
```

import Test.QuickCheck
import Control.Monad -- defines liftM, liftM2, used below

-- 1a

f :: [Int] -> Int
f xs = maximum (0 : [ x | x <- xs, x > 0 ])

test1a =
  f [1,2,3,4,5]      == 5
  && f [-1,2,-3,4,-5] == 4
  && f [-1,-2,-3]     == 0
  && f [2,42,-7]      == 42

-- 1b

g :: [Int] -> Int
g [] = 0
g (x:xs) | x > 0 = x `max` g xs
         | otherwise = g xs

test1b =
  g [1,2,3,4,5]      == 5
  && g [-1,2,-3,4,-5] == 4
  && g [-1,-2,-3]     == 0
  && g [2,42,-7]      == 42

test1 = test1a && test1b
prop_1 xs = f xs == g xs
check1 = quickCheck prop_1

-- 2a

p :: [Int] -> Int
p xs | even (length xs)
    = sum [ xs!!(i+1) * xs!!i | i <- [0..length xs-1], even i ]

test2a =
  p [1,2,3,4]      == 14
  && p [3,5,7,5,-2,4] == 42
  && p []           == 0

-- 2b

q :: [Int] -> Int
q [] = 0
q (x:y:zs) = x*y + q zs

test2b =
  q [1,2,3,4]      == 14
  && q [3,5,7,5,-2,4] == 42
  && q []           == 0

-- 2c

r :: [Int] -> Int
r xs | even (length xs)
    = foldr (+) 0 (map (\i -> xs!!(i+1) * xs!!i) (filter even [0..length xs-1]))

test2c =
  r [1,2,3,4]      == 14

```

```

&& r [3,5,7,5,-2,4] == 42
&& r []              == 0

test2 = test2a && test2b && test2c
prop_2 xs = even (length xs) ==> p xs == q xs && q xs == r xs
check2 = quickCheck prop_2

-- 3

data Expr = Var String
          | Expr :+: Expr
          | Expr **: Expr
          deriving (Eq, Show)

-- code that enables QuickCheck to generate arbitrary values of type Expr

instance Arbitrary Expr where
  arbitrary = sized arb
    where
      arb 0 = liftM Var arbitrary
      arb n | n > 0 = oneof [liftM Var arbitrary,
                             liftM2 (:+:) sub sub,
                             liftM2 (**) sub sub]
    where
      sub = arb (n `div` 2)

-- 3a

isNorm :: Expr -> Bool
isNorm (a :+: b) = isNorm a && isNorm b
isNorm a         = isTerm a

isTerm :: Expr -> Bool
isTerm (Var x)   = True
isTerm (a :+: b) = False
isTerm (a **: b) = isTerm a && isTerm b

test3a =
  isTerm (Var "x") == True
  && isTerm ((Var "x" **: Var "y") **: Var "z") == True
  && isTerm ((Var "x" **: Var "y") :+: Var "z") == False
  && isTerm (Var "x" **: (Var "y" :+: Var "z")) == False
  && isNorm (Var "x") == True
  && isNorm (Var "x" **: Var "y" **: Var "z") == True
  && isNorm ((Var "x" **: Var "y") :+: Var "z") == True
  && isNorm (Var "x" **: (Var "y" :+: Var "z")) == False
  && isNorm ((Var "x" **: Var "y") :+: (Var "x" **: Var "z")) == True
  && isNorm ((Var "u" :+: Var "v") **: (Var "x" :+: Var "y")) == False
  && isNorm (((Var "u" **: Var "x") :+: (Var "u" **: Var "y")) :+:
            ((Var "v" **: Var "x") :+: (Var "v" **: Var "y"))) == True

-- 3b

norm :: Expr -> Expr
norm (Var v) = Var v
norm (a :+: b) = norm a :+: norm b
norm (a **: b) = norm a *** norm b
  where
    (a :+: b) *** c = (a *** c) :+: (b *** c)
    a *** (b :+: c) = (a *** b) :+: (a *** c)

```

```
a *** b          = a *: b

test3b =
  norm (Var "x")
  == (Var "x")
  && norm ((Var "x" *: Var "y") *: Var "z")
  == ((Var "x" *: Var "y") *: Var "z")
  && norm ((Var "x" *: Var "y") :+: Var "z")
  == ((Var "x" *: Var "y") :+: Var "z")
  && norm (Var "x" *: (Var "y" :+: Var "z"))
  == ((Var "x" *: Var "y") :+: (Var "x" *: Var "z"))
  && norm ((Var "u" :+: Var "v") *: (Var "x" :+: Var "y"))
  == (((Var "u" *: Var "x") :+: (Var "u" *: Var "y")) :+:
      ((Var "v" *: Var "x") :+: (Var "v" *: Var "y")))

test3 = test3a && test3b
prop_3 a = isNorm (norm a) && norm (norm a) == norm a
check3 = quickCheck prop_3

test = test1 && test2 && test3
check = check1 >> check2 >> check3
```