Programare declarativă

Introducere în programarea funcțională folosind Haskell

Traian Florin Șerbănuță - seria 33
Ioana Leuștean - seria 34

Departamentul de Informatică, FMI, UB
traian.serbanuta@fmi.unibuc.ro
ioana@fmi.unibuc.ro

Semantica denotationala

Feluri de a da semantica

- Limbaj de programare: sintaxă și semantică
- Feluri de semantică
  - Limbaj natural — descriere textuală a efectelor
  - Operațională — asocierea unei demonstrații a execuției
  - Axiomatică — Descrierea folosind logică a efectelor unei instrucțiuni
  - Denotațională — prin asocierea unui obiect matematic (denotație)
  - Statică — Asocierea unui sistem de tipuri care exclude programe eronate

Limbajul unui mini calculator

Definim în Haskell limbajul unui mini calculator:

```
data Prog  = On Instr
data Instr  = Off | Expr :> Instr
data Expr  =  Mem | V Int | Expr :+ Expr
```

Limbajul unui mini calculator

Definim în Haskell limbajul unui mini calculator:

```
data Prog  = On Instr
data Instr  = Off | Expr :> Instr
data Expr  =  Mem | V Int | Expr :+ Expr
```

Semantica în limbaj natural

Dorim ca un program să afișeze lista valorilor corespunzătoare expresiilor, unde Mem reprezintă ultima valoare calculată. Valoarea inițială a lui Mem este 0.

De exemplu, programul

On ((V 3) :> ((Mem :+ (V 5)) :> Off))

va afișa lista [3,8]

Semantica denotațională - domenii semantice

Categoriilor sintactice le corespund domenii semantice

Categorii sintactice	**Domenii semantice**
Prog	$\mathbb{Z}^*$
Instr	$\mathbb{Z} \to \mathbb{Z}^*$
Expr	$\mathbb{Z} \to \mathbb{Z}$

unde $\mathbb{Z}$ este mulțimea numerelor întregi. Observăm că domeniile semantice pentru Instr și Expr sunt funcții deoarece depind de valoarea din memorie.

Semantica denotațională - domenii semantice

Categoriilor sintactice le corespund domenii semantice

Categorii sintactice	Domenii semantice
Prog	$\mathbb{Z}^*$
Instr	$\mathbb{Z} \to \mathbb{Z}^*$
Expr	$\mathbb{Z} \to \mathbb{Z}$

unde $\mathbb{Z}$ este mulțimea numerelor întregi. Observăm că domeniile semantice pentru Instr și Expr sunt funcții deoarece depind de valoarea din memorie.

Domeniile semantice în Haskell

```
type Env = Int
type DomProg = [Int]
type DomInstr = Int -> [Int]
type DomExpr = Int -> Int
```

Semantica denotațională

Pentru a defini semantica denotațională trebuie să evaluăm (interpretăm) categoriile sintactice în domeniile semantice corespunzătoare.

Interpretări (Evaluări)

```
prog :: Prog -> DomProg
stmt :: Instr -> DomInstr
expr ::  Expr -> DomExpr
```

Observație. Interpretările trebuie să reflecte cerințele semantice explicate în limbaj natural. De exemplu

```
prog :: Prog -> DomProg
prog (On s) = stmt s 0
```

deoarece am precizat că valoarea inițială a celulei de memorie Mem este 0.

Semantica denotațională

Etape în definirea semanticii denotaționale

- identificăm categoriile sintactice;
- asociem fiecărei categorii sintactice un domeniu semantic;
- definim interpretări ale categoriilor sintactice în domeniile semantice.

Avantaje și dezavantaje

+ Formală, matematică, foarte precisă
+ Compozițională (morfisme și compuneri de funcții)
- Greu de stăpânit (domeniile devin din ce în ce mai complexe)

Semantica denotațională în Haskell

```
type DomProg = [Int]
type DomInstr =  Int -> [Int]
type DomExpr = Int -> Int

prog :: Prog -> DomProg
prog (On s) = stmt s 0

stmt :: Instr -> DomInstr
stmt  (e :> s) m = let v = expr e m in  (v : (stmt s v))
stmt Off _ = []

expr ::  Expr -> DomExpr
expr (e1 :+  e2) m = (expr  e1 m) + (expr  e2 m)
expr (V n) _  =  n
expr  Mem m  = m
```

Mini-Haskell

Mini-Haskell

Vom defini folosind Haskell un mini limbaj funcțional și semantica lui denotațională.

- Limbajul Mini-Haskell conține:
  - expresii de tip **Bool** și expresii de tip **Int**
  - expresii de tip funcție (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 (memoria) in care vom retine variabilele si valorile curente asociate.

Sintaxă

```
data Hask = HTrue
          | HFalse
          | Hlf Hask Hask Hask
          | HLit Int
          | Hask :==: Hask
          | Hask :+: Hask
          | HVar Name
          | HLam Name Hask
          | Hask :$: Hask
  deriving (Read, Show)
infix  4 :==:
infixl 6 :+:
infixl 9 :$:
```

Domenii

Domeniul valorilor

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

Mediul de evaluare

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

Domeniul de evaluare

Fiecărei expresii i se va asocia ca denotație o funcție de la medii de evaluare la valori. Deci Domeniul de evaluare al expresiilor este

```
type    DomHask    =    HEnv -> Value
```

Afișarea expresiilor din Hask

```
instance Show Value where
  show (VBool b)    = show b
  show (VInt i)     = show i
  show (VFun _)     = "<function>"
  show VError       = "<error>"
```

Observație

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

Egalitate pentru valori

```
instance Eq Value where
  (VBool b) == (VBool c)  = b == c
  (VInt i)  == (VInt j)   = i == j
  (VFun _)  == (VFun _)   = error "Unknown"
  VError    == VError     = error "Unknown"
  _         == _          = False
```

Observație

Funcțiile si erorile nu pot fi testate dacă sunt egale.

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask --*type DomHask = 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
```

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask --*type DomHask = HEnv -> Value*

```
hEval (HLit i) r   = VInt i

hEval (d :==: e) r = heq (hEval d r) (hEval e r)
  where
    heq (VInt i) (VInt j) = VBool (i == j)
    heq _ _               = VError


hEval (d :+: e) r = hadd (hEval d r) (hEval e r)
  where
    hadd (VInt i) (VInt j) = VInt (i + j)
    hadd _ _               = VError
```

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask --*type DomHask = HEnv -> Value*

```
hEval (HVar x) r        = fromMaybe VError (lookup x r)
```

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask --*type DomHask = HEnv -> Value*

```
hEval (HVar x) r        = fromMaybe VError (lookup x r)

-- 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

-- fromMaybe din modulul Data.Maybe
fromMaybe :: a -> Maybe a -> a
fromMaybe d x = case x of
    Nothing -> d
    Just v  -> v
```

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask --*type DomHask = HEnv -> Value*

```
hEval :: Hask -> HEnv -> Value

hEval (HLam x e) r     = VFun (\ v -> hEval e ((x,v):r))
```

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask --*type DomHask = HEnv -> Value*

```
hEval :: Hask -> HEnv -> Value

hEval (HLam x e) r     = VFun (\ v -> hEval e ((x,v):r))

hEval (d :$: e) r      = happ (hEval d r) (hEval e r)
  where
    happ (VFun f) v = f v
    happ _ _        = VError
```

Evaluarea expresiilor Mini-Haskell în Haskell

hEval :: Hask -> DomHask *--type DomHask = 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 (HLit i) r       = VInt i
hEval (d :=: e) r      = heq (hEval d r) (hEval e r)
  where heq (VInt i) (VInt j) = VBool (i == j)
        heq _ _               = VError
hEval (d :+: e) r    = hadd (hEval d r) (hEval e r)
  where hadd (VInt i) (VInt j) = VInt (i + j)
        hadd _ _               = VError
hEval (HVar x) r       = fromMaybe VError (lookup x r)
hEval (HLam x e) r     = VFun (\ v -> hEval e ((x,v):r))
hEval (d :$: e) r    = happ (hEval d r) (hEval e r)
  where happ (VFun f) v = f v
        happ _ _        = VError
```

Evaluarea expresiilor Mini-Haskell în Haskell

Test

```
h0 =
    (HLam "x" (HLam "y" (HVar "x" :+: HVar "y")))
    :$: HLit 3
    :$: HLit 4

test_h0 = hEval h0 [] == VInt 7
```