Потоци

доц. Атанас Семерджиев

1

Съдържание

- Интуиция
- Поддръжка в Racket
- Функции от по-висок ред за работа с потоци
- Комбиниране на потоци

2

2

Отлагане на операции

```
(define (infinite n)
  (printf "~a\n" n)
  (infinite (+ n 1)))
```

```
(infinite 0)
```

```
0
1
2
3
4
5
6
...
```

3

3

```
(define (infinite n)
  (printf "~a\n" n)
  (delay (infinite (+ n 1)))))
```

```
(infinite 0)
```

```
0
#<promise:infinite>
```

```
(force (infinite 0))
```

```
0
1
#<promise:infinite>
```

```
(force (force (infinite 0))
```

...

4

4

```
(define (infinite n)
  (cons n (delay (infinite (+ n 1)))))
```

```
(define (run p n)
  (cond [(> n 0) (printf "~a: ~a\n" n (car p))
                 (run (force (cdr p)) (- n 1))]

        [else    (display "End\n")]))
```

```
(run (infinite 0) 10)
```

```
10: 0
9: 1
8: 2
7: 3
6: 4
5: 5
4: 6
3: 7
2: 8
1: 9
End
```

5

5

Поддръжка в Racket

6

```
(require racket/stream)
```

```
(define (infinite n)
  (cons n (delay (infinite (+ n 1)))))
```

```
(define (infinite n)
  (stream-cons n (infinite (+ n 1))))
```

7

7

```
(define (run p n)
  (cond ((> n 0) (printf "~a: ~a\n" n (car p))
                 (run (force (cdr p)) (- n 1)))

        (else    (display "End\n"))))
```

```
(define (run p n)
  (cond [(> n 0) (printf "~a: ~a\n" n (stream-first p))
                 (run (stream-rest p) (- n 1))]

        [else    (display "End\n")]))
```

8

8

Някои полезни функции

Функция	Описание
`(stream-length s)`	Дължина на потока `s` (`s` трябва да е краен)
`(stream? v)`	Проверява дали `v` е поток
`(stream-empty? s)`	Проверява дали `s` е празен
`(stream->list s)`	Преобразува поток до списък (`s` трябва да е краен)
`(stream-ref s i)`	Връща `i`-ия елемент на `s`
`(stream-tail s i)`	Поток получен от `s`, след премахване на първите `i`-елемента
`(stream-take s i)`	Поток от първите `i`-елемента на `s`

Източник: https://docs.racket-lang.org/reference/streams.html

9

9

Специални форми (stream) и (stream*)

`stream` и `stream*` са аналогични на `list` и `list*`

```
(stream 3 4 5)
```

```
(stream-cons 3
             (stream-cons 4
                          (stream-cons 5
                                       empty-stream)))
```

```
(stream* 1 2 (stream 3 4))
```

```
(stream 1 2 3 4)
```

10

10

Ръчно конструиране на поток

```
(define (interval a b)
  (if (> a b) empty-stream
      [stream-cons a (interval (+ a 1) b)]))
```

```
(define (fib-stream)
  (define (gen n-2 n-1)
    [stream-cons n-2 (gen n-1 (+ n-1 n-2))])
  (gen 1 1))
```

11

11

(stream-append)

И двата потока могат да са безкрайни!

```
(define (stream-append s1 s2)
    (if (stream-empty? s1)
        s2
        (stream-cons
           (stream-head s1)
           (stream-append (stream-rest s1) s2))))
```

12

12

Функции от по-висок ред за работа с потоци

13

(stream-map)

Потокът може да е безкраен!

```
(define (stream-map f s)
  (if (stream-empty? s)
      empty-stream
      (stream-cons (f (stream-first s))
                   (stream-map f (stream-rest s)))))
```

14

14

(stream-filter)

Потокът може да е безкраен!

```
(define (stream-filter pred? s)
  (cond
    [(stream-empty? s)
     empty-stream]

    [(pred? (stream-first s))
     (stream-cons (stream-first s)
                  (stream-filter pred? (stream-rest s)))]

    [else (stream-filter pred? (stream-rest s))]
    ))
```

15

15

(stream-fold)

Потокът НЕ може да е безкраен!

```
(define (stream-fold f init s)
    (if (stream-empty? s)
        init
        (stream-fold f
                     (f init (stream-first s))
                     (stream-rest s))))
```

16

16

Комбиниране на потоци

17

Комбиниране на потоци

И двата потока може да са безкрайни

```
(define (stream-sum s1 s2)
  (cond [(stream-empty? s1) s2]
        [(stream-empty? s2) s1]
        [else
         (stream-cons (+ (stream-first s1)
                         (stream-first s2))
                      (stream-sum (stream-rest s1)
                                  (stream-rest s2)))]
        ))
```

18

18

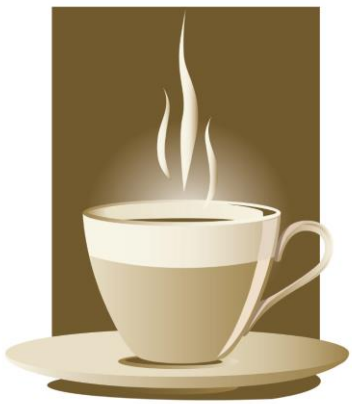
Комбиниране на потоци

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

(define (nat)
  (stream-cons 0
               (stream-sum (nat) (ones))))
```

19

19



20