Higher Order Functions and Algebraic Datatypes

dannymaate

13 July 2022

1 Higher Order Functions

Higher order functions capture common programming patterns as functions. In practice, they accept functions as arguments.

Map applies a function to all elements of a list, e.g. `map(2*) [1..10]`
Zip combines two lists into a single list of tuples, e.g. `zipWith(+) [1,2,3] [4,5,6,7,8]`
Filter selects all elements of a list that satisfy some predicate, e.g. `filter(> 5) [1..10]`

Folds

Many functions that accept a list are defined with the following pattern of recursion. Folds are left or right, this is an indicator of the associativity of the funtion being folded.
`f [] = v`
`f (x:xs) = x # f xs` *operator # is applied to the head and result of recursion on tail*

```
sum :: Num a => [a] -> a
sum = foldr (+) 0

product :: Num a => [a] -> a
product = foldr (*) 1

or :: [Bool] -> Bool
or = foldr (||) False

and :: [Bool] -> Bool
and = foldr (&&) True

foldr :: (a -> b -> b) -> b -> [a] -> b
```

The behaviour of fold can be summarised as follows
`foldr (#) v [x0, x1,...,xn] = x0 # (x1 # (... (xn # v) ...)).`
`foldl (#) v [x0, x1,...,xn] = (... ((v # x0) # x1) ...) # xn`

Composition Operator

```
(.) :: (b -> c) -> (a -> b) -> (a -> c)
f . g = \x -> f (g x)
```