Programare funcţională – Laboratorul 4

Recursivitate, Recursivitate finală

Isabela Drămnesc

March 17, 2014

1 Concepte

- Recursivitate
- Recursivitate finala
- Tehnica variabilei colectoare
- Definire de funcţii final recursive
- Operaţii asupra listelor la nivel superficial
- Operaţii asupra listelor la orice nivel
- Operaţii cu mulţimi
- Utilizare (time <expresie>)
- Utilizare (trace <expresie>)

2 Întrebări din Laboratorul 3

- Daţi exemple pentru a evidenţia diferenţa dintre variabile locale şi variabile globale în Racket.
- Ce este recursivitatea? Daţi cel puţin 2 exemple de funcţii recursive (scrieţi definiţia în Racket).
- Cât de importantă este ordinea declarării clauzelor în definirea unei funcţii recursive în Racket?
- Scrieţi câte o funcţie recursivă în Racket pentru fiecare din următoarele:
  1. Concatenarea a doua liste;
  2. Inversa unei liste;
  3. Lungimea unei liste.

3 Recursivitate finală. Metoda variabilei colectoare

O funcţie este *liniar recursivă* dacă se apelează o singură dată pe ea insăşi pentru a întoarce rezultatul.

O funcţie este *"gras"/"exploziv" recursivă* dacă se apelează de mai multe ori pe ea insăşi pentru a întoarce rezultatul.

O funcţie recursivă este *final-recursivă* dacă:

- apelurile recursive nu sunt argumente pentru alte funcţii şi nu sunt utilizate ca şi teste;
- dacă valoarea obţinută pe ultimul nivel de recursivitate rămâne neschimbată până la revenirea pe nivelul de sus;
- la ultima copie creată se obţine rezulatul, rezultat ce rămâne neschimbat la revenire;

Exemplele cu definirea de funcţii recursive de până acum reprezintă funcţii nefinal-recursive. La astfel de funcţii nefinal recursive apelul recursiv este conţinut într-un apel de funcţie (+, -, cons, append etc).

Recursivitatea grasă este foarte ineficientă, recursivitatea finală este cea mai eficientă.

Pentru transformarea unei funcţii recursive într-o funcţie final-recursivă se foloseşte tehnica variabilelor colectoare.

3.1 Factorial

```
;testare functii definite in lab3
; factorial din lab 3 (fara acumulatori)
(require racket/trace)

(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))
      )
  )
(trace fact)

> (fact 5)

> (untrace fact)

>(time (fact 1000))
```

```
; tehnica variabilei colectoare ;;

(define (factf n)
  (fact-aux n 1)
 )

(define (fact-aux n rez)
  (cond ((= n 0) rez)
        (#t (fact-aux (- n 1) (* n rez)))
   )
  )
(trace fact-aux)

 >(fact-aux 5 1)

> (untrace fact-aux)

> (factf 10)

> (factf 100)

> (factf 10000)

> (time (factf 1000))
```

3.2 Fibonacci

```
; alt exemplu din laboratorul 3
; fibonacci fara acumulatori

(define (fibonacci n)
  (cond ((= n 0) 1)
        ((= n 1) 1)
        (#t (+ (fibonacci (- n 1))
               (fibonacci (- n 2))
           )
        )
  )
)
; >(time (fibonacci 300))

(trace fibonacci)

> (fibonacci 11)

> (untrace fibonacci)

;;;; Varianta final recursiva
(define (rfib n)
  (if (< n 1)
```

```
        1
        (fib-aux n 1 1)
    )
  )

(define (fib-aux n fn-1 fn-2)
  (if (= n 1)
        fn-1
        (fib-aux (- n 1) (+ fn-1 fn-2) fn-1)
    )
  )
(trace fib-aux)

> (fib-aux 5 1 1)
 > (rfib 2)

> (rfib 3)

> (rfib 4)

> (rfib 5)

> (rfib 11)

> (untrace rfib)

> (rfib 12)

> (untrace fib-aux)

> (rfib 11)

> (time (rfib 100))

> (time (rfib 1000))
```

3.3 Definiţi o funcţie final recursivă pentru calcularea înmulţirii a două numere x * y astfel:

```
> (inmultire 2 -1)
-2
> (inmultire 2 -2)
-4
> (inmultire 2 -3)
-6
> (inmultire 2 -7)
-14
> (inmultire 2 -10000)
-20000
> (inmultire 0 -10000)
```

```
0
> (inmultire -6 -10000)
60000
```

3.4 Inversa

```
;;; definire reverse cu o variabila colectoare

(define (rev l)
  (rev-aux l '())
  )

(define (rev-aux l aux)
  (cond ((null? l) aux)
        (#t (rev-aux (cdr l) (cons (car l) aux)))
   )
 )

> (rev '(a b d))

> (rev '(a b d (1 2)))

(trace rev-aux)

> (rev-aux '(1 2 a b d) '())

> (untrace rev-aux)

>(rev '(1 (a b) 2 3))
```

3.5 Lungimea unei liste

Urmând exemplele de mai sus scrieţi o funcţie final recursivă care returnează lungimea unei liste. De exemplu:

```
> (rlen '(1 2 3 4))
4

> (rlen '(1 2 3 (j k) (m n o p) 4))
6
```

3.6 Scrieţi o funcţie final recursivă pentru calcularea sumei numerelor unei liste (ignorând simbolurile).

```
>(rsum '(1 2 d 4))
7
>(rsum '(1 2 d 4 (5 lalala 5)))
17
```

4 Nivel superficial, Orice nivel

4.1 Definiţi o funcţie în Racket care determină primul atom dintr-o listă. Funcţia se va comporta astfel:

La nivel superficial:

```
>(prim-elem '(1 2 3))
1

>(prim-elem '())
#f

>(prim-elem '((a b) (c d e) l (o p)))
'l
```

La orice nivel (fără să folosiţi FLATTEN):

```
>(prim-elem2 '((((2 3 4) 8) 8 9) 9))
2

>(prim-elem2 '((((a b 4) 8) 8 9) 9))
'a
```

5 Tema

5.1 Ridicare la putere - final recursivă

Scrieţi o funcţie final recursivă pentru calculul x^y analizând toate cazurile (inclusiv când y este negativ).

5.2 Mulţimi - operaţii

Fiind date două mulţimi A şi B, să se scrie o funcţie recursivă ce determină reuniunea celor două mulţimi ($A \cup B$). Similar, să se scrie câte o funcţie pentru calculul intersecţiei ($A \cap B$), diferenţei ($A\ B$) şi diferenţei simetrice ($A \triangle B$). Hint: verificaţi prima dată dacă lista de input este mulţime.

Notă: Termen de realizare: laboratorul următor.