Laboratorul 11: MicroHaskell

MicroHaskell

Vom defini un mini-limbaj funcțional, împreună cu semantica lui denotațională.

Limbajul Mini-Haskell conține:

- expresii de tip `Bool`,
- expresii de tip `Int`
- expresii de tip `funcție` (lambda-expresii)
- expresii provenite din `aplicarea functiilor`

Tipul `Name` va reprezenta valorile `String` asociate numelor de variabile.

```
type Name = String
```

Tipul `Value` reprezintă valorile asociate expresiilor. Valorile pot fi elemente Booleene, elemente întregi, funcții sau erori.

```
data  Value  =  VBool Bool
     |VInt Int
     |VFun (Value -> Value)
     |VError
```

```
data  Hask  = HTrue | HFalse
     |HIf Hask Hask Hask
     |HLit Int
     |Hask :==: Hask
     |Hask :+:  Hask
     |HVar Name
     |HLam Name Hask
     |Hask :$: Hask
```

```
infix 4 :==:
infixl 6 :+:
infixl 9 :$:
```

Tipul `HEnv` reprezintă mediul de evaluare al expresiilor, asociind fiecărei variabile o valoare de tip `Value`.

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

1. Afișarea valorilor expresiilor din `Hask`

Să se instanțieze clasa `Show` pentru tipul `Value`, astfel încât să se afișeze valorile fără constructori. Funcțiile și erorile nu se pot afișa. Se va afișa un mesaj corespunzător.

```
instance Show Value where
     show = undefined
```

2. Egalitate pentru valori

Să se instanțieze clasa `Eq` pentru tipul `Value`, astfel încât să se verifice egalitatea între elemente de tip `Value`. Funcțiile și valorile nu se pot compara. Putem întoarce o eroare folosind funcția `error`.

```
instance Eq Value where
     x == y = undefined
```

3. Evaluarea Expresiilor de tip `Hask`

Completați funcția `hEval` astfel încât să evalueze fiecare expresie de tip `Hask`. Pentru a căuta o variabilă în mediul de evaluare, puteți folosi funcția `lookup` din modulul `Data.List` sau să vă definiți funcția proprie.

```
hEval :: Hask -> HEnv -> Value
hEval HTrue r      =  VBool True
hEval HFalse r        =  VBool False
hEval (HIf c d e) r   = hif (hEval c r) (hEval d r) (hEval e r)
  where  hif (VBool b) v w  =  if b then v else w
         hif _ _ _ = VError
hEval _ _ = undefined
```

4. Rularea unui program

Definim următoarea funcție:

```
run :: Hask -> String
run pg = show (hEval pg [])
```

Astfel, `run pgm` va întoarce rezultatul evaluării (rulării) programului `pgm`.

4.1) Scrieți mai multe programe si rulați-le pentru a vă familiariza cu sintaxa.

4.2) Adăugați operația de înmulțire pentru expresii, cu precedență mai mare decât a operației de adunare. Definiți semantica operației de înmulțire.

4.3) Folosind funcția `error`, înlocuiți acolo unde este posibil valoarea `VError` cu o eroare care să precizeze motivul apariției erorii.