Recursivitate
Final recursivitate

Isabela Drămnesc

Department of Computer Science,
West University of Timișoara,
Romania
idramnesc@info.uvt.ro

March 19, 2014

Întrebări:

- Cum definim recursivitatea?
- Are importanță ordinea declarării clauzelor?
- Definiți în Racket:
  - O funcție care **concatenează** două liste;
  - O funcție care returnează **inversa** unei liste;
  - O funcție care returnează **produsul** a două numere.

Concatenarea a două liste

Input: l1: [1,2,3] l2: [a,b,c,d]
Output: [1,2,3,a,b,c,d]

```
(define (concatenare l1 l2)
(cond ((null? l1) l2)
        (#t (cons (car l1)
                  (concatenare (cdr l1) l2)))
)
)

(trace concatenare)
```

Inversa unei liste

Input: [1,2,3]
Output: [3,2,1]

```
(define (inversa l)
(cond ((null? l1) '())
        (#t (concatenare (cdr l) (list (car l)))))
)
)

(trace inversa)
```

Înmulțirea a două numere

```
(define (x-ori-y x t)
(if (= y 0)
      0
      (if (< y 0)
          (- (x-ori-y x (+ y 1)) x)
          (+ x (x-ori-y x (- y 1))))
)
)

(trace x-ori-y)
```

Înmulțirea a două numere

```
(define (x-ori-y x t)
(if (= y 0)
      0
      (if (< y 0)
          (- (x-ori-y x (+ y 1)) x)
          (+ x (x-ori-y x (- y 1))))
)
)

(trace x-ori-y)
```

O funcție recursivă este **final-recursivă** dacă:

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

La funcțiile nefinal recursive apelul recursiv este conținut într-un apel de funcție (+, -, cons, append etc).
Pentru transformarea unei funcții recursive într-o funcție final-recursivă se folosește tehnica variabilelor colectoare.

Factorial - Final recursivă

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

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

(trace fact-aux)
```

Inversa unei liste - Final recursivă

```
(define (rev l)
    (rev-aux l '())
)
(define (rev-aux l aux)
   (cond ((null? l) aux)
         (t (rev-aux (cdr l) (cons (car l) aux))))
)

(trace rev-aux)
```

Lungimea unei liste - Final recursivă

```
(define (lung l)
   (lung-aux l 0)
)

(define (lung-aux l aux)
   (if (null? l)
       aux
       (lung-aux (cdr l) (+ 1 aux)))
)
(trace lung-aux)
```

Suma elementelor unei liste - Final recursivă

```
(define (suma-el l)
  (suma-aux l 0))

(define (suma-aux l aux)
  (if (null? l)
      aux
      (suma-aux (cdr l) (+ (car l) aux)))
)

(trace suma-aux)
```

Înmulțirea a două numere - Final recursivă

```
(define (xy x y)
    (xy-aux x y 0))

(define (xy-aux x y aux)
  (if (= y 0)
        aux
        (if (< y 0)
            (xy-aux x (+ y 1) (- aux x))
             (xy-aux x (- y 1) (+ aux x)))
  )
)

(trace xy-aux)
```

Recursivitate vs Final recursivitate

```
(time (fact 200))

(time (factf 200))

(trace x-ori-y)

(trace xy)
```