

Списъци

Трифон Трифонов

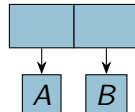
Функционално програмиране, 2025/26 г.

20–23 октомври 2025 г.

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

Наредени двойки

(A . B)

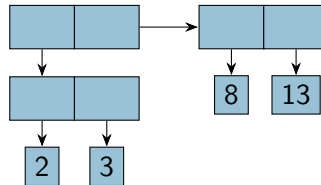


- (cons <израз₁> <израз₂>)
- Наредена двойка от оценките на <израз₁> и <израз₂>
- (car <израз>)
- **Първият** компонент на двойката, която е оценката на <израз>
- (cdr <израз>)
- **Вторият** компонент на двойката, която е оценката на <израз>
- (pair? <израз>)
- Проверява дали оценката на <израз> е наредена двойка

Примери

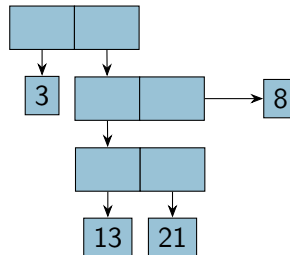
```
(cons (cons 2 3) (cons 8 13))
```

↓
((2 . 3) . (8 . 13))



```
(cons 3 (cons (cons 13 21) 8))
```

↓
(3 . ((13 . 21) . 8))



S-изрази

Дефиниция

S-израз наричаме:

- атоми (булеви, числа, знаци, символи, низове, функции)
- наредени двойки ($S_1 . S_2$), където S_1 и S_2 са S-изрази

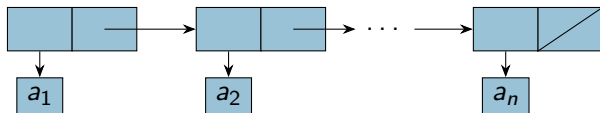
S-изразите са най-общият тип данни в Scheme.

С тяхна помощ могат да се дефинират произволно сложни структури от данни.

Списъци в Scheme

Дефиниция

- ❶ Празният списък $()$ е списък
- ❷ $(h . t)$ е списък ако t е списък
 - h — глава на списъка
 - t — опашка на списъка



$$(a_1 . (a_2 . (\dots (a_n . ())))) \iff (a_1 \ a_2 \ \dots \ a_n)$$

Вградени функции за списъци

- `(null? <израз>)` — дали <израз> е празният списък `()`
- `(list? <израз>)` — дали <израз> е списък
 - `(define (list? l) (or (null? l) (and (pair? l) (list? (cdr l)))))`
- `(list {<израз>})` — построява списък с елементи <израз>
- `(list <израз1> <израз2> ... <изразn>)` \iff
`(cons <израз1> (cons <израз2> ... (cons <изразn> '())))`
- `(cons <глава> <опашка>)` — списък с <глава> и <опашка>
- `(car <списък>)` — главата на <списък>
- `(cdr <списък>)` — опашката на <списък>
- `()` не е наредена двойка!
- `(car '())` \longrightarrow Грешка!, `(cdr '())` \longrightarrow Грешка!

Съкратени форми на car и cdr

Нека $l = (a_1 a_2 a_3 \dots a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 a_3 \dots a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$
- $(\text{cdr } (\text{cdr } l)) \longrightarrow (a_3 \dots a_n) \longleftarrow (\text{cddr } l)$
- $(\text{car } (\text{cdr } (\text{cdr } l))) \longrightarrow a_3 \longleftarrow (\text{caddr } l)$
- имаме съкратени форми за до 4 последователни прилагания на car и cdr

Форми на равенство в Scheme

- **(eq? <израз₁> <израз₂>)** — връща #t точно тогава, когато оценките на <израз₁> и <израз₂> заемат едно и също място в паметта
- **(eqv? <израз₁> <израз₂>)** — връща #t точно тогава, когато оценките на <израз₁> и <израз₂> заемат едно и също място в паметта или са едни и същи по стойност **атоми** (без функции), дори и да заемат различно място в паметта
 - Ако (eq? <израз₁> <израз₂>),
то със сигурност (eqv? <израз₁> <израз₂>)
- **(equal? <израз₁> <израз₂>)** — връща #t точно тогава, когато оценките на <израз₁> и <израз₂> са едни и същи по стойност **атоми** или **наредени двойки**, чиито компоненти са равни в смисъла на equal?
 - В частност, equal? проверява за равенство на списъци
 - Ако (eqv? <израз₁> <израз₂>),
то със сигурност (equal? <израз₁> <израз₂>)

Вградени функции за списъци

- `(length <списък>)` — връща дължината на <списък>
- `(append {<списък>})` — конкатенира всички <списък>
- `(reverse <списък>)` — елементите на <списък> в обратен ред
- `(list-tail <списък> n)` — елементите на <списък> без първите n
- `(list-ref <списък> n)` — n-ти елемент на <списък> (от 0)
- `(member <елемент> <списък>)` — проверява дали <елемент> се среща в <списък>
 - По-точно, връща <списък> от първото срещане на <елемент> нататък, ако го има
 - Връща #f, ако <елемент> го няма в <списък>
 - Сравнението на елементи става с `equal`?
- `(memv <елемент> <списък>)` — като `member`, но сравнява с `eqv`?
- `(memq <елемент> <списък>)` — като `member`, но сравнява с `eq`?

Обхождане на списъци

При обхождане на `l`:

- Ако `l` е празен, връщаме базова стойност (**дъно**)
- Иначе, комбинираме главата (`car l`) с резултата от рекурсивното извикване над опашката (`cdr l`) (**стъпка**)

Примери: `length`, `list-tail`, `list-ref`, `member`, `memqv`, `memq`

Конструиране на списъци

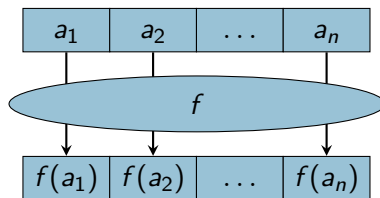
Използваме рекурсия по даден параметър (напр. число, списък...)

- На дъното връщаме фиксиран списък (например `()`)
- На стъпката построяваме с `cons` списък със съответната глава, а опашката строим чрез рекурсивно извикване на същата функция

Примери: `from-to`, `collect`, `append`, `reverse`

Изобразяване на списък (map)

Да се дефинира функция (**map** <функция> <списък>), която връща нов списък съставен от елементите на <списък>, върху всеки от които е приложена <функция>.



```
(define (map f l)
  (if (null? l) '()
      (cons (f (car l)) (map f (cdr l)))))
```

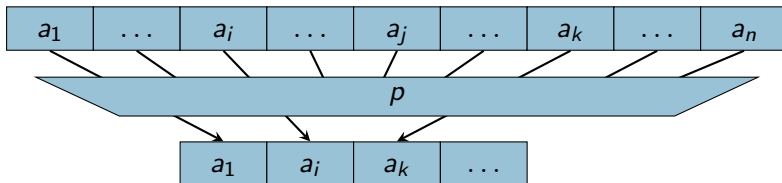
Изобразяване на списък (map) — примери

```
(define (map f l)
  (if (null? l) '()
      (cons (f (car l)) (map f (cdr l)))))
```

- `(map square '(1 2 3))` \longrightarrow `(1 4 9)`
- `(map cadr '((a b c) (d e f) (g h i)))` \longrightarrow `(b e h)`
- `(map (lambda (f) (f 2)) (list square 1+ odd?))` \longrightarrow `(4 3 #f)`

Филтриране на списък (filter)

Да се дефинира функция (**filter** <условие> <списък>), която връща само тези от елементите на <списък>, които удовлетворяват <условие>.



```
(define (filter p? l)
  (cond ((null? l) l)
        ((p? (car l)) (cons (car l) (filter p? (cdr l))))
        (else (filter p? (cdr l)))))
```

Филтриране на списък (filter)

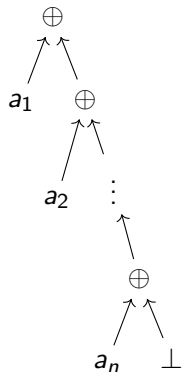
```
(define (filter p? l)
  (cond ((null? l) l)
        ((p? (car l)) (cons (car l) (filter p? (cdr l))))
        (else (filter p? (cdr l)))))
```

- (filter odd? '(1 2 3 4 5)) \longrightarrow (1 3 5)
- (filter pair? '((a b) c () d (e))) \longrightarrow ((a b) (e))
- (map (lambda (x) (filter even? x)) '((1 2 3) (4 5 6) (7 8 9)))
 \longrightarrow ((2) (4 6) (8))
- (map (lambda (x) (map (lambda (f) (filter f x)) (list negative? zero? positive?))) '((-2 1 0) (1 4 -1) (0 0 1)))
 \longrightarrow (((-2) (0) (1)) ((-1) () (1 4)) ((0 0) (1)))

Дясно свиване (foldr)

Да се дефинира функция, която по даден списък $l = (a_1 a_2 a_3 \dots a_n)$ пресмята:

$$a_1 \oplus \left(a_2 \oplus \left(\dots \oplus (a_n \oplus \perp) \dots \right) \right),$$



```
(define (foldr op nv l)
  (if (null? l) nv
      (op (car l) (foldr op nv (cdr l)))))
```


Дясно свиване (foldr) — примери

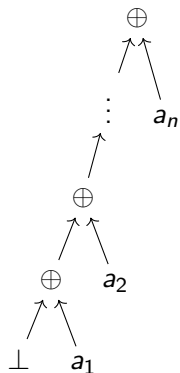
```
(define (foldr op nv l)
  (if (null? l) nv
      (op (car l) (foldr op nv (cdr l)))))
```

- `(foldr * 1 (from-to 1 5))` \rightarrow 120
- `(foldr + 0 (map square (filter odd? (from-to 1 5))))` \rightarrow 35
- `(foldr cons '() '(1 5 10))` \rightarrow (1 5 10)
- `(foldr list '() '(1 5 10))` \rightarrow (1 (5 (10 ())))
- `(foldr append '() '((a b) (c d) (e f)))` \rightarrow (a b c d e f)
- `map`, `filter` и `accumulate` могат да се реализират чрез `foldr`

Ляво свиване (foldl)

Да се дефинира функция, която по даден списък $l = (a_1 a_2 a_3 \dots a_n)$ пресмята:

$$\left(\dots \left((\perp \oplus a_1) \oplus a_2 \right) \oplus \dots \right) \oplus a_n$$



```
(define (foldl op nv l)
  (if (null? l) nv
      (foldl op (op nv (car l)) (cdr l))))
```

Ляво свиване (foldl) — примери

```
(define (foldl op nv l)
  (if (null? l) nv
      (foldl op (op nv (car l)) (cdr l))))
```

- $(\text{foldl } * 1 (\text{from-to } 1 5)) \rightarrow 120$
- $(\text{foldl } \text{cons } '() '(1 5 10)) \rightarrow (((() . 1) . 5) . 10)$
- $(\text{foldl } (\text{lambda } (x y) (\text{cons } y x)) '() '(1 5 10)) \rightarrow (10 5 1)$
- $(\text{foldl } \text{list } '() '(1 5 10)) \rightarrow (((() 1) 5) 10)$
- $(\text{foldl } \text{append } '() '((a b) (c d) (e f))) \rightarrow (a b c d e f)$
- foldr генерира линеен рекурсивен процес, а foldl — линеен итеративен

Функции от по-висок ред в Racket

В R^5RS е дефинирана само функцията `map`.

В Racket са дефинирани функциите `map`, `filter`, `foldr`, `foldl`

Внимание: `foldl` в Racket е дефинирана по различен начин!

`foldl` от лекции

```
(define (foldl op nv l)
  (if (null? l) nv
      (foldl op (op nv (car l))
              (cdr l))))
```

$$\left(\dots \left((\perp \oplus a_1) \oplus a_2 \right) \oplus \dots \right) \oplus a_n$$

`foldl` в Racket

```
(define (foldl op nv l)
  (if (null? l) nv
      (foldl op (op (car l) nv)
              (cdr l))))
```

$$a_n \oplus \left(\dots \left(a_2 \oplus (a_1 \oplus \perp) \right) \dots \right),$$

Свиване на непразен списък (foldr1, foldl1)

Задача. Да се намери максималният елемент на списък.

```
(define (maximum l) (foldr max ? 1))
```

Можем ли да пропуснем нулевата стойност за непразен списък?

$$a_1 \oplus (\dots \oplus (a_{n-1} \oplus a_n) \dots)$$

```
(define (foldr1 op l)
  (if (null? (cdr l)) (car l)
      (op (car l)
          (foldr1 op (cdr l)))))
```

$$(\dots ((a_1 \oplus a_2) \oplus \dots) \oplus a_n$$

```
(define (foldl1 op l)
  (foldl op (car l) (cdr l)))
```

Прилагане на функция над списък от параметри (apply)

- `(apply <функция> <списък>)`
- прилага <функция> над <списък> от параметри
- **Примери:**
- `(apply + '(1 2 3 4 5)) → 15`
- `(apply append '((1 2) (3 4) (5 6))) → (1 2 3 4 5 6)`
- `(apply list '(1 2 3 4)) → (1 2 3 4)`

Оценяване на списък като комбинация (eval)

- `(eval <S-израз> <среда>)`
- връща оценката на <S-израз> в <среда>
- `(interaction-environment)` — текущата среда, в която оценяваме
- `(define (evali x) (eval x (interaction-environment)))`
- **Примери:**
- `(define a 2)`
- $a \longrightarrow 2$
- `(evali a) \longrightarrow 2`
- `(evali 'a) \longrightarrow 2`
- `(evali ''a) \longrightarrow a`
- `(evali (evali ''a)) \longrightarrow 2`

Примери за eval

- $(\text{evali } (\text{list } '+ \ 5 \ 7 \ 'a)) \longrightarrow 14$
- $(\text{evali } (\text{list } 'define \ b \ 5)) \longrightarrow \text{Грешка!}$
- $(\text{evali } (\text{list } 'define \ 'b \ 5)) \iff (define \ b \ 5)$
- $b \longrightarrow 5$
- $(\text{evali } (\text{list } 'if \ (\text{list } '< \ 2 \ 5) \ (\text{list } 'quote \ 'a) \ 'b)) \longrightarrow a$
- $(define \ (\text{apply } f \ l) \ (\text{evali } (\text{cons } f \ l)))$

Програмите на Scheme могат да се разглеждат като данни!