Examen PP – Seria 2CC

11.06.2016

ATENȚIE: Aveți 2 ore . 10p per subiect . 100p necesare pentru nota maximă . **Justificați** răspunsurile!

1. Ilustrați cele două posibile secvențe de reducere pentru expresia: $(\lambda y.(\lambda x.\lambda y.x\ \ y)\ \ 2)$
   Soluție:
   - $(\lambda \underline{y}.(\lambda x.\lambda y.x\ \ \underline{y})\ \ 2) \xrightarrow{\textit{stanga-dreapta}}_\beta\ (\lambda \underline{x}.\lambda y.\underline{x}\ \ 2) \rightarrow_\beta\ \lambda y.2$
   - $(\lambda y.(\lambda x.\lambda \underline{y}.x\ \ y)\ \ 2) \rightarrow_\alpha\ (\lambda y.(\lambda \underline{x}.\lambda z.\underline{x}\ \ y)\ \ 2) \xrightarrow{\textit{dreapta-stanga}}_\beta\ (\lambda y.\lambda z.y\ \ 2) \rightarrow_\beta\ \lambda z.2$

2. Implementați în Racket o funcție `myAndMap` care să aibă un comportament similar cu `andmap` – primește o listă și întoarce o valoare booleană egală cu rezultatul operației `and` pe elementele listei. Folosiți cel puțin o funcțională. Nu folosiți `andmap`.
   Soluție:
   `(define (myAndMap L) (foldl (λ (x y) (and x y)) #t L))` (am acceptat și `foldl/r` direct cu `and`, soluție cu `filter`, etc)

3. Ce întoarce următoarea expresie în Racket? Justificați!
```
(let ((n 2))
  (letrec ((f (lambda (n)
    (if (zero? n) 1 (* n (f (- n 1)))))))
  (f 5))
)
```
   Soluție:
   Este factorial. $5! = 120$. `n` din let nu are niciun efect pentru că în cod se folosește `n` legat de `lambda`.

4. Cum se poate îmbunătăți următorul cod Racket pentru ca funcția `calcul-complex` să se evalueze doar atunci când este necesar, adică doar atunci când `variant` este fals (fără a o muta apelul lui `calcul-complex` în interiorul lui `calcul`) ?
```
1. (define (calcul x y z) (if x y z))
2. (define (test variant) (calcul variant 2 (calcul-complex 3)))
```
   Soluție:
```
1. (define (calcul x y z) (if x y (force z)))
2. (define (test variant) (calcul variant 2 (delay (calcul-complex 3))))
```
   Se mai poate și folosind închidere lambda și `(z)`, `if` peste apelul lui `calcul-complex`, sau chiar `quote` și `eval`.

5. Sintetizați tipul funcției `f` în Haskell: `f x y = (y x) x`
   Soluție:
```
f :: a → b →c
y :: d → e
b = d → e (y este argumentul lui f)
d = a (y ia ca argument pe x)
e = a → c (valoarea întoarsă de y)
⇒ f :: a -> (a -> a -> c) -> c
```

6. Instanțiați în Haskell clasa `Eq` pentru tripluri, considerând că `(a1, a2, a3)` este egal cu `(b1, b2, b3)` dacă `a1 == b1` și `a2 == b2`.
   Soluție:
```
instance (Eq a, Eq b) => Eq (a, b, c) where (a1, a2, _) == (b1, b2, _) = (a1 == b1)
&& (a2 == b2)
```

7. Implementați în Haskell, fără a utiliza recursivitate explicită, funcția `setD` care realizează diferența a două mulțimi `a` și `b` (`a` \ `b`) date ca liste (fără duplicate). Care este tipul funcției?
Soluție:
`setD a b = [x | x <- a, not $ elem x b]`
sau
`setD a b = filter (not . (flip elem) b) a`
`setD :: Eq t => [t] -> [t] -> [t]`

8. Traduceți în logica cu predicate de ordinul întâi propoziția: *Orice naș își are nașul.*
Soluție:
$\forall x \forall y.nas(x, y) \Rightarrow \exists z.nas(z, x)$
sau
$\forall x.(\exists y.nas(x, y)) \Rightarrow \exists z.nas(z, x)$

9. Știind că $\forall x.Trezit(x, Dimineata) \Rightarrow \forall y.AjungeLa(x, y)$ și că $Trezit(Eu, Dimineata)$, demonstrați, folosind **metoda rezoluției**, că $AjungeLa(Eu, Examen)$.
Soluție:
FNC:
$\neg Trezit(x, Dimineata) \vee Ajunge(x, y)$ (1)
$Trezit(Eu, Dimineata)$ (2)
$\neg AjungeLa(Eu, Examen)$ (3) (negarea concluziei)
Rezoluție:
(1) rezolvă cu (2), cu rezolventul $Trezit(Eu, Dimineata)$, sub substituția $x \leftarrow Eu$ obținem clauza $AjungeLa(Eu, y)$ (4)
(3) rezolvă cu (4), sub substituția $y \leftarrow Examen$, rezultă clauza vidă.

10. Care este efectul aplicării predicatului `p` asupra listelor `L1` și `L2` (la ce este legat argumentul `R` în apelul `p(L1, L2, R)` ?):
`p(A, [], A). p(A, [E|T], [E|R]) :- p(A, T, R).`
Soluție:
`R = L2 ++ L1`

11. Implementați un algoritm Markov care primește un șir de simboluri `0` și `1` și verifică dacă șirul începe cu `0` și se termină cu `1` și, în caz afirmativ, adaugă la sfârșitul șirului simbolurile "`ok`", altfel nu schimbă șirul cu nimic. Exemple: `010111011` → `010111011ok` ; `010` → `010` ; `1010` → `1010`
Soluție:
    1. `Check(); {0, 1}` g_1
    2. `a0`→`0b`
    3. `b`g_1→g_1`b`
    4. `1b`→`1okb`
    5. `b`→`.`
    6. `a`→`.`
    7. →`a`

12. Explicați care dintre următoarele apeluri dă eroare și care nu, și justificați pentru fiecare:
1. `(if #t 5 (/ 2 0))` (Racket)
2. `(let ((f (λ (x y) x))) (f 5 (/ 2 0)))` (Racket)
3. `let f x y = x in f 5 (div 2 0)` (Haskell)
4. `X = 2 / 0, Y = X.` (Prolog)
Soluție:
1. Nu este eroare → `if` este funcție nestrictă.

2. Eroare, din cauza evaluării aplicative.
3. Nu este eroare, datorită evaluării leneșe (`y` nu este folosit).
4. Nu este eroare, `=` nu evluează calculele aritmetice.