Основни понятия в Scheme

Трифон Трифонов

Функционално програмиране, 2018/19 г.

3–10 октомври 2018 г.

Що за език е Scheme?

- Създаден през 1975 г. от Guy L. Steele и Gerald Jay Sussman
- Диалект на LISP, създаден с учебна цел
- “Structure and Interpretation of Computer Programs”, Abelson & Sussman, MIT Press, 1985.
- Минималистичен синтаксис
- Най-разпространен стандарт: R^5RS

Програмиране на Scheme

- Среда за програмиране: DrRacket
- Има компилатори и интерпретатори
  - Ние ще ползваме интерпретатор
- REPL = Read-Eval-Print-Loop
- Програма = списък от дефиниции
- Изпълнение = оценка на израз

Синтаксис в Scheme

- Литерали
  - Булеви константи (`#f`, `#t`)
  - Числови константи (`15`, `2/3`, `-1.532`)
  - Знакови константи (`#\a`, `#\newline`)
  - Низови константи (`"Scheme"`, `"hi "`)
- Символи (`f`, `square`, `+`, `find-min`)
- Комбинации

  (<израз$_1$> <израз$_2$> . . . <израз$_n$>)

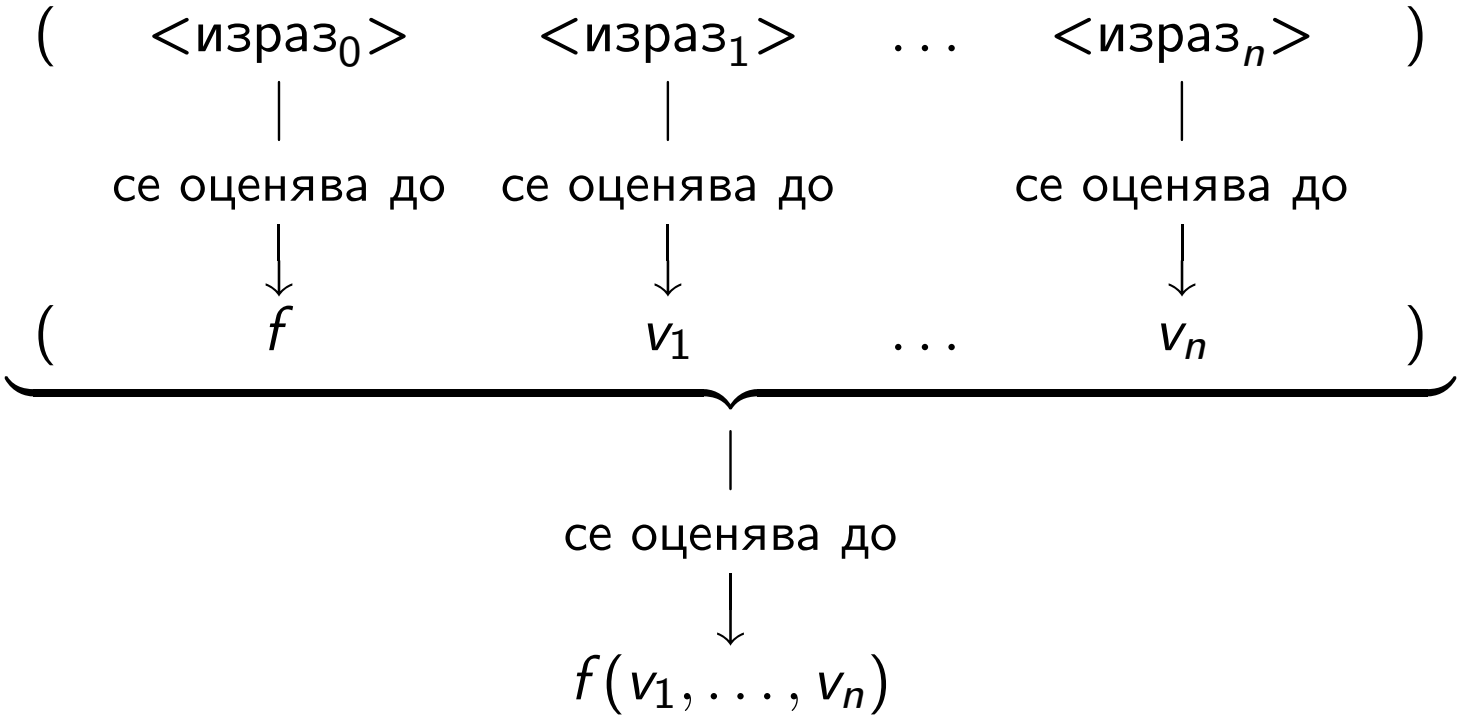
Оценки на литерали и символи

На всеки израз се дава оценка.

- Оценката на булевите константи, знаците, числата и низовете са самите те
  - `5` ⟶ `5`
  - `#t` ⟶ `#t`
  - `#\a` ⟶ `#\a`
  - `"scheme"` ⟶ `"scheme"`
- Оценката на символ е стойността, свързана с него
  - `+` ⟶ `#<procedure:+>`
  - `a` ⟶ Грешка!
  - `(define a 5)`
  - `a` ⟶ `5`

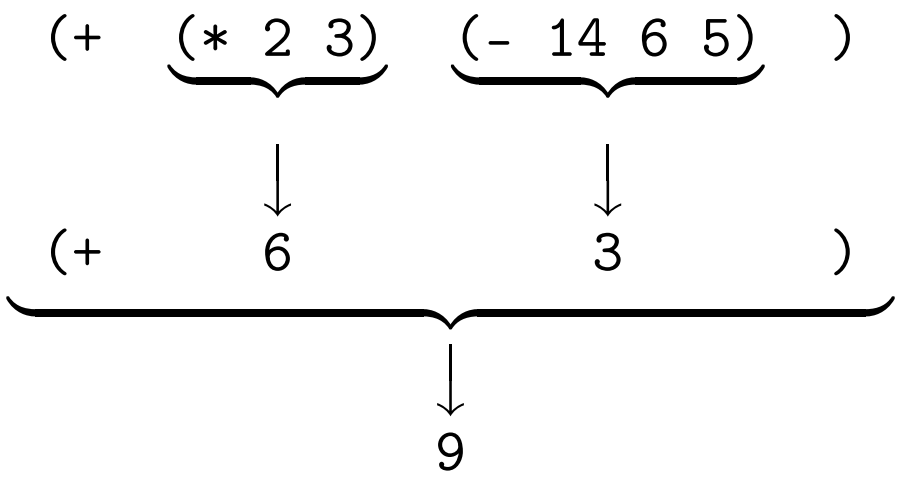
Основно правило за оценяване

Оценка на комбинация **(основно правило за оценяване)**



Ако f не е функция — грешка!

Пример за оценяване на комбинации



(1 2 3) ⟶ Грешка!

Дефиниция на символи

- `(define` <символ> <израз>`)`
- Оценява <израз> и свързва <символ> с оценката му.
- Примери:
  - `(define s "Scheme is cool")`
  - `s` $\longrightarrow$ `"Scheme is cool"`
  - `(define x 2.5)`
  - `x` $\longrightarrow$ `2.5`
  - `(+ x 3.2)` $\longrightarrow$ `5.7`
  - `(define y (+ x 3.2))`
  - `(> y 3)` $\longrightarrow$ `#t`
  - `(define z (+ y z))` $\longrightarrow$ Грешка!

Специални форми

- По основното правило ли се оценява `(define x 2.5)`?
- Не!
- В синтаксиса на Scheme има конструкции, които са изключение от стандартното правило
- Такива конструкции се наричат **специални форми**
- `define` е пример за специална форма

Цитиране

- `(quote` <израз>`)`
- Алтернативен запис: `'`<израз>
- Оценката на `(quote` <израз>`)` или `'`<израз> е самият <израз>
- **Примери:**
  - `'2` $\longrightarrow$ `2`
  - `'+` $\longrightarrow$ `+`
  - `'(+ 2 3)` $\longrightarrow$ `(+ 2 3)`
  - `(quote quote)` $\longrightarrow$ `quote`
  - `('+ 2 3)` $\longrightarrow$ Грешка!
  - `(/ 2 0)` $\longrightarrow$ Грешка!
  - `'(/ 2 0)` $\longrightarrow$ `(/ 2 0)`
  - `'(+ 1 '(* 3 4))` $\longrightarrow$ `(+ 1 (quote (* 3 4)))`

Дефиниция на функции

- `(define (`<функция> {<параметър>}`)` <тяло>`)`
- <функция> и <параметър> са символи
- <тяло> е <израз>
- Символът <функция> се свързва с поредица от инструкции, които пресмятат <тяло> при подадени стойности на <параметър>

Примери за дефиниция на функции

- `(define (square x) (* x x))`
- `(square 5)` $\longrightarrow$ `25`
- `(define (1+ k) (+ k 1))`
- `(square (1+ (square 3)))` $\longrightarrow$ `100`
- `(define (f x y) (+ (square (1+ x)) (square y) 5))`
- `(f 2 4)` $\longrightarrow$ `30`
- `(define (g x) (- (g (+ x 1)) 1))`
- `(g 0)` $\longrightarrow$ `...`
- `(define (h) (+ 2 3))`
- `h` $\longrightarrow$ `#<procedure:h>`
- `(h)` $\longrightarrow$ `5`

Оценяване на комбинации с дефинирани функции