Programare declarativă

Introducere în programarea funcțională folosind Haskell

Ioana Leuștean
Traian Șerbănuță

Departamentul de Informatică, FMI, UB

ADT: expresii, foldr generalizat

Tipuri de date algebrice

Tipurile de date algebrice se definesc folosind "operațiile" sumă și produs.

Forma generală

$$
\begin{aligned}
data\ \ Typename \quad = \quad & Cons_1 \ \ t_{11} \ldots t_{1k_1} \\
& \mid Cons_2 \ \ t_{21} \ldots t_{2k_2} \\
& \mid \ldots \\
& \mid Cons_n \ \ t_{n1} \ldots t_{nk_n}
\end{aligned}
$$

unde $k_1, \ldots, k_n \geq 0$

- Se pot folosi tipuri sumă și tipuri produs.
- Se pot defini tipuri parametrizate.
- Se pot folosi definiții recursive.

din nou **foldr**

Problema: să generalizăm **foldr** la alte structuri recursive.

```
data Exp = Lit Int
         | Add Exp Exp
         | Mul Exp Exp
```

din nou **foldr**

Problema: să generalizăm **foldr** la alte structuri recursive.

```
data Exp     =   Lit Int
             |   Add Exp Exp
             |   Mul Exp Exp
```

Cum definim **"foldr"** înlocuind listele cu date de tip **Exp** ?

```
evalExp :: Exp -> Int
evalExp (Lit n)   = n
evalExp (Add e1 e2) = evalExp e1 + evalExp e2
evalExp (Mul e1 e2) = evalExp e1 * evalExp e2
```

Vrem să definim **"foldExp"** astfel încât

```
evalExp = foldExp fLit (+) (*)
```

din nou **foldr**

```
data Exp = Lit Int
         | Add Exp Exp
         | Mul Exp Exp
```

din nou **foldr**

```
data Exp     =    Lit Int
             |    Add Exp Exp
             |    Mul Exp Exp


foldExp fLit fAdd fMul (Lit n)    = fLit  n
```

din nou **foldr**

```
data Exp = Lit Int
         | Add Exp Exp
         | Mul Exp Exp


foldExp fLit fAdd fMul (Lit n)   = fLit n
foldExp fLit fAdd fMul (Add e1 e2) = fAdd v1 v2
                    where
                          v1 = foldExp fLit fAdd fMul e1
                          v2 = foldExp fLit fAdd fMul e2

foldExp fLit fAdd fMul (Mul e1 e2) = fMul v1 v2
                    where
                          v1 = foldExp fLit fAdd fMul e1
                          v2 = foldExp fLit fAdd fMul e2

evalExp = foldExp fLit (+) (*)
          where fLit (Lit x) = x
```

din nou **foldr**

```
data Exp = Lit Int
         | Add Exp Exp
         | Mul Exp Exp

foldExp fLit fAdd fMul (Lit n)   = fLit n
foldExp fLit fAdd fMul (Add e1 e2) = fAdd v1 v2
                       where ...

foldExp fLit fAdd fMul (Mul e1 e2) = fMul v1 v2
                       where ...
```

Ce tip are **foldExp**?

din nou **foldr**

```
data Exp     =   Lit Int
             |   Add Exp Exp
             |   Mul Exp Exp

foldExp fLit fAdd fMul (Lit n)     = fLit n
foldExp fLit fAdd fMul (Add e1 e2) = fAdd v1 v2
                       where ...

foldExp fLit fAdd fMul (Mul e1 e2) = fMul v1 v2
                       where ...
```

Ce tip are **foldExp**?

```
foldExp :: (Int->b)->(b->b->b)->(b->b-> b)-> Exp Int->b
```

Pe săptămâna viitoare!