Informatics 1

Functional Programming Lecture 2

Lists and Comprehensions

Don Sannella

University of Edinburgh

Part I

Lists

The List

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

chars  :: [Char]
chars  =  [′I′,′n′,′f′,′1′,′A′]

-- or, equivalently
str  :: String
str  =  "Inf1A"

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

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

oops  =  [1,"Inf1A",[2,3]]  -- type error!

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

Putting together and taking apart lists

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

> [1,2] : 3  -- type error!
<interactive>:1:1: error:
     Non type-variable argument in the constraint: Num [[t]]
      (Use FlexibleContexts to permit this)
     When checking the inferred type
        it :: forall t. (Num [[t]], Num t) => [[t]]

head :: [a] -> a
head (x : xs) = x

> head [1,2,3]
1

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

Part II

List Comprehensions

List comprehensions — Generators

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

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

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

> [ 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

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

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

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

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

`odd x` is called a *guard*

Sum, Product

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

> sum []
0

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

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

> product []
1

factorial :: Int -> Int
factorial n = product [1..n]
> 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

```haskell
-- 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 8.0.2: http://www.haskell.org/ghc/ :? for help
> quickCheck prop_sumSqOdd
+++ OK, passed 100 tests.
```