Introduction to Functional Programming

Lists

Overview

1. Lists
   - Definitions of predefined functions
   - Fibonacci

init, last

`init` selects everything but the last element (compare with `last`!).

```
init :: [a] → [a]
init [x] = []
init [x : xs] = [x : init xs]

last :: [a] → [a]
last [x] = x
last [x : xs] = last xs
```

flatten

`flatten` join lists to form one single one.

```
flatten :: [[a]] → [a]
flatten [] = []
flatten [x : xs] = x ++ flatten xs

last :: [a] → [a]
last [x] = x
last [x : xs] = last xs
```

Comparing and ordering lists

Equality of lists (operators are also functions written between the arguments)

```
(==) infix 4 :: [a] [a] -> Bool | == a
(==) [] [] = True
(==) [] [y : ys] = False
(==) [x : xs] [] = False
(==) [x : xs] [y : ys] = x == y && xs == ys
```

Ordering lists

Lexicographical ordering (dictionary ordering)

E.g. `[2, 3] < [3, 0]` or `[10, 1] < [10, 2]`

```
(<) infix 4 :: [a] [a] → Bool | <, == a
(<) [] [] = False
(<) [] _ = True
(<) _ [] = False
(<) [x : xs] [y : ys] = x < y || ( x == y && xs < ys)
```

Other comparisons

Once we have $<$ and $==$ all others can be defined.

```
(≠)  x y = not ( x == y )
(>)  x y  = y < x
(>=) x y  = not (x < y)
(≤)  x y  = not (y < x)
```

Fibonacci

Compute the nth Fibonacci number

```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

// Start = fib 5

Fibonacci

Compute the nth Fibonacci number

```
fib1 :: Int → (Int, Int)
fib1 0 = (1,1)
fib1 1 = (1,1)
fib1 n = (b,a+b)
where
(a,b) = fib1 (n-1)
```

// Start = fib1 8

Fibonacci

Compute the nth Fibonacci number

```
fib2 :: Int Int Int → Int
fib2 a b 0 = a
fib2 a b c = fib2 b (a+b) (c-1)
```

// Start = fib2 1 1 10

Fibonacci

Compute the nth Fibonacci number

```
fib3 :: Int → Int
fib3 n = fibAux n 1 1

fibAux 0 a b           = a
fibAux i a b | i > 0 = fibAux (i-1) b (a+b)

// Start = fib3 8
```