Paradigme de programare

2010-2011, semestrul 2

Curs 7

Cuprins

- Fluxuri

Context

Scopuri

- Modularitate
- Eficienta

+

Viziune asupra lumii

- Obiecte care isi schimba starea
- Aceasta e inglobata in interiorul obiectelor (din ratiuni de modularitate)

Context (2)

Aceste scopuri + aceasta viziune =>

- Atribuiri
- Stare
- Schimbare
- Timp
- Identitate
- Partajarea obiectelor

=>

Probleme!

Idee

Poate ca:

- viziunea asupra lumii este gresita
- nimic nu se schimba
- in loc de obiecte care se schimba avem un tot (descris prin colectia stadiilor sale de evolutie)

Poate ca:

- “time is on our side” (nu e nevoie sa il controlam explicit)

Exemplu – determinarea primalitatii unui numar

Ce este un numar prim?

Un numar care nu are divizori in intervalul [2 .. sqrt(n)]

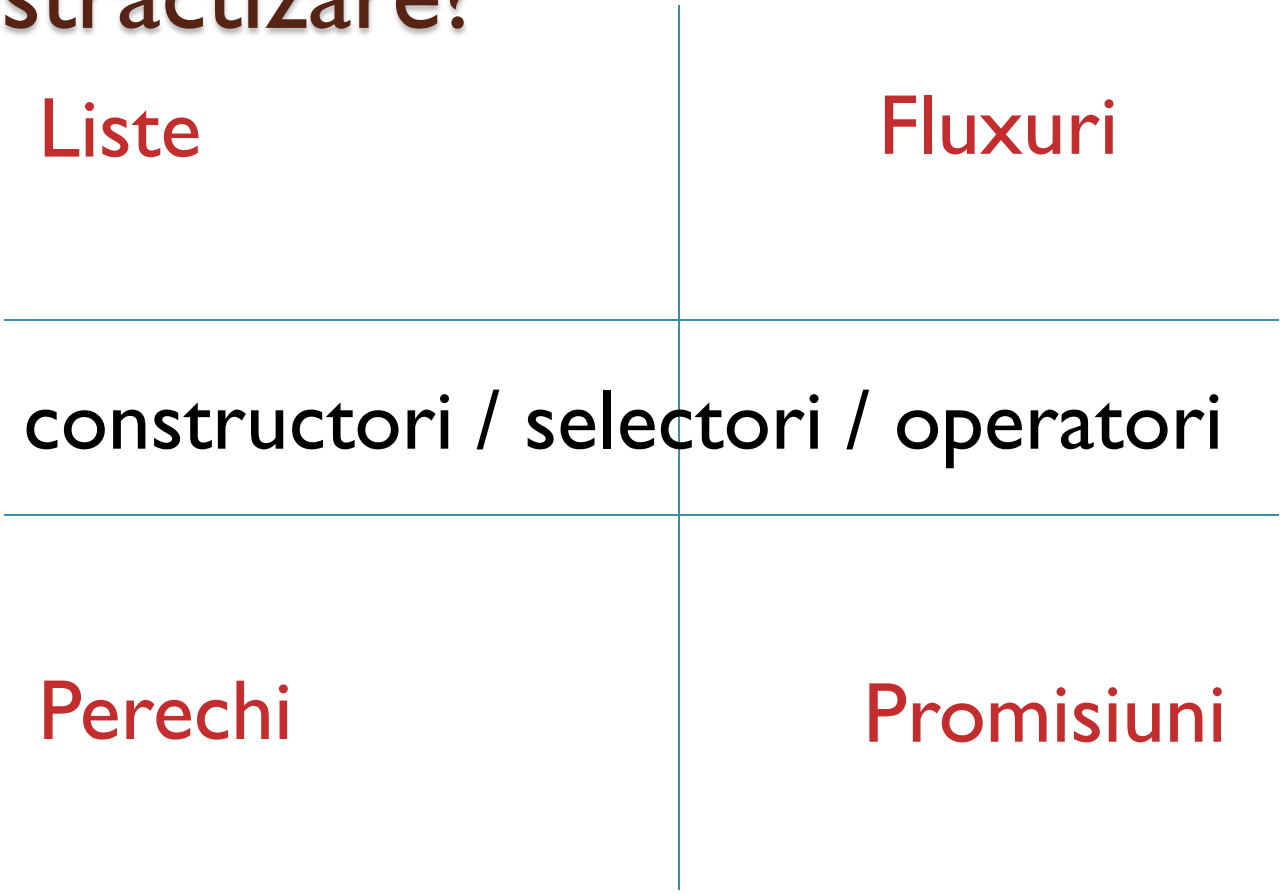
Comparatie intre variantele de implementare

- la calculator

Care e diferenta esentiala intre varianta 1 (cu liste) si varianta 3 (cu streamuri)?

Diferenta se afla sub bariera de abstractizare.

Ce se afla sub bariera de abstractizare?



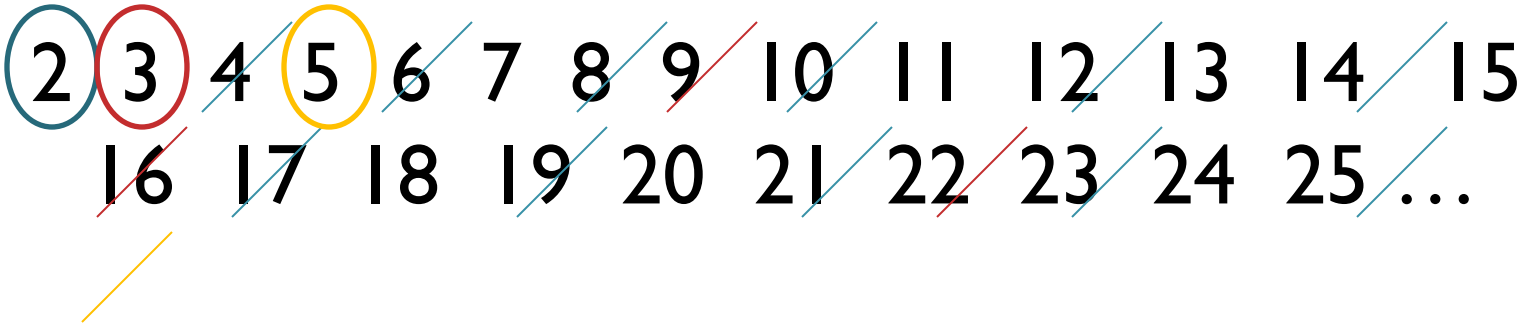
Promisiune = procedura al carei corp spune ce m-am angajat sa fac si ce intelegeam prin asta (cat erau variabilele in momentul in care am promis)

Exemplu de rulare cu fluxuri

```
(define (prim? n)
  (null-stream?
   (filter-stream (λ (d) (zero? (modulo n d)))
                  (interval-stream 2 (sqrt n)))))
```

```
(prim? 1000000000000) =>
(null-stream? (filter-stream ….‘(2 .
(null-stream? ‘(2 .
#f
```

Ciurul lui Eratostene



Cum se implementeaza promisiunile?

O varianta ar fi sa folosim inchideri functionale.

```
(define-syntax cons-stream
  (syntax-rules ()
    ((_ term rest)
     (cons term (lambda () rest)))))
```

rest nu se evalueaza pana cand functia nu se aplica pe 0 argumente.

```
(define cdr-stream (λ (s) ((cdr s))))
```

Problema cu inchiderile functionale

Trebuie sa recalculez portiunile deja calculate din flux – de fiecare data cand reapare nevoia de ele.

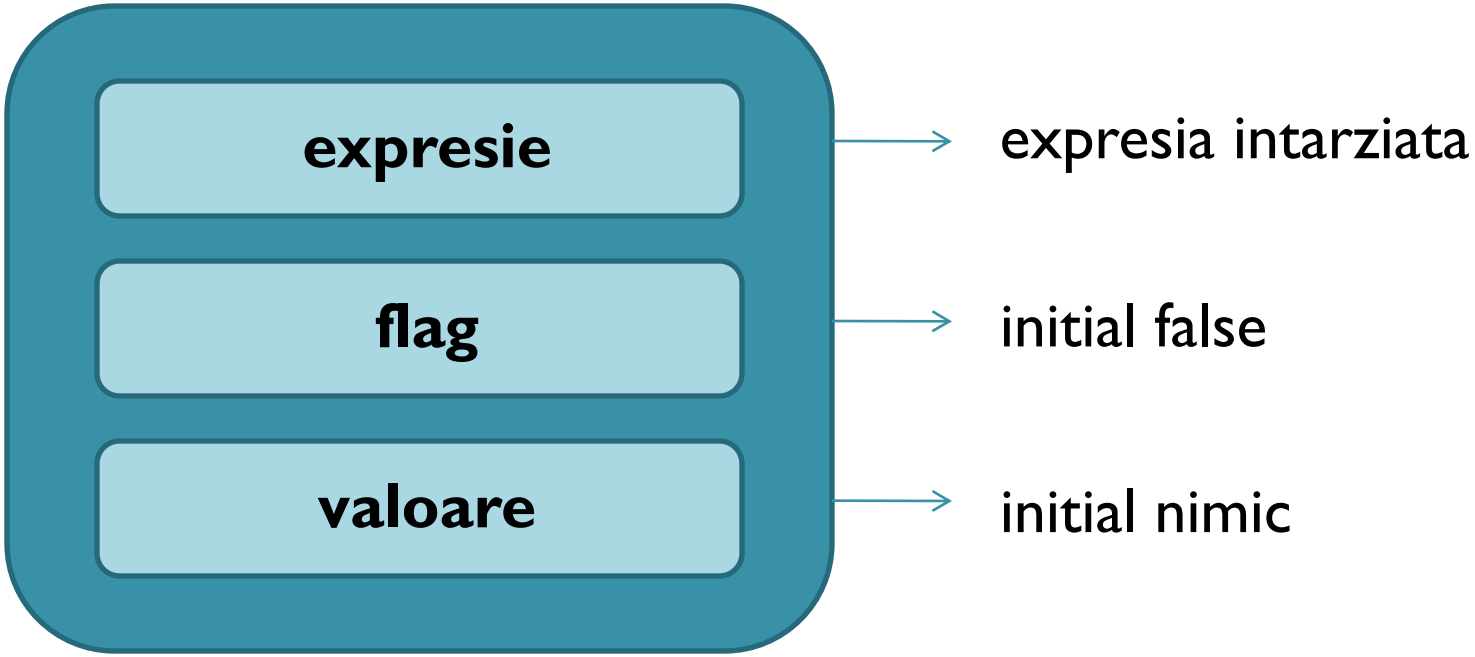
Solutia: functiile delay si force

(delay expresie) => promisiune

(force promisiune) => valoare

delay

- Creeaza o promisiune care arata cam asa:



force

(force promisiune)

daca flag = true

    intoarce valoare

altfel

    valoare <- evalueaza expresie

    flag <- true

    intoarce valoare

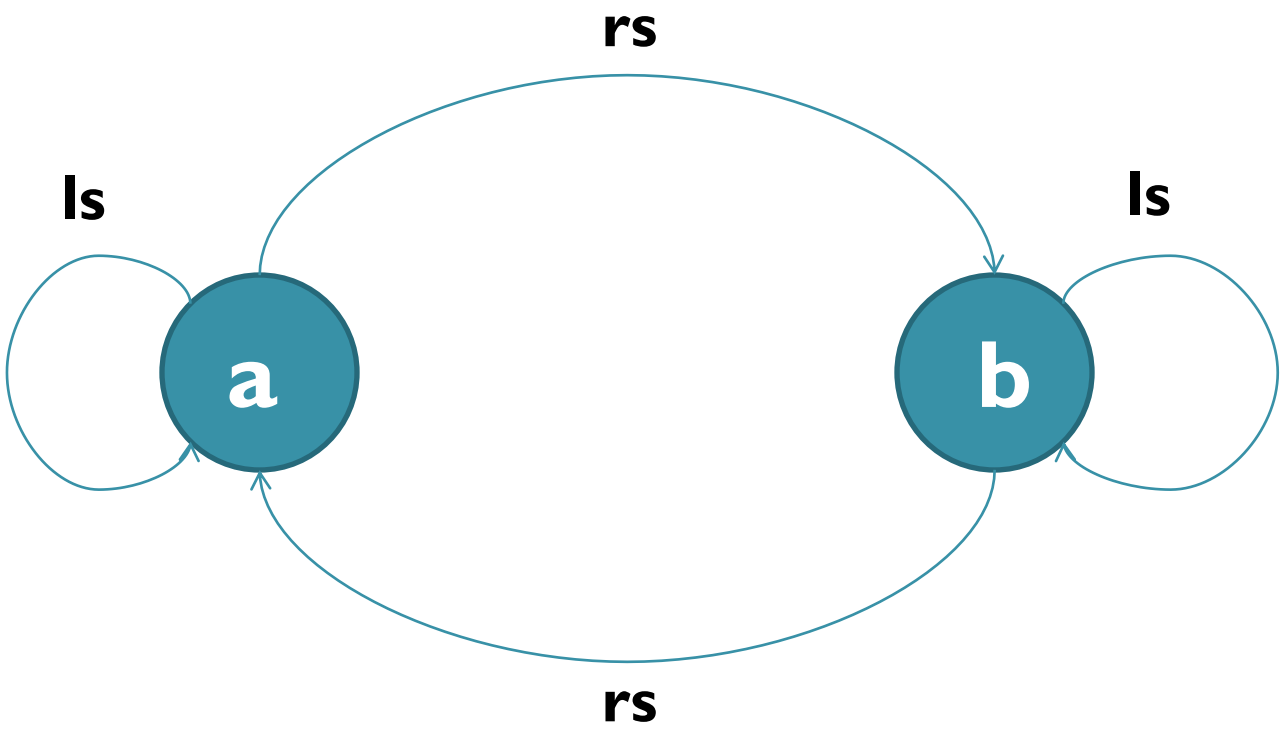
Implementarea promisiunilor folosind delay

```
(define-syntax cons-stream
  (syntax-rules ()
    ((_ term rest)
     (cons term (delay rest)))))

(define cdr-stream (λ (s) (force (cdr s))))
```

Posibila implementare pentru delay

```
(define (delay expr) (memoize (λ () expr))

(define (memoize thunk)
  (let ((need-val #t)
        (val 'whatever))
    (λ ()
     (if need-val
         (begin
           (set! val (thunk))
           (set! need-val #f)))
      val)))
```

Un exemplu cu grafuri



Graful rezultat folosind inchideri functionale