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

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)) (let ((b a)) (+ a b)))
   (let* ((a 1) (b a)) (+ a b))
   ```
   Soluție:
   Nu este nicio diferență; `let*` este același lucru cu câte un `let` imbricat pentru fiecare definiție.

3. Implementați în Racket funcția `f` care primește o listă și determină cel mai mare 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 g h l = map (g .  h) l`
   Soluție:
   ```
   f ::  a → b → c → d
   map ::  (e → i) → [e] → [i]
   l ::  c ⇒ c = [e]
   (.)  ::  (t2 → t3) → (t1 → t2) → (t1 → t3)
   a = g1 → g2 = t2 → t3
   b = h1 → h2 = t1 → t2
   g .  h ::  e → i = t1 → t3
   c = [e] = [t1] = [h1]
   d = [i] = [t3] = [g2]
   f ::  (t2 → t3) → (t1 → t2) → [t1] → [t3]
   ```

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 listă Haskell.
   Soluție:
   ```
   class Ended t where frontEnd ::  t v -> v; backEnd ::  t v -> v
   instance Ended  []  where

    frontEnd = head
    backEnd = last sau head .  reverse
   ```

6. Știind că *Cine spune multe, spune mai puțin decât cine tace* , și că $spune\_multe(Ion)$ și $tace(Marcu)$, demonstrați folosind rezoluția că $spune\_mai\_putin(Ion, Marcu)$ este adevărat .
   Soluție:

$\forall x.\forall y.spune\_multe(x) \wedge tace(y) \rightarrow spune\_mai\_putine(x, y)$
$\neg spune\_multe(x) \vee \neg tace(y) \vee spune\_mai\_putine(x, y)$ (1)
$spune\_multe(Ion)$ (2)
$tace(Marcu)$ (3)
$\neg spune\_mai\_putine(Ion, Marcu)$ (4)
(1) + (2) $\{x \leftarrow Ion\} \Rightarrow \neg tace(y) \vee spune\_mai\_putine(Ion, y)$
+ (3) $\{y \leftarrow Marcu\} \Rightarrow spune\_mai\_putine(Ion, Marcu)$
+ (4) $\Rightarrow$ clauza vidă

7. Implementați în Prolog predicatul `x(L, A, B, N)` care detemină, pentru o listă `L`, numărul `N` de elemente care 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 adună 1 la acest număr. Exemple: 0 + 1 = 1; 1 + 1 = 10; 11 + 1 = 100; 100 + 1 = 101.
Soluție:
```
ag → ga
a → b
0b → 1c
1b → b0
b → 1c
c → .
→ a
```