Einführung in die funktionale Programmierung

Valentin Hauner – WS 2013/14

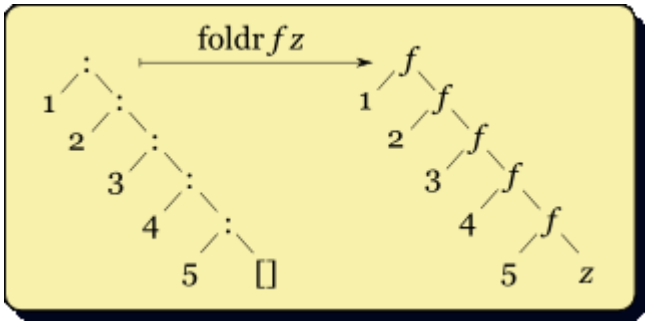
Reducing lists with foldr, foldl

foldr

foldr

- from the Prelude:

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



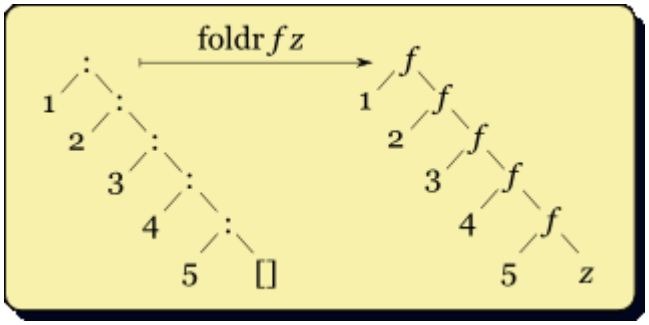
foldr

- from the Prelude:

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

- example:

```
foldr (+) 0 [1,2,3] =
```



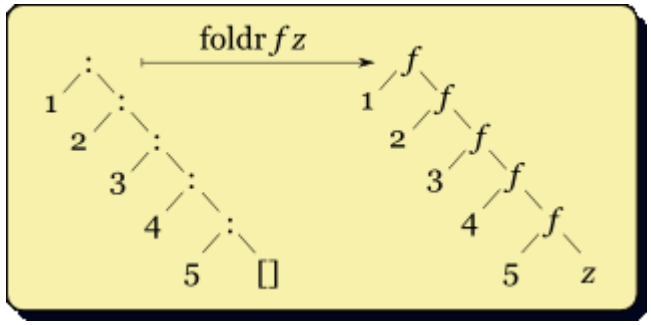
foldr

- from the Prelude:

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

- example:

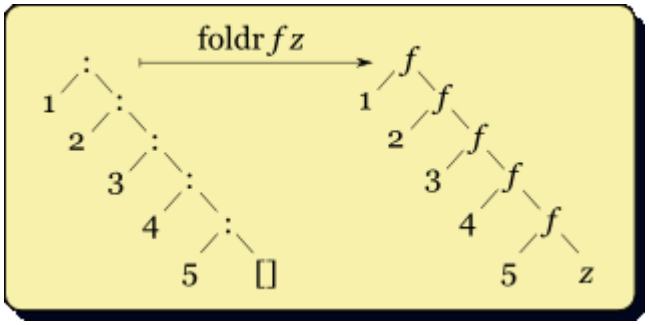
```
foldr (+) 0 [1,2,3] =
1 + (foldr (+) 0 [2,3]) =
```



foldr

- from the Prelude:

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



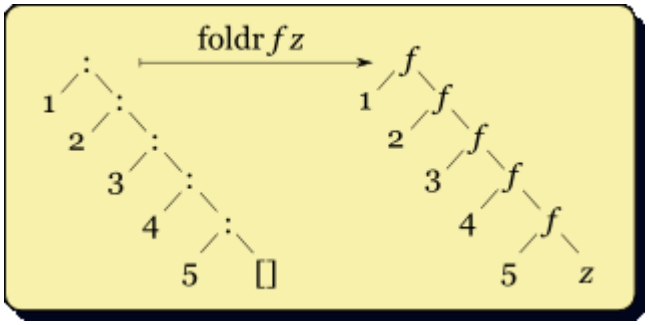
- example:

```
foldr (+) 0 [1,2,3] =
1 + (foldr (+) 0 [2,3]) =
1 + (2 + (foldr (+) 0 [3])) =
```

foldr

- from the Prelude:

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



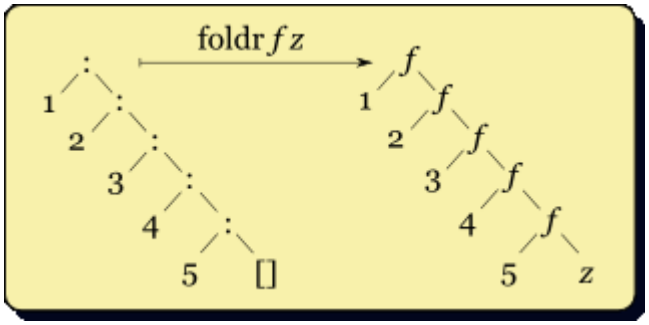
- example:

```
foldr (+) 0 [1,2,3] =
1 + (foldr (+) 0 [2,3]) =
1 + (2 + (foldr (+) 0 [3])) =
1 + (2 + (3 + (foldr (+) 0 []))) =
```

foldr

- from the Prelude:

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



- example:

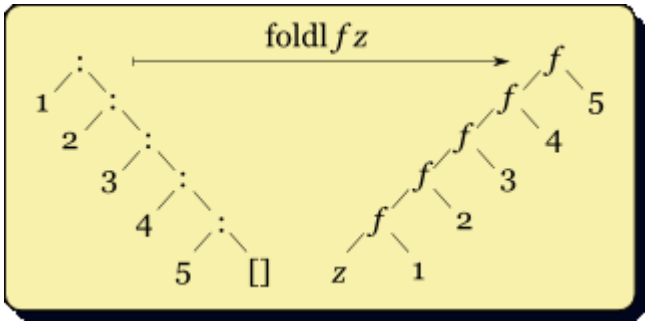
```
foldr (+) 0 [1,2,3] =
1 + (foldr (+) 0 [2,3]) =
1 + (2 + (foldr (+) 0 [3])) =
1 + (2 + (3 + (foldr (+) 0 []))) =
1 + (2 + (3 + 0)) =
6
```

foldl

foldl

- from the Prelude:

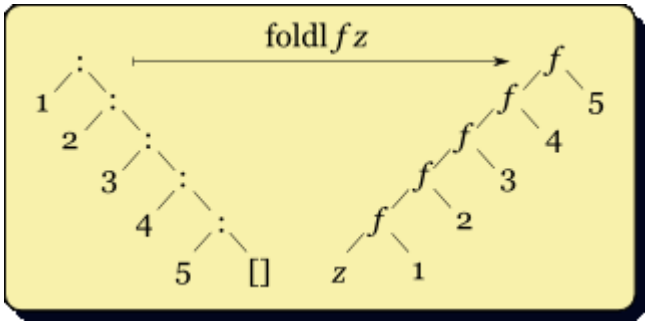
```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f z []     = z
foldl f z (x:xs) = foldl f (f z x) xs
```



foldl

- from the Prelude:

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



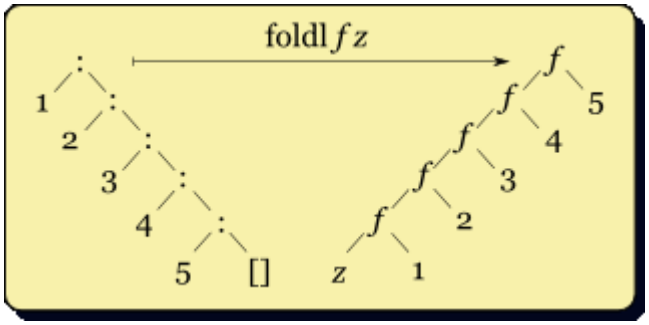
- example:

```
foldl (+) 0 [1,2,3] =
```

foldl

- from the Prelude:

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



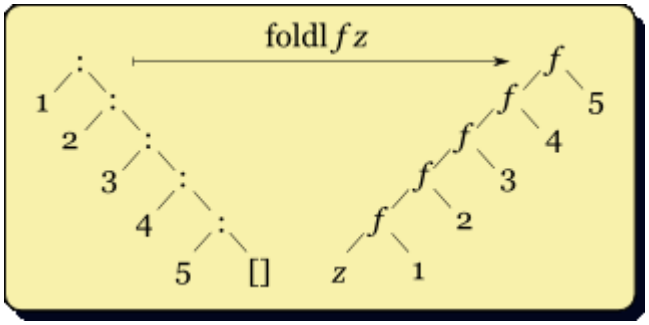
- example:

```
foldl (+) 0 [1,2,3] =
foldl (+) (0 + 1) [2,3] =
```

foldl

- from the Prelude:

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



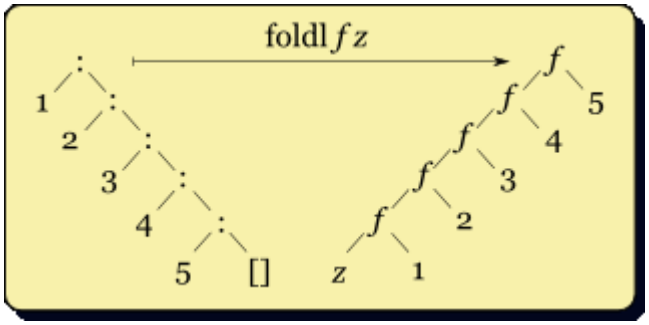
- example:

```
foldl (+) 0 [1,2,3] =
foldl (+) (0 + 1) [2,3] =
foldl (+) ((0 + 1) + 2) [3] =
```

foldl

- from the Prelude:

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



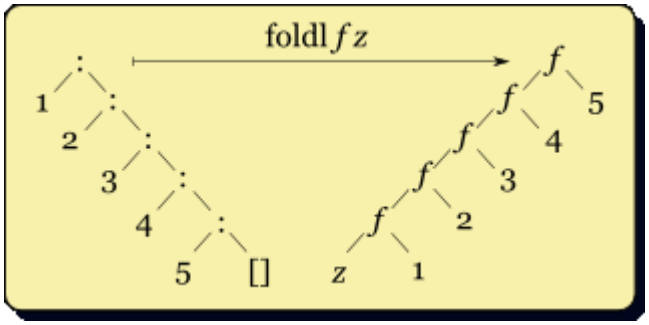
- example:

```
foldl (+) 0 [1,2,3] =
foldl (+) (0 + 1) [2,3] =
foldl (+) ((0 + 1) + 2) [3] =
foldl (+) (((0 + 1) + 2) + 3) [] =
```

foldl

- from the Prelude:

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



- example:

```
foldl (+) 0 [1,2,3] =
foldl (+) (0 + 1) [2,3] =
foldl (+) ((0 + 1) + 2) [3] =
foldl (+) (((0 + 1) + 2) + 3) [] =
(((0 + 1) + 2) + 3 =
6
```

foldr/foldl

```
foldr (+) 0 [1,2,3] =
1 + (foldr (+) 0 [2,3]) =
1 + (2 + (foldr (+) 0 [3])) =
1 + (2 + (3 + (foldr (+) 0 []))) =
1 + (2 + (3 + 0)) =
6
```

```
foldl (+) 0 [1,2,3] =
foldl (+) (0 + 1) [2,3] =
foldl (+) ((0 + 1) + 2) [3] =
foldl (+) (((0 + 1) + 2) + 3) [] =
(((0 + 1) + 2) + 3 =
6
```

=> foldr and foldl produce the same result if the operation used is associative

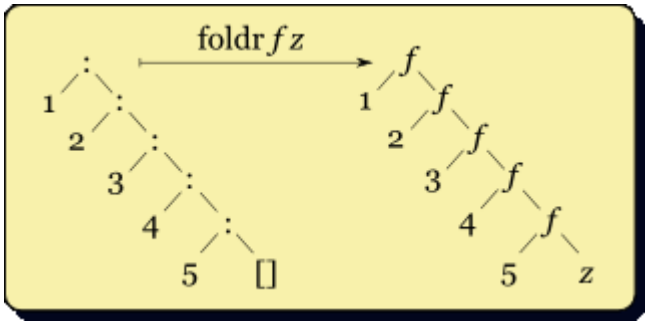
foldr

- from the Prelude:

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

- example:

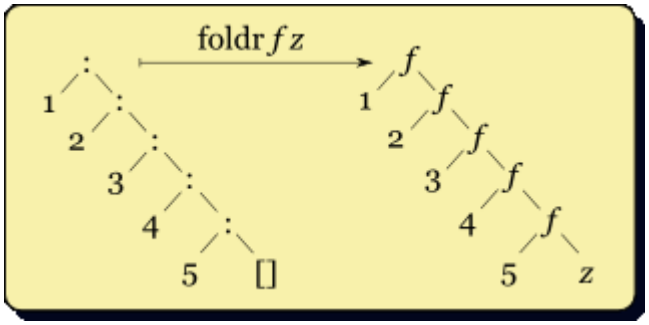
```
foldr (-) 0 [1,2,3] =
```



foldr

- from the Prelude:

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



- example:

```
foldr (-) 0 [1,2,3] =
1 - (foldr (-) 0 [2,3]) =
```

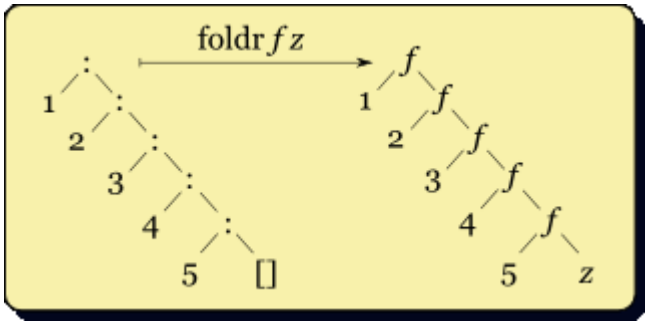
foldr

- from the Prelude:

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

- example:

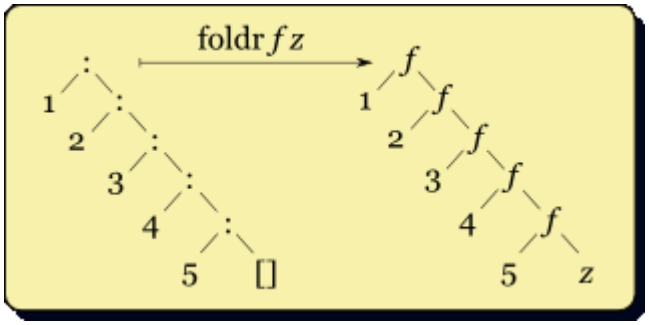
```
foldr (-) 0 [1,2,3] =
1 - (foldr (-) 0 [2,3]) =
1 - (2 - (foldr (-) 0 [3])) =
```



foldr

- from the Prelude:

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



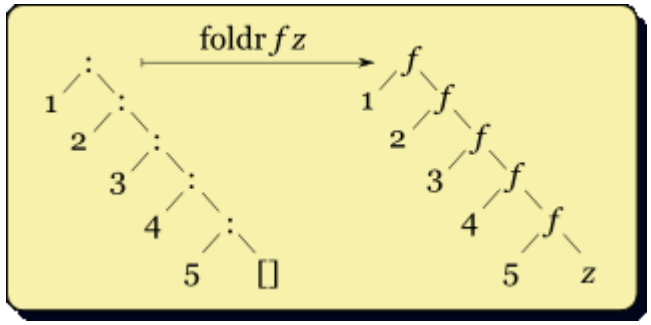
- example:

```
foldr (-) 0 [1,2,3] =
1 - (foldr (-) 0 [2,3]) =
1 - (2 - (foldr (-) 0 [3])) =
1 - (2 - (3 - (foldr (-) 0 []))) =
```

foldr

- from the Prelude:

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



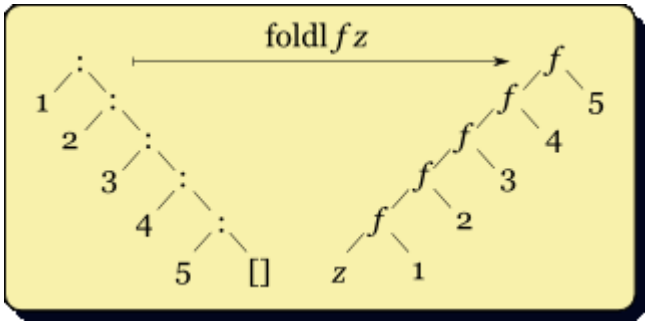
- example:

```
foldr (-) 0 [1,2,3] =
1 - (foldr (-) 0 [2,3]) =
1 - (2 - (foldr (-) 0 [3])) =
1 - (2 - (3 - (foldr (-) 0 []))) =
1 - (2 - (3 - 0)) =
2
```

foldl

- from the Prelude:

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



- example:

```
foldl (-) 0 [1,2,3] =
```

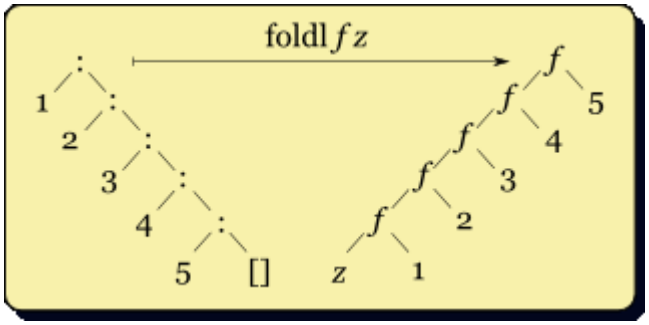
foldl

- from the Prelude:

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

- example:

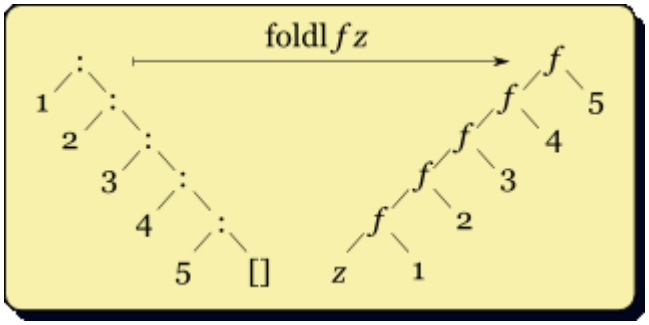
```
foldl (-) 0 [1,2,3] =
foldl (-) (0 - 1) [2,3] =
```



foldl

- from the Prelude:

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



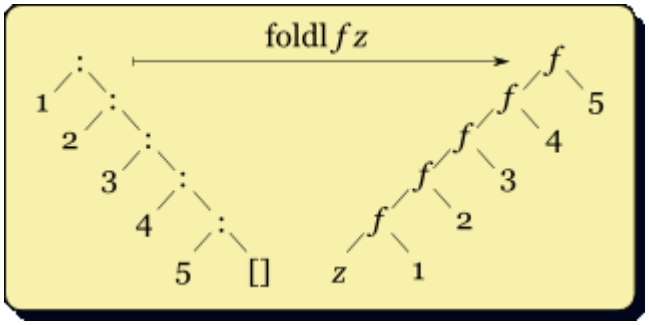
- example:

```
foldl (-) 0 [1,2,3] =
foldl (-) (0 - 1) [2,3] =
foldl (-) ((0 - 1) - 2) [3] =
```

foldl

- from the Prelude:

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



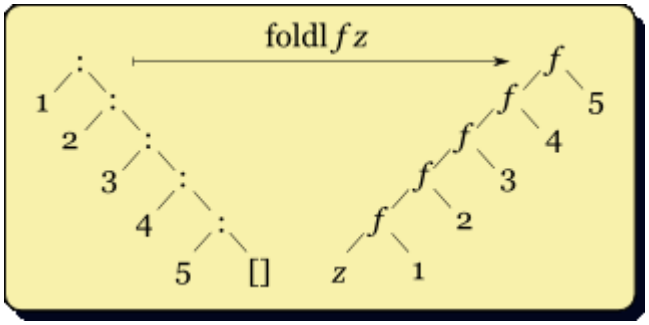
- example:

```
foldl (-) 0 [1,2,3] =
foldl (-) (0 - 1) [2,3] =
foldl (-) ((0 - 1) - 2) [3] =
foldl (-) (((0 - 1) - 2) - 3) [] =
```

foldl

- from the Prelude:

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



- example:

```
foldl (-) 0 [1,2,3] =
foldl (-) (0 - 1) [2,3] =
foldl (-) ((0 - 1) - 2) [3] =
foldl (-) (((0 - 1) - 2) - 3) [] =
(((0 - 1) - 2) - 3 =
-6
```

foldr/foldl

```
foldr (-) 0 [1,2,3] =
1 - (foldr (-) 0 [2,3]) =
1 - (2 - (foldr (-) 0 [3])) =
1 - (2 - (3 - (foldr (-) 0 []))) =
1 - (2 - (3 - 0)) =
2
```

```
foldl (-) 0 [1,2,3] =
foldl (-) (0 - 1) [2,3] =
foldl (-) ((0 - 1) - 2) [3] =
foldl (-) (((0 - 1) - 2) - 3) [] =
(((0 - 1) - 2) - 3 =
-6
```

=> foldr and foldl do not produce the same result if the operation used is not associative

foldr: cons

- from the Prelude:

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

- example:

```
foldr (:) [] [1,2,3] =
```

foldr: cons

- from the Prelude:

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

- example:

```
foldr (:) [] [1,2,3] =
1 : (foldr (:) [] [2,3]) =
```

foldr: cons

- from the Prelude:

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

- example:

```
foldr (:) [] [1,2,3] =
1 : (foldr (:) [] [2,3]) =
1 : (2 : (foldr (:) [] [3])) =
```

foldr: cons

- from the Prelude:

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

- example:

```
foldr (:) [] [1,2,3] =
1 : (foldr (:) [] [2,3]) =
1 : (2 : (foldr (:) [] [3])) =
1 : (2 : (3 : (foldr (:) [] []))) =
```

foldr: cons

- from the Prelude:

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

- example:

```
foldr (:) [] [1,2,3] =
1 : (foldr (:) [] [2,3]) =
1 : (2 : (foldr (:) [] [3])) =
1 : (2 : (3 : (foldr (:) [] []))) =
1 : (2 : (3 : [])) =
[1,2,3]
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3]
```

type error! why?

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
foldl (\xs x -> x:xs) (1:[]) [2,3] =
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
foldl (\xs x -> x:xs) (1:[]) [2,3] =
foldl (\xs x -> x:xs) (2:(1:[])) [3] =
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
foldl (\xs x -> x:xs) (1:[]) [2,3] =
foldl (\xs x -> x:xs) (2:(1:[])) [3] =
foldl (\xs x -> x:xs) (3:(2:(1:[]))) [] =
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
foldl (\xs x -> x:xs) (1:[]) [2,3] =
foldl (\xs x -> x:xs) (2:(1:[])) [3] =
foldl (\xs x -> x:xs) (3:(2:(1:[]))) [] =
(3:(2:(1:[]))) =
[3,2,1]
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
foldl (\xs x -> x:xs) (1:[]) [2,3] =
foldl (\xs x -> x:xs) (2:(1:[])) [3] =
foldl (\xs x -> x:xs) (3:(2:(1:[]))) [] =
(3:(2:(1:[]))) =
[3,2,1]

=> tail recursion, will be efficiently compiled as a loop
```

foldl: cons

- from the Prelude:

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

- example:

```
foldl (:) [] [1,2,3] = foldl (\a b -> a:b) [] [1,2,3]
type error! why?

foldl (\xs x -> x:xs) [] [1,2,3] =
foldl (\xs x -> x:xs) (1:[]) [2,3] =
foldl (\xs x -> x:xs) (2:(1:[])) [3] =
foldl (\xs x -> x:xs) (3:(2:(1:[]))) [] =
(3:(2:(1:[]))) =
[3,2,1]

=> tail recursion, will be efficiently compiled as a loop
=> does not work with infinite lists! why?
```

Infinite lists

- example:

```
> let xs = [1..]
> xs
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
 27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, . . .]
```

Infinite lists

- example:

```
> let xs = [1..]
> xs
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
 27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, . . .]
```

```
head ( foldr (:) [] xs ) =
```

Infinite lists

- example:

```
> let xs = [1..]
> xs
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
 27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, . . .]
```

```
head ( foldr (:) [] xs ) =
head ( 1 : (foldr (:) [] [2..]) ) =
```

Infinite lists

- example:

```
> let xs = [1..]
> xs
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
 27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, . . .]
```

```
head ( foldr (:) [] xs ) =
head ( 1 : (foldr (:) [] [2..]) ) =
1
```

Infinite lists

- example:

```
> let xs = [1..]
> xs
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
 27,28,29,30,31,32,33,34,35,36,37,38,39,40,41, . . .]
```

```
head ( foldr (:) [] xs ) =
head ( 1 : (foldr (:) [] [2..]) ) =
1
```

```
head ( foldl f [] xs )
=> counts to infinity!
```

References

[1] http://hackage.haskell.org/package/base-4.6.0.1/docs/src/GHC-Base.html#foldr
[2] http://hackage.haskell.org/package/base-4.6.0.1/docs/src/GHC-List.html#foldl
[3] http://www.haskell.org/haskellwiki/Fold

(all web resources requested on November 11th, 2013)