Examen PP – Seria 2CC

28.05.2013

NOTĂ: Fiecare subiect valorează 10 puncte. Este suficientă rezolvarea completă a 10 subiecte pentru nota maximă. Timpul de lucru este de 2 ore. Examenul este open-book.

---

1. Reduceți expresia E la forma normală:
   $E \equiv ((\lambda x.\lambda y.\lambda x.x\ \ (\lambda x.(x\ \ x)\ \ \lambda x.(x\ \ x)))\ \ y)$

   Soluție:
   $\lambda x.x$

2. Implementați funcția `(oddStarts L)` în Scheme, care pentru o listă de liste întoarce câte dintre listele componente încep cu un număr impar. Utilizați **funcționale** și nu utilizați recursivitate explicită – soluțiile care nu respectă cele două constrângeri **nu** sunt punctate.
   Exemplu: `(oddStarts '((2 3 4) (1 2 3) (5 6) (8 9)))` → `2`

   Soluție:
   ```
   (define (oddStarts L) (length (filter odd?  (map car L))))
   ```

3. Ce întoarce următoarea expresie în Scheme? Justificați!
   ```
    (apply
     (lambda (x y) (let ([x 1] [y 2]) (+ x y)))
     (let ([x 2] [y 3]) (list x y))
   )
   ```

   Soluție:
   Expresia întoarce 3, parametri funcției nu sunt utilizați.

4. Ce afișează următorul cod:
   ```
   1.  (define f (delay (lambda (a) (display a) (newline))))
   2.  (define ff (force f))
   3.  (and (ff 1) (ff 2) #f)
   ```

   Soluție:
   ```
   1
   2
   #f
   ```

5. Câte dintre cele trei adunări se vor realiza în evaluarea expresiei Haskell de mai jos? Justificați!
   ```
   let selector x y z = x in selector (1 + 2) (2 + 3) (3 + 4)
   ```

   Soluție:
   Una – (1 + 2), pentru că evaluarea este leneșă și parametrii `y` și `z` nu sunt evaluați.

6. Sintetizați tipul funcției Haskell de mai jos. Justificați!
   ```
   f x y = (x y) y
   ```

   Soluție:
   ```
   f ::  a → b → c
   ```

```
x :: d → e
d = b
a = d → e
e = b → c
f :: (b → b → c) → b → c
```

7. Supraîncărcați în Haskell ordonarea funcțiilor care au ca parametru un număr, ordonând funcțiile după valoarea lor pentru argumentul `0`. Se consideră că operatorul de egalitate între astfel de funcții a fost deja definit.

   Soluție:
   ```
   instance (Num a, Ord b) => Ord (a -> b) where f > g = f 0 > g 0
   ```

8. Traduceți în logica cu predicate de ordinul I următoarea propoziție: *Pentru fiecare om există un moment în care își cunoaște perechea.*

   Soluție:
   $\forall x.(om(x) \Rightarrow \exists t.\exists p.pereche(p, x) \wedge cunoaste(x, p, t))$

9. Scrieți predicatul `filter(+L, +T, -LF)` în Prolog care ia o listă și o filtrează, întorcând în `LF` doar acele elemente mai mari (strict) decât pragul `T`, fără a utiliza recursivitate explicită. Exemplu: `filter([1,5,9,3,2,6,3,8], 5, LF)` leagă `LF` la `[9, 6, 8]`.

   Soluție:
   ```
   filter(L, T, LF) :- findall(X, (member(X, L), X > T), LF).
   ```

10. Realizați un predicat Prolog care elimină duplicatele dintr-o listă.

    Soluție:
    ```
    nodups([], []).
    nodups([H|T], [H|LO]) :- findall(X, (member(X, T), X \= H), T1),
    nodups(T1, LO).
    ```
    sau
    ```
    rem(_, [], []).
    rem(X, [X|L], LO) :- rem(X, L, LO), !.
    rem(X, [H|L], [H|LO]) :- rem(X, L, LO).
    nodups([], []).
    nodups([H|T], [H|LO]) :- rem(H, T, T1), nodups(T1, LO).
    ```

11. Scrieți un algoritm Markov care lucrează pe un șir de simboluri `0` și `1` și înlocuiește toate aparițiile de pe poziții pare ale simbolului `1` cu simbolul `0` (prima poziție din șir este impară). De exemplu, pentru șirul `010010101101` se obține `000010101000`.

    Soluție:

```
Replace(); Ab g1, g2
  ag11 -> g10a
  ag1g2-> g1g2a
  a -> .
  -> a
```

12. Scrieți un program CLIPS care concatenează două liste aflate în memorie ca fapte `listA`, respectiv `listB`, de exemplu `(listA a b c)` și `(listB d e f)`, marcând elementele provenind din a doua listă prin precedarea cu simbolul `B`, și lasă în memorie faptul corespunzător listei rezultat, de exemplu `(list a b c B d B e B f)`.

    Soluție:

```
(deffacts facts
  (listA a b c)
  (listB d e f)
)

(defrule create
  (listA $?pre)
  =>
  (assert (list $?pre))
)

(defrule append
  ?f <- (list $?pre)
  ?g <- (listB ?x $?post)
  =>
  (retract ?f)
  (retract ?g)
  (assert (list $?pre B ?x))
  (assert (listB $?post))
  )
```