Funkcionális nyelvek 2 (MSc)

Páli Gábor János
pgj@elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

Programozási Nyelvek és
Fordítóprogramok Tanszék

Tematika

A (tervezett) tematika rövid összefoglalása

`http://people.inf.elte.hu/pgj/fny2_msc/`

- Funktorok, applikatív funktorok
- Monádok, monádtípusok, ezek kombinációja
- Tulajdonságalapú tesztelés
- Konkurens és párhuzamos programozás
- Perzisztens (tisztán funkcionális) adatszerkezetek
- Hatékonyság funkcionális nyelvekben
- Beágyazott nyelvek

Bevezetés

A Haskellről röviden...

http://haskell.org/

- általános célú, tisztán funkcionális (deklaratív) programozási nyelv
- nem mohó (lényegében lusta) kiértékelés
- erős, statikus típusrendszer
- közös kutatási, oktatási, ipari törekvések összefogása
- *de jure* szabványok: Haskell 98/2010, Haskell Prime
- *de facto* szabvány: Glasgow Haskell Compiler (8.0.2)

A típusdefiníciók szintjei

Típusszinonima. Már meglevő típusok vagy azok kombinációjának elnevezése.

$$\textbf{type}\ \mathit{Bool} = \mathit{Int}$$

Származtatott típus. Már meglevő típusok vagy azok kombinációjával izomorf típus létrehozása.

$$\textbf{newtype}\ \mathit{Bool} = \mathit{B}\ \mathit{Int}$$

-- B :: Int $\rightarrow$ Bool, konverziós függvény

$$\begin{aligned} &\mathit{mkBool} :: \mathit{Int} \rightarrow \mathit{Bool} \text{ -- "smart constructor"} \\ &\mathit{mkBool}\ n \mid 0 \leq n \wedge n \leq 1 = \mathit{B}\ n \\ &\qquad\qquad \mid \mathit{otherwise} \qquad\quad = \mathit{error}\ \text{''}\mathit{Invalid\ value}\text{''} \end{aligned}$$

Algebraic Data Type (ADT). Lehetőségünk van a típus elemeinek pontos megadására.

$$\textbf{data}\ \mathit{Bool} = \mathit{True} \mid \mathit{False}$$

-- True, False :: Bool, adatkonstruktorok

Típusok algebrai konstrukciója

$$\textbf{data}\ \mathit{Unit} = \mathit{Unit}$$
$$\text{-- } \mathit{Unit} :: \star \text{ – unit (1)}$$
$$\textbf{data}\ \mathit{Pair}\ \alpha\ \beta = \mathit{Pair}\ \alpha\ \beta$$
$$\text{-- } \mathit{Pair} :: \star \to \star \to \star \text{ – product } (\times)$$
$$\textbf{data}\ \mathit{Either}\ \alpha\ \beta = \mathit{Left}\ \alpha \mid \mathit{Right}\ \beta$$
$$\text{-- } \mathit{Either} :: \star \to \star \to \star \text{ – sum vagy coproduct (+)}$$
$$\textbf{data}\ \mathit{Fix}\ \varphi = \mathit{Fix}\ (\varphi\ (\mathit{Fix}\ \varphi))$$
$$\text{-- } \mathit{Fix} :: (\star \to \star) \to \star \text{ – fixpont } (\mu)$$

Ezen kombinátorok alkalmazására egy példa:

List α = $\mu(\lambda\beta.1 + (\alpha \times \beta))$

$$\textbf{type}\ \mathit{List}\ \alpha = \mathit{Fix}\ (L\ \alpha)$$
$$\textbf{data}\ L\ \alpha\ \beta = L\ (\mathit{Either}\ \mathit{Unit}\ (\mathit{Pair}\ \alpha\ \beta))$$

vagy általában:

$$\textbf{data}\ \mathit{List}\ \alpha = \mathit{Nil} \mid \mathit{Cons}\ \alpha\ (\mathit{List}\ \alpha)$$

Információábrázolás típusokkal

Az algebrai típusokkal készíthetünk felsorolásokat:

$$\textbf{data}\ \mathit{Bool} = \mathit{True} \mid \mathit{False}$$

$$\begin{aligned}
&(\wedge) :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool} \\
&\mathit{True} \wedge \mathit{True} = \mathit{True} \\
&_ \wedge _ = \mathit{False}
\end{aligned}$$

vagy leírhatunk adatszerkezeteket:

$$\textbf{data}\ \mathit{Tree}\ \alpha = L \mid B\ \alpha\ (\mathit{Tree}\ \alpha)\ (\mathit{Tree}\ \alpha)$$

$$\begin{aligned}
&\mathit{aTree} :: \mathit{Tree\ Int} \\
&\mathit{aTree} = B\ 0\ (B\ 1\ L\ (B\ 2\ L\ L))\ L
\end{aligned}$$

$$\begin{aligned}
&\mathit{size} :: \mathit{Tree}\ \alpha \to \mathit{Integer} \\
&\mathit{size}\ L = 0 \\
&\mathit{size}\ (B\ _\ \mathit{lt}\ \mathit{rt}) = 1 + \mathit{size}\ \mathit{lt} + \mathit{size}\ \mathit{rt}
\end{aligned}$$

Számítás nélküli függvények

Tekintsük a Peano-számokat:

$Nat :: \star$ – **N**
$Zero :: Nat$ – 0
$Succ :: Nat \to Nat$ – S

$$\textbf{data}\ Nat\ =\ Zero \mid Succ\ Nat$$

$$aNat\ ::\ Nat$$
$$aNat\ =\ Succ\,(Succ\,(Succ\ Zero)))$$ -- S (S (S 0))

Nem közvetlenül a természetes számokat modellezzük, ehhez külön ilyen jelentést kell hozzá rendelnünk:

$$fromNat\ ::\ Nat\ \to\ Integer$$
$$fromNat\ Zero\ \ = 0$$
$$fromNat\ (Succ\ n)\ = 1\ +\ fromNat\ n$$

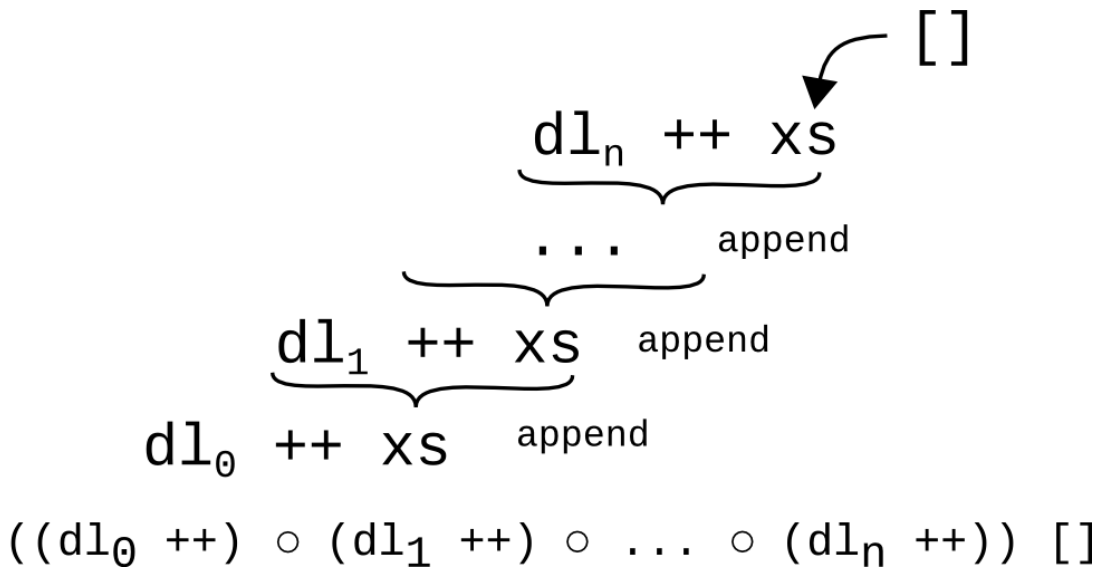
$aNat \not\to^*_\beta 3$, ám $fromNat\ aNat \to^*_\beta 3$

Adatszerkezetek függvényekkel

A differencialista egy függvény, amely egy előre tárolt listát fűz a paramétere után.

$$\textbf{newtype}\ \mathit{DiffList}\ \alpha\ =\ \mathit{DFL}\,([\alpha]\ \rightarrow\ [\alpha])$$

Segítségével egy lista elé és mögé tudunk elemeket $O(1)$ időben fűzni.



Invariánsok kódolása a típusban

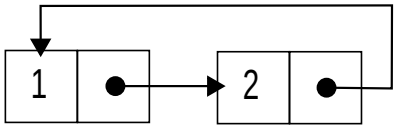
Lista váltakozó típusú elemekkel:

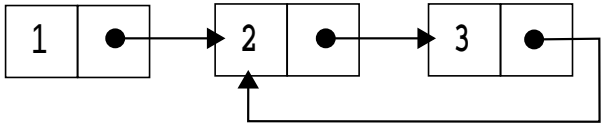
$$\textbf{data}\ \mathit{AList}\ \alpha\ \beta\ =\ \mathit{Nil} \mid \mathit{Cons}\ \alpha\ (\mathit{AList}\ \beta\ \alpha)$$

$$\mathit{alist}\ ::\ \mathit{AList\ Int\ Bool}$$
$$\mathit{alist}\ =\ \mathit{Cons}\ 1\ (\mathit{Cons\ True}\ (\mathit{Cons}\ 2\ (\mathit{Cons\ False\ Nil})))$$

Ciklikus listák:

$$\textbf{data}\ \mathit{Void}$$
$$\textbf{data}\ \mathit{Maybe}\ \alpha\ =\ \mathit{Nothing} \mid \mathit{Just}\ \alpha$$
$$\textbf{data}\ \mathit{CList}\ \alpha\ \beta\ =\ \mathit{Var}\ \beta \mid \mathit{Nil} \mid \mathit{C}\ \alpha\ (\mathit{CList}\ \alpha\ (\mathit{Maybe}\ \beta))$$





$$\mathit{clist}_1,\ \mathit{clist}_2\ ::\ \mathit{CList\ Int\ Void}$$
$$\mathit{clist}_1\ =\ \mathit{C}\ 1\ (\mathit{C}\ 2\ (\mathit{Var\ Nothing}))$$
$$\mathit{clist}_2\ =\ \mathit{C}\ 1\ (\mathit{C}\ 2\ (\mathit{C}\ 3\ (\mathit{Var}\ (\mathit{Just\ Nothing}))))$$

Nyelvbeágyazás típusokkal

```
data Expr
  = ValI Integer | ValB Bool
  | Add Expr Expr | And Expr Expr
  | Eq Expr Expr | If Expr Expr Expr

eval :: Expr → Either Integer Bool
eval (ValI n)      = Left n
eval (ValB b)      = Right b
eval (x `Add` y)   = case (eval x, eval y) of
                       (Left m , Left n) → Left (m + n)
eval (x `And` y)   = case (eval x, eval y) of
                       (Right a, Right b) → Right (a ∧ b)
eval (x `Eq` y)    = case (eval x, eval y) of
                       (Left m , Left n)  → Right (m ≡ n)
                       (Right m, Right n) → Right (m ≡ n)
eval (If x y z)    = case (eval x) of
                       Right b → if b then (eval y) else (eval z)
```

Típusozás típusokkal

De a Haskell-fordító meggyőzhető arról, hogy akár statikusan is tudja típusozni a beágyazott nyelvünket:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Expr} :: \star \to \star\ \textbf{where} \\
\quad \mathit{ValI} :: \mathit{Integer} \to \mathit{Expr}\ \mathit{Integer} \\
\quad \mathit{ValB} :: \mathit{Bool} \to \mathit{Expr}\ \mathit{Bool} \\
\quad \mathit{Add} :: \mathit{Expr}\ \mathit{Integer} \to \mathit{Expr}\ \mathit{Integer} \to \mathit{Expr}\ \mathit{Integer} \\
\quad \mathit{And} :: \mathit{Expr}\ \mathit{Bool} \to \mathit{Expr}\ \mathit{Bool} \to \mathit{Expr}\ \mathit{Bool} \\
\quad \mathit{Eq} :: \mathit{Eq}\ \alpha \Rightarrow \mathit{Expr}\ \alpha \to \mathit{Expr}\ \alpha \to \mathit{Expr}\ \mathit{Bool} \\
\quad \mathit{If} :: \mathit{Expr}\ \mathit{Bool} \to \mathit{Expr}\ \alpha \to \mathit{Expr}\ \alpha \to \mathit{Expr}\ \alpha
\end{array}
$$

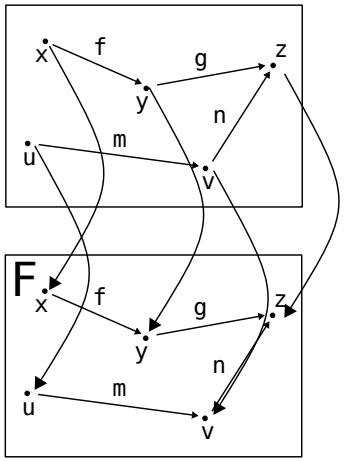
$$
\begin{array}{ll}
\mathit{eval} :: \mathit{Eq}\ \alpha \Rightarrow \mathit{Expr}\ \alpha \to \alpha & \\
\mathit{eval}\ (\mathit{ValI}\ n) & = n \\
\mathit{eval}\ (\mathit{ValB}\ b) & = b \\
\mathit{eval}\ (x\ \text{`}\mathit{Add}\text{`}\ y) & = (\mathit{eval}\ x) + (\mathit{eval}\ y) \\
\mathit{eval}\ (x\ \text{`}\mathit{And}\text{`}\ y) & = (\mathit{eval}\ x) \wedge (\mathit{eval}\ y) \\
\mathit{eval}\ (x\ \text{`}\mathit{Eq}\text{`}\ y) & = (\mathit{eval}\ x) \equiv (\mathit{eval}\ y) \\
\mathit{eval}\ (\mathit{If}\ x\ y\ z) & = \textbf{if}\ (\mathit{eval}\ x)\ \textbf{then}\ (\mathit{eval}\ y)\ \textbf{else}\ (\mathit{eval}\ z)
\end{array}
$$

Funktorok

Funktorok: Bevezetés

$$\textbf{class}\ \mathit{Functor}\ (\varphi :: \star \to \star)\ \textbf{where}$$
$$\quad \mathit{fmap} :: (\alpha \to \beta) \to (\varphi\,\alpha \to \varphi\,\beta)$$
$$(<\$>) = \mathit{fmap}\ \text{-- } \mathit{Control.Applicative}$$

$$\textbf{class}\ (\mathit{Functor}\ \varphi) \Rightarrow \mathit{Applicative}\ (\varphi :: \star \to \star)\ \textbf{where}$$
$$\quad \mathit{pure} :: \alpha \to \varphi\,\alpha$$
$$\quad (<*>) :: \varphi\,(\alpha \to \beta) \to \varphi\,\alpha \to \varphi\,\beta$$
$$\textbf{infixl}\ 4\ <\$>,\ <*>$$



Funktortörvények:

$\mathit{fmap}\ \mathit{id}$	$\equiv \mathit{id}$	(identitás)
$\mathit{fmap}\ (f \circ g)$	$\equiv \mathit{fmap}\ f \circ \mathit{fmap}\ g$	(kompozíció)
$\mathit{pure}\ \mathit{id}\ <*>\ v$	$\equiv v$	(identitás)
$\mathit{pure}\ (\circ)\ <*>\ u\ <*>\ v\ <*>\ w$	$\equiv u\ <*>\ (v\ <*>\ w)$	(kompozíció)
$\mathit{pure}\ f\ <*>\ \mathit{pure}\ x$	$\equiv \mathit{pure}\ (f\ x)$	(homomorfizmus)
$u\ <*>\ \mathit{pure}\ y$	$\equiv \mathit{pure}\ (\$\ y)\ <*>\ u$	(felcserélhetőség)

Egyszerű példák funktorokra

```
instance Functor [] where
  fmap = map

instance Applicative [] where
  pure x       = [x]
  fs <*> xs    = [f x | f <- fs, x <- xs]
instance Functor Maybe where
  fmap _ Nothing  = Nothing
  fmap f (Just x) = Just (f x)
instance Applicative Maybe where
  pure                    = Just
  (Just f) <*> (Just x)   = Just (f x)
  _        <*> _          = Nothing
```