Examen PP – Seria CC — NOT EXAM MODE

16.06.2017

Timp de lucru 2 ore . 100p necesare pentru nota maximă

---

1. Determinați forma normală pentru următoarea expresie, ilustrând pașii de reducere:
   $((\lambda z.\lambda y.\lambda x.(y\ \ z)\ \ y)\ \ \lambda y.y)$
   Soluție:
   $((\lambda \underline{z}.\lambda y.\lambda x.(y\ \ \underline{z})\ \ y)\ \ \lambda y.y) \rightarrow_\alpha ((\lambda \underline{z}.\lambda w.\lambda x.(w\ \ \underline{z})\ \ y)\ \ \lambda y.y) \rightarrow_\beta (\lambda \underline{w}.\lambda x.(\underline{w}\ \ y)\ \ \lambda y.y) \rightarrow_\beta \lambda x.(\lambda \underline{y}.\underline{y}\ \ y) \rightarrow_\beta \lambda x.y$

2. Este vreo diferență (ca efect, la execuție) între cele două linii de cod Racket? Dacă da, care este diferența?; dacă nu, de ce nu diferă?
   ```
   (let ((a 1) (b 2)) (+ a b))
   ((lambda (a b) (+ a b)) 1 2)
   ```
   Soluție:
   Nu este nicio diferență; `(let ((a_i v_i)) corp)` este echivalent cu `(lambda (a_i) corp)` aplicat parametrilor `v_i`

3. Implementați în Racket funcția `f` care primește o listă și determină elementul mai mare decât modulul oricărui alt element. Folosiți, în mod obligatoriu, cel puțin o funcțională.
   Soluție:
   ```
   (car (filter (λ(e) (null?  (filter (compose ((curry <) e) abs) L))) L))
   ```
   sau
   ```
   (car (filter (λ(e) (null?  (filter (λ(a) (< e (abs a))) L))) L))
   ```
   sau
   ```
   (last (sort L <))
   ```

4. Sintetizați tipul funcției `f` (în Haskell): `f g h l1 l2 = filter (g .  h) (l1 ++ l2)`
   Soluție:
   ```
   f ::  a → b → c → d → e
   filter ::  (i → Bool) → [i] → [i]
   l1 ::  c și l2 ::  d ⇒ c = d = [i]
   (.)  ::  (t2 → t3) → (t1 → t2) → (t1 → t3)
   a = g1 → g2 = t2 → t3
   b = h1 → h2 = t1 → t2
   g .  h ::  i → Bool = t1 → t3
   c = d = [i] = [t1] = [h1]
   [t3] = [g2] = Bool
   f ::  (t2 → Bool) → (t1 → t2) → [t1] → [t1] → [t1]
   ```

5. Scrieți definiția în Haskell a clasei `Ended` care, pentru un tip colecție `t` construit peste un alt tip `v`, definește o funcție `frontEnd` care extrage primul element din colecție și o funcție `backEnd` care extrage ultimul element din colecție.
   Instanțiați această clasă pentru tipul `data Triple a = T a a a`
   Soluție:
   ```
   class Ended t where frontEnd ::  t v -> v; backEnd ::  t v -> v
   instance Ended Triple where

     frontEnd (T x _ _) = x
     backEnd (T _ _ x) = x
   ```

6. Știind că *Ucenicul vrea sa învețe pe dascăl* , și că *ucenic*$(Luke)$ și *dascal*$(Yoda)$, demonstrați folosind rezoluția că *vrea-să-învețe*$(Luke, Yoda)$ este adevărat .

Soluție:
$\forall x.\forall y.ucenic(x) \wedge dascal(y) \rightarrow vrea\_sa\_invete(x, y)$
$\neg ucenic(x) \vee \neg dascal(y) \vee vrea\_sa\_invete(x, y)$
$+ucenic(Luke)\{x \leftarrow Luke\} \Rightarrow \neg dascal(y) \vee vrea\_sa\_invete(Luke, y)$
$+\neg vrea\_sa\_invete(Luke, Yoda)\{y \leftarrow Yoda\} \Rightarrow \neg dascal(Yoda)$
$+dascal(Yoda) \Rightarrow$ clauza vidă

7. Implementați în Prolog predicatul `x(L, M)` care detemină, pentru o listă `L`, `M`, minimul listei. Nu folosiți recursivitate explicită.
Soluție:
```
x(L, M) :- member(M, L), forall(member(E, L), X < E).
```

8. Implementați un algorim Markov care primește în șirul de intrare un număr binar și scade 2 din acest număr. Exemple: 10 - 10 = 0; 100 - 10 = 10; 111 - 10 = 101; 10001 - 10 = 1111. Este ok dacă numărul rezultat începe cu 0.
Soluție:
`ag` $\rightarrow$ `ga`
`ga` $\rightarrow$ `gb`
`1b` $\rightarrow$ `0c`
`0b` $\rightarrow$ `b1`
`c` $\rightarrow$ `.`
$\rightarrow$ `a`