Examen PP

31.05.2019

ATENȚIE: Aveți 2 ore · 1-9: 10p; 10: 30p · 100p pentru nota maximă · **Justificați** răspunsurile!

---

1. Reduceți expresia lambda $E = (\lambda x.(x\ \ (\lambda y.z\ \ x))\ \ \lambda x.x)$

2. Se dă următorul cod Racket:
```
(define computation (delay (+ 5 5)))
(* 5 5)
(define (f x) (cons x (force computation)))
(map f '(1 2 3 4))
```
   (a) De câte ori se realizează adunarea?
   (b) Prima evaluare a adunării se realizează înainte sau după înmulțire?
   (c) Rescrieți codul pentru `computation` și pentru `f` folosind închideri funcționale în loc de promisiuni și răspundeți din nou la întrebările de la (a) și (b).

3. Date fiind două liste de numere `L1` și `L2`, scrieți în Racket codul care produce o listă de perechi `(x . n)`, unde `x` este un element din `L1`, iar `n` este numărul de apariții ale lui `x` în `L2`. E.g. pentru `L1 = (1 4 5 3)` și `L2 = (1 3 2 4 1 5 3 9)` rezultatul este `((1 . 2) (4 . 1) (5 . 1) (3 . 2))`. Nu folosiți recursivitate explicită.

4. Sintetizați tipul următoarei funcții în Haskell: `f x y = x y (y x)`

5. (a) Câți pași de concatenare sunt realizați pentru evaluarea expresiei Racket `(car (append '(1 2) '(3 4)))` ?
   (b) Dar pentru expresia Haskell `head $ [1, 2] ++ [3, 4]` ?

6. Evidențiați o posibilă instanță a clasei Haskell de mai jos:
```
class MyClass c where
  f :: c a -> a
```

7. Transformați propoziția „Nu tot ce zboară se mănâncă.” în logică cu predicate de ordinul întâi.

8. Se dă programul Prolog:
```
p(R, S) :- member(X, R),
        findall(Y, (member(Y, R), Y \= X), T), !, q(X, T, S).
q(X, A, [X|A]). q(X, [A|B], [A|C]) :- q(X, B, C).
```
   Dacă predicatul `p` primește în primul argument o listă, la ce valori leagă al doilea argument? Câte soluții are interogarea `p([1, 2, 3, 4], S)` ?

9. Se dau următoarele relații genealogice prin predicatul `c(Parinte, Copil)`. Implementați predicatul `frati(X, F)`, care leagă `F` la lista de frați ai lui `X` (dacă există). De exemplu, pentru definițiile de mai jos, interogarea `frati(herodot, F)` leagă `F` la `[faramir, george]`.
```
c(alex, celia).  c(alex, delia).  c(alex, marcel).
c(barbra, celia).  c(barbra, delia).  c(barbra, marcel).
c(delia, faramir).  c(delia, george).  c(delia, herodot).
c(erus, faramir).  c(erus, george).  c(erus, herodot).
```

10. PROBLEMA (Poate fi implementată în orice limbaj studiat la PP.) Se urmărește implementarea unui *multi-map*, care este un tabel asociativ în care unei chei i se pot asocia oricâte valori.
   (a) Descrieți reprezentarea *multi-map*-ului. Pentru Haskell, dați definiția tipului de date polimorfic. Definiți funcția/predicatul `lookup'`, care extrage lista tuturor valorilor asociate cu o cheie.
   (b) Definiți funcția/predicatul `insert'`, pentru adăugarea unei noi asocieri între o cheie și o valoare.
   (c) Definiți funcția/predicatul `map'`, care aplică o funcție/predicat pe fiecare valoare din *multi-map*. NOTĂ: în Prolog, `map'` va aplica întotdeauna un același predicat `p(+VIn, -VOut)`.