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: $((\lambda x.\lambda y.\lambda z.(x\ \ y)\ \ \lambda x.y)\ \ a)$
   Soluție:
   $((\lambda \underline{x}.\lambda y.\lambda z.(\underline{x}\ \ y)\ \ \lambda x.y)\ \ a) \rightarrow_\alpha ((\lambda \underline{x}.\lambda w.\lambda z.(\underline{x}\ \ w)\ \ \lambda x.y)\ \ a) \rightarrow_\beta (\lambda \underline{w}.\lambda z.(\lambda x.y\ \ \underline{w})\ \ a) \rightarrow_\beta \lambda z.(\lambda x.y\ \ a) \rightarrow_\beta \lambda z.y$
   15p

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 ((c 2)) (let ((a 1) (b a)) (+ a b)))
   (define a 2) (let* ((c 2) (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 înca `a=2`; prima linie dă 3, a două dă 2.
   15p

3. Implementați în Racket funcția `f` care primește o listă și determină elementul cu cel mai mare modul. Folosiți, în mod obligatoriu, cel puțin o funcțională.
   Soluție:
   ```
   (car (filter (λ(e) (null? (filter (compose ((curry <) (abs e)) abs) L))) L))
   ```
   sau
   ```
   (car (filter (λ(e) (null? (filter (λ(a) (< (abs e) (abs a))) L))) L))
   ```
   sau
   ```
   (let ((M (last (sort (map abs L) <)))) (if (member M L) M (- 0 M)))
   ```
   15p

4. Sintetizați tipul funcției `f` (în Haskell): `f x y z g = map g [x, y, z]`
   Soluție:
   ```
   f :: a → b → c → d → e
   d = g1 → g2
   map :: (t1 → t2) → [t1] → [t2]
   a = b = c
   ```
   (parte din aceeași listă)
   ```
   t1 = a = b = c
   e = [t2]
   f :: t1 → t1 → t1 → (t1 → t2) → [t2]
   ```
   15p

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 NestedL a = A a | L [NestedL a]`

Soluție:

```
class Ended t where frontEnd ::  t v -> v; backEnd ::  t v -> v
instance Ended ][Pair][NestedL][Triple] where
   frontEnd (A a) = a; frontEnd (L l) = frontEnd $ head l
  backEnd (A a) = a; backEnd (L l) = backEnd $ last l
```

15p

6. Știind că *Un bogat când moare, săracul fluieră* , și că *bogat*(*Bill*), *sarac*(*Bob*), și *moare*(*Bill*) , demonstrați folosind rezoluția că *fluieră*(*Bob*) este adevărat .
   Soluție:
   $\forall x.\forall y.bogat(x) \wedge sarac(y) \wedge moare(x) \rightarrow fluiera(y)$
   $\neg bogat(x) \vee \neg sarac(y) \vee \neg moare(x) \vee fluiera(y)$
   $+bogat(Bill)\{x \leftarrow Bill\} \Rightarrow$
   $\neg sarac(y) \vee \neg moare(Bill) \vee fluiera(y)$
   $+\neg fluiera(Bob)\{\Rightarrow \neg sarac(Bob) \vee \neg moare(Bill)$
   $+moare(Bill) \Rightarrow \neg sarac(Bob)$
   $+sarac(Bob) \Rightarrow$ clauza vidă

   15p

7. Implementați în Prolog predicatul `x(L, M)` care detemină, pentru o listă `L`, `M`, maximul listei. Nu folosiți recursivitate explicită.
   Soluție:

   ```
   x(L, M) :- member(M, L), forall(member(E, L), X > E).
   ```

   15p

8. Implementați un algorim Markov care primește în șirul de intrare un număr binar și adună 2 la acest număr. Exemple: $1 + 10 = 10$; $10 + 10 = 100$; $1000 + 10 = 1010$; $101 + 10 = 111$; $111 + 10 = 1001$
   Soluție:

   ```
   ag → ga
   ga → bg
   0b → 1c
   1b → b0
   b → 1c
   c → .
   → a
   ```

   15p

C