Laboratorul 7

O implementare pentru lambda-calcul

Descriere generală a problemei

În acest laborator vom implementa funcțiile și predicatele descrise în cursul despre lambda-calcul pentru a obține o modalitate de a evalua lambda-expresiile și a le testa egalitatea.

Variabile

Generarea de variabile noi

O parte importantă din procesul de substituție modulo alfa-echivalență este posibilitatea de a redenumi variabilele legate cu versiuni noi ale lor, evitând astfel problema *capturării* variabilelor.

Vom considera Variabilele ca fiind *indexate* de un întreg:

```
data Variable = Variable String Int
```

Primul argument al constructorului `Variable` va reprezenta numele variabilei iar al doilea va folosi pentru a putea diferenția între variabile cu același nume.

Putem defini și un constructor simplu pentru variabile:

```
var :: String -> Variable
var x = Variable x 0
```

Exerciții

Exercițiul 1

Scrieți o funcție
`fresh :: Variable -> [Variable] -> Variable` care produce o variabilă nouă, cu același nume ca primul argument, dar diferită de acesta și de oricare variabilă din lista dată.

```
fresh (Variable "x" 0) [Variable "y" 2, Variable "x" 1]
    == Variable "x" 2
```

Exercițiul 2

Faceți `Variable` instanță a clasei `Show`, pentru a obține un comportament de genul:

```
show (Variable "x" 0) == "x"
show (Variable "x" 14) == "x_14"
```

Sintaxă abstractă și variabile libere

Sintaxă abstractă și scurtături

```
data Term
  = V Variable
  | App Term Term
  | Lam Variable Term

v :: String -> Term
v x = V (var x)

lam :: String -> Term -> Term
lam x = Lam (var x)

lams :: [String] -> Term -> Term
lams xs t = foldr lam t xs

($$) :: Term -> Term -> Term
($$) = App
infixl 9 $$
```

Exercițiul 3

Faceți `Term` instanță a clasei `Show`, pentru a obține o reprezentare complet parantezată a lambda-expresiilor.

```
show (v "x")                    == "x"
show (v "x" $$ v "y")           == "(x y)"
show (lam "x" (v "x"))          == "(\x.x)"
show (lams ["x","y"] (v "x")) == "(\x.(\y.x))"
```

Variabile libere

Exercițiul 4

Definiți o funcție `freeVars :: Term -> [Variable]` care calculează variabilele libere dintr-un termen.

```
freeVars (lam "x" (v "x" $$ v "y")) == [var "y"]
```

Exercițiul 5

Definiți o funcție `allVars :: Term -> [Variable]` care calculează *toate* variabilele (libere sau legate) dintr-un termen.

```
allVars (lam "x" (v "x" $$ v "y")) == [var "x", var "y"]
```

Substituția

Substituție cu redenumire — Exercițiul 6

Substituția definită la curs e parțială. Putem să o facem totală redenumind variabila legată cu una nouă (`fresh`).

Definiți o funcție care implementează substituția unui termen pentru o variabilă într-un termen dat.

```
subst :: Term -> Variable -> Term -> Term  -- [u/x]t
subst u x (Lam y t)  -- usor diferita fata de curs
  | x == y                    =
  | y `notElem` freeVarsU =
  | x `notElem` freeVarsT =
  | otherwise                 =
  where
    freeVarsT = freeVars t
    freeVarsU = freeVars u
    allFreeVars = nub ([x] ++ freeVarsU ++ freeVarsT)
    y' = freshVariable y allFreeVars
    t' = (subst (V y') y t)  -- t' = [y'/y]t
```

Alfa conversie și alfa-echivalență

Alfa-conversie — Exercițiul 7

Definiția de la curs e corectă matematic, dar nu f. algoritmică.
Vom folosi o altă definiție:

$$\frac{}{x \equiv_\alpha x} \qquad \frac{t_1 \equiv_\alpha t_1' \quad t_2 \equiv_\alpha t_2'}{t_1\ t_2 \equiv_\alpha t_1'\ t_2'}$$

$$\frac{t \equiv_\alpha t'}{\lambda x.t \equiv_\alpha \lambda x.t'} \qquad \frac{[y/x]t \equiv_\alpha [y/x']t'}{\lambda x.t \equiv_\alpha \lambda x'.t'} \text{ where } y \text{ fresh}$$

Scrieți o funcție `aEq :: Term -> Bool` care implementează regulile de mai sus.

```
aEq (lam "x" (v "x")) (lam "y" (v "y")) == True
aEq (lam "x" (v "x")) (lam "y" (v "z")) == False
```

Beta reducție și (alfa-)beta echivalență

Beta reducție până la o formă normală — Exercițiul 8

Scrieți o funcție `reduce :: Term -> Term` care folosește beta-reducția pentru a evalua un termen până la o formă normală.

```
reduce
  (lams ["m", "n"] (v "n" $$ v "m")
  $$ lams ["s", "z"] (v "s" $$ (v "s" $$ v "z"))
  $$ lams ["s", "z"] (v "s" $$ (v "s" $$ (v "s" $$ v "z")))
  )
  ==
  lams ["s", "z"] (v "s" $$ (v "s" $$ (v "s" $$ (v "s"
      $$ (v "s" $$ (v "s" $$ (v "s" $$ v "z"))))))
```

(Alfa-)Beta echivalența — Exercițiul 9

Scrieți o funcție `abEq :: Term -> Term -> Bool` care testează dacă doi termeni sunt beta-convertibili astfel:

- reduce ambii termeni la o formă normală
- verifică dacă formele normale sunt alfa-convertibile