Informatics 1

Functional Programming Lecture 3

Lists and Comprehensions

Philip Wadler

University of Edinburgh

Required text and reading

Haskell: The Craft of Functional Programming (Third Edition),
Simon Thompson, Addison-Wesley, 2011.

or

Learn You a Haskell for Great Good!
Miran Lipovača, No Starch Press, 2011.

See the web page for weekly reading assignments.

Part I

Lists

The List

```
nums  :: [Int]
nums  =  [1,2,3]

chars  :: [Char]
chars  =  ['I','n','f','1']

-- or, equivalently
str  :: String
str  =  "Inf1"

numss  ::  [[Int]]
numss  =   [[1],[2,4,2],[],[3,5]]

funs  :: [Picture -> Picture]
funs  =  [invert,flipV]

oops  =  [1,"Inf1",[2,3]]  -- type error!

count  :: [Int]
count  =  [1..10]
```

Part II

List Comprehensions

List comprehensions — Generators

```
Prelude> [ x*x | x <- [1,2,3] ]
[1,4,9]

Prelude> [ toLower c | c <- "Hello, World!" ]
"hello, world!"

Prelude> [ (x, even x) | x <- [1,2,3] ]
[(1,False),(2,True),(3,False)]

Prelude> [ if even x then x else x+1 | x <- [4,5,6] ]
[4,6,6]
```

`x <- [1,2,3]` is called a *generator*

`<-` is pronounced *drawn from*

List comprehensions — Guards

```
Prelude> [ x | x <- [1,2,3], odd x ]
[1,3]

Prelude> [ x*x | x <- [1,2,3], odd x ]
[1,9]

Prelude> [ x | x <- [42,-5,24,0,-3], x > 0 ]
[42,24]

Prelude> [ toLower c | c <- "Hello, World!", isAlpha c ]
"helloworld"
```

`odd x` is called a *guard*

Sum, Product

```
Prelude> sum [1,2,3]
6

Prelude> sum []
0

Prelude> sum [ x*x | x <- [1,2,3], odd x ]
10

Prelude> product [1,2,3,4]
24

Prelude> product []
1

Prelude> let factorial n = product [1..n]
Prelude> factorial 4
24
```

Example uses of comprehensions

```
squares :: [Int] -> [Int]
squares xs  =  [ x*x | x <- xs ]

odds :: [Int] -> [Int]
odds xs  =  [ x | x <- xs, odd x ]

sumSqOdd :: [Int] -> Int
sumSqOdd xs  =  sum [ x*x | x <- xs, odd x ]
```

QuickCheck

```
-- sumSqOdd.hs

import Test.QuickCheck

squares :: [Int] -> [Int]
squares xs  =  [ x*x | x <- xs ]

odds :: [Int] -> [Int]
odds xs  =  [ x | x <- xs, odd x ]

sumSqOdd :: [Int] -> Int
sumSqOdd xs  =  sum [ x*x | x <- xs, odd x ]

prop_sumSqOdd :: [Int] -> Bool
prop_sumSqOdd xs  =  sum (squares (odds xs)) == sumSqOdd xs
```

Running QuickCheck

```
[melchior]dts: ghci sumSqOdd.hs
GHCi, version 7.6.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( sumSqOdd.hs, interpreted )
*Main> quickCheck prop_sumSqOdd
Loading package old-locale-1.0.0.0 ... linking ... done.
Loading package old-time-1.0.0.0 ... linking ... done.
Loading package random-1.0.0.0 ... linking ... done.
Loading package mtl-1.1.0.1 ... linking ... done.
Loading package QuickCheck-2.1 ... linking ... done.
+++ OK, passed 100 tests.
*Main>
```

Breaking up and putting together lists

```
Prelude> head [1,2,3]
1

Prelude> tail [1,2,3]
[2,3]

Prelude> 1 : [2,3]
[1,2,3]
```

QuickCheck

```
-- headTail.hs

import Test.QuickCheck

prop_headTail :: [Int] -> Bool
prop_headTail xs  =  (xs == []) or (head xs : tail xs == xs)
```