Akumulační funkce nad vlastními datovými typy, opakování

IB015 Neimperativní programování

Tomáš Szaniszlo, Vladimír Štill, Martin Ukrop

Fakulta informatiky, Masarykova univerzita

Podzim 2017

Foldy na různých datových typech

Foldy na datových strukturách (odpovídá pojmu *katamorfismus*):

- „projdou“ rekurzivně celou danou strukturu
- nahradí všechny datové konstruktory zadanými funkcemi
  - arita musí souhlasit
  - typová struktura musí souhlasit
- výsledkem je nová struktura

Příklad ze standardní knihovny: `foldr`

- náhrada `(:)` za první argument `foldr`
- náhrada `[]` za druhý argument `foldr`

Pozor: `foldl` není katamorfismem (foldem) v tomhle smyslu!

Fold na seznamech (definice)

Datový typ seznam:

```
data [a] = (:) a [a] | []
```

Typy jeho datových konstruktorů:

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

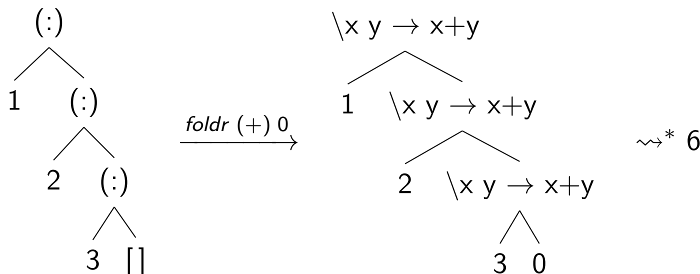
Fold na seznamech:

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

Fold na seznamech (příklad)

```
data [a] = (:) a [a] | []

foldr (+) 0 [1,2,3]
    ⇝ foldr (\x y -> x+y) 0 (1:(2:(3:[])))
   ⇝* 1+2+3+0 ⇝* 6
```



Rekurzivní funkce na binárních stromech

```haskell
data BinTree a = Node a (BinTree a) (BinTree a)
               | Empty
               deriving (Eq, Ord, Show)

treeSum :: Num a => BinTree a -> a
treeSum Empty = 0
treeSum (Node v l r) = (\x y z -> x + y + z) v
    (treeSum l) (treeSum r)
```

Fold na binárních stromech (definice)

Datový typ binární strom:

```
data BinTree a = Node a (BinTree a) (BinTree a)
               | Empty
```

Typy jeho datových konstruktorů:

```
Node  :: a -> BinTree a -> BinTree a -> BinTree a
Empty :: BinTree a
```

Fold na binárních stromech:

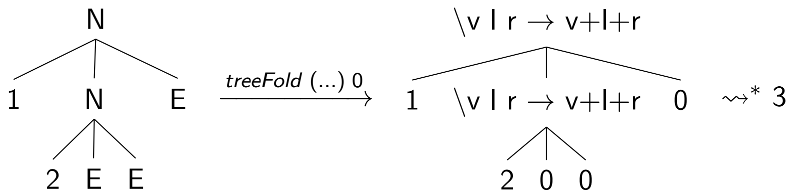
```
treeFold :: (a -> b -> b -> b) -> b -> BinTree a -> b
treeFold n e (Node v l r) = n v (treeFold n e l)
                                (treeFold n e r)
treeFold n e Empty        = e
```

Fold na binárních stromech (příklad)

```
data BinTree a = Node a (BinTree a) (BinTree a)
               | Empty
```

```
treeFold (\v l r -> v+l+r) 0 (N 1 (N 2 E E) E)
```

$\leadsto^*$ `1+(2+0+0)+0` $\leadsto^*$ `3`



Foldy na binárních stromech

Zhluboka se nadechněte, ve sbírce si otevřete příklad 8.1.2 (implementace funkcí na binárních stromech pomocí foldů) a řešte jednotlivé podpříklady.

Soubor s definicí datového typu `BinTree a`, funkce `treeFold` a ukázkovými stromy ze zadání najdete v ISu:

`https://is.muni.cz/auth/el/1433/podzim2016/IB015/um/seminars/code/08_treeFold.hs`

Opakování

IB015 Neimperativní programování

Tomáš Szaniszlo, Vladimír Štill, Martin Ukrop

Fakulta informatiky, Masarykova univerzita

Podzim 2017

Datové typy

Příklad 8.2.1: Otypujte následující výrazy:

a) `head [id, not]`

b) `\f -> f 42`

c) `\t x -> x + x > t x`

d) `\xs -> filter (> 2) xs`

e) `\f -> map f [1,2,3]`

f)
```
foo f True  = map f [1,2,3]
foo f False = filter f [1,2,3]
```

g) `\(p,q) z  ->  q (tail z) : p (head z)`

Části a, b, d, e případně f

Seznamy

Příklad 8.2.3: Definujte funkci
`minmax :: Ord a => [a] -> (a, a)`, která pro daný neprázdný seznam *v jednom průchodu* spočítá minimum i maximum.
Funkci definujte jednou rekurzivně a jednou pomocí `foldr` nebo `foldl` (ne `foldr1`, `foldl1`).
Dále implementujte funkci
`minmaxBounded :: (Ord a, Bounded a) => [a] -> (a, a)`, která funguje i na prázdných seznamech. Využijte konstant
`minBound :: Bounded a => a`,
`maxBound :: Bounded a => a`.
Najděte datový typ, pro který `minmaxBounded` nefunguje.
Poznámka: Může se stát, že interpretr bude zmatený z požadavku `Bounded a` a nebude schopen sám vyhodnotit výrazy jako `minmaxBounded [1,2,3]`. V takovém případě explicitně určete typ prvků seznamu, například:
`minmaxBounded [1,2,3::Int]`.

Příklad 8.2.4: Převeďte následující funkce do pointwise tvaru a přepište je s pomocí intensionálních seznamů a bez použití funkcí `map`, `filter`, `curry`, `uncurry`, `zip`, `zipWith`.

a) `map . uncurry`

b) `\f xs -> zipWith (curry f) xs xs`

c) ``map (* 2) . filter odd . map (* 3) . map (`div` 2)``

d) `map (\f -> f 5) . map (+)`

Různé funkce

Příklad 8.2.13: Co dělají následující funkce?

a) `f1 = flip id 0`

b) `f2 = flip (:) []`

c) `f3 = zipWith const`

d) `f4 p = if p then ('/':) else id`

e) `f5 = foldr id 0`

f) `f6 = foldr (const not) True`

Příklad 8.2.2: Uvažte následující datový typ:

```
data Foo a = Bar [a]
           | Baz a
```

Určete typy následujících výrazů:

a)
```
getlist (Bar xs) = xs
getlist (Baz x ) = x : []
```

b)
```
\foo -> foldr (+) 0 (getlist foo)
```

Vstup a výstup

Příklad 8.2.5: Otypujte následující `IO` výrazy a převeďte je do do-notace: Uvažujte při tom následující typy `(>>=)`, `(>>)`, `return`:

```
(>>=)  :: IO a -> (a -> IO b) -> IO b
(>>)   :: IO a -> IO b -> IO b
return :: a -> IO a
```

a) `readFile "/etc/passwd" >> putStrLn "bla"`
b) `\f -> putStrLn "bla" >>= f`
c) `getLine >>= \x -> return (read x)`

d)
```
foo :: Integer
foo = getLine >>= \x -> read x
```