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

foldr _ start [] = start

foldr f start (x:xs) = f x $ foldr f start xs

1 + (2 + (3 + (4 + (5 + 0))))

O(n) paměti

f = (:)  s = []

[] 1:xs => 1 : [] : xs

product a = foldl' (*) 1 a

length = foldl' (\x y -> x+1) 0

deleteAll p y = foldr (\x xs -> if x == p then xs else x:xs) []

Stavy nechť jsou šipky

b = a -> c

zipWith g xs ys = ... ...
  where
    ...
    g (x:xs) f (y:ys) = (g x y) : (f xs ys)

maximum = foldl' max

scalarProd a b = foldr' (+) 0 $ zipWith (*) a b

f [] = []

f = map (+1)

f (x:xs) = (x+1) : (f xs)

foldr (:) [] == id

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

foldl _ acc [] = acc

foldl f acc (x:xs) = foldl (f acc x) xs

[1,2,3,4,5] f start = 0

(((((0 + 1) + 2) + 3) + 4) + 5)

(1 + 2)

(3 + 3)

(6 + 4)

(10 + 5)

O(1) paměti ... líné O(n)!

akumulátor

foldl' ... striktní

zip : b => [x] -> [y]  y :: (a, x)

f :: x -> ([x] -> [y] | -) ([x] -> [y])

foldr :: TS -> b -> [a] -> [x] -> [(a,x)]

take 5 [1,2,3,4,5,6]

k -> 1 : k-1 -> 2 : k-3 -> 3 : k-4 -> 4 ...

l@(x:xs)

scanl' (+) 0 ... prefixové součty

scanl (\x y -> x+k) 0 [a..]

prefix [k,k...]

fib = 0 : 1 : zipWith (+) (fib) (tail fib)