Examen PP – Seria 2CC — NOT EXAM MODE

29.05.2018

ATENȚIE: Aveți 2 ore · 100p pentru nota maximă · **Justificați** răspunsurile!

---

1. Reduceți la forma normală următoarea expresie, ilustrând pașii de reducere:
   $\lambda x.\lambda y.((\lambda x.\lambda y.x\ \ (y\ \ x))\ \ (x\ \ y))$
   Soluție:
   $\lambda x.\lambda y.((\lambda x.\lambda y.x \quad (y \quad x)) \quad (x \quad y)) \rightarrow_\alpha \lambda x.\lambda y.((\lambda x.\lambda z.x \quad (y \quad x)) \quad (x \quad y)) \rightarrow_\beta \lambda x.\lambda y.(\lambda z.(y\ \ x)\ \ (x\ \ y)) \rightarrow_\beta \lambda x.\lambda y.(y\ \ x)$

2. Care este diferența între următoarele două linii de cod Racket
   ```
   (let ((a 1) (b 2)) (let ((b 3) (c (+ b 2))) (+ a b c)))
   (let* ((a 1) (b 2)) (let* ((b 3) (c (+ b 2))) (+ a b c)))
   ```
   Soluție:
   În prima definiția `(b 3)` nu este vizibilă în legarea lui `c`; rezultatul este 8, iar în a doua linie rezultatul este 9.

3. Scrieți în Racket o funcție echivalentă cu `zip` din Haskell, știind că
   `zip :: [a] -> [b] -> [(a, b)]`. Folosiți cel puțin o funcțională.
   Soluție:
   ```
   (define (zip L1 L2) (map cons L1 L2))
   ```

4. Sintetizați tipul funcției `f` în Haskell: `f x y z = x y . z`
   Soluție:
   `y :: t`
   `z :: a → b`
   `x y :: b → c`
   `x :: t → b → c`
   `f x y z :: a → c`
   `f :: (t → b → c) → t → (a → b) → a →c`

5. Instanțiați clasa `Show` pentru funcții Haskell care iau un argument numeric, astfel încât afișarea unei funcții `f` va produce afișarea rezultatelor aplicării funcției pe numerele de la 1 la 10. E.g. afișarea lui `(+1)` va produce: `234567891011`.
   Soluție:
   ```
   -# LANGUAGE FlexibleInstances #- -- nu este cerut în rezolvarea din examen
   instance (Enum a, Num a, Show b) => Show (a -> b) where
            show f = concatMap (show .  f) [1..10]
   -- Enum nu este cerut în rezolvarea din examen
   ```

6. Folosiți list comprehensions pentru aproduce fluxul listelor formate din primii 5 multipli ai fiecărui număr natural:
   `[[1,2,3,4,5],[2,4,6,8,10],[3,6,9,12,15],[4,8,12,16,20] ...] .`
   Soluție:
   ```
   [take 5 [m | m <- [n..], mod m n == 0] | n <- [1..]]
   ```

7. Folosiți rezoluția pentru a demonstra că dacă *Ion este om* și *orice om are o bicicletă* atunci este adevărat că *Ion are bicicletă sau Ion este bogat* (folosiți predicatele $om(X)$, $areBicicletă(X)$ și $bogat(X)$).
   Soluție:
   Avem premisele: $om(ion)$ și $\forall x.om(x) \Rightarrow areBicicletă(x)$
   Concluzia: $areBicicletă(ion) \vee bogat(ion)$

Clauzele:
(a) $\{om(ion)\}$
(b) $\{\neg om(x) \vee areBicicletă(x)\}$
(c) $\{\neg areBicicletă(ion)\}$ (prima parte a concluziei negate)
(d) $\{\neg bogat(ion)\}$ (a doua parte a concluziei negate)
(b) + (c) $\{x \leftarrow ion\} \rightarrow \neg om(ion)(e)$
(a) + (e) $\rightarrow$ clauza vidă

8. Scrieți un predicat Prolog `diff(A, B, R)` care leagă R la diferența mulțimilor (reprezentate ca liste) A și B.
Soluție:
```
intersect(A, B, R) :- findall(X, (member(X, A), member(X, B)), R).
```

9. Dat fiind un șir de date binare, scrieți un algoritm Markov care plasează la sfârșitul șirului suma modulo 2 a biților din șir. Exemple: `101010110000111` $\rightarrow$ `1010101100001110; 100110110110` $\rightarrow$ `1001101101101; 100110110111` $\rightarrow$ `1001101101110`
Soluție:
CheckSum; 0,1 g_1
$ag_1 0 \rightarrow 0ag_1$
$a01 \rightarrow 1a1$
$a11 \rightarrow 1a0$
$a \rightarrow .$
$\rightarrow a0$

10. Considerăm o structură de date de tip listă circulară, caracterizată de conținutul său și de un cursor intrinsec structurii, poziționat la orice moment pe un element al listei. Avem următoarele funcționalități:
    - Structura va putea fi creată pe baza unei liste obișnuite L; la creare cursorul va fi inițial poziționat pe elementul care era primul element din L;
    - Operația *get*, care întoarce elementul de la poziția unde este cursorul;
    - Operația *next*, care avansează cursorul cu o poziție spre dreapta;

    Exemplu: avem lista circulară C, construită pe baza listei 1,2,3,1,5. Astfel:
    $get(C) = 1$ $\quad$ $get(next(next(next(next(C))))) = 5$
    $get(next(C)) = 2$ $\quad$ $get(next(next(next(next(next(C)))))) = 1$
    Se cere implementarea în Racket, Haskell **sau** Prolog a celor 3 funcționalități: crearea listei circulare, operația *get* și operația *next*.
    Soluție:
    Exemplu în Haskell:
```
circular l = concat .  repeat $ l -- desfășurăm lista originală
get = head -- elementul curent
next = tail -- avansăm cursorul
```

A