Списъци
част 1: увод

доц. Атанас Семерджиев

1

Съдържание на поредицата

Част 1: Увод

Част 2: Основни операции

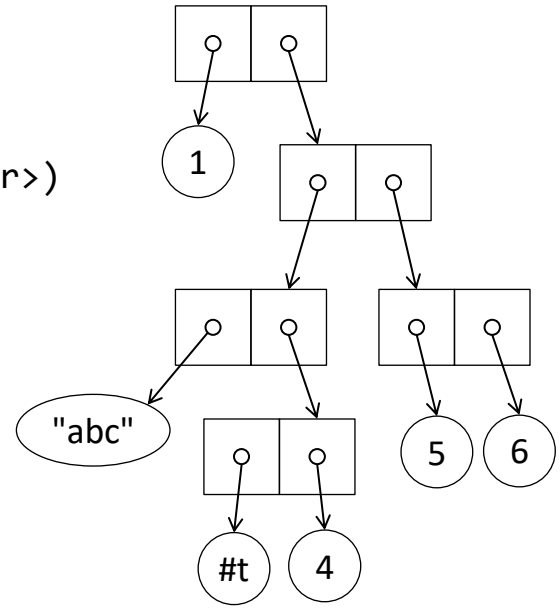
Част 3: Функции от по-висок ред за работа със списъци

2

S-изрази

```
<S-expr> ::=
   <atom> | (<S-expr>.<S-expr>)
```

```
(cons
   1
   (cons
      (cons
         "abc"
         (cons #t 4))
      (cons 5 6))))
```



3

3

Списъци

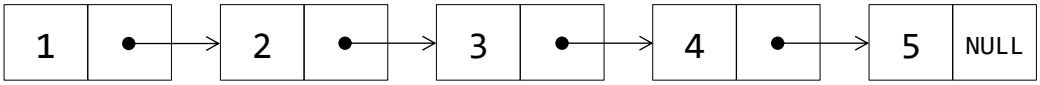
Списъците в Scheme се конструират като поредица от наредени двойки.

Краят на списък се обозначава с празния списък – ( ).

Например бихме могли да дефинираме списъка [1 2 3 4 5] така:

```
(cons 1 (cons 2 (cons 3 (cons 4 (cons 5 '()))))
```

Може да се потърси аналогия със СД свързан списък (въпреки, че двете неща НЕ СА напълно еквивалентни):



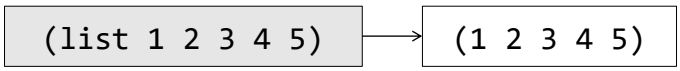
4

4

Специална форма (list)

Обикновено последователното прилагане на `cons` се прилага когато трябва да генерираме списък динамично.

В останалите случаи бихме могли да използваме `list`:



5

5

Терминиращ елемент

ВНИМАНИЕ: Когато генерираме списъка на ръка, често срещана грешка е да се пропусне терминиращия елемент.

Например следните две дефиниции не са еквивалентни:

```
(cons 1 (cons 2 (cons 3 (cons 4 5))))
```

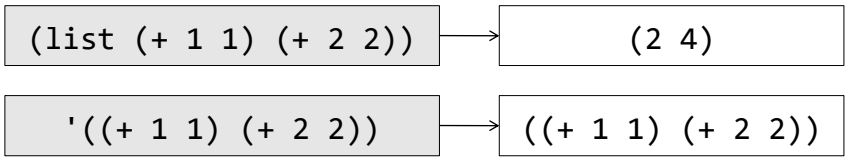
```
(cons 1 (cons 2 (cons 3 (cons 4 (cons 5 '()))))))
```

6

6

(list) и (quote)

Като алтернатива на `list`, бихме могли да използваме `quote`, но между тях има съществена разлика:



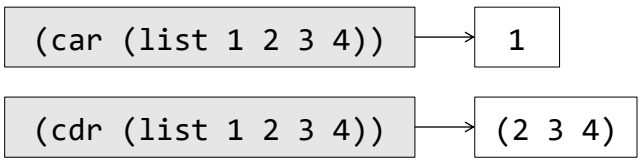
7

7

Обхождане на списък

Обхождането на даден списък се извършва чрез `car` и `cdr`:

- `car` връща първия елемент на списъка (глава);
- `cdr` връща списък от останалите елементи (опашка).



8

8

Многократно прилагане на car и cdr

Обхождането на списък става с многократно прилагане на `car` и `cdr`:

Израз	Оценка
`(car '(1 2 3))`	`1`
`(cdr '(1 2 3))`	`(2 3)`
`(car (cdr '(1 2 3)))`	`2`
`(cdr (cdr '(1 2 3)))`	`(3)`
`(car (cdr (cdr '(1 2 3))))`	`3`
`(cdr (cdr (cdr '(1 2 3))))`	`'()`

9

9

Многократно прилагане на car и cdr

За да избегнем утежняването на записа, бихме могли да използваме съответните кратки форми:

```
(define x '(1 2 3 4))
```

`(car (cdr x))`	⟺	`(cadr x)`
`(cdr (cdr x))`	⟺	`(cddr x)`
`(car (cdr (cdr x)))`	⟺	`(caddr x)`

10

10

Предикат (null?)

`null?` е истина само за празния списък.

```
(null? #f)  → #f
(null? "")  → #f
(null? 0)   → #f
(null? '()) → #t
(null? (cons 1 2)) → #f
(null? (list 1 2 3 4)) → #f
```

11

11

Предикат (pair?)

Чрез `pair?` Можем да различим наредените двойки (и в частност списъците) от атомите.

Забележете, че празният списък също е атом!

```
(pair? #f)  → #f
(pair? "")  → #f
(pair? 0)   → #f
(pair? '()) → #f
(pair? (cons 1 2)) → #t
(pair? (list 1 2 3 4)) → #t
```

12

12

Предикат (atom?)

Понякога е удобно да имаме предикат и за обратната проверка:

```
(atom? 0) → #t
(atom? '()) → #t
(atom? (cons 1 2)) → #f
(atom? (list 1 2 3 4)) → #f
```

```
(define (atom? x)
   (not (pair? x))
)
```

13

13

14