

Списъци

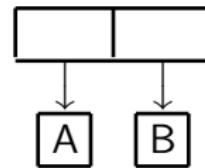
Трифон Трифонов

Функционално програмиране, 2017/18 г.

2 ноември 2017 г.

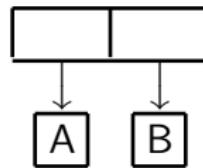
Наредени двойки

(A . B)



Наредени двойки

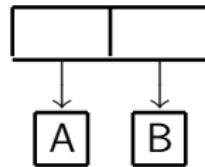
(A . B)



- (cons <израз₁> <израз₂>)

Наредени двойки

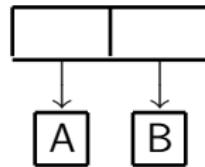
(A . B)



- (`cons <израз1> <израз2>`)
- Наредена двойка от оценките на <израз₁> и <израз₂>

Наредени двойки

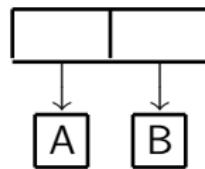
(A . B)



- (`cons <израз1> <израз2>`)
- Наредена двойка от оценките на <израз₁> и <израз₂>
- (`car <израз>`)

Наредени двойки

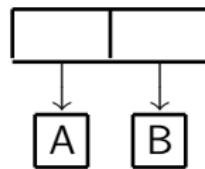
(A . B)



- (`cons <израз1> <израз2>`)
- Наредена двойка от оценките на <израз₁> и <израз₂>
- (`car <израз>`)
- **Първият** компонент на двойката, която е оценката на <израз>

Наредени двойки

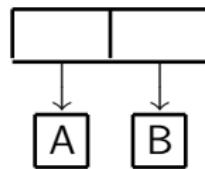
(A . B)



- (`cons <израз1> <израз2>`)
- Наредена двойка от оценките на <израз₁> и <израз₂>
- (`car <израз>`)
- **Първият** компонент на двойката, която е оценката на <израз>
- (`cdr <израз>`)

Наредени двойки

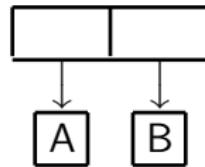
(A . B)



- (`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>

Наредени двойки

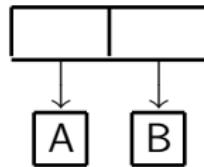
(A . B)



- (`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>`)

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))

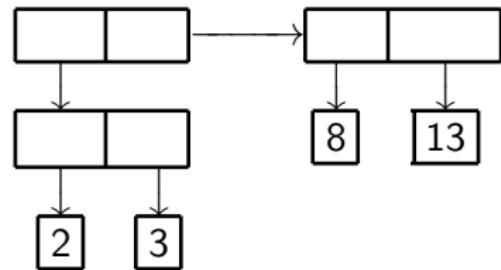


((2 . 3) . (8 . 13))

Примери

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

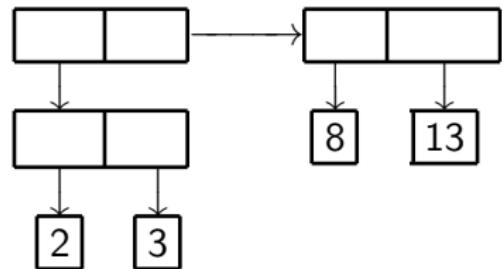
↓
((2 . 3) . (8 . 13))



Примери

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

↓
((2 . 3) . (8 . 13))



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

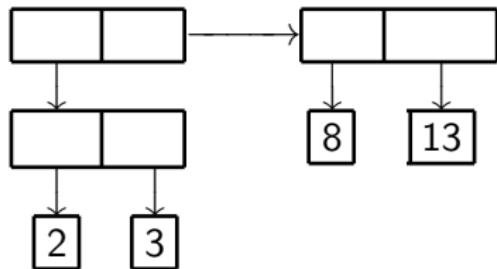
↓
(3 . ((13 . 21) . 8))

Примери

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



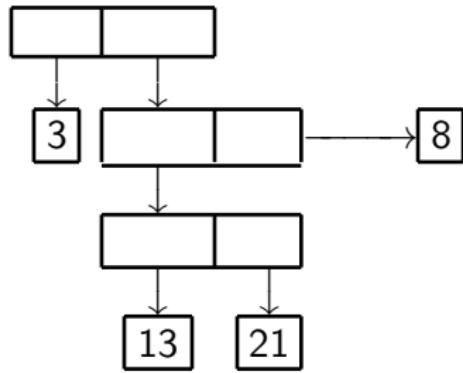
((2 . 3) . (8 . 13))



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



(3 . ((13 . 21) . 8))



All you need is λ — наредени двойки

Можем да симулираме cons, car и cdr чрез lambda!

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))
```

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)))
```

Списъци в Scheme

Дефиниция

- ① Празният списък () е списък
- ② ($h . t$) е списък ако t е списък
 - h — глава на списъка
 - t — опашка на списъка

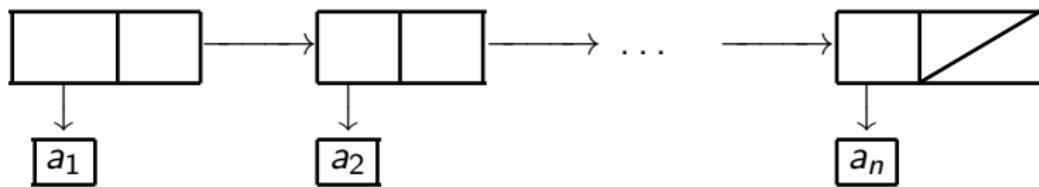
[]

[H | T]

Списъци в Scheme

Дефиниция

- ① Празният списък () е списък
- ② ($h . t$) 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 ()

Вградени функции за списъци

- **(null? <израз>)** — дали <израз> е празният списък ()
- **(list? <израз>)** — дали <израз> е списък

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))))))`

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>

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> ... <израз_n>)** \iff
`(cons <израз1> (cons <израз2> ... (cons <изразn> '()))))`

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> ... <израз_n>)** \iff
`(cons <израз1> (cons <израз2> ... (cons <изразn> '()))))`
- **(cons <глава> <опашка>)** — списък с <глава> и <опашка>

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>`

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> ... <израз_n>) ⇔
(cons <израз₁> (cons <израз₂> ... (cons <израз_n> '()))))
- (**cons** <глава> <опашка>) — списък с <глава> и <опашка>
- (**car** <списък>) — главата на <списък>
- (**cdr** <списък>) — опашката на <списък>

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!

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 $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- (car l) $\longrightarrow a_1$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- (car l) $\longrightarrow a_1$
- (cdr l) $\longrightarrow (a_2 \ a_3 \ \dots \ a_n)$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow ? \longleftarrow (\text{cadr } l)$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$
- $(\text{cdr } (\text{cdr } l)) \longrightarrow ? \longleftarrow (\text{caddr } l)$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$
- $(\text{cdr } (\text{cdr } l)) \longrightarrow (a_3 \ \dots \ a_n) \longleftarrow (\text{caddr } l)$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$
- $(\text{cdr } (\text{cdr } l)) \longrightarrow (a_3 \ \dots \ a_n) \longleftarrow (\text{caddr } l)$
- $(\text{car } (\text{cdr } (\text{cdr } l))) \longrightarrow ? \longleftarrow (\text{caddr } l)$

Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$
- $(\text{cdr } (\text{cdr } l)) \longrightarrow (a_3 \ \dots \ a_n) \longleftarrow (\text{caddr } l)$
- $(\text{car } (\text{cdr } (\text{cdr } l))) \longrightarrow a_3 \longleftarrow (\text{caddr } l)$

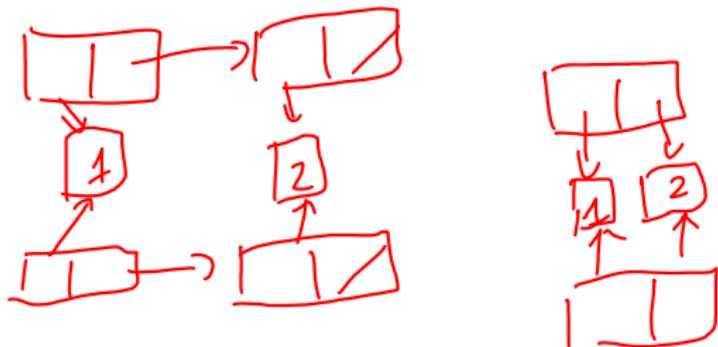
Съкратени форми на car и cdr

Нека $l = (a_1 \ a_2 \ a_3 \ \dots \ a_n)$.

- $(\text{car } l) \longrightarrow a_1$
- $(\text{cdr } l) \longrightarrow (a_2 \ a_3 \ \dots \ a_n)$
- $(\text{car } (\text{cdr } l)) \longrightarrow a_2 \longleftarrow (\text{cadr } l)$
- $(\text{cdr } (\text{cdr } l)) \longrightarrow (a_3 \ \dots \ a_n) \longleftarrow (\text{caddr } l)$
- $(\text{car } (\text{cdr } (\text{cdr } l))) \longrightarrow a_3 \longleftarrow (\text{caddr } l)$
- 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> <израз2> заемат едно и също място в паметта



Форми на равенство в Scheme

- (**eq?** <израз₁> <израз₂>) — връща #t точно тогава, когато оценките на <израз₁> <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 (дори и да заемат различно място в паметта)

Форми на равенство в Scheme

- (**eq?** <израз1> <израз2>) — връща #t точно тогава, когато оценките на <израз1> <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 (дори и да заемат различно място в паметта)
 - Ако (eq? <израз1> <израз2>),
то със сигурност (eqv? <израз1> <израз2>)

Форми на равенство в Scheme

- (**eq?** <израз₁> <израз₂>) — връща #t точно тогава, когато оценките на <израз₁> <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 (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**, чийто компоненти са равни в смисъла на equal?

Форми на равенство в Scheme

- (**eq?** <израз₁> <израз₂>) — връща #t точно тогава, когато оценките на <израз₁> <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 (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**, чийто компоненти са равни в смисъла на equal?
 - В частност, equal? проверява за равенство на списъци

Форми на равенство в Scheme

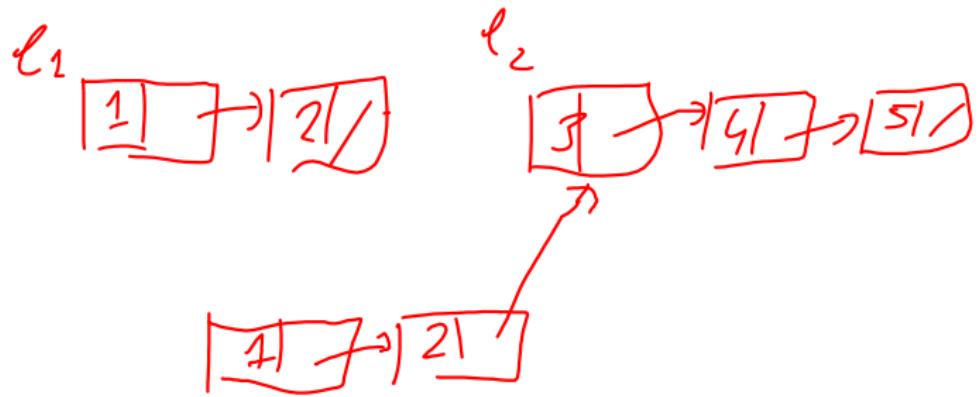
- (**eq?** <израз1> <израз2>) — връща #t точно тогава, когато оценките на <израз1> <израз2> заемат едно и също място в паметта
- (**eqv?** <израз1> <израз2>) — връща #t точно тогава, когато оценките на <израз1> и <израз2> заемат едно и също място в паметта или са едни и същи по стойност атоми (дори и 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**, чийто 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>

Вградени функции за списъци

- (`length <списък>`) — връща дължината на `<списък>`
- (`append {<списък>}`) — конкатенира всички `<списък>`



Вградени функции за списъци

- **(length <списък>)** — връща дължината на <списък>
- **(append {<списък>})** — конкатенира всички <списък>
- **(reverse <списък>)** — елементите на <списък> в обратен ред

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`

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)