Curs 11

Implementarea Mini-Haskell în Haskell

Mini-Haskell

Vom defini folosind Haskell un mini limbaj funcţional şi semantica lui denotaţională.

- Limbajul Mini-Haskell conţine:
  - expresii de tip **Int**
  - expresii de tip funcţie (λ-expresii)
  - expresii provenite din aplicarea functiilor
- Pentru a defini semantica limbajului vom introduce domeniile semantice (valorile) asociate expresiilor limbajului.
- Pentru a evalua (interpreta) expresiile vom defini un mediu de evaluare in care vom retine variabilele si valorile curente asociate.

Mini-Haskell (λ-calcul cu întregi). Sintaxă

```
type Name = String

data Term = Var Name
          | Con Integer
          | Term :+: Term
          | Lam Name Term
          | App Term Term
deriving (Show)
```

Program - Exemplu

λ-expresia $(\lambda x.x + x)(10 + 11)$
este definită astfel:

```
pgm :: Term
pgm = App
          (Lam "x" ((Var "x") :+: (Var "x")))
          ((Con 10) :+: (Con 11))
```

Program - Exemplu

```
pgm :: Term
pgm = App
   (Lam "y"
      (App
         (App
            (Lam "f"
               (Lam "y"
                  (App (Var "f") (Var "y"))
               )
            )
            (Lam "x"
               (Var "x" :+: Var "y")
            )
         )
         (Con 3)
      )
   )
   (Con 4)
```

Domenii

Domeniul valorilor

```
data Value = Num Integer
           | Fun (Value -> Value)
           | Wrong -- pentru reprezentarea erorilor
```

Mediul de evaluare

```
type Environment = [(Name, Value)]
```

Domeniul de evaluare

Fiecărei expresii i se va asocia ca denotație o funcție de la medii de evaluare la valori:

```
interp :: Term -> Environment -> Value
```

Afișarea expresiilor

```
instance Show Value where
  show (Num x) = show x
  show (Fun _) = "<function>"
  show Wrong   = "<wrong>"
```

Observație

Funcțiile nu pot fi afișate ca atare, ci doar generic.

Evaluarea expresiilor Mini-Haskell în Haskell

```
interp :: Term -> Environment ->  Value

interp (Con i) _ =  Num i

interp (t1 :+: t2) env = add (interp t1 env) (interp
    t2 env)

add :: Value -> Value ->  Value
add (Num i) (Num j) = Num $ i + j
add _ _             =  Wrong
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
interp :: Term -> Environment ->  Value

interp (Var x) env = lookupM x env
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
interp :: Term -> Environment ->  Value

interp (Var x) env = lookupM x env


lookupM :: Name -> Environment ->  Value
lookupM x env = case lookup x env of
  Just v  -> v
  Nothing -> Wrong

-- lookup din modulul Data.List
   lookup :: (Eq a) => a -> [(a,b)] -> Maybe b
   lookup _key []          =  Nothing
   lookup  key ((x,y):xys)
       | key == x          =  Just y
       | otherwise         =  lookup key xys
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
interp :: Term -> Environment ->  Value

interp (Lam x e) env = Fun $\v ->interp e((x,v):env)
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
interp :: Term -> Environment ->  Value

interp (Lam x e) env = Fun $\v ->interp e((x,v):env)

interp (App t1 t2) env = apply  f v
  where
      f = interp t1 env
      v = interp t2 env

apply :: Value -> Value ->  Value
apply (Fun k) v = k v
apply _ _       =  Wrong
```

Implementarea Mini-Haskell în Haskell

```
interp :: Term -> Environment ->  Value
interp (Var x) env = lookupM x env
     where lookupM x env = case lookup x env of
                                Just v  -> v
                                Nothing -> Wrong
interp (Con i) _ =  Num i
interp (t1 :+: t2) env = add (interp t1 env) (interp
    t2 env)
       where add (Num i) (Num j) = Num $ i + j
             add _ _             =  Wrong
interp (Lam x e) env =  Fun $ \ v -> interp e ((x,v):
    env)
interp (App t1 t2) env = apply (interp t1 env) (
    interp t2 env)
       where apply (Fun k) v = k v
             apply _ _       =  Wrong
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
pgm :: Term
pgm  = App
          (Lam "x" ((Var "x") :+: (Var "x")))
          ((Con 10) :+:  (Con 11))

test :: Term -> String
test t =  show $ interp t []

*Main> test pgm
"42"
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
pgmW :: Term
pgmW  = App
          (((Var "x") :+: (Var "x")))
          ((Con 10) :+:  (Con 11))

test :: Term -> String
test t =  show $ interp t []

*Main> test pgmW
"<wrong>"
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
pgm :: Term
pgm  = App
          (Lam "x" ((Var "x") :+: (Var "x")))
          ((Con 10) :+:  (Con 11))

testIO :: Term -> String
testIO t =  putStrLn . show $ interp t []

> test pgm
42
```

Evaluarea expresiilor Mini-Haskell în Haskell

```
pgm :: Term
pgm  = App
          (Lam "x" ((Var "x") :+: (Var "x")))
          ((Con 10) :+:  (Con 11))

testIO :: Term -> String
testIO t =  putStrLn . show $ interp t []

> test pgm
42
```

Ce este **IO**?