

LABORATOR 5: Întărirea evaluării

EVALUARE

APLICATIVĂ

→ parametrii și
evaluarea lor înainte de
aplicarea funcției
"eager evaluation"

LENESĂ

→ evaluarea param.
se întârzie până
când e nevoie de ei
"lazy evaluation"

Ex :

$$\begin{aligned}
 & (\text{st. sy}(\#y)) \\
 & 1 \quad (\lambda z. (z + z) 3) \\
 & \downarrow \qquad \qquad \qquad \searrow \\
 & (\text{st. sy}(\#y) \quad 5) \\
 & \downarrow \\
 & 6
 \end{aligned}$$

$$\begin{aligned}
 & (\text{st. sy}(\#y)) \\
 & 1 \quad (\lambda z. (z + z) 3) \\
 & \downarrow \\
 & 1 + (\lambda z. (z + z) 3) \\
 & \downarrow \\
 & 1 + 5 \Rightarrow 6
 \end{aligned}$$

Racket

parametrii prin valoare

Haskell

parametrii prin nume

(Ex)

$$\text{Fix } f = \text{sf. } (f(\text{Fix } f))$$

$$d = \text{st. } b$$

$$(\text{Fix } d) ?$$

1) Eager-evaluation,

$$(\text{Fix } d) = ((\text{sf. } f(\text{Fix } f)) d)$$

$$= (d (\text{Fix } d))$$

$$= (d (\text{sf. } (f(\text{Fix } f)) d))$$

$$= (d (d (\text{Fix } d)))$$

=) Such a infinite

2) Lazy - evaluation

$$\begin{aligned}(\bar{f}x \quad ct) &= \left(s.f. \left(f \left(\bar{f}x \quad f \right) \right) \quad ct \right) \\&= \left(ct \quad (\bar{f}x \quad ct) \right) \\&= \left(st. b \quad \underbrace{(\bar{f}x \quad ct)}_{\text{parametru}} \right) \\&= b \quad \text{mai e evaluat pt. ca}\end{aligned}$$

*parametru
mai e evaluat pt. ca
nu e nuanță de el.*

EVALUAREA LENEŞĂ IN RACKET

INCHIDERI
FUNCTIONALE

PROMISE

```
(define sum1
  (lambda (x y)
    (lambda () (+ x y))))
```

```
(define sum2
  (lambda (x y)
    (delay (+ x y))))
```

$(\text{sum } 1 \ 2) \Rightarrow$ procedure
 $((\text{sum } 1 \ 2)) \Rightarrow$ întârziere evaluării

$(\text{sum2 } 1 \ 2) \Rightarrow$ promise
 $(\text{force } (\text{sum2 } 1 \ 2))$
intârziere evaluării

!!!

```
(define x (delay (+ 1 2)))
```

$x \leftarrow \langle \text{promise} : 2 \rangle$

$(\text{force } x) \leftarrow$

$x \leftarrow \langle \text{promise} : 3 \rangle$

de la force
rezultatul se
păstrază în
cache

FLUXURI

→ obiecte infinite create cu
generatorul evaluatorului limbaj

(elem, current, generator)

primul element

Promisiune /
Închidere funcț.

=> generatorul ne ajută să producă
elementele următoare la cerere

Interfață Fluxuri Racket

LISTE

FLUXURI

cons

stream-cons

car

stream-first

cdr

stream-rest

'()

empty-stream

null?

stream-empty?

map / filter

stream-map / stream-filter

SIRUL CONSTANT DE 1

- NICHEZERI FUNCT.

```
(define ones-stream1  
  (cons 1 (lambda () ones-stream1)))
```

generator

- PROMISIUNI

```
(define ones-stream2  
  (cons 1 (delay ones-stream2)))
```

generator

- FU XURI RACKET

```
(define ones-stream3  
  (stream-cons 1 ones-stream3))
```

SIRUL NR. NATURALE

→ aia genera'm elementul următor
pr baza elem. curent

```
(define (get-naturals k)
  (stream-cons k (get-naturals (add1 k))))
```

```
(define naturals (get-naturals 0))
```

generator cu parametru

(stream-take (get-naturals 0) 3)

(0, get-nat 1)

(1, get-nat 2)

(2, get-nat 3)



Când mai e nevoie de
elemente se extindeaza
generatorul

DECLARAȚII IMPLICITE

= fără generator explicit

- se dau 1/2 termeni și după se aplicază funcționale pe streamuri

• Stream-map (!! pt fluxuri - acceptă & setui cu 1 parametru)

$s = s_0, s_1, s_2, \dots$

map (f, s) = $f(s_0), f(s_1), f(s_2), \dots$

Ex: 1, 2, 4, 8, 16, ...

1 * $\xrightarrow{2}$ 2 * $\xrightarrow{2}$ 4 * $\xrightarrow{2}$ 8, ...

$s = 1, \text{ map}(*^2, s)$

$s = \underline{1}, f(s_0), f(s_1), f(s_2), \dots$

$s = 1, f(1), f(s_1), f(s_2), \dots$

$s = 1, 2, \underline{f(s_1)}, f(s_2) \dots$

$s = 1, 2, 4, \underline{f(s_2)} \dots$

• Stream - Filter

Ex: 1, 3, 5, 7 ...

nat = 0, 1, 2, 3, 4 ...

=> filter (odd?, naturals)

=> (filter odd? naturals)

Alte et. - Stream

0

Definiții în Racket următorul flux infinit: ((1) (1 2 1) (1 2 3 2 1) (1 2 3 4 3 2 1) ...).

Idee: urmărt. lista se adaugă din lista anterioră

'(1) - elem. de inițiat
~~~~~  
↓ map add1

((1 2 1))  
~~~~~  
↓ map add1

((1 2 3 2 1))

=)

```
(define pyramids
  (stream-cons '()
    (stream-map (lambda (L)
      (append '() (map add1 L) '()))
      pyramids)))
```

2

4. Determinați primele 8 elemente ale fluxului s din Racket:

```
1 (define s
2   (stream-cons 1
3     (stream-cons 2
4       (stream-zip-with - s naturals))))
```

s: 30 31 32 33 34 35 ... (-)

0 1 2 3 4 5 ...

| | | | | | | | |
|----|----|-----|-----|-----|-----|-----|-----|
| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| 1 | 2 | 30- | 31- | 32- | 33- | 34- | 35- |
| 0 | 1 | 2 | 3 | 4 | 5 | | |

1 2 1 1 -1 -2 -5 -7 ...