Racket CheatSheet

Laborator 5

Promisiuni

delay force

```
(define p (delay (+ 1 2)))
p                                    #<promise:p>

;; force forteaza evaluarea
(force p)                                       3

;; un force subsecvent ia rezultatul din cache
(force p)                                       3
```

Constructori fluxuri

empty-stream stream-cons

```
empty-stream                            #<stream>
(stream-cons 1 empty-stream)            #<stream>

(define ones (stream-cons 1 ones))    fluxul de 1

;; fluxul numerelor naturale
(define naturals
  (let loop ((n 0))
     (stream-cons n (loop (add1 n)))))
```

Operatori pe fluxuri

stream-first stream-rest stream-empty?

```
(stream-first naturals)                         0
(stream-rest (stream-cons 2 ones))    fluxul de 1

(stream-empty? empty-stream)                   #t
(stream-empty? ones)                           #f
```

Funcționale pe fluxuri

stream-map stream-filter

```
;; stream-map merge numai cu functii unare
(stream-map sqr naturals)        fluxul 0, 1, 4..

(stream-filter even? naturals)     fluxul nr pare
```

Fluxuri definite explicit

Generator recursiv cu oricâți parametri definit în mod uzual cu named let

```
;; fluxul puterilor lui 2
(define powers-of-2
  (let loop ((n 1))
    (stream-cons n (loop (* n 2)))))

;; fluxul Fibonacci
(define fibonacci
  (let loop ((n1 0) (n2 1))
    (stream-cons n1 (loop n2 (+ n1 n2)))))

;; fluxul 1/(n!)
;; (cu care putem aproxima constanta lui Euler)
(define rev-factorials
  (let loop ((term 1) (n 1))
    (stream-cons term (loop (/ term n) (add1
        n)))))

;; testare: stream-take este definita de noi
;; in laborator, nu cel definit in Racket

;; rezultat '(1 2 4 8 16 32 64 128 256 512)
(stream-take powers-of-2 10)

;; rezultat '(0 1 1 2 3 5 8 13 21 34)
(stream-take fibonacci 10)

;; rezultat 2.7182815255731922
(apply + 0.0 (stream-take rev-factorials 10))
```

AȘA DA / AȘA NU

Folosiți interfața Racket pentru fluxuri!

```
DA: (stream-cons x S) NU: (cons x (lambda () S))
                      NU: (cons x (delay S))
DA: (stream-rest S)   NU: ((cdr S))
                      NU: (force (cdr S))
```

Fluxuri definite implicit

Fără generator explicit

Dă explicit primii 1-2 termeni, apoi inițiază o prelucrare folosind (de obicei) funcționale pe fluxuri

```
;; stream-zip-with este definita de voi
;; in laborator, nu exista in Racket

;; fluxul puterilor lui 2
(define powers-of-2-a
  (stream-cons
   1
   (stream-zip-with +
                    powers-of-2-a
                    powers-of-2-a)))

(define powers-of-2-b
  (stream-cons
   1
   (stream-map (lambda (x) (* x 2))
               powers-of-2-b)))

;; fluxul Fibonacci
(define fibonacci
  (stream-cons
   0
   (stream-cons
    1
    (stream-zip-with +
                     fibonacci
                     (stream-rest fibonacci)))))

;; fluxul 1/(n!)
(define rev-factorials
  (stream-cons
   1
   (stream-zip-with /
                    rev-factorials
                    (stream-rest naturals))))
```

Folositi cu incredere!

http://docs.racket-lang.org/