Функциональное программирование

part 4

Гавриков Антон Александрович

2022

ленивые вычисления

```
;(define (delay a) (lambda () a))
;(define (force a) (a))
;delay откладывает выполнение a до вызова force

;; Таким образом задаётся "ленивая пара", пока не обращайте
;; внимание на синтаксис
(define-syntax cons-stream
    (syntax-rules ()
          [(_ a b) (cons a (delay b))]
              ))
(define (stream-car stream)
    (car stream))
(define (stream-cdr stream)
    (force (cdr stream)))
(define the-empty-stream '())
(define stream-null? null?)
```

stream map, for-each

```
;аналог for-each для stream
(define (stream-for-each proc stream)
  (if (stream-null? stream)
    'done
    (begin (proc (stream-car stream))
           (stream-for-each proc (stream-cdr stream)))))

;аналог map для stream
(define (stream-map proc . argstreams)
  (if (null? (car argstreams))
    the-empty-stream
    (cons-stream
      (apply proc (map stream-car argstreams))
      (apply stream-map
             (cons proc (map stream-cdr argstreams))))))
```

Пример с созданием стрима

```
;выводит каждый элемент стрима на новой строчке
(define (display-stream stream)
    (stream-for-each println stream))

(define (stream-enumerate-interval low hi)
  (if (> low hi)
    the-empty-stream
    (cons-stream low
                 (stream-enumerate-interval (+ 1 low) hi))))

(define yyy (stream-map
              square
              (stream-enumerate-interval 1 10)))

(display-stream yyy)
;1
;4
;9
;16
;...
```

перевод стрима в лист

```
(define (stream->list stream)
  (if (stream-null? stream)
    '()
    (cons (stream-car stream)
          (stream->list (stream-cdr stream)))))

(display (stream->list yyy)) (newline)
;(1 4 9 16 25 36 49 64 81 100)
```

filter для stream

```
(define (stream-filter pred stream)
  (cond ((stream-null? stream) the-empty-stream)
        ((pred (stream-car stream))
         (cons-stream (stream-car stream)
                      (stream-filter
                        pred
                        (stream-cdr stream))))
        (else (stream-filter pred (stream-cdr stream)))))

(display-stream (stream-filter prime?
                               (stream-enumerate-interval
                                 10 100)))
;11
;13
;17
;...
```

аналог list-ref

```
(define (stream-ref s n)
  (if (= n 0)
    (stream-car s)
    (stream-ref (stream-cdr s) (- n 1))))

(define (show x)
  (println x)
  x)

(define x
  (stream-map show
              (stream-enumerate-interval 0 10)))

(stream-ref x 5) ; 0 .. 5
(println "next command")
(stream-ref x 7) ;  6 7
;delay запоминает результаты вычислений
```

Пример с моментом вычисления

```
(define sum 0)
(define (accum x) (set! sum (+ x sum)) sum)
(define seq
  (stream-map accum
              (stream-enumerate-interval 1 20)))
(define (print-sum) (display "sum: ") (println sum))
(print-sum) ;1
(define y (stream-filter even? seq))
(print-sum) ;6
(define z
  (stream-filter (lambda (x) (= (remainder x 5) 0))
                 seq))
(print-sum) ;10
(println (stream-ref y 7)) ;136
(print-sum) ;136
(display-stream z)
;10 15 45 55 105 120 190 210
(print-sum) ;210
```

Бесконечный поток

```
(define (integers-starting-from n)
  (cons-stream n (integers-starting-from (+ n 1))))
(define integers (integers-starting-from 1))

(define (stream-print-first s n)
  (if (= n 0)
    (newline)
    (begin (display (stream-car s)) (display " : ")
           (stream-print-first (stream-cdr s) (- n 1)))))

(stream-print-first integers 10)
;1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 : 9 : 10 :
(define no-sevens
  (stream-filter (lambda (x) (not (divisible? x 7)))
                 integers))

(stream-print-first no-sevens 20)
;1 : 2 : 3 : 4 : 5 : 6 : 8 : 9 : 10 : 11 : 12 : 13 : 15 : 16
;: 17 : 18 : 19 : 20 : 22 : 23 :
```

Фибоначчи поток

```
(define (fibgen a b) (cons-stream a (fibgen b (+ a b))))
(define fibs (fibgen 0 1))

(stream-print-first fibs 10)
;0 : 1 : 1 : 2 : 3 : 5 : 8 : 13 : 21 : 34
```

Простые числа с помощью решета Эратосфена

```
;по очереди убираем каждое число, которое делится на текущее
(define (sieve stream)
  (cons-stream
    (stream-car stream)
    (sieve (stream-filter
             (lambda (x)
               (not (divisible? x (stream-car stream))))
             (stream-cdr stream)))))
(define primes (sieve (integers-starting-from 2)))

(stream-print-first primes 10)
;2 : 3 : 5 : 7 : 11 : 13 : 17 : 19 : 23 : 29 :
```

Определение стрима через стрим

```
(define ones (cons-stream 1 ones))
(stream-print-first ones 10)
;1 : 1 : 1 : 1 : 1 : 1 : 1 : 1 : 1 : 1

(define (add-streams s1 s2) (stream-map + s1 s2))

(define integers
  (cons-stream 1 (add-streams ones integers)))

(stream-print-first integers 10)
;1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 : 9 : 10
```

Фибоначчи через сложения

```
(define fibs
  (cons-stream
    0
    (cons-stream 1 (add-streams (stream-cdr fibs) fibs))))

;            1 : 1 : 2 : 3 : 5 : 8  : 13 : 21 : 34
;            0 : 1 : 1 : 2 : 3 : 5  : 8  : 13 : 21 : 34
;    0 : 1 : 1 : 2 : 3 : 5 : 8 : 13 : 21 : 34

(stream-print-first fibs 10)
;0 : 1 : 1 : 2 : 3 : 5 : 8 : 13 : 21 : 34
```

Степени двойки

```
(define (scale-stream stream factor)
  (stream-map (lambda (x) (* x factor))
              stream))

(define double (cons-stream 1 (scale-stream double 2)))

(stream-print-first double 10)
;1 : 2 : 4 : 8 : 16 : 32 : 64 : 128 : 256 : 512
```

Опять простые числа

```
(define primes
  (cons-stream
    2
    (stream-filter prime? (integers-starting-from 3))))


(define (prime? n)
  (define (iter ps)
    (cond ((> (square (stream-car ps)) n) #t)
          ((divisible? n (stream-car ps)) #f)
          (else (iter (stream-cdr ps)))))
  (iter primes))

(stream-print-first primes 10)
```

Опять факториал

```
(define s (cons-stream 1 (add-streams s s)))

(stream-print-first s 10)

(define factorials
  (cons-stream
    1
    (mul-streams (stream-cdr integers) factorials)))

(define (mul-streams s1 s2) (stream-map * s1 s2))

(stream-print-first factorials 10)
```

Частичные суммы

```
(define (partial-sums s)
  (if (stream-null? s)
    0
    (cons-stream
      (stream-car s)
      (stream-map (lambda (x) (+ x (stream-car s)))
                  (partial-sums (stream-cdr s))))))

(stream-print-first (partial-sums integers) 10)
```

Вычисление квадратного корня

```
(define (average x y)
  (/ (+ x y) 2))

(define (sqrt-improve guess x)
  (average guess (/ x guess)))

(define (sqrt-stream x)
  (define guesses
    (cons-stream
      1.0
      (stream-map (lambda (guess) (sqrt-improve guess x))
                  guesses)))
  guesses)
(stream-print-first (sqrt-stream 2) 10)
; 1.0 : 1.5 : 1.4166666666666665 : 1.4142156862745097 :
; 1.4142135623746899 : ...
```

Вычисление Pi

```
(define (pi-summands n)
  (cons-stream (/ 1.0 n)
               (stream-map - (pi-summands (+ n 2)))))
(define pi-stream
  (scale-stream (partial-sums (pi-summands 1)) 4))

(stream-print-first pi-stream 20)
;4.0 : 2.666666666666667 : 3.466666666666667 :
;2.895238095238095 : ...
```

На размышление 30 секунд

```
;почему это хуже, чем предыдущее определение?
(define (sqrt-stream x)
  (cons-stream
    1.0
    (stream-map (lambda (guess) (sqrt-improve guess x))
                (sqrt-stream x))))

;предыдущее
(define (sqrt-stream x)
  (define guesses
    (cons-stream
      1.0
      (stream-map (lambda (guess) (sqrt-improve guess x))
                  guesses)))
  guesses)
```

Подсказочка

```
(define (sqrt-improve guess x)
  (display ".")(average guess (/ x guess)))

(define (sqrt-stream x)
  (cons-stream
    1.0
    (stream-map (lambda (guess) (sqrt-improve guess x))
                (sqrt-stream x))))

(stream-ref (sqrt-stream 10) 10) (newline) ;55 .

(define (sqrt-stream x)
  (define guesses
    (cons-stream
      1.0
      (stream-map (lambda (guess) (sqrt-improve guess x))
                  guesses)))
  guesses)

(stream-ref (sqrt-stream 10) 10) (newline) ;10 .
```

Стрим пар

```
(define (interleave s1 s2)
  (if (stream-null? s1)
    s2
    (cons-stream (stream-car s1)
                 (interleave s2 (stream-cdr s1)))))

(define (pairs s t)
  (cons-stream
    (list (stream-car s) (stream-car t))
    (interleave
      (stream-map (lambda (x) (list (stream-car s) x))
                  (stream-cdr t))
      (pairs (stream-cdr s) (stream-cdr t)))))

(define int-pairs (pairs integers integers))

(stream-print-first int-pairs 20)
;(1 1) : (1 2) : (2 2) : (1 3) : (2 3) : (1 4) : (3 3) : (1
;5) : (2 4) : (1 6) : (3 4) : (1 7) : (2 5) : (1 8) : (4 4)
;: (1 9) : (2 6) : (1 10) : (3 5) : (1 11) :
```

Стрим пар с простой суммой

```
(define prime-sum (
  stream-filter
  (lambda (pair) (prime? (+ (car pair) (cadr pair))))
  int-pairs))

(stream-print-first prime-sum 20)
;(1 1) : (1 2) : (2 3) : (1 4) : (1 6) : (3 4) : (2 5) : (1
;10) : (1 12) : (2 9) : (1 16) : (1 18) : (2 11) : (1 22) :
;(3 8) : (5 6) : (2 15) : (1 28) : (4 7) : (1 30) :
```