

# Haskell

*A Purely Functional Language*

featuring static typing, higher-order functions,  
polymorphism, type classes and monadic effects

## Funkcie a funkcionály

referečná transparentosť

Peter Borovanský

I-18

<http://dai.fmph.uniba.sk/courses/FPRO/>



# Zoznamová rekurzia

---

```
-- vyber prvých n prvkov zo zoznamu
take      :: Int -> [a] -> [a]
take 0 _  = []
take _ [] = []
take n (x:xs) = x : take (n-1) xs
```

```
-- dĺžka zoznamu
```

```
length      :: [a] -> Int
length []   = 0
length (x:xs) = 1 + length xs
```

Hypotéza (pre ľubovoľné n a xs):

- $\text{length (take } n \text{ xs)} = n$
- $\text{length } \$ \text{ take } n \text{ xs} = n$
- $(\text{length} . \text{take } n) \text{ xs} = n$

```
"?: " take 5 [1,3..100]
```

```
[1,3,5,7,9]
```

```
"?: " length (take 5 [1,3..100])
```

```
5
```

```
"?: " length $ take 5 [1,3..100]
```

```
5
```



# Dôkaz - $\text{length} (\text{take } n \text{ } xs) = n$

Indukcia (vzhľadom na dĺžku xs):

-  $xs = []$

$\text{length} (\text{take } n []) = 0$

$0 = 0$

*č.b.t.d.*

$\text{take} \quad \quad \quad :: \text{Int} \rightarrow [a] \rightarrow [a]$   
 $\text{take } 0 \_ = []$   
 $\text{take } \_ [] = []$   
 $\text{take } n (x:xs) = x : \text{take } (n-1) \text{ } xs$

-  $xs = (y:ys)$

$\text{length} (\text{take } n (y:ys)) = n$

$\text{length} (y:\text{take } (n-1) \text{ } ys) = n$

$1 + \text{length} (\text{take } (n-1) \text{ } ys) = n$

indukčný predpoklad,  $|ys| < |xs|$

$1 + (n-1) = n$

*č.b.t.d.*

$\text{length} \quad \quad \quad :: [a] \rightarrow \text{Int}$   
 $\text{length} [] = 0$   
 $\text{length } (x:xs) = 1 + \text{length } xs$



# QuickCheck

---

Elegantný nástroj na testovanie (!!! nie dôkaz !!!) hypotéz

```
"?: " import Test.QuickCheck
```

```
"?: " quickCheck (\(xs,n) -> length (take n xs) == n)
```

```
*** Failed! Falsifiable (after 2 tests and 1 shrink):
```

```
"?: " verboseCheck (\(xs,n) -> length (take n xs) == n)
```

```
Passed:
```

```
([],0)
```

```
Passed:
```

```
([()],1)
```

```
Failed:
```

```
([],-1)
```

```
*** Failed! Failed:
```

Neplatí to pre  $n$  záporne, lebo napr. `take (-3) [1..100] = []`,  
resp. naša definícia nepokrýva prípad  $n < 0$



# QuickCheck

Podmienka: miesto písania

if  $n \geq 0$  then length (take n s) == n else True

"?: " verboseCheck (\(xs,n) ->  $n \geq 0 \implies$  length (take n xs) == n)

Passed:

([],0)

Failed:

([()],2)

Neplatí to pre ak length xs < n ☹️

"?: " quickCheck (\(xs,n) ->  $n \geq 0 \ \&\& \text{length xs} \geq n \implies$

length (take n xs) == n)

\*\*\* Gave up! Passed only 35 tests.



Tvrdenie sme **overili** na niekoľkých prípadoch, ale to **nie je dôkaz**.

V dôkaze môžeme urobiť chybu (ako na slajde 2), QuickCheck slúži ako nástroj na hľadanie/odhaľovanie kontrapríkladov, kedy naše tvrdenie neplatí.

# Kvíz - platí/neplatí ?

(neseriózný prístup ale intuíciu treba tiež trénovať)

- `length [m..n] == n-m+1` 😞  
"?: " `quickCheck ((\ (n,m) -> length [m..n] == n-m+1))`  
\*\*\* Failed! Falsifiable (after 3 tests and 1 shrink):  
"?: " `quickCheck ((\ (n,m) -> m <= n ==> length [m..n] == n-m+1))` 😊  
+++ OK, passed 100 tests.
- `length (xs ++ ys) == length xs + length ys` 😊  
"?: " `quickCheck((\xs->\ys->(length (xs++ys)==length xs + length ys)))`  
+++ OK, passed 100 tests.
- `length (reverse xs) == length xs` 😊  
`quickCheck((\xs -> (length (reverse xs) == length xs )))`  
+++ OK, passed 100 tests.
- `(xs, ys) == unzip (zip xs ys)` 😞  
`quickCheck((\xs -> \ys -> ( (xs, ys) == unzip (zip xs ys) )))`  
\*\*\* Failed! Falsifiable (after 3 tests and 1 shrink):  
`quickCheck((\xs -> \ys -> ( length xs == length ys ==>`  
`(xs, ys) == unzip (zip xs ys) )))` 😊



# Funkcia/predikát argumentom

- zober zo zoznamu tie prvky, ktoré spĺňajú podmienku (test)  
Booleovská podmienka príde ako argument funkcie a má typ  $(a \rightarrow \text{Bool})$ :

`filter`  $:: (a \rightarrow \text{Bool}) \rightarrow [a] \rightarrow [a]$

`filter p xs`  $= [x \mid x \leftarrow xs, p\ x]$

**> filter even [1..10]  
[2,4,6,8,10]**

alternatívna definícia:

`filter p []`  $= []$

`filter p (x:xs)`  $= \text{if } p\ x \text{ then } x:(\text{filter } p\ xs) \text{ else } \text{filter } p\ xs$

vlastnosti (zväčša úplne zrejmé ?):

- `filter True xs`  $= xs$   $\dots [x \mid x \leftarrow xs, \text{True}] = [x \mid x \leftarrow xs] = xs$
- `filter False xs`  $= []$   $\dots [x \mid x \leftarrow xs, \text{False}] = []$
- `filter p1 (filter p2 xs)`  $= \text{filter } (p1 \ \&\& \ p2)\ xs$
- `(filter p1 xs) ++ (filter p2 xs)`  $= \text{filter } (p1 \ || \ p2)\ xs$

$$\begin{aligned}\text{filter } p \ [] &= [] \\ \text{filter } p \ (x:xs) &= \text{if } p \ x \text{ then } x:(\text{filter } p \ xs) \text{ else } \text{filter } p \ xs\end{aligned}$$

# Dôkaz

$\text{filter } p1 \ (\text{filter } p2 \ xs) = \text{filter } (p1 \ \&\& \ p2) \ xs$

Indukcia vzhľadom na parameter  $xs$

- $[]$   
L.S. =  $\text{filter } p1 \ (\text{filter } p2 \ []) = \text{filter } p1 \ [] = [] = \text{filter } (p1 \ \&\& \ p2) \ [] = \text{P.S.}$
- $(x:xs)$   
L.S. =  $\text{filter } p1 \ (\text{filter } p2 \ (x:xs)) = \dots \text{definícia}$   
 $\text{filter } p1 \ (\text{if } p2 \ x \text{ then } x:(\text{filter } p2 \ xs) \text{ else } \text{filter } p2 \ xs) = \dots \text{filter dnu cez if}$   
 $\text{if } p2 \ x \text{ then } \text{filter } p1 \ (x:(\text{filter } p2 \ xs)) \text{ else } \text{filter } p1 \ (\text{filter } p2 \ xs) = \dots \text{indukcia}$   
 $\text{if } p2 \ x \text{ then } \text{filter } p1 \ (x:(\text{filter } p2 \ xs)) \text{ else } \text{filter } (p1 \ \&\& \ p2) \ xs = \dots \text{definícia}$   
 $\text{if } p2 \ x \text{ then}$ 
  - $\text{if } p1 \ x \text{ then } x:(\text{filter } p1 \ (\text{filter } p2 \ xs)) \text{ else } \text{filter } p1 \ (\text{filter } p2 \ xs)$ $\text{else } \text{filter } (p1 \ \&\& \ p2) \ xs = \dots \text{2 x indukcia}$   
 $\text{if } p2 \ x \text{ then}$ 
  - $\text{if } p1 \ x \text{ then } x:(\text{filter } (p1 \ \&\& \ p2) \ xs) \text{ else } \text{filter } (p1 \ \&\& \ p2) \ xs$ $\text{else } \text{filter } (p1 \ \&\& \ p2) \ xs =$



filter p [] = []  
filter p (x:xs) = if p x then x:(filter p xs) else filter p xs

# Dôkaz

filter p1 (filter p2 xs) = filter (p1 && p2) xs

---

if p2 x then

if p1 x then x:(filter (p1 && p2) xs) else filter (p1 && p2) xs

else filter (p1 && p2) xs = ... **požívame vlastnosť** if-then-else

if A then

if A && B then C

if B then C

else D

else D

else D

if (p1 && p2) x then x:(filter (p1 && p2) xs) else filter (p1 && p2) xs = ... **def.**

filter (p1 && p2) (x:xs) = P.S.

*č.b.t.d.*



# QuickCheck a funkcie

Funkcie sú hodnoty ako každé iné  
Ako vie QuickCheck pracovať s funkciami ?

- je skladanie funkcií komutatívne ?

```
"?: " import Text.Show.Functions
```



```
"?: " quickCheck(
```

```
  (\x -> \f -> \g -> (f.g) x == (g.f) x)::Int->(Int->Int)->(Int->Int)->Bool)
```

```
*** Failed! Falsifiable (after 2 tests):
```

- je skladanie funkcií asociatívne ?

```
"?: " quickCheck(
```

```
  (\x -> \f -> \g -> \h -> (f.(g.h)) x == ((f.g).h) x)
  ::Int->(Int->Int)->(Int->Int)->(Int->Int)->Bool)
```



```
+++ OK, passed 100 tests.
```

Opäť to NIE je DÔKAZ, len 100 pokusov.

# QuickCheck a predikáty

Predikát je len funkcia s výsledným typom Bool

- `filter p1 (filter p2 xs) = filter (p1 && p2) xs` ☹️

```
?: " quickCheck ( \xs -> \p1 -> \p2 ->
    filter p1 (filter p2 xs) == filter (p1 && p2) xs)
    :: [Int] -> (Int->Bool) -> (Int->Bool) -> Bool)
```

<interactive>:113:91:

Couldn't match expected type 'Bool' --- NEPLATÍ LEBO ANI TYPY NESEDIA

- `filter p1 (filter p2 xs) = filter (\x-> p1 x && p2 x) xs` 😊

+++ OK, passed 100 tests.

Opäť to NIE je DÔKAZ (ten už bol), len 100 pokusov.

- `(filter p1 xs) ++ (filter p2 xs) = filter (\x -> p1 x || p2 x) xs`

```
"?: " quickCheck ( \xs -> \p1 -> \p2 ->
    (filter p1 xs) ++ (filter p2 xs) == filter (\x -> p1 x || p2 x) xs)
    :: [Int] -> (Int->Bool) -> (Int->Bool) -> Bool)
```

\*\*\* Failed! Falsifiable (after 3 tests):

[0] <function> <function>

# Funkcia argumentom

## map

- funktor, ktorý aplikuje funkciu (1.argument) na všetky prvky zoznamu

`map`  $:: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

`map f []`  $= []$

`map f (x:xs)`  $= f\ x : \text{map } f\ xs$

`map f xs`  $= [ f\ x \mid x \leftarrow xs ]$

- Príklady:

`map (+1) [1,2,3,4,5]`  $= [2,3,4,5,6]$

`map odd [1,2,3,4,5]`  $= [\text{True}, \text{False}, \text{True}, \text{False}, \text{True}]$

`and (map odd [1,2,3,4,5])`  $= \text{False}$

`map head [ [1,0,0], [2,1,0], [3,0,1] ]`  $= [1, 2, 3]$

`map tail [ [1,0,0], [2,1,0], [3,0,1] ]`  $= [ [0,0], [1,0], [0,1] ]$

`map (0:) [[1],[2],[3]]`  $= [[0,1],[0,2],[0,3]]$



# Vlastnosti map

- $\text{map id } xs = xs$  ☒  $\text{map id} = \text{id}$
- $\text{map } (f.g) \text{ } xs = \text{map } f (\text{map } g \text{ } xs)$  ☒  $\text{map } f . \text{map } g = \text{map } (f.g)$
- ~~$\text{head } (\text{map } f \text{ } xs) = f (\text{head } xs)$~~  ☒  ~~$\text{head} . \text{map } f = f . \text{head}$~~
- ~~$\text{tail } (\text{map } f \text{ } xs) = \text{map } f (\text{tail } xs)$~~  ☒  ~~$\text{tail} . \text{map } f = \text{map } f . \text{tail}$~~
- $\text{map } f (xs ++ ys) = \text{map } f \text{ } xs ++ \text{map } f \text{ } ys$  ☒
- $\text{length } (\text{map } f \text{ } xs) = \text{length } xs$  ☒  $\text{length} . \text{map } f = \text{length}$
- $\text{map } f (\text{reverse } xs) = \text{reverse } (\text{map } f \text{ } xs)$  ☒  $\text{map } f . \text{reverse} = \text{reverse} . \text{map } f$
- ~~$\text{sort } (\text{map } f \text{ } xs) = \text{map } f (\text{sort } xs)$~~  ☒  ~~$\text{sort} . \text{map } f = \text{map } f . \text{sort}$~~
- $\text{map } f (\text{concat } xss) = \text{concat } (\text{map } (\text{map } f) \text{ } xss)$  ☒

$\text{map } f . \text{concat} = \text{concat} . \text{map } (\text{map } f)$

$\text{concat} :: [[a]] \rightarrow [a]$

$\text{concat } [] = []$

$\text{concat } (xs:xss) = xs ++ \text{concat } xss$



$\text{concat } [[1], [2,3], [4,5,6], []] = [1,2,3,4,5,6]$



# Vlastnosti map, filter

---

Na zamyslenie:

- `filter p (map f xs)` = `??? (filter (p.f) xs)` 
- `filter p (map f xs)` = `map f (filter (p.f) xs)` 
- `filter p . map f` = `map f . filter (p.f)`

Dôkaz:

`filter p (map f xs)`  
= `filter p [ f x | x<-xs]`  
= `[y | y <- [ f x | x<-xs], p y]`  
= `[f x | x<-xs, p (f x)]`  
= `map f [x | x<-xs, p (f x)]`  
= `map f (filter (p.f))`



# Quíz - prémia

nájdite pravdivé a zdôvodnite

---

- $\text{map } f . \text{take } n = \text{take } n . \text{map } f$
- $\text{map } f . \text{filter } p = \text{map } \text{fst} . \text{filter } \text{snd} . \text{map } (\text{fork } (f,p))$   
where  $\text{fork} :: (a \rightarrow b, a \rightarrow c) \rightarrow a \rightarrow (b,c)$   
 $\text{fork } (f,g) x = (f x, g x)$
- $\text{filter } (p . g) = \text{map } (\text{inverzna\_g}) . \text{filter } p . \text{map } g$   
ak  $\text{inverzna\_g} . g = \text{id}$
- $\text{reverse} . \text{concat} = \text{concat} . \text{reverse} . \text{map } \text{reverse}$
- $\text{filter } p . \text{concat} = \text{concat} . \text{map } (\text{filter } p)$

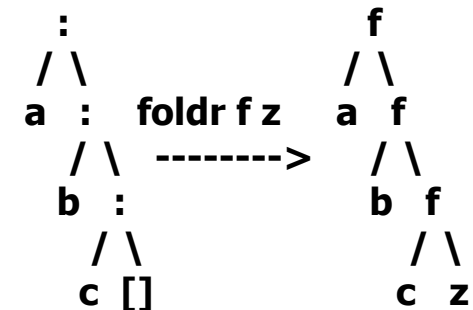
# Haskell – foldr

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

`foldr f z [] = z`

`foldr f z (x:xs) = f x (foldr f z xs)`

`a : b : c : [] -> f a (f b (f c z))`



```
Main> foldr (+) 0 [1..100]
```

```
5050
```

```
Main> foldr (\x y->10*y+x) 0 [1,2,3,4]
```

```
4321
```

-- g je vnorená lokálna funkcia

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

`foldr f z = g`

where `g [] = z`

`g (x:xs) = f x (g xs)`





# Haskell – foldl

---

`foldl`  $:: (a \rightarrow b \rightarrow a) \rightarrow a \rightarrow [b] \rightarrow a$

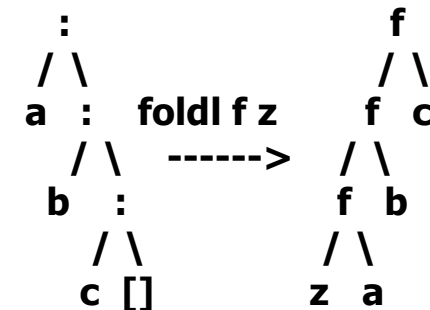
`foldl f z []`  $= z$

`foldl f z (x:xs)`  $= \text{foldl } f (f\ z\ x)\ xs$

`a : b : c : []`  $\rightarrow f (f (f\ z\ a)\ b)\ c$

```
Main> foldl (+) 0 [1..100]
5050
```

```
Main> foldl (\x y->10*x+y) 0 [1,2,3,4]
1234
```





# Vypočítajte

---

- `foldr max (-999) [1,2,3,4]`  
`foldl max (-999) [1,2,3,4]`
- `foldr (\_ -> \y ->(y+1)) 0 [3,2,1,2,4]`  
`foldl (\x -> \_ ->(x+1)) 0 [3,2,1,2,4]`

- `foldr (-) 0 [1..100] =`

$$(1-(2-(3-(4-\dots-(100-0)))))) = 1-2 + 3-4 + 5-6 + \dots + (99-100) = -50$$

- `foldl (-) 0 [1..100] =`

$$(\dots(((0-1)-2)-3) \dots - 100) = -5050$$

# Kvíz

`foldr (:) [] xs = xs`

`foldr (:) ys xs = xs++ys`

`foldr ? ? xs = reverse xs`

<http://foldl.com/>



Pre tých, čo zvládli kvíz, odmena !

kliknite si podľa vašej politickej orientácie

<http://foldr.com/>





# Funkcia je hodnotou

- $[a \rightarrow a]$  je zoznam funkcií typu  $a \rightarrow a$   
napríklad:  $[(+1), (+2), (*3)]$  je  $[\backslash x \rightarrow x+1, \backslash x \rightarrow x+2, \backslash x \rightarrow x*3]$

- čo je foldr (.) id  $[(+1), (+2), (*3)]$  ??

akého je typu

foldr (.) id  $[(+1), (+2), (*3)]$  100

foldl (.) id  $[(+1), (+2), (*3)]$  100

$[a \rightarrow a]$

303

???

lebo skladanie fcií je asociatívne:

- $((f \cdot g) \cdot h) x = (f \cdot g) (h x) = f (g (h x)) = f ((g \cdot h) x) = (f \cdot (g \cdot h)) x$
- funkcie nevieme porovnávať, napr.  $\text{head } [(+1), (+2), (*3)] = \text{id}$
- funkcie vieme permutovať,  $\text{length } \$ \text{permutations } [(+1), (+2), (*3), (^2)]$



# Maximálna permutácia funkcií

- zoznam funkcií aplikujeme na zoznam argumentov

```
apply      :: [a -> b] -> [a] -> [b]  
apply fs args = [ f a | f <- fs, a <- args]
```

```
apply [(+1),(+2),(*3)] [100, 200]  
[101,201,102,202,300,600]
```

- čo počíta tento výraz

```
maximum $  
  apply  
    (map (foldr (.) id) (permutations [(+1),(^2),(*3),(+2),(/3)]))  
    [100]
```

31827

- $(+1).(+2).(*3).(^2).(/3) 100$

3336.333333333334

- $((/3).(^2).(*3).(+2).(+1)) 100$

31827.0



# Zákon fúzie

---

Fussion Law:

Ak platí

$$f \ z1 = z2 \ \&\& \ f \ (g1 \ a \ b) = g2 \ a \ (f \ b)$$

potom platí

$$f \ . \ foldr \ g1 \ z1 = foldr \ g2 \ z2$$

Príklad použitia Fussion Law:

$$(n^*). \ foldr \ (+) \ 0 = foldr \ (+). (n^*) \ 0$$

Dôkaz (pomocou Fussion Law): overíme predpoklady  
čo je čo:

$$f = (n^*), \ z1 = z2 = 0, \ g1 = (+), \ g2 = (+). (n^*)$$

treba overiť:

- $(n^*) \ 0 = 0$
- $(n^*) \ (a+b) = (n^*a + n^*b) = (+). (n^*) \ a \ ((n^*) \ b)$

# Vlastnosti



Acid Rain (fold/build/deforestation theorem)

$$\text{foldr } f \ z \ . \ g \ (:) \ [] = g \ f \ z$$

Intuícia: Keď máme vytvoriť zoznam pomocou funkcie  $g$  zo zoznamových konštruktorov  $(:) []$ , na ktorý následne pustíme  $\text{foldr}$ , ktorý nahradí  $(:)$  za  $f$  a  $[]$  za  $z$ , namiesto toho môžeme konštruovať priamo výsledný zoznam pomocou  $g \ f \ z$ .

Otypujme si to:

Ak  $z :: u$ , potom  $f :: x \rightarrow u \rightarrow u$ ,  $\text{foldr } f \ z :: [x] \rightarrow u$

Ľavá strana:  $([x] \rightarrow u) \cdot (t \rightarrow [x])$  výsledkom je typ  $t \rightarrow u$

Pravá strana:  $g :: (x \rightarrow u \rightarrow u) \rightarrow u \rightarrow (t \rightarrow u)$



length . map \_ = length

foldr f z . g (:) [] = g f z

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

map h = foldr ((:) . h) []

-- (:) . h a as = (:) (h a as) = h a : as

= (\f z -> foldr (f . h) z) (:) []

length :: [a] -> Int

length = foldr (\ \_ -> \n -> n+1) 0

length . map h = length

(foldr (\ \_ -> \n -> n+1) 0) . (foldr ((:) . h) []) =

= podľa Acid Rain theorem (f = (\ \_ -> \n -> n+1), z = 0, ale čo je g ? ...

g x y = (foldr (x . h) y)

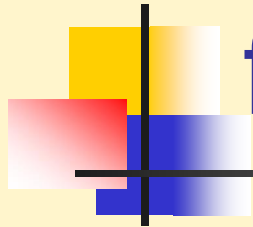
g f z = (foldr (f . h) z) = foldr ((\ \_ -> \n -> n+1) . h) 0 =

foldr ((\ \_ -> \n -> n+1) ) 0 = length

lebo

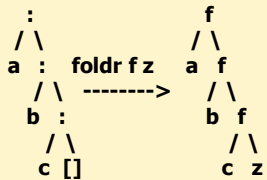
((\ \_ -> \n -> n+1) . h) x y = (\ \_ -> \n -> n+1) (h x) y = (\n -> n+1) y = y+1





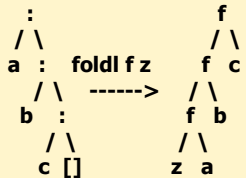
# foldr a foldl pre pokročilejších

definujte foldl pomocou foldr, alebo naopak:



$\text{myfoldl } f \ z \ xs = \text{foldr } (\backslash x \rightarrow \backslash y \rightarrow (f \ y \ x)) \ z \ (\text{myReverse } xs)$

$\text{myfoldr } f \ z \ xs = \text{foldl } (\backslash x \rightarrow \backslash y \rightarrow (f \ y \ x)) \ z \ (\text{myReverse } xs)$



- odstránime myReverse

$\text{myReverse } xs = \text{foldr } (\backslash x \rightarrow \backslash y \rightarrow (y ++ [x])) \ [] \ xs$

$\text{myfoldl}' f \ z \ xs = \text{foldr } (\backslash x \rightarrow \backslash y \rightarrow (f \ y \ x)) \ z$   
 $(\text{foldr } (\backslash x \rightarrow \backslash y \rightarrow (y ++ [x])) \ [] \ xs)$

- odstránime ++

$xs ++ ys = \text{foldr } (:) \ ys \ xs$

$\text{myfoldl}'' f \ z \ xs = \text{foldr } (\backslash x \rightarrow \backslash y \rightarrow (f \ y \ x)) \ z$   
 $(\text{foldr } (\backslash x \rightarrow \backslash y \rightarrow (\text{foldr } (:) \ [x] \ y)) \ [] \ xs)$

hmmm..., teoreticky (možno) zaujímavé, prakticky nepoužiteľné ...

# foldr a foldl posledný krát

Zamyslime sa, ako z foldr urobíme foldl:

induktívne predpokladajme, že rekurzívne volanie foldr nám vráti výsledok, t.j. hodnotu  $y$ , ktorá zodpovedá foldl:

- $y = \text{myfoldl } f \text{ } [b,c] = \lambda z \rightarrow f (f z b) c$

nech  $x$  je ďalší prvok zoznamu, t.j.

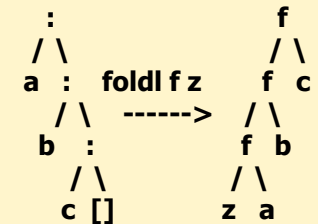
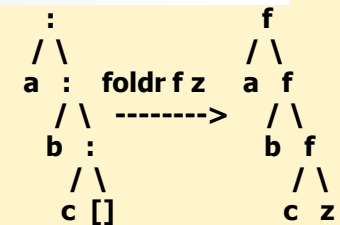
- $x = a$

ako musí vyzerat' funkcia  $?$ , ktorou fold-r-ujeme, aby sme dostali  $\text{myfoldl } f \text{ } [a,b,c] = \lambda z' \rightarrow f (f (f z' a) b) c = ? \ x \ y$

- $? = (\lambda x \ y \ z' \rightarrow y (f z' x))$

dosad'me:

- $(\lambda z' \rightarrow (\lambda z \rightarrow f (f z b) c) (f z' a)) =$
- $(\lambda z' \rightarrow f (f (f z' a) b) c) =$
- $\lambda z' \rightarrow f (f (f z' a) b) c$



# Pre tých, čo neveria, fakt posledný krát

$$? = (\lambda x y z' \rightarrow y (f z' x))$$

$$\blacksquare \text{ myfoldl'''' } f \text{ xs } z = \text{ foldr } (\lambda x y z \rightarrow y (f z x)) \text{ id } \text{ xs } z$$

- $\text{myfoldl'''' } f [] = \text{id}$
- $\text{myfoldl'''' } f [c] = (\lambda x y z \rightarrow y (f z x)) c \text{ id} = \lambda z \rightarrow f z c$
- $\text{myfoldl'''' } f [b,c] = (\lambda x y z \rightarrow y (f z x)) b (\lambda w \rightarrow f w c) =$   
 $\lambda z \rightarrow (\lambda w \rightarrow f w c) (f z b) =$   
 $\lambda z \rightarrow f (f z b) c$
- $\text{myfoldl'''' } f [a,b,c] = (\lambda x y z \rightarrow y (f z x)) a (\lambda w \rightarrow f (f w b) c) =$   
 $\lambda z \rightarrow (\lambda w \rightarrow f (f w b) c) (f z a) =$   
 $\lambda z \rightarrow f (f (f z a) b) c$
- $\text{myfoldl'''''' } f z \text{ xs} = \text{ foldr } (\lambda x y z \rightarrow y (f x z)) \text{ id } \text{ xs } z$

... doma skúste foldr pomocou foldl ...