Examen PP – Seria 2CC

31.05.2014

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. Pentru puctarea răspunsurilor este necesară **justificarea** acestora.

---

1. Reduceti expresia E la forma normală: $E \equiv ((\lambda y.(\lambda x.\lambda y.x\ \ y)\ \ \lambda x.x)\ \ \Omega)$
   Soluție:
   $((\lambda y.(\lambda x.\lambda y.x\ \ y)\ \ \lambda x.x)\ \ \Omega) \to ((\lambda x.\lambda y.x\ \ \lambda x.x)\ \ \Omega) \to (\lambda y.\lambda x.x\ \ \Omega) \to \lambda x.x$

2. Implementați o funcție în Racket care ia o listă de numere L1 ca prim parametru și o listă de liste L2 ca al doilea parametru și întoarce acele liste din L2 ale căror lungimi se regăsesc în L1. Utilizați funcționale și nu utilizați recursivitate explicită – soluțiile care nu respectă cele două constrângeri nu vor fi punctate.
   Exemplu: `(f '(1 2 3 4) '((4 5 6) () (a b)))` → `'((4 5 6) (a b))`
   Soluție:
   ```
   (λ (L1 L2) (filter (λ (l) (member (length l) L1)) L2))
   ```

3. La ce se evaluează următoarea expresie în Racket? (cu `()` ≡ `[]`)
   ```
   (let* [(x 1) (y 2) (f (delay (λ (y) (+ x y))))] (let [(x 5)] ((force f) x)))
   ```
   Soluție:
   6. x=5 + x=1.

4. Scrieți în Haskell o funcție care realizează produsul cartezian a două mulțimi oarecare (liste) `A` și `B`. Utilizați facilitățile oferite de limbaj. Care este tipul funcției create?
   Soluție:
   ```
   cart a b = [(x, y) | x <- a, y <- b]
   cart ::  [t] -> [t1] -> [(t, t1)]
   ```

5. Construiți în Haskell fluxul puterilor lui 2.
   Soluție:
   ```
   fluxus = 1 :  map (* 2) fluxus
   ```

6. Sintetizați tipul funcției f în Haskell:
   ```
   f x = [(y, y) | (y, z, t) <- x, y == z]
   ```
   Soluție:
   ```
   f ::  Eq t1 => [(t1, t1, t)] -> [(t1, t1)]
   ```

7. Supraîncărcați în Haskell afișarea funcțiilor care au ca parametru un număr, afișându-se valoarea funcției în punctul 0.
   Soluție:
   ```
   instance (Show b, Num a) => Show (a -> b) where show f = show (f 0)
   ```

8. Traduceți în logica cu predicate de ordinul I următoarea propoziție:
   Cine tace e mai înțelept decât cine vorbește.
   Soluție:
   $\forall x.\forall y.(tace(x) \wedge vorbeste(y)) \Rightarrow mai\_intelept(x, y)$

9. Folosiți rezoluția pentru a demonstra că, din moment ce toți oamenii sunt muritori, Socrate, om el însuși, este muritor. Folosiți predicatele *om* și *muritor*.

Soluție:
Premise: $\forall x.om(x) \Rightarrow muritor(x)$; $om(Socrate)$
Concluzia: $muritor(Socrate)$ (negată: $\neg muritor(Socrate)$)
în FNC: $\{\neg om(x), muritor(x)\}, \{om(Socrate)\}, \{\neg muritor(Socrate)\}$
Pas de rezoluție cu rezolvent $om(x)\{Socrate/x\} \rightarrow \{muritor(Socrate)\}$
Pas de rezoluție cu rezolvent $muritor(Socrate) \rightarrow \{\}$

10. Ce rezultat are în Prolog evaluarea lui `p(L, X)`, cu `L` o listă și `X` nelegat:
```
r([], _).
r([H|T], X) :- member(H, X), r(T, X).
p(L, X) :- length(L, N), length(X, N), r(L, X).
```

Soluție:
Lista X are aceeași lungime ca și L și aceeași membri, în orice ordine (diversele soluții pentru X sunt permutările listei L).

11. Scrieți un predicat Prolog `up` (și eventual predicate ajutătoare) care identifică secvențele (cel puțin două elemente) strict crescătoare dintr-o listă. Exemplu:
`up([5, 1, 2, 3, 2, 3, 1, 1, 0, 9, 10], LS)` → `LS = [1, 2, 3, 2, 3, 0, 9, 10]`

Soluție:
```
up([], []).
up([H, H1 | T], [H, H1 | LS]) :- H1 > H, !, up(T, H1, LS).
up([_ | T], LS) :- up(T, LS).
up([], _, []) :- !.
up([H | T], E, [H | LS]) :- H > E, !, up(T, H, LS).
up(L, _, LS) :- up(L, LS).
```

12. O transmisiune de date transmite câte doi biți de date urmați de un bit de control (suma modulo 2 a celor doi biți transmiși anterior). Scrieți un algoritm Markov care identifică secvențele eronate (bit de control greșit) și le marchează cu trei de `x`.
Exemplu: 101010110000111 → 101xxx110000xxx
Soluție:
a000 →000a
a011 →011a
a101 →101a
a110 →110a
$ag_1g_2g_3$→xxxa
a →.
→a

A