Module Title: Inf1-FP
Exam Diet (Dec/April/Aug): Aug 2018
Brief notes on answers:

-- Informatics 1 Functional Programming
-- August 2018

module Aug2018 where

import Test.QuickCheck (quickCheck,
    Arbitrary (arbitrary ), Gen, suchThat,
    oneof, elements, sized, (==>) )
import Control.Monad -- defines liftM, liftM2, liftM3, used below
import Data.Char

-- Question 1

f :: [String] -> [String]
f [] = []
f ss = [last t : s | (_:s,t) <- zip ss (tail ss) ]

test1a =
    f ["pattern","matching","rules","ok"] == ["gattern","satiching","kules"]
    && f ["word"] == []
    && f ["almost","all","students","love","functional","programming"]
        == ["llmost","sl1","etudents","love","uunctional"]
    && f ["make","love","not","war"] == ["ake","ove","ort"]

f :: [String] -> [String]

f [] = []
f [s] = []
f ((_:s):t:ss) = (last t : s) : f (t:ss)

test1b =
    g ["pattern","matching","rules","ok"] == ["gattern","satiching","kules"]
    && g ["word"] == []
    && g ["almost","all","students","love","functional","programming"]
        == ["llmost","sl1","etudents","love","uunctional"]
    && g ["make","love","not","war"] == ["ake","ove","ort"]

prop1 ss = all (\s -> not (null s)) ss ==> f ss == g ss

-- Question 2

-- 2a

tla :: String -> Bool
tla [a,b,c] = isUpper a && isUpper b && isUpper c
tla_ = False

p :: [String] -> Int
p ss = length [ s | s <- ss, tla s ]

test2a =
p ["I","played","the","BBC","DVD","in","the","USA"] == 3
&& p ["The","DUP","MP","travelled","to","LHR"] == 2
&& p ["The","SNP","won","in","South","Morningside"] == 1
&& p [] == 0

-- 2b

q :: [String] -> Int
q [] = 0
q (s:ss) | tla s = 1 + q ss
    | otherwise = q ss

test2b =
q ["I","played","the","BBC","DVD","in","the","USA"] == 3
&& q ["The","DUP","MP","travelled","to","LHR"] == 2
&& q ["The","SNP","won","in","South","Morningside"] == 1
&& q [] == 0

-- 2c

r :: [String] -> Int
r ss = foldr (\_ -> \n -> n+1) 0 (filter tla ss)

test2c =
r ["I","played","the","BBC","DVD","in","the","USA"] == 3
&& r ["The","DUP","MP","travelled","to","LHR"] == 2
&& r ["The","SNP","won","in","South","Morningside"] == 1
&& r [] == 0

prop2 ss = p ss == q ss && q ss == r ss

-- Question 3

data Expr = X            -- variable X
    | Y            -- variable Y
    | Const Int    -- integer constant
    | Expr :+: Expr     -- addition
    | Expr :*: Expr     -- multiplication
    deriving (Eq, Ord)

-- turns an Expr into a string approximating mathematical notation
showExpr :: Expr -> String
showExpr X = "X"
showExpr Y = "Y"
showExpr (Const n) = show n
showExpr (p :+: q) = "(" ++ showExpr p ++ "+" ++ showExpr q ++ ")"
showExpr (p :*: q) = "(" ++ showExpr p ++ "*" ++ showExpr q ++ ")"

-- For QuickCheck
instance Show Expr where
    show = showExpr

instance Arbitrary Expr where
    arbitrary = sized expr
    where
        expr n | n <= 0 = oneof [ return X
                          , return Y
                          , liftM Const arbitrary ]
        | otherwise = oneof [ return X
                            , return Y
                            , liftM Const arbitrary
                            , liftM2 ( :+: ) subform2 subform2
                            , liftM2 ( :* : ) subform2 subform2

        where
            subform2 = expr (n `div` 2)

-- 3a

eval :: Expr -> Int -> Int -> Int
eval X i j = i
eval Y i j = j
eval (Const n) _ _ = n
eval (p :+: q) i j = eval p i j + eval q i j
eval (p :*: q) i j = eval p i j * eval q i j

test3a =
eval ((X :*: Const 3) :+: (Const 0 :*: Y)) 2 4 == 6
&& eval (X :*: (Const 3 :+: Y)) 2 4 == 14
&& eval (Y :+: (Const 1 :*: X)) 3 2 == 5
&& eval (((Const 1 :*: Const 1) :*: (X :+: Const 1)) :*: Y) 3 4 == 16

-- 3b

isSimple :: Expr -> Bool
isSimple X = True
isSimple Y = True
isSimple (Const _) = True
isSimple \( (p :+: q) \) = isSimple p && isSimple q
isSimple \( ((\text{Const 0}) :*: q) \) = False
isSimple \( ((\text{Const 1}) :*: q) \) = False
isSimple \( (p :*: (\text{Const 0})) \) = False
isSimple \( (p :*: (\text{Const 1})) \) = False
isSimple \( (p :*: q) \) = isSimple p && isSimple q

\[
\text{test3b} = \\
isSimple \((X :*: \text{Const 3}) :+: (\text{Const 0} :*: Y)\) == False \\
&& isSimple \((X :*: (\text{Const 3} :+: Y))\) == True \\
&& isSimple \((Y :+: (\text{Const 1} :*: X))\) == False \\
&& isSimple \(((\text{Const 1} :*: \text{Const 1}) :*: (X :+: \text{Const 1})) :*: Y)\) == False
\]

-- 3c

\[
simplify :: \text{Expr} \to \text{Expr} \\
simplify X = X \\
simplify Y = Y \\
simplify \((\text{Const } n)\) = \text{Const } n \\
simplify \((p :+: q)\) = simplify p :+: simplify q \\
simplify \((\text{Const 0} :*: q)\) = \text{Const 0} \\
simplify \((p :*: \text{Const 0})\) = \text{Const 0} \\
simplify \((p :*: \text{Const 1})\) = simplify p \\
simplify \((p :*: q)\) = simplify' (simplify p :*: simplify q) \\
\]

\[
\text{where} \\
simplify' \((\text{Const 0} :*: q)\) = simplify \((\text{Const 0} :*: q)\) \\
simplify' \((p :*: \text{Const 0})\) = simplify \((p :*: (\text{Const 0}))\) \\
simplify' \((\text{Const 1} :*: q)\) = simplify \((\text{Const 1} :*: q)\) \\
simplify' \((p :*: \text{Const 1})\) = simplify \((p :*: (\text{Const 1}))\) \\
simplify' \(p\) = p
\]

\[
\text{test3c} = \\
simplify \((X :*: \text{Const 3}) :+: (\text{Const 0} :*: Y)\) == (X :+: \text{Const 3}) :+: \text{Const 0} \\
&& simplify \((X :*: (\text{Const 3} :+: Y))\) == (X :+: (\text{Const 3} :+: Y)) \\
&& simplify \((Y :+: (\text{Const 1} :*: X))\) == Y :+: X \\
&& simplify \(((\text{Const 1} :*: \text{Const 1}) :*: (X :+: \text{Const 1})) :*: Y)\) == \\
(X :+: \text{Const 1}) :*: Y
\]

prop1_simplify :: \text{Expr} \to \text{Bool} \\
prop1_simplify p = isSimple \((\text{simplify } p)\)

prop2_simplify :: \text{Expr} \to \text{Int} \to \text{Int} \to \text{Bool} \\
prop2_simplify p i j = eval p i j == eval \((\text{simplify } p)\) i j