N-ární stromy a akumulační funkce nad vlastními datovými typy

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Opakování

Naprogramujte následující funkce z minulého cvičení pomocí funkce `foldr` na seznamech.

- `idFold :: [a] -> [a]`, který projde vstupní seznam a vrátí stejný seznam (použijte funkci `fold`)

  `idFold [5,3,2,1,4,1]` $\rightsquigarrow^*$ `[5,3,2,1,4,1]`

- `evenFold :: [Int] -> Int`, který projde vstupní seznam a vrátí součin všech sudých čísel v seznamu.

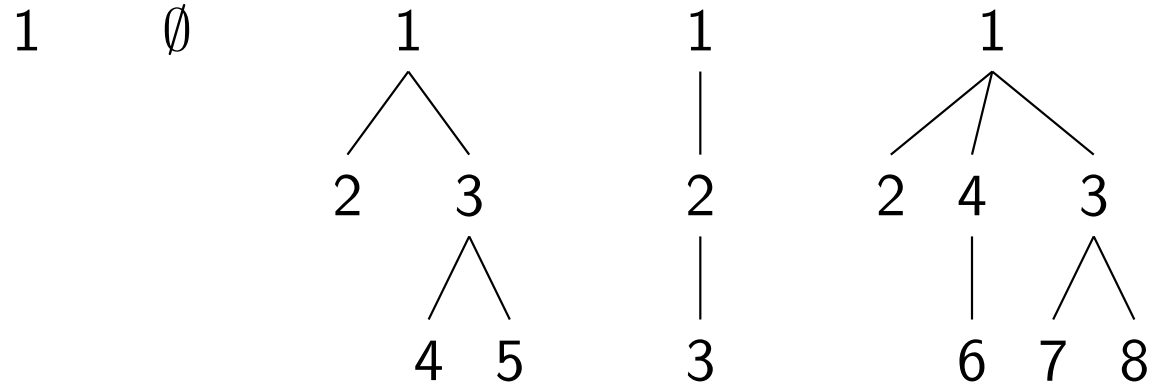
  `evenFold [1,3,4,5,6]` $\rightsquigarrow^*$ `24`

N-ární stromy

Uvažme následující definici *n*-árních stromů:

```
data NTree a = NNode a [NTree a]
```

- identifikujte v definici hodnotový a typový konstruktor a určete jejich aritu
- zkuste pomocí tohoto datového typu zapsat stromy zobrazené níže



N-ární stromy

Příklad 7.1.1: Uvažte typ n-árních stromů definovaný následovně:

```
data NTree a = NNode a [NTree a]
             deriving (Show, Read)
```

Definujte následující:

a) funkci `ntreeSize :: NTree a -> Integer`, která spočítá počet uzlů ve stromě

b) funkci `ntreeSum :: Num a => NTree a -> a`, která sečte ohodnocení všech uzlů stromu

c) funkci `ntreeMap :: (a -> b) -> NTree a -> NTree b`, která bere funkci a strom, a aplikuje danou funkci na hodnotu v každém uzlu:

```
ntreeMap (+1) (NNode 0 [NNode 1 [], NNode 41 []])
    ⇝* NNode 1 [NNode 2 [], NNode 42 []]
```

Foldy na různých datových typech

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

- „projdou“ rekurzivně celou danou strukturu
- nahradí všechny hodnotové 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 tomto smyslu!

Fold na seznamech (definice)

Datový typ seznam:

```
data MyList a = ListElement a (MyList a) | Empty
```

Typy jeho hodnotových konstruktorů:

```
ListElement :: a -> MyList a -> MyList a
Empty  :: MyList a
```

Fold na seznamech:

```
foldr :: (a -> b -> b) -> b -> MyList a -> b
foldr f z (ListElement x xs) = f x (foldr f z xs)
foldr f z Empty              = z
```

Fold na seznamech (příklad)

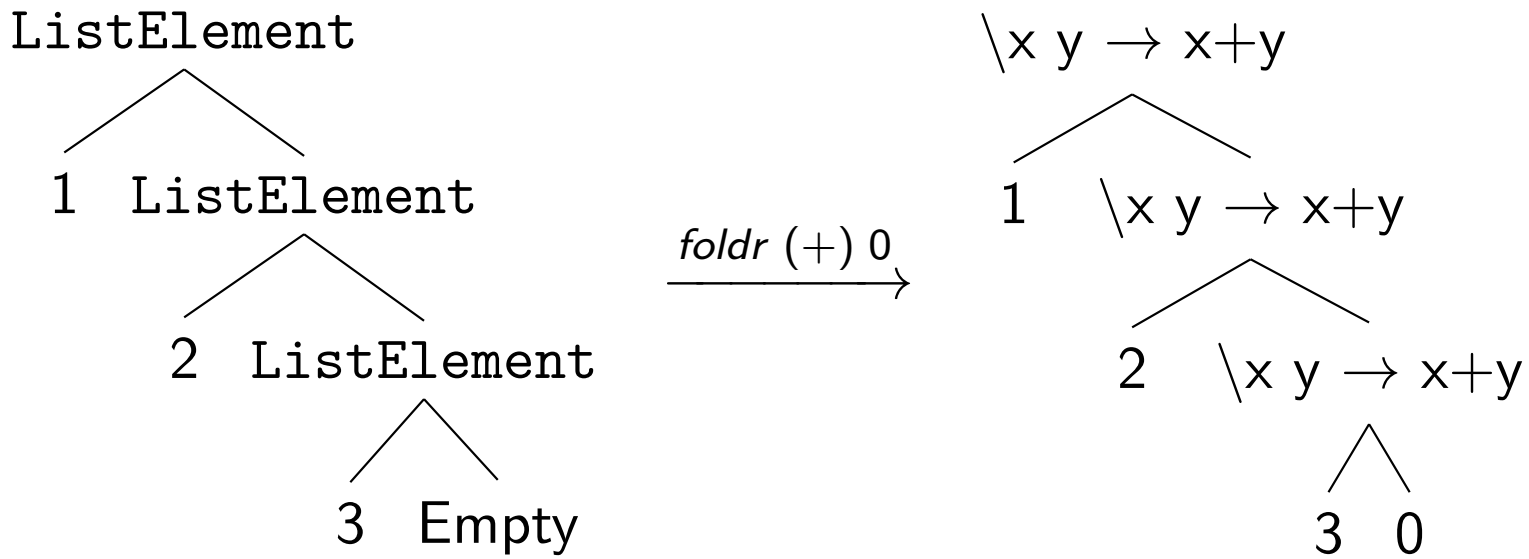
```
data MyList a = ListElement a (MyList a) | Empty

foldr (+) 0 (ListElement 1 (ListElement 2
                            (ListElement 3 Empty)))

    ~ foldr (\x y -> x+y) 0 (ListElement 1
              (ListElement 2 (ListElement 3 Empty)))

                ⇝* 1 + (2 + (3 + (0))) ⇝* 6
```

```
foldr (+) 0 (ListElement 1 (ListElement 2
                                (ListElement 3 Empty)))
```



$\rightsquigarrow^* 6$

Fold na binárních stromech (definice)

Datový typ binární strom:

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

Typy jeho hodnotových konstruktorů:

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

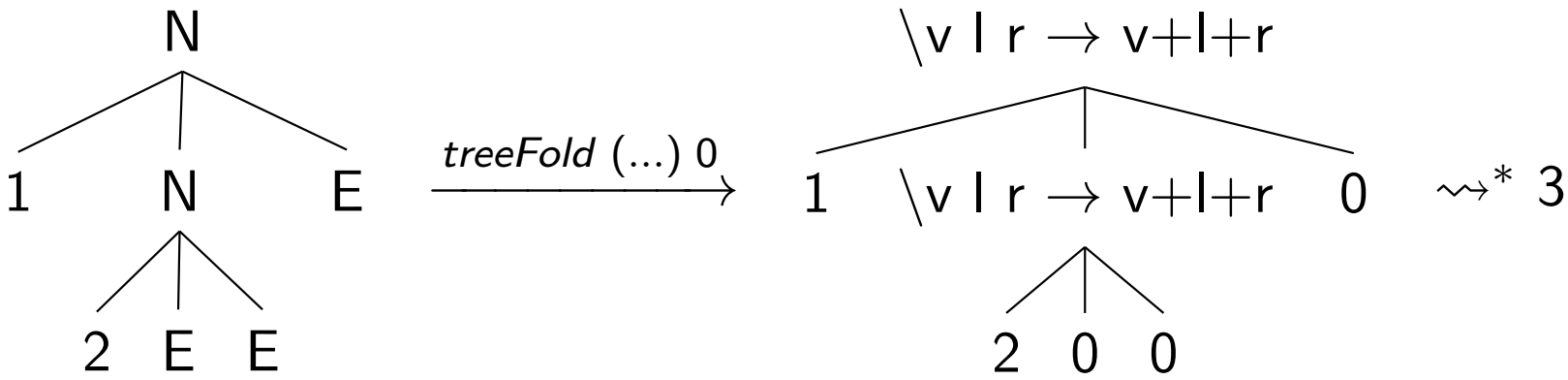
Fold na binárních stromech:

```
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)
    ⇝* 1+(2+0+0)+0 ⇝* 3
```



Foldy na binárních stromech

Zhluboka se nadechněte, ve sbírce si otevřete příklad 7.2.2 (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/podzim2018/IB015/um/seminars/code/07_treeFold.hs`