

Функции от по-висок ред – част 2

Трифон Трифонов

Функционално програмиране, 2025/26 г.

30 октомври 2025 г.

Тази презентация е достъпна под лиценза Creative Commons Признание-Некомерсиално-Споделяне на споделеното 4.0 Международен 

Функции, които връщат функции

Да разгледаме функцията, която прилага дадена функция два пъти над аргумент.

- `(define (twice f x) (f (f x)))`
- `(twice square 3) → 81`
- `(define (twice f) (lambda (x) (f (f x))))`
- `(twice square 3) → Грешка!`
- `(twice square) → #<procedure>`
- `((twice square) 3) → 81`
- `((twice (twice square)) 2) → 65536`

Примери

- `(define (n+ n) (lambda (i) (+ i n)))`
- `(define 1+ (n+ 1))`
- $(1+ 7) \longrightarrow 8$
- `(define 5+ (n+ 5))`
- $(5+ 7) \longrightarrow 12$
- `(define (compose f g) (lambda (x) (f (g x))))`
- $((\text{compose square } 1+) 3) \longrightarrow 16$
- $((\text{compose } 1+ \text{ square}) 3) \longrightarrow 10$
- $((\text{compose } 1+ (\text{compose square } (n+ 2))) 3) \longrightarrow 26$

Оценка на lambda

{E} (define (n+ n)
 (lambda (i) (+ i n)))

{E} (define 1+ (n+ 1))



{E₁} (lambda (i) (+ i n))

{E} (define 5+ (n+ 5))



{E₂} (lambda (i) (+ i n))

{E} (1+ 7)



{E₃} (+ i n)



8

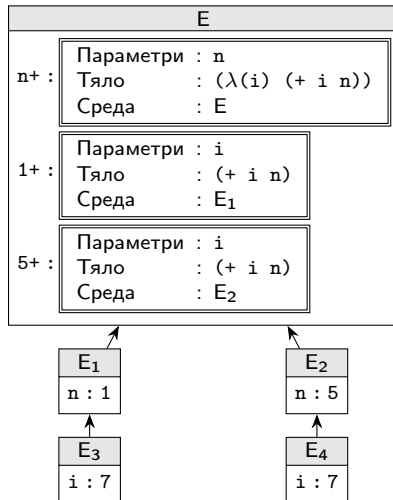
{E} (5+ 7)



{E₄} (+ i n)



12



Намиране на производна

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

```
(define (derive f dx)
  (lambda (x) (/ (- (f (+ x dx)) (f x)) dx)))
```

- `(define 2* (derive square 0.01))`
- `(2* 5) → 10.0099999999999764`
- `((derive square 0.0000001) 5) → 10.000000116860974`
- `((derive (derive (lambda (x) (* x x x)) 0.001) 0.001) 3) → 18.006000004788802`

Повторено прилагане

Да се намери n -кратното прилагане на дадена едноместна функция.

$$f^n(x) = \underbrace{f(f(f(\dots(f(x))\dots)))}_n$$

Решение №1: $f^0(x) = x, f^n(x) = f(f^{n-1}(x))$

```
(define (repeated f n)
  (lambda (x) (if (= n 0) x (f ((repeated f (- n 1)) x))))
```

Решение №2: $f^0 = id, f^n = f \circ f^{n-1}$

```
(define (repeated f n)
  (if (= n 0) id (compose f (repeated f (- n 1)))))
```

Решение №3: $f^n = \underbrace{f \circ f \circ \dots \circ f}_n \circ id$

```
(define (repeated f n)
  (accumulate compose id 1 n (lambda (i) f) 1+))
```

n -та производна

Да се намери n -та производна на дадена едноместна функция.

Решение №1: $f^{(0)} = f, f^{(n)} = (f^{(n-1)})'$

```
(define (derive-n f n dx)
  (if (= n 0) f (derive (derive-n f (- n 1) dx) dx)))
```

Решение №2: $f^{(n)} = f \overset{n}{\overbrace{''''\dots''}}$

```
(define (derive-n f n dx)
  ((repeated (lambda (f) (derive f dx)) n) f))
```

Решение №3: $f^{(n)} = \underbrace{f \circ f \circ \dots \circ f}_n$

```
(define (derive-n f n dx)
  ((accumulate compose id 1 n
    (lambda (i) (lambda (f) (derive f dx))) 1+) f))
```

All you need is λ — let

Специалната форма `lambda` е достатъчна за реализацията на (почти) всички конструкции в Scheme!

```
(let ((<символ> <израз>)) <тяло>)
```

Симулация на `let`:

 \Longleftrightarrow

```
((lambda (<символ>) <тяло>) <израз>)
```

```
(let ((<символ1> <израз1>)
      (<символ2> <израз2>)
      ...
      (<символn> <изразn>))
  <тяло>)
```

 \Longleftrightarrow

```
((lambda (<символ1> ... <символn>) <тяло>)
  <израз1> ... <изразn>)
```


All you need is λ — булева логика

Симулация на булеви стойности и `if`:

```
(define #t (lambda (x y) x))
(define #f (lambda (x y) y))
(define (lambda-if b x y) ((b x y)))
```

Примери:

- `(lambda-if #t (lambda () (+ 3 5)) (lambda () (/ 4 0)))` \longrightarrow 8
- `(lambda-if #f (lambda () +) (lambda () "abc"))` \longrightarrow "abc"
- `(define (not b) (lambda (x y) (b y x)))`

All you need is λ — числа

Симулация на естествени числа (*нумерали на Чърч*)

Идея: представяне на числото n като $\lambda f, x \ f^n(x)$

- `(define c3 (lambda (f x) (f (f (f x)))))`
- `(define c5 (lambda (f x) (f (f (f (f (f x)))))))`
- `(define c1+ (lambda (a) (lambda (f x) (f (a f x)))))`
- `(define c+ (lambda (a b) (lambda (f x) (a f (b f x)))))`

All you need is λ — наредени двойки

Можем да симулираме cons, car и cdr чрез lambda!

Вариант №1:

```
(define (lcons x y) (lambda (p) (if p x y)))  
(define (lcar z) (z #t))  
(define (lcdr z) (z #f))
```

Вариант №2:

```
(define (lcons x y) (lambda (p) (p x y)))  
(define (lcar z) (z (lambda (x y) x)))  
(define (lcdr z) (z (lambda (x y) y)))
```