Paradigme de Programare

Conf. dr. ing. Andrei Olaru

andrei.olaru@cs.pub.ro | cs@andreiolaru.ro
Departamentul de Calculatoare

2020

Cursul 5

Anexă opțională: Implementare promisiuni în Racket

Cursul 5: Anexă opțională: Implementare promisiuni în Racket

Promisiuni

Promisiuni
Implementare în Racket (1)

- `p` = o promisiune – expresie care se evaluează numai când este necesar prima oară, și reține valoarea la care s-a evaluat;
- `my-delay` construiește promisiunea;
- `my-force` evaluează promisiunea;

```
1 (define-macro my-delay (lambda (expr)
2     `(make-promise (lambda () ,expr))
3 ))
4
5 (define my-force (lambda (p)
6     (p)
7 ))
```

Promisiuni
Implementare în Racket (2a)

```
1 (define make-promise (lambda (closure)
2     (let ((ready? #f) (result #f))
3       (lambda ()     ; promisiunea
4         (if (not ready?)
5           (begin (set! ready? #t)
6                  (set! result (closure))))
7         result
8 ))))
```

Dar dacă:

```
1 (define x 1)
2 (define p (my-delay (if (= x 3) 0
3           (begin (set! x (+ x 1)) (my-force p) 100)
4 )))
```

`(my-force p)` returnează 100, deși prima valoare calculată de o promisiune terminată a fost 0 (când `x` a ajuns la 3).

Promisiuni

Implementare în Racket (2b – corect)

```
1  (define make-promise (lambda (closure)
2      (let ((ready? #f) (result #f))
3        ; promisiunea
4        (lambda ()
5          (if ready?
6              result
7              (let ((r (closure)))
8                (if ready?
9                    result
10                   (begin (set! ready? #t)
11                          (set! result r)
12                          result)
13             ))
14         ))
15 )))
```

Promisiuni
Implementare în Racket – discuție

- Situații în care evaluarea expresiei împachetate declanșează, ea însăși, forțarea promisiunii → a doua verificare a lui `ready?`.
- Promisiuni → obiecte cu stare.
- Prima forțare → efecte laterale.