two* programming languages named after them.

6.
```
-- We use letters M, D, C, L, X, V, and I, with values
--
--   M = 1000
--   D =  500
--   C =  100
--   L =   50
--   X =   10
--   V =    5
--   I =    I
--
-- and five rules for well-formedness of Roman numerals
-- (plus some more obvious ones, not stated explicitly):
--
-- (1) I can precede V and X, but not L, C, D, or M.
-- (2) X can precede L or C, but not D or M.
-- (3) C can precede D or M.
-- (4) V, L, or D can not be followed by a numeral of
--     greater or equal value.
-- (5) A symbol with two consecutive occurrences can
--     not be followed by a symbol of larger value.
--
-- The function r2a both checks for well-formedness
-- and performs the conversion to arabic numbers.
-- We accept the empty string, translating it to 0.

r2a :: String -> Int
r2a ('M':s) = r2a s + 1000
r2a s       = get100s s

get100s :: String -> Int
get100s ('C':'M':s)         = get10s s + 900
get100s ('D':'C':'C':'C':s) = get10s s + 800
get100s ('D':'C':'C':s)     = get10s s + 700
get100s ('D':'C':s)         = get10s s + 600
get100s ('D':s)             = get10s s + 500
get100s ('C':'D':s)         = get10s s + 400
get100s ('C':'C':'C':s)     = get10s s + 300
get100s ('C':'C':s)         = get10s s + 200
get100s ('C':s)             = get10s s + 100
get100s s                   = get10s s

get10s :: String -> Int
get10s ('X':'C':s)          = get1s  s +  90
get10s ('L':'X':'X':'X':s)  = get1s  s +  80
get10s ('L':'X':'X':s)      = get1s  s +  70
get10s ('L':'X':s)          = get1s  s +  60
get10s ('L':s)              = get1s  s +  50
get10s ('X':'L':s)          = get1s  s +  40
get10s ('X':'X':'X':s)      = get1s  s +  30
get10s ('X':'X':s)          = get1s  s +  20
get10s ('X':s)              = get1s  s +  10
get10s s                    = get1s  s

get1s :: String -> Int
get1s "IX"   = 9
get1s "VIII" = 8
```

```
get1s "VII"  = 7
get1s "VI"   = 6
get1s "V"    = 5
get1s "IV"   = 4
get1s "III"  = 3
get1s "II"   = 2
get1s "I"    = 1
get1s ""     = 0
get1s s      = error ("Bad input, from: " ++ s)

-- Translate arabic numerals to roman numerals:

a2r :: Int -> String
a2r n
  = if n < 0
      then error "Input must be a natural number"
      else subtractivise (convert n)

convert :: Int -> String
convert n
  =    take m (repeat 'M')
    ++ take d (repeat 'D')
    ++ take c (repeat 'C')
    ++ take l (repeat 'L')
    ++ take x (repeat 'X')
    ++ take v (repeat 'V')
    ++ take i (repeat 'I')
    where
       last_3_digits = n `mod` 1000
       last_2_digits = n `mod`  100
       last_digit    = n `mod`   10
       m = n `div` 1000
       d = if last_3_digits < 500 then 0 else 1
       c = last_3_digits `div` 100 - 5*d
       l = if last_2_digits < 50 then 0 else 1
       x = last_2_digits `div` 10 - 5*l
       v = if last_digit < 5 then 0 else 1
       i = last_digit - 5*v

-- Use the subtractive convention of writing IV instead of IIII etc.

subtractivise :: String -> String
subtractivise ('D':'C':'C':'C':'C':s) = "CM" ++ subtractivise s
subtractivise ('C':'C':'C':'C':s)     = "CD" ++ subtractivise s
subtractivise ('L':'X':'X':'X':'X':s) = "XC" ++ subtractivise s
subtractivise ('X':'X':'X':'X':s)     = "XL" ++ subtractivise s
subtractivise ('V':'I':'I':'I':'I':s) = "IX" ++ subtractivise s
subtractivise ('I':'I':'I':'I':s)     = "IV" ++ subtractivise s
subtractivise (c:s)                   = c :    subtractivise s
subtractivise ""                      = ""

testQ6 :: Bool
testQ6
  = and [n == r2a (a2r n) | n <- [1..2000]]
```