Examen PP – Seria CD — NOT EXAM MODE

14.06.2017

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

---

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

2. Ce întoarce următorul cod Racket și ce reprezintă valoarea întoarsă?
   ```
   (let f ((n 120) (m 5)) (if (zero?  m) 1 (f (/ n m) (- m 1))))
   ```
   Soluție:
   1, împarte n succesiv la m, m-1, m-2... cât timp m > 0.

3. De câte ori se evaluează expresia `(F 1)` în codul Racket de mai jos?
   ```
   (let ((proc (λ (x) (* (x) (x))))) (proc (λ () (F 1))))
   ```
   Tranformați codul dat pentru a folosi promisiuni.
   Este vreo diferență în execuție?
   Soluție:
   de două ori; `(let ((proc (λ (x) (* (force x) (force x))))) (proc (delay (F 1))))` se va evalua F o singură dată.

4. Implementați în Haskell o funcție care primește 3 liste infinite și determină maximul dintre ele la fiecare index. E.g. `f [1,6,2 ...] [8,3,4 ...] [5,10,3 ...] = [8,10,4 ...]`
   Soluție:
   ```
   f = zipWith3 (\x y z -> max x $ max y z) sau
   f (h1:t1) (h2:t2) (h3:t3) = max h1 (max h2 h3) :  f t1 t2 t3
   ```

5. Se dă clasa Haskell: `class Addable t where add ::  (Num a) => t a -> t a -> t a`, cu semnificația că funcția din clasă realizează adunarea, element cu element, a elementelor din cele două valori date ca argumente.
   - Instanțiați clasa dată pentru tipul listăeste suficient să acoperiți cazul cu liste de lungimi egale;
   - Definiți un tip *pereche de elemente de tip* `a`*, în care ambele elemente ale perechii sunt de același tip*;
   - Instanțiați clasa dată pentru tipul definit.

   Soluție:
   ```
   instance Addable [] where add = zipWith (+)
   data P a = P a a deriving (Eq, Show)
   instance Addable P where add (P a b) (P c d) = P (a+c) (b+d)
   ```

6. Știind că *Tâlhar pe tâlhar nu fură* , și că $talhar(Ben)$ și $talhar(Don)$, demonstrați folosind rezoluția că *fură(Ben, Don)* este fals .
   Soluție:

$\forall x.\forall y.talhar(x) \wedge talhar(y) \rightarrow \neg fura(x, y)$
$\neg talhar(x) \vee \neg talhar(y) \vee fura(x, y)$
$+talhar(Ben)\{x \leftarrow Ben\} \Rightarrow \neg talhar(y) \vee fura(Ben, y)$
$+talhar(Don)\{y \leftarrow Don\} \Rightarrow fura(Ben, Don)$
$+\neg fura(Ben, Don) \Rightarrow$ clauza vidă

7. Implementați în Prolog un predicatul `p(L1, L2, L3)`, care primește în L3 o listă de perechi și pune în L1 elementele de pe prima poziție din fiecare pereche și în L2 elementele de pe a doua poziție din fiecare pereche (similar `unzip` din Haskell) . Exemplu: este adevărat `p([1, 2, 3], [a, b, c], [(1, a), (2, b), (3, c)])`.
   Soluție:
   ```
   unzip(A, B, C) :- findall(X, member((X, _), C), A), findall(X, member((_, X), C), B).
   ```

8. Implementați un algoritm Markov care primește un șir format din simboluri `0` și `1` și verifică dacă șirul începe și se termină cu simboluri diferite. Dacă simbolurile de la început și de la sfârșit sunt diferite, scrie la sfârșitul șirului simbolurile "`ok`", altfel nu schimbă șirul cu nimic. Încercați să scrieți algoritmul pentru un alfabet de bază A oarecare.
   Soluție:
   1. `StartEnd(); A` g_1, g_2
   2. $ag_1 \rightarrow g_1 bg_1$
   3. $bg_1 g_2 \rightarrow g_2 bg_1$
   4. $g_1 bg_1 \rightarrow g_1 b$
   5. $g_1 bg_2 \rightarrow g_1 okb$
   6. $b \rightarrow .$
   7. $\rightarrow a$

B