Lab 3

16 March 2023 16:33

FUNCȚII CA VALORI DE ORD. I

Funcțiile pot fi:

→ valori ale unor variabile (ex: (define id (lambda (x) x)))

→ argumente pt. alte funcții

→ întoarse de o funcție

→ membri ai unor structuri (ex: (list 1 2 even? max))

FCȚII CURRY/UNCURRY

① FCȚII UNCURRIED

→ primește toți parametrii

ex: (define (add x y) (+ x y))

primește ambii parametrii deodată

② FCȚII CURRY

→ primește parametrii pe rând

(define (add x) (λ (y) (+ x y)))

primește parametrul (x)

întoarce o funcție care primește parametrul (y)

apelare: ((add 1) 2) → parametrii sunt pasați pe rând

- TRANSFORMARE [curry → uncurry]

((f 2) 3) → ((curry f) 2 3)

forma curry — forma uncurry

- TRANSFORMARE [uncurry → curry]

(f 2 3) → (((uncurry f) 2) 3)

forma uncurry — forma curry

FUNCȚIONALE

FUNCȚIONALE

= funcții care primesc ca parametru o funcție

① MAP

(map f L_1 ... L_n)

↓ are n parametri

⇒ întoarce o listă formată din aplicările lui f pe elementele listelor

Ex:
- map f [1, 2, 3] ⇒ [f 1, f 2, f 3]
- map f [1, 2, 3] [4, 5, 6] ⇒ [f 1 4, f 2 5, f 3 6]

② FILTER

(filter predicat L)

→ returnează o listă cu toate el. din L care îndeplinesc predicatul

Ex:
- (filter negative? '(-2 3 5 -4 2)) → '(-2 -4)
- (filter (λ (x) (> x 5)) '(1 5 7 9)) ⇒ '(7 9)

! filter elimină din listă doar elementele pentru care funcția dată ca parametru întoarce #f

Ex: (filter (λ (x) x) '(1 #t 3 #f 5)) ⇒ '(1 #t 3 5)

③ FOLDL = fold left

(foldl f acc L_1 L_2 ... L_n)

Ordinea parcurgerii: stga → dr

f - ia ca parametrii
- param 1 = (car L_1)
- param 2 = (car L_2)
- ...
- param n = (car L_n)
- param n+1 = acc

→ aplică f pe câte element din listă / liste și pe acumulator

foldl f acc [] = acc

foldl f acc [h : L] = foldl f (f h acc) L

⇒ rec. pe coadă

foldl cons '() '(1 2 3)

cons 1 '() → '(1)

cons 2 '(1) → '(2 1)

cons 3 '(2 1) → '(3 2 1)

rezultat inversat pt. rezultat corect: REVERSE

- (foldl (λ (h acc) (+ h acc)) '(1 2 3)) ⇒ 6 (suma)
- '(2 5 7) → 257 : vreau să formez un nr. din elementele listei

acc: 0 → 2 + 0*10 = 2 → 5 + 2*10 = 25 → 7 + 25*10 = 257

(define (get-num L)

(foldl (λ (x acc) (+ x (* 10 acc))) 0 L)

! atenție la ordine

④ FOLDR = fold right

(foldr f acc $L_1 \dots L_n$)

Ordine parcurgere: dr → stg

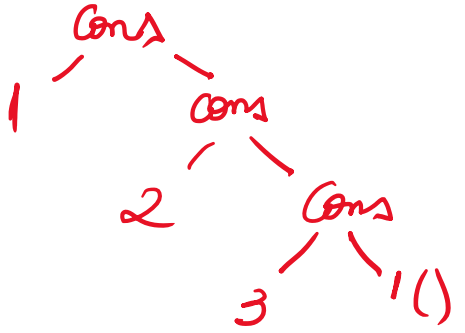
f - asem cu f de la foldl

$$\begin{cases} \text{foldr } f \text{ acc } [] = \text{acc} \\ \text{foldr } f \text{ acc } [x : L] = (f\ x\ (\text{foldr } f \text{ acc } L)) \end{cases}$$

⇓ rec - pe stivă

Ex:

- (foldr (λ (x acc) (cons x acc)) '() '(1 2 3))



'(1 2 3)
'(2 3)
'(3)

- (foldr + 0 '(1 2 3)) → 6 (suma)
- my-filter : <=> filter din Racket

(define (my-filter f L)
(foldr (λ (x acc)
(if (f x)
(cons x acc)
acc))

'( ) L)

⑤ APPLY

(apply f $x_1 \dots x_n$ [$y_1 \dots y_m$])

⇔ (f $x_1 \dots x_n$ $y_1 \dots y_m$)

optionali

Ex:

- (apply + 5 '(2 3 1)) ⇔ (+ 5 2 3 1)
- (apply cons '(1 2)) ⇔ (cons 1 2) → '(1 . 2)

  '(1 2 3) ⇔ (cons 1 2 3) → eroare
- (apply list '(2 3 4) '(1 2 3))

  ⇔ (list '(2 3 4) 1 2 3)

  ⇒ '((2 3 4) 1 2 3)

- transpose

'((1 2 3) (4 5 6) (7 8 9)) → '((1 4 7) (2 5 8) (3 6 9))

(1 2 3) (4 5 6) (7 8 9)

map list (1 2 3) (4 5 6) (7 8 9)

```
(define (transpose L)
  (apply map list L))
(define L '((1 2 3) (4 5 6) (7 8 9)))
```

(transpose L) → '((1 4 7) (2 5 8) (3 6 9))