

Модел на средите и изчислителни процеси

Трифон Трифонов

Функционално програмиране, 2018/19 г.

10 октомври 2018 г.

Среди в Scheme

- Връзката между символите и техните оценки се записват в речник, който се нарича **среда**.
- Всеки символ има най-много една оценка в дадена среда.
- В даден момент могат да съществуват много среди.
- Символите винаги се оценяват в една конкретна среда.
- **Символите могат да има различни оценки в различни среди.**
- При стартиране Scheme по подразбиране работи в **глобалната среда**.
- В глобалната среда са дефинирани символи за стандартни операции и функции.

Пример за среда

- `(define a 8)`
- `r` \rightarrow **Грешка!**
- `(define r 5)`
- `(+ r 3)` \rightarrow 8
- `(define (f x) (* x r))`
- `(f 3)` \rightarrow 15
- `(f r)` \rightarrow 25



Функции и среди

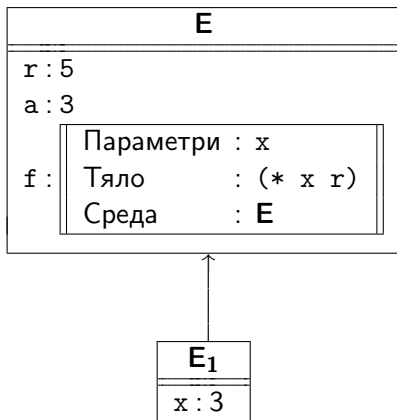
- Всяка функция f пази указател към средата E , в която е дефинирана.
- При извикване на f :
 - създава се нова среда E_1 , която разширява E
 - в E_1 всеки символ означаващ формален параметър се свързва с оценката на фактическия параметър
 - тялото на f се оценява в E_1

Дърво от среди

- Всяка среда пази указател към своя “родителска среда”, която разширява
- така се получава дърво от среди
- при оценка на символ в дадена среда **E**
 - първо се търси оценката му в **E**
 - ако символът не е дефиниран в **E**, се преминава към родителската среда
 - при достигане на най-горната среда, ако символът не е дефиниран и в нея се извежда съобщение за грешка

Извикване на дефинирана функция

- `(define r 5)`
- `(define a 3)`
- `(define (f x) (* x r))`
- $\{E\}$ `(f a)`
 ↓
 $\{E\}$ `(f 3)`
 ↓
 $\{E_1\}$ `(* x r)`
 ↓
 15



Какво е рекурсия?



Какво е рекурсия?



Какво е рекурсия?

Повторение чрез позоваване на себе си

Рекурсивна функция: дефинира се чрез себе си

$$n! = \begin{cases} 1, & \text{при } n = 0, & \text{(база)} \\ n \cdot (n - 1)!, & \text{при } n > 0. & \text{(стъпка)} \end{cases}$$

- Дава се отговор на най-простата задача (база, дъно)
- Показва се как сложна задача се свежда към една или няколко по-прости задачи от същия вид (стъпка)

Рекурсивни уравнения

Какво означава “да дефинираме функция чрез себе си”?

Да разгледаме *рекурсивното уравнение*, в което F е неизвестно:

$$F(n) = \underbrace{\begin{cases} 1, & \text{при } n = 0, \\ n \cdot F(n - 1), & \text{при } n > 0. \end{cases}}_{\Gamma(F)(n)}$$

$n!$ е “най-малкото” решение на уравнението $F = \Gamma(F)$.

Най-малка неподвижна точка

Теорема (Knaster-Tarski)

Ако Γ е изчислим оператор, то уравнението $F = \Gamma(F)$ има единствено най-малко решение f (най-малка неподвижна точка на Γ). Нещо повече, решението точно съответства на рекурсивна програма пресмятаща f чрез Γ .

```
(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))
```

Кое е най-малкото решение на уравнението $F(x) = 1 + F(x - 1)$?

```
(define (f x) (+ 1 (f (- x 1))))
(f 0) → ?
```

f е “празната функция”, т.е. $\text{dom}(f) = \emptyset$.

Операционна и денотационна семантика

Два подхода за описание на смисъла на функциите в Scheme.
Нека $(\text{define } (f\ x) \Gamma[f])$ е рекурсивно дефинирана функция.
Коя е математическата функция f , която се пресмята от f ?

Денотационна семантика

f е най-малката неподвижна точка на уравнението $F = \Gamma(F)$.

Операционна семантика

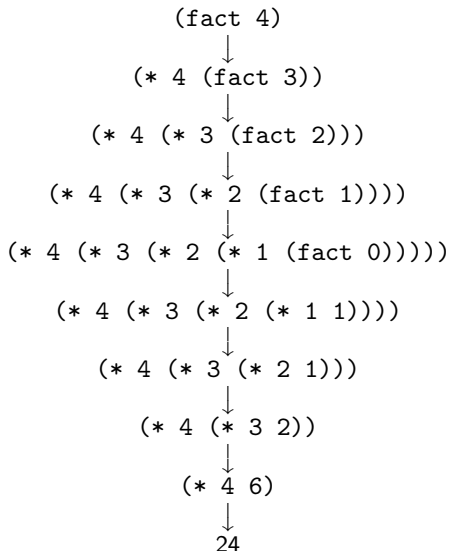
Разглеждаме редицата от последователни оценки на комбинации
 $(f\ a) \rightarrow \Gamma[f][x \mapsto a] \rightarrow \dots$

Ако стигнем до елемент b , който не е комбинация, то $f(a) := b$.

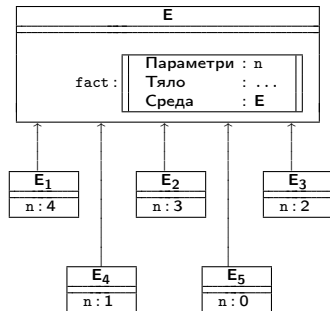
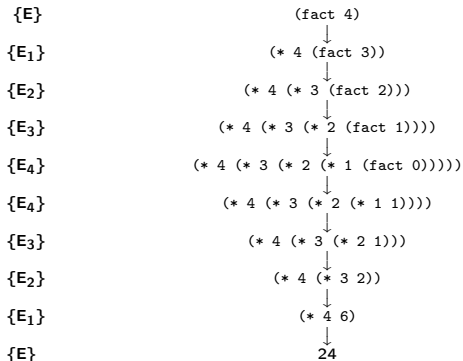
Функциите в Scheme имат дуален, но еквивалентен смисъл:

- решения на рекурсивни уравнения
- изчислителни процеси, генериращи се при оценка

Оценка на рекурсивна функция



Оценка на рекурсивна функция в среда



Линеен рекурсивен процес

Факториел с цикъл

Факториел на C++

```

int fact(int n) {
    int r = 1;
    for(int i = 1; i <= n; i++)
        r *= i;
    return r;
}

```

Превод на Scheme

```

(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))

(define (fact n)
  (for n 1 1))

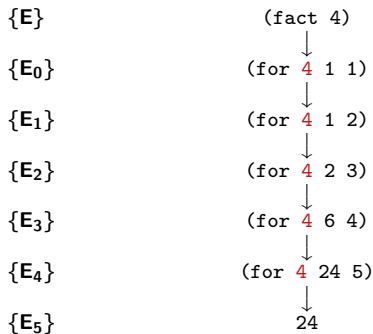
```

Оценка на итеративен факториел

```
(fact 4)
  ↓
(for 4 1 1)
  ↓
(for 4 1 2)
  ↓
(for 4 2 3)
  ↓
(for 4 6 4)
  ↓
(for 4 24 5)
  ↓
24
```

Линеен итеративен процес

Оценка на итеративен факториел със среди



Рекурсивен и итеративен процес

```

      (fact 4)
        ↓
    (* 4 (fact 3))
        ↓
  (* 4 (* 3 (fact 2)))
        ↓
(* 4 (* 3 (* 2 (fact 1))))
        ↓
(* 4 (* 3 (* 2 (* 1 (fact 0)))))
        ↓
  (* 4 (* 3 (* 2 (* 1 1))))
        ↓
    (* 4 (* 3 (* 2 1)))
        ↓
      (* 4 (* 3 2))
        ↓
        (* 4 6)
          ↓
            24
  
```

```

(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))
  
```

```

      (fact 4)
        ↓
    (for 4 1 1)
        ↓
    (for 4 1 2)
        ↓
    (for 4 2 3)
        ↓
    (for 4 6 4)
        ↓
    (for 4 24 5)
        ↓
            24
  
```

```

(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))
  
```

```

(define (fact n)
  (for n 1 1))
  
```

Опашкова рекурсия

- Функциите, в които има отложени операции генерират същински **рекурсивни процеси**
- Рекурсивно извикване, при което няма отложена операция се нарича **опашкова рекурсия**
- Функциите, в които всички рекурсивни извиквания са опашкови генерират **итеративни процеси**
- При итеративните процеси липсва етап на свиването на рекурсията
- Опашковата рекурсия се използва за симулиране на цикли
- В Scheme опашковата рекурсия **по стандарт** се интерпретира като цикъл
 - т.е. не се заделя памет за всяко рекурсивно извикване

Рекурсивен и итеративен процес

```

      (fact 4)
        ↓
      (* 4 (fact 3))
        ↓
      (* 4 (* 3 (fact 2)))
        ↓
      (* 4 (* 3 (* 2 (fact 1))))
        ↓
      (* 4 (* 3 (* 2 (* 1 (fact 0)))))
        ↓
      (* 4 (* 3 (* 2 (* 1 1))))
        ↓
      (* 4 (* 3 (* 2 1)))
        ↓
      (* 4 (* 3 2))
        ↓
      (* 4 6)
        ↓
      24
  
```

```

(define (fact n)
  (if (= n 0) 1
      (* n (fact (- n 1)))))
  
```

```

      (fact 4)
        ↓
      (for 4 1 1)
        ↓
      (for 4 1 2)
        ↓
      (for 4 2 3)
        ↓
      (for 4 6 4)
        ↓
      (for 4 24 5)
        ↓
      24
  
```

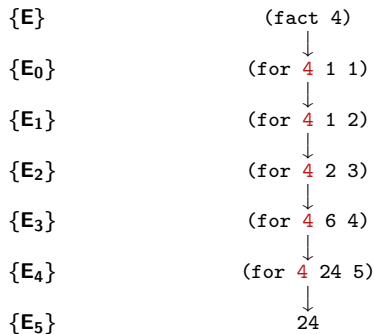
```

(define (for n r i)
  (if (<= i n)
      (for n (* r i) (+ i 1))
      r))
  
```

```

(define (fact n)
  (for n 1 1))
  
```

Оценка на итеративен факториел със среди



Вложени дефиниции

- `(define (<функция> {<параметър>} {<дефиниция>} <тяло>)`
- При извикване на <функция> първо се оценяват всички <дефиниция> и след това се оценява <тяло>
- Вложените дефиниции се оценяват и записват в средата, която се оценява функцията, а не в средата, в която е дефинирана
- Пример:

```
(define (dist x1 y1 x2 y2)
  (define dx (- x2 x1))
  (define dy (- y2 y1))
  (define (sq x) (* x x))
  (sqrt (+ (sq dx) (sq dy))))
```

Оценка на вложени функции

```

{E}      (dist 2 5 -1 9)
          ↓
{E1}   (define dx (- x2 x1))
{E1}   (define dy (- y2 y1))
{E1}   (define (sq x) (* x x))
{E1}   (sqrt (+ (sq dx) (sq dy)))
          ↓
{E2}   (sqrt (+ (* x x) (sq dy)))
          ↓
{E3}   (sqrt (+ 9 (* x x)))
          ↓
{E1}   (sqrt (+ 9 16))
          ↓
{E1}   (sqrt 25)
          ↓
{E1}   5
  
```



Вложена помощна итеративна функция

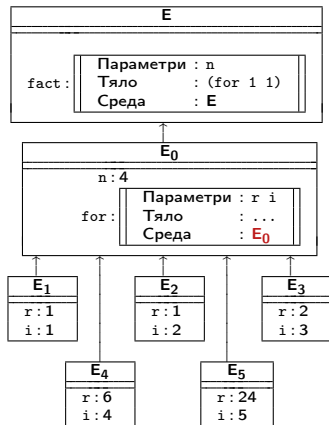
При итеративни функция е удобно помощната функция да е вложена.

```
(define (fact n)
  (define (for r i)
    (if (<= i n)
        (for (* r i) (+ i 1))
        r))
  (for 1 1))
```

Вложените дефиниции “виждат” символите на обхващащите им дефиниции.

Оценка на итеративен факториел с вложена функция

{E}	(fact 4)
	↓
{E ₀ }	(for 1 1)
	↓
{E ₁ }	(for 1 2)
	↓
{E ₂ }	(for 2 3)
	↓
{E ₃ }	(for 6 4)
	↓
{E ₄ }	(for 24 5)
	↓
{E ₅ }	24



Специална форма let

- $(\text{let } (\{(\langle \text{символ} \rangle \langle \text{израз} \rangle)\}) \langle \text{тяло} \rangle)$
- $(\text{let } ((\langle \text{символ}_1 \rangle \langle \text{израз}_1 \rangle)$
 $(\langle \text{символ}_2 \rangle \langle \text{израз}_2 \rangle)$
 ...
 $(\langle \text{символ}_n \rangle \langle \text{израз}_n \rangle))$
 $\langle \text{тяло} \rangle)$
- При оценка на let в среда E :
 - Създава се нова среда E_1 разширение на текущата среда E
 - Оценката на $\langle \text{израз}_1 \rangle$ в E се свързва със $\langle \text{символ}_1 \rangle$ в E_1
 - Оценката на $\langle \text{израз}_2 \rangle$ в E се свързва със $\langle \text{символ}_2 \rangle$ в E_1
 - ...
 - Оценката на $\langle \text{израз}_n \rangle$ в E се свързва със $\langle \text{символ}_n \rangle$ в E_1
 - Връща се оценката на $\langle \text{тяло} \rangle$ в средата E_1
- let няма странични ефекти върху средата!
 - за разлика от define

Пример за let

```
(define (dist x1 y1 x2 y2)
  (let ((dx (- x2 x1))
        (dy (- y2 y1)))
    (sqrt (+ (sq dx) (sq dy)))))
```

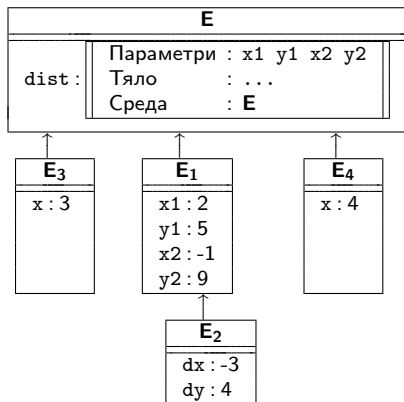
```
(define (area x1 y1 x2 y2 x3 y3)
  (let ((a (dist x1 y1 x2 y2))
        (b (dist x2 y2 x3 y3))
        (c (dist x3 y3 x1 y1))
        (p (/ (+ a b c) 2)))
    (sqrt (* p (- p a) (- p b) (- p c)))))
```

Оценка на let

```

{E}      (dist 2 5 -1 9)
          ↓
{E1}  (let ((dx (- x2 x1))
              (dy (- y2 y1)))
          (sqrt (+ (sq dx) (sq dy))))
          ↓
{E2}  (sqrt (+ (sq dx) (sq dy)))
          ↓
{E3}  (sqrt (+ (* x x) (sq dy)))
          ↓
{E4}  (sqrt (+ 9 (* x x)))
          ↓
{E2}  (sqrt (+ 9 16))
          ↓
{E2}  (sqrt 25)
          ↓
{E2}  5

```



Специална форма let*

- $(\text{let}^* (\{(\langle \text{символ} \rangle \langle \text{израз} \rangle)\}) \langle \text{тяло} \rangle)$
- $(\text{let}^* ((\langle \text{символ}_1 \rangle \langle \text{израз}_1 \rangle)$
 $(\langle \text{символ}_2 \rangle \langle \text{израз}_2 \rangle)$
 ...
 $(\langle \text{символ}_n \rangle \langle \text{израз}_n \rangle))$
 $\langle \text{тяло} \rangle)$
- При оценка на let^* в среда \mathbf{E} :
 - Създава се нова среда \mathbf{E}_1 разширение на текущата среда \mathbf{E}
 - Оценката на $\langle \text{израз}_1 \rangle$ в \mathbf{E} се свързва със $\langle \text{символ}_1 \rangle$ в \mathbf{E}_1
 - Създава се нова среда \mathbf{E}_2 разширение на текущата среда \mathbf{E}_1
 - Оценката на $\langle \text{израз}_2 \rangle$ в \mathbf{E}_1 се свързва със $\langle \text{символ}_2 \rangle$ в \mathbf{E}_2
 - ...
 - Създава се нова среда \mathbf{E}_n разширение на текущата среда \mathbf{E}_{n-1}
 - Оценката на $\langle \text{израз}_n \rangle$ в \mathbf{E}_{n-1} се свързва със $\langle \text{символ}_n \rangle$ в \mathbf{E}_n
 - Връща се оценката на $\langle \text{тяло} \rangle$ в средата \mathbf{E}_n

Пример за let*

```
(define (area x1 y1 x2 y2 x3 y3)
  (let* ((a (dist x1 y1 x2 y2))
         (b (dist x2 y2 x3 y3))
         (c (dist x3 y3 x1 y1))
         (p (/ (+ a b c) 2)))
```

Редът има значение!

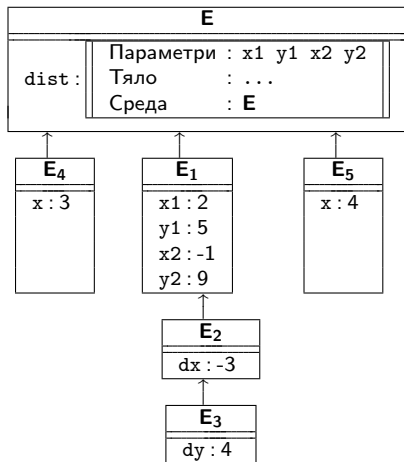
```
(define (area x1 y1 x2 y2 x3 y3)
  (let* ((p (/ (+ a b c) 2))
         (a (dist x1 y1 x2 y2))
         (b (dist x2 y2 x3 y3))
         (c (dist x3 y3 x1 y1)))
    (sqrt (* p (- p a) (- p b) (- p c)))))
```

Оценка на let*

```

{E}      (dist 2 5 -1 9)
           ↓
{E1}  (let* ((dx (- x2 x1))
              (dy (- y2 y1)))
          (sqrt (+ (sq dx) (sq dy))))
           ↓
{E3}  (sqrt (+ (sq dx) (sq dy)))
           ↓
{E4}  (sqrt (+ (* x x) (sq dy)))
           ↓
{E5}  (sqrt (+ 9 (* x x)))
           ↓
{E3}  (sqrt (+ 9 16))
           ↓
{E3}  (sqrt 25)
           ↓
{E3}  5

```



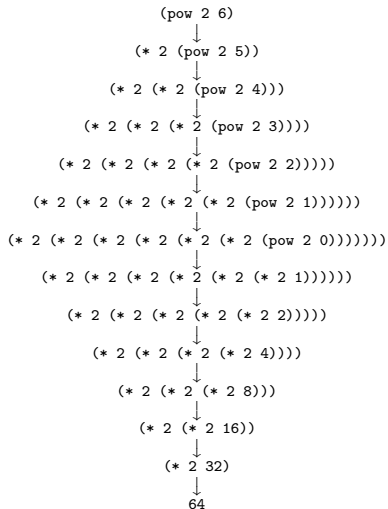
Степенуване

Функцията x^n може да се дефинира по следния начин:

$$x^n = \begin{cases} 1, & \text{ако } n = 0, \\ \frac{1}{x^{-n}}, & \text{ако } n < 0, \\ x \cdot x^{n-1}, & \text{ако } n > 0. \end{cases}$$

```
(define (pow x n)
  (cond ((= n 0) 1)
        ((< n 0) (/ 1 (pow x (- n))))
        (else (* x (pow x (- n 1))))))
```


Оценка на степенуване



Линеен рекурсивен процес

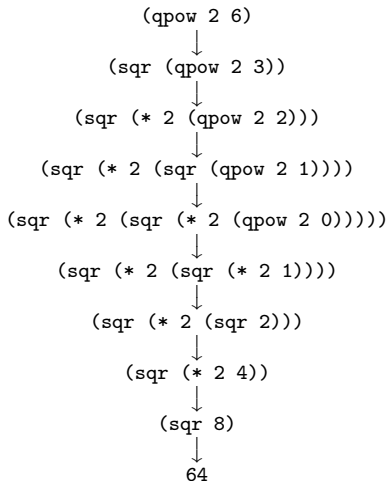
Бързо степенуване

Алтернативна дефиниция на x^n :

$$x^n = \begin{cases} 1, & \text{ако } n = 0, \\ \frac{1}{x^{-n}}, & \text{ако } n < 0, \\ (x^{\frac{n}{2}})^2, & \text{ако } n > 0, n \text{ — четно,} \\ x \cdot x^{n-1}, & \text{ако } n > 0, n \text{ — нечетно.} \end{cases}$$

```
(define (qpow x n)
  (define (sqr x) (* x x))
  (cond ((= n 0) 1)
        ((< n 0) (/ 1 (qpow x (- n))))
        ((even? n) (sqr (qpow x (quotient n 2))))
        (else (* x (qpow x (- n 1))))))
```

Оценка на бързо степенуване



Логаритмичен рекурсивен процес

Числа на Фибоначи

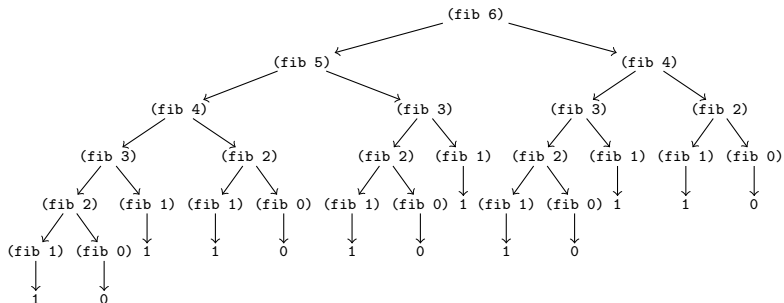
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...

$$f_n = \begin{cases} 0, & \text{за } n = 0, \\ 1, & \text{за } n = 1, \\ f_{n-1} + f_{n-2}, & \text{за } n \geq 2. \end{cases}$$

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))
```

$f_{40} = ?$

Дървовидна рекурсия



Дървовиден рекурсивен процес