Introduction to Functional Programming

Tuples and list comprehensions

Department of Programming Languages and Compilers
Faculty of Informatics
Eötvös Loránd University
Budapest, Hungary
zsv@elte.hu

Overview

1. Other types
   - Tuples
2. List comprehensions

Tuples

```
(1,'f')           :: (Int,Char)
("world",True,2)  :: (String,Bool,Int)
([1,2],sqrt)      :: ([Int],Real→Real)
(1,(2,3))         :: (Int,(Int,Int))
// any number 2-tuples pair, 3-tuples, no 1-tuple (8) is just integer

fst :: (a,b) → a
fst (x,y) = x
Start = fst (10, "world") // 10

snd :: (a,b) → b
snd (x,y) = y
Start = snd (1,(2,3))     // (2,3)

f :: (Int, Char) → Int
f (n, x) = n + toInt x
Start = f (1,'a') // 98
```

Tuples

```
splitAt :: Int [a] → ([a],[a])
splitAt n xs = (take n xs, drop n xs)

Start = splitAt 3 ['hello'] // (['h','e','l'],['l','o'])

search :: [(a,b)] a → b | == a
search [(x,y):ts] s
| x == s = y
| otherwise = search ts s

Start = search [(1,1), (2,4), (3,9)] 3 // 9
```

Zipping

```
zip :: [a] [b] → [(a,b)]
zip [] ys = []
zip xs [] = []
zip [x : xs] [y : ys] = [(x , y) : zip1 xs ys]

Start = zip [1,2,3] ['abc'] // [(1,'a'),(2,'b'),(3,'c')]
```

List comprehensions

```
Start :: [Int]
Start = [x * x \\ x ← [1..10]] // [1,4,9,16,25,36,49,64,81,100]

// expressions before double backslash
// generators after double backslash
// i.e. expressions of form x <- xs x ranges over values of xs
// for each value value the expression is computed

Start = map (λx = x * x) [1..10] // [1,4,9,16,25,36,49,64,81,100]

// constraints after generators
Start :: [Int]
Start = [x * x \\ x ← [1..10] | x rem 2 == 0]   // [4,16,36,64,100]
```

List comprehensions

```
// nested combination of generators
// , - every possible combination of the corresponding variables
// last variable changes faster
// for each x value y traverses the given list

Start :: [(Int,Int)]
Start = [(x,y) \\ x ← [1..2], y ← [4..6]]
          // [(1,4),(1,5),(1,6),(2,4),(2,5),(2,6)]

// parallel combination of generators

Start = [(x,y) \\ x ← [1..2] & y ← [4..6]]
          // [(1,4),(2,5)]

// multiple generators with constraints

Start = [(x,y) \\ x ← [1..5], y ← [1..x] | isEven x]
        // [(2,1),(2,2),(4,1),(4,2),(4,3),(4,4)]
```

List comprehensions - equivalences

```
map :: (a→b) [a] → [b]
map f l = [ f x \\ x ← l ]

filter :: (a→Bool) [a] → [a]
filter p l = [ x \\ x ← l | p x ]

zip :: [a] [b] → [(a,b)]
zip as bs = [(a , b) \\ a ← as & b ← bs]

Start = zip [1,2,3] [10, 20, 30] // [(1,10),(2,20),(3,30)]

// functions like sum, reverse, isMember, take
// are hard to write using list comprehensions
```

Quick sort

```
qsort :: [t] → [t] | Ord t
qsort [] = []
qsort [a : xs] = qsort [x \\ x ← xs | x <  a] ++ [a] ++
                 qsort [x \\ x ← xs | x >= a]

Start = qsort [2,1,5,3,6,9,0,1] // [0,1,1,2,3,5,6,9]
```

Warm up for practice

```
// 1. exists x xs checks whether x exists as an element in the list xs
// logical operator or is ||
exists :: Int [Int] → Bool
exists x []     = False
exists x [y:ys] = x == y || exists x ys

Start = exists 3 [1, 2, 1, 1, 2, 3, 2, 1, 3] // True

// 2. write the function duplicates which checks if there are duplicates
// in the list xs
duplicates :: [Int] → Bool
duplicates []     = False
duplicates [x:xs] = exists x xs || duplicates xs

Start = duplicates [1, 2, 1, 1, 2, 3, 2, 1, 3] // True
```

Warm up for practice

```
// 3. remove x xs removes x from the list xs
remove :: Int [Int] → [Int]
remove x []                  = []
remove x [y:ys]
| x == y     = remove x ys
| otherwise = [y : remove x ys]

Start = remove 3 [1, 2, 1, 1, 2, 3, 2, 1, 3] // [1,2,1,1,2,2,1]

// 4. removeDuplicates l returns the list l with all duplicate elements removed
removeDuplicates :: [Int] → [Int]
removeDuplicates []     = []
removeDuplicates [x:xs] = [x : removeDuplicates (remove x xs)]

Start = removeDuplicates [1, 2, 1, 2, 3, 1, 2, 4, 2, 3] // [1,2,3,4]
```