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)
```

2 Algebraic Datatypes

To declare a new type introduce a new name for an existing type, e.g. `type Pos = (Int, Int)`.
The `data` mechanism is another way to declare a new type:
`data Bool = False | True` *type constructor* `Bool` *and data constructors* `False` *and* `True`

Stages of Execution

Compile-Time When a program is read into the REPL or *compile* type checking occurs.
Runtime Our program is *interpreted* into execution. *Data can be created* and expressions can be evaluated.

newtype If a new type has a single constructor and argument, then declare it with the `newtype` mechanism.
`newtype Nat = N Int` `N` *takes a single argument of type* `Int`

Differences with using `newtype` vs. `type` vs. `data`?
`Nat` and `Int` are different types and not synonymous. Using `newtype` over `data` brings an efficiency benefit that improves type safety