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: $(\lambda x.(\lambda y.\lambda x.y\ \ x)\ \ 5)$
   Soluție:
   - $(\lambda \underline{x}.(\lambda y.\lambda x.y\ \ \underline{x})\ \ 5) \xrightarrow{stanga\text{-}dreapta}_{\beta} (\lambda \underline{y}.\lambda x.\underline{y}\ \ 5) \rightarrow_{\beta} \lambda x.5$
   - $(\lambda x.(\lambda y.\underline{\lambda x}.y\ \ x)\ \ 5) \rightarrow_{\alpha} (\lambda x.(\lambda \underline{y}.\lambda z.\underline{y}\ \ x)\ \ 5) \xrightarrow{dreapta-stanga}_{\beta} (\lambda x.\lambda z.x\ \ 5) \rightarrow_{\beta} \lambda x.5$

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

3. Ce întoarce următoarea expresie în Racket? Justificați!
```
(letrec ((f (lambda (n)
  (let ((n (- n 1)))
    (if (eq?  n -1) 1 (* (+ n 1) (f n)))))))
  (f 5))
)
```
   Soluție:
   Este factorial. $5! = 120$.

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 = x (y x)`
   Soluție:
```
f ::  a → b → c
x ::  d → e
y ::  g → h
d = h (x ia valoarea întoarsă de y)
e = c (f întoarce valoarea întoarsă de x)
a = g = d → e (y și f iau ca argument pe x)
b = g → h (tipul lui x în f)
⇒ b = (d → c) → d; a = d → c
⇒ f ::  (d → c) → ((d → c) → d) → c
```

6. Instanțiați în Haskell clasa `Ord` pentru tripluri (știind că `Eq` este deja instanțiată), considerând că `(a1, a2, a3)` este mai mic decât `(b1, b2, b3)` dacă `a1 < b1`.
   Soluție:
```
instance Ord a => Ord (a, b, c) where (a1, _, _) < (b1, _, _) = a1 < b1
```

7. Implementați în Haskell, fără a utiliza recursivitate explicită, funcția `setN` care realizează intersecția a două mulțimi `a` și `b` date ca liste (fără duplicate). Care este tipul funcției?
Soluție:
```
setN a b = [x | x <- a, elem x b]
```
sau
```
setN a b = filter ((flip elem) b) a
setN ::  Eq t => [t] -> [t] -> [t]
```

8. Traduceți în logica cu predicate de ordinul întâi propoziția: *Orice copil are o mamă.*
Soluție:
$\forall x.copil(x) \Rightarrow \exists y.mama(y, x)$

9. Știind că $\forall x.Are(x, Carte) \Rightarrow \forall y.Are(x, y)$ și că $Are(Eu, Carte)$, demonstrați, folosind **metoda rezoluției**, că $Are(Eu, Parte)$.
Soluție:
FNC:
$\neg Are(x, Carte) \vee Are(x, y)$ (1)
$Are(Eu, Carte)$ (2)
$\neg Are(Eu, Parte)$ (3) (negarea concluziei)
Rezoluție:
(1) rezolvă cu (2), cu rezolventul $Are(Eu, Carte)$, sub substituția $x \leftarrow Eu$
obținem clauza $Are(Eu, y)$ (4)
(3) rezolvă cu (4), sub substituția $y \leftarrow Carte$, 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([E|T], A, [E|R]) :- p(T, A, R).
```
Soluție:
```
R = L1 ++ L2
```

11. Implementați un algoritm Markov care primește un șir de simboluri `0` și `1` și verifică dacă șirul începe cu `1` și se termină cu `0` și, în caz afirmativ, adaugă la sfârșitul șirului simbolurile "`ok`", altfel nu schimbă șirul cu nimic. Exemple: `1110100` → `1110100ok` ; `0101` → `0101` ; `010` → `010` ; `1010` → `1010ok`
Soluție:
1. `Check(); {0, 1}` g_1
2. `a1`→`1b`
3. `b`g_1→g_1`b`
4. `0b`→`0okb`
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 lenește (`y` nu este folosit).
4. Nu este eroare, `=` nu evluează calculele aritmetice.