List and Folding Lists

Sanchayan Maity

Agenda

- Lists
- Folds
- **Disclaimer:** No original material in this presentation.

Lists recap

- Data type

```
data []  a   =   []  |   a : [a]
--  [1] [2] [3] [4] [5]   [6]
```

1. The datatype with the type constructor [],
2. which takes a single type constructor argument of type a,
3. at the term level can be constructed via
4. the nullary list constructor [],
5. or it can be constructed by
6. infix data constructor (or cons) :, which is a product of a value of type a from the type constructor and a value of type [a], that is, "more list."

Pattern matching

```haskell
ourTail :: [a] -> [a]
ourTail [] = []
ourTail (_ : xs) = xs
```

Syntactic sugar

```
ghci> [1, 2, 3] ++ [4]
[1, 2, 3, 4]
ghci> (1 : 2 : 3 : []) ++ 4 : []
[1,2,3,4]
```

Construction lists

```
ghci> [1..10]
[1,2,3,4,5,6,7,8,9,10]
ghci> enumFromTo 1 10
[1,2,3,4,5,6,7,8,9,10]
ghci> [1,2..10]
[1,2,3,4,5,6,7,8,9,10]
ghci> enumFromThenTo 1 2 10
[1,2,3,4,5,6,7,8,9,10]
ghci> [1,3..10]
[1,3,5,7,9]
ghci> enumFromThenTo 1 3 10
[1,3,5,7,9]
ghci> ['t'..'z']
"tuvwxyz"
ghci> enumFromTo 't' 'z'
"tuvwxyz"
```

Extracting from lists

```haskell
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
splitAt :: Int -> [a] -> ([a], [a])

takeWhile :: (a -> Bool) -> [a] -> [a]
dropWhile :: (a -> Bool) -> [a] -> [a]
```

List comprehensions

```
ghci> [x^y | x <- [1..5], y <- [2, 3]]
[1,1,4,8,9,27,16,64,25,125]
```

Evaluation

```
1 : (2 : [])
  :
  / \
  1
    :
    / \
    2 []
```

See `sprint` command.

```
ghci> blah = enumFromTo 'a' 'z'
ghci> :sprint blah
```

Spines are evaluated independently of values.

Miscellaneous

- `map`
- `filter`
- `zip`

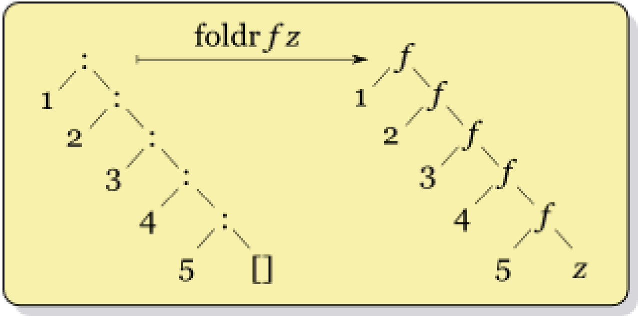
Patterns

```haskell
sum :: [Integer] -> Integer
sum [] = 0
sum (x:xs) = x + sum xs

length :: [a] -> Integer
length [] = 0
length (_:xs) = 1 + length xs

product :: [Integer] -> Integer
product [] = 1
product (x:xs) = x * product xs

concat :: [[a]] -> [a]
concat [] = []
concat (x:xs) = x ++ concat xs
```

Folds types

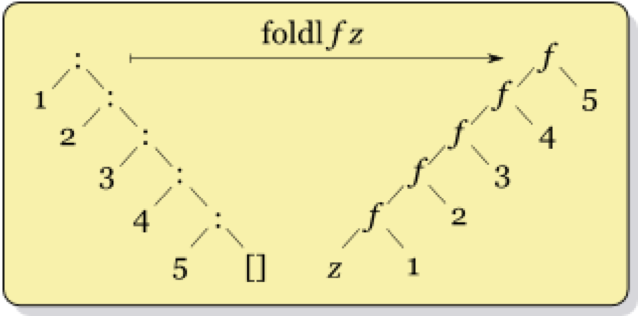
```haskell
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
foldr :: (a -> b -> b) -> b -> [] a -> b

foldl :: (b -> a -> b) -> b -> [a] -> b
foldl f acc [] = acc
foldl f acc (x:xs) = foldl f (f acc x) xs
```

Right fold transformation[1]



[1] Haskell Wiki - Fold

Left fold transformation[2]



[2] Haskell Wiki - Fold

Folds in-depth

- An aside from Alexis King.

https://github.com/hasura/graphql-engine/pull/2933#discussion_r328821960

Questions

- Reach out on
  - Email: sanchayan@sanchayanmaity.net
  - Mastodon: https://sanchayanmaity.com/@sanchayan
  - Blog: https://sanchayanmaity.net
  - Telegram:
    - `t.me/fpncr`
    - `t.me/SanchayanMaity`