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:
   $((\lambda x.\lambda y.\lambda z.(y\ \ x)\ \ y)\ \ \lambda z.z)$
   Soluție:
   $((\lambda \underline{x}.\lambda y.\lambda z.(y\ \ \underline{x})\ \ y)\ \ \lambda z.z) \rightarrow_\alpha ((\lambda \underline{x}.\lambda w.\lambda z.(w\ \ \underline{x})\ \ y)\ \ \lambda z.z) \rightarrow_\beta (\lambda \underline{w}.\lambda z.(\underline{w}\ \ y)\ \ \lambda z.z) \rightarrow_\beta \lambda z.(\lambda \underline{z}.\underline{z}\ \ y) \rightarrow_\beta \lambda z.y$

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?
   ```
   (define a 2) (let ((a 1) (b a)) (+ a b))
   (define a 2) (letrec ((a 1) (b a)) (+ a b))
   ```
   Soluție:
   În prima linie, definiția `(a 1)` este vizibilă în corpul `let`-ului, dar nu și în definiția lui `b`, care vede încă `a=2`; prima linie dă 3, a două dă 2.

3. Implementați în Racket funcția `f` care primește o listă și determină cel mai mic element. Folosiți, în mod obligatoriu, cel puțin o funcțională.
   Soluție:
   ```
   (car (filter (λ(e) (null?  (filter ((curry >) e) L))) L))
   ```
   sau
   ```
   (car (filter (λ(e) (null?  (filter (λ(a) (> e a)) L))) L))
   ```
   sau
   ```
   (last (sort L >))
   ```

4. Sintetizați tipul funcției `f` (în Haskell): `f x y z g = filter g [x, y, z]`
   Soluție:
   `f ::  a → b → c → d → e`
   `d = g1 → g2`
   `filter ::  (t1 → Bool) → [t1] → [t1]`
   `a = b = c` (parte din aceeași listă)
   `t1 = a = b = c`
   `e = [t1]`
   `f ::  t1 → t1 → t1 → (t1 → Bool) → [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 Pair a = MakePair a a`
   Soluție:
   ```
   class Ended t where frontEnd ::  t v -> v; backEnd ::  t v -> v
   instance Ended Pair where

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

6. Știind că *Un om cum își așterne, așa doarme* , și că $asterne(Nectarie, bine)$ și $om(Nectarie)$, demonstrați folosind rezoluția că $doarme(Nectarie, bine)$ este adevărat .
   Soluție:
   $\forall x.\forall y.om(x) \wedge asterne(x,y) \rightarrow doarme(x,y)$
   $\neg om(x) \vee \neg asterne(x,y) \vee doarme(x,y)$

$+\neg doarme(Nectarie, bine)\{x \leftarrow Nectarie, y \leftarrow bine\} \Rightarrow$
$\neg om(Nectarie) \vee \neg asterne(Nectarie, bine)$
$+om(Nectarie) \Rightarrow \neg asterne(Nectarie, bine)$
$+asterne(Nectarie, bine) \Rightarrow$ clauza vidă

7. Implementați în Prolog predicatul `x(L, A, B, N)` care detemină, pentru o listă `L`, numărul `N` de elemente care nu sunt mai mari decât `A` și mai mici decât `B`. Nu folosiți recursivitate explicită.
   Soluție:
   ```
   x(L, A, B, N) :- findall(X, (member(X, L), (X < A; X > B)), S), length(S, N).
   ```

8. Implementați un algorim Markov care primește în șirul de intrare un număr binar și scade 1 din acest număr. Exemple: 1 - 1 = 0; 10 - 1 = 1; 11 - 1 = 10; 100 - 1 = 11; 1010 - 1 = 1001. Este ok dacă numărul rezultat începe cu 0.
   Soluție:
   ```
   ag → ga
   a → b
   1b → 0c
   0b → b1
   c → .
   → a
   ```