Tutorium Funktionale Programmierung 2019

Part 10 – Lists, List Comprehension, „filter“, „map“, „sortBy“ and „fold“

VO – Part 5

Benedikt Dornauer, 08.12.2019

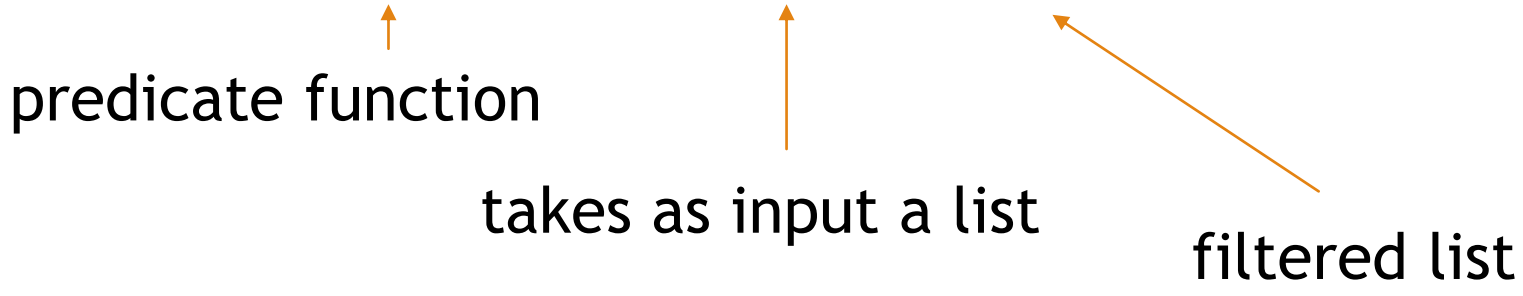
Exercise 10.1.: OVERVIEW

The following text should be converted:

encrypt “Hello World !!!!!!” → “KHOORZRUOG”

a. punctuation marks (‘!’, ‘.’, ‘,’, ‘[space character]’) should be ignored
b. convert all small letters to uppercase letters
c. cipher the text with the *Caesar*-cipher (function is given, but only works with uppercase letters)

filter :: (a -> Bool) -> [a] -> [a]




```
1.filter :: (a -> Bool) -> [a] -> [a]
2.filter _ [] = []
3.filter p (x:xs)
4.    | p x       = x : filter p xs
5.    | otherwise = filter p xs
```

Example:

- filter (>=2) [1,2,3] → [2,3]
- filter (==True) [True, False, True] → [True,True]

Exercise 10.1. a)

All punctuation marks (‘!’, ‘.’, ‘,’ and ‘[space character]’) should be ignored in a string.

Hint: You can use `notElem`

map :: (a -> b) -> [a] -> [b]

"Express a function *myMap* that takes a function *f* :: *a* -> *b* and a list of type [a]. The function f is applied to each element of the list." (Part 8)

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

Example:

- map (*2) [1,2,3] → [2,4,6]
- map (head) [[1,2], [2,3], [3,4]] → [1,2,3]

Exercise 10.1. b) and c)

b) Convert all small letters to uppercase letters

c) Cipher the text with the *Caesar*-Cipher (function is given)

```
encryptChar :: Char -> Char
encryptChar x =
  let a = fromEnum x
  in toEnum$ 65 + mod ((a-65) + 3)  26
```

sortBy: (a -> a -> Ordering) -> [a] -> [a]

sort: Ord a => [a] -> [a]

Example:

- sortBy compare [3,2,5,2,1]→ [1,2,2,3,5]
- sort [3,2,5,2,1] → [1,2,2,3,5]

import Data.List

List Comprehensions

- syntactic sugar
- similar to sets in mathematics

$$\{ 2^n \mid 1 \leq n \leq 10 \ \wedge n \ \neq 3\}$$

```
[2^n| n<- [0..10], n /= 3 ]
```

```
[1,2,4,16,32,64,128,256,512,1024]
```

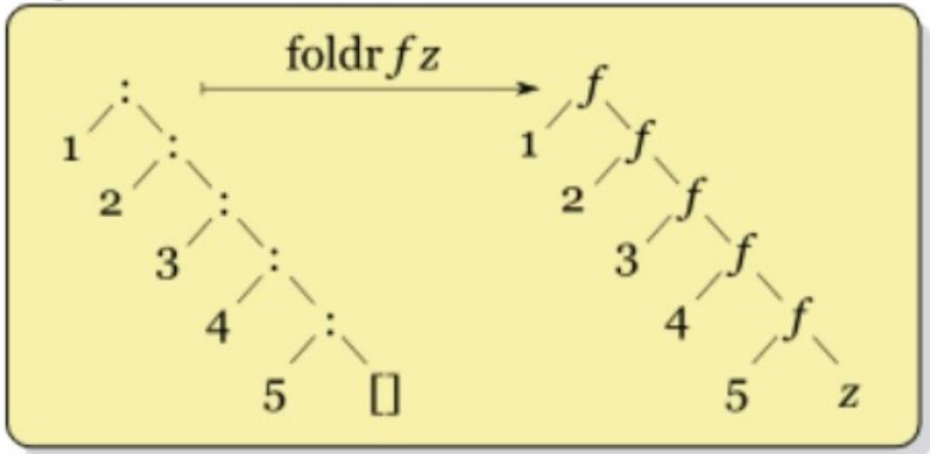
Exercise 10.2.: List Comprehension

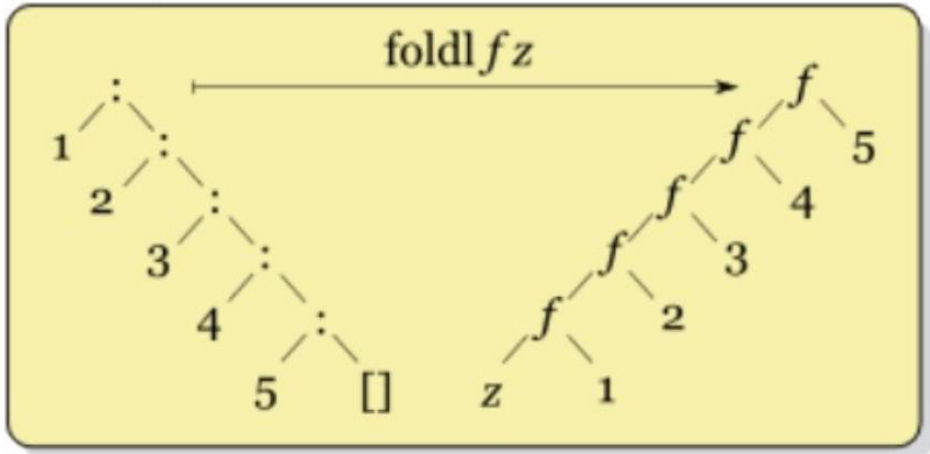
Sort the given list by the sum of the tuple (a,b) = a+b

- myList = [(Integer, Integer)]
- myList = [(x,y) | x<-[1..4], y<-[1..4]]

foldr

In graphical form, foldr and foldl can be represented as:





In other words - we replace the empty list ([]) by start element and list constructor (:) by our operation.

And this is the formal (source code) definition of foldr:

Definition

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr _ z []     = z
foldr f z (x:xs) = f x (foldr f z xs)
-- foldr op x0 (x:xs) = x 'op' (foldr op x0 xs)
```

Quelle: URL: Ihar I Hacharenka (2011). Slide 28
https://www.slideshare.net/goncharenko/functional-programming-by-ex

Exercise 10.3.: List Comprehension, foldr

```
ls :: [String]
ls = ["Franz", " Anton",  "Ingrid","Niklas",
 "Zoe",  "Dors", "Viktoria", "Ludwig",
 "Lutz", "Thomas", "Josef","Martina",
 "Gregor", "Emil", "Anna"]

myOrd :: [Int]
myOrd = [0, 13, 8, 2, 4, 1, 3, 14, 6, 2, 5, 14, 5]
```

1) Define a function that returns a list of words in a specific order both given as a list. Use list comprehension.
   e.g. *orderWords [0,3] ls → [“Franz”, “Niklas”]*

2) Define a function that only takes the first letter of each String in a list. Use *foldr*.
   e.g *extractLetters [“Franz”, “Niklas”] → “Fl”*

Questions? Need help? Feedback? etc.

- benedikt.dornauer@student.uibk.ac.at