ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ

Магдалина Тодорова
magda@fmi.uni-sofia.bg
todorova_magda@hotmail.com
кабинет 517, ФМИ

Тема 4

Ламбда изрази и локални дефиниции.

Процедурите като върнати оценки

1. Ламбда изрази

В Лисп е дадена специална форма **lambda**, чрез която се дефинират т. нар. „анонимни функции“ („анонимни процедури“) или функции без имена.

Специална форма lambda

Взаимствана е от λ-смятането – математически формализъм, въведен от Алонсо Чърч (1941 год.).

Специална форма lambda

Синтаксис:

(lambda (<формални_параметри>) <тяло>)

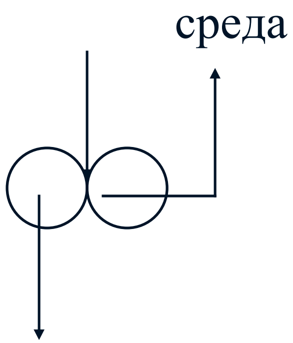
където:

- <формални_параметри> е редица от различни символи, отделени с бял знак. Означават имената на аргументите, използвани в тялото на lambda-израза.

- <тяло> е израз или редица от изрази и определя оценката на дефинираната безименна процедура.

Семантика:

В резултат на оценяването на обръщението към ***lambda*** в някаква *среда* се получава процедура (процедурен обект), която не се свързва с име в средата, в която се извършва оценката. Тази процедура става оценка на обръщението към ***lambda***-израза.



параметри: <формални_параметри>
тяло: <тяло>

1. Ламбда изрази

Примери:

λ : x -> (x+1)	**(lambda (x) (+ x 1))**
λ : x -> (x+4)	**(lambda (x) (+ x 4))**
λ : x -> 1/(x*(x+2))	**(lambda (x) (/ 1 (* x (+ x 2))))**
λ : x, y -> x+y	**(lambda (x y) (+ x y))**

Пример:

Процедурата

```
(define (sum_pi a b)
   (define (pi_term x)
      (/ 1 (* x (+ x 2))))
   (define (pi_next x)
      (+ x 4))
   (sum pi_term a pi_next b))
```

може да се запише по следния начин:

```
(define (sum_pi a b)
   (sum
      (lambda (x) (/ 1 (* x (+ x 2))))
      a
      (lambda (x) (+ x 4))
      b)
)
```

Връзка между дефинирането на процедура чрез *define* и чрез lambda дефиниции

Дефинициите:

(define (<име> <формални параметри>) <тяло>)

и

(define <име> (lambda (<формални параметри>)

<тяло>)

)

са еквивалентни.

Приложения на специалната форма lambda

Специалната форма lambda може да се използва:

а) *вместо име на процедура*

Пример:

```
(define (sum_pi a b)
    (sum (lambda (x) (/ 1 (* x (+ x 2))))
          a
          (lambda (x) (+ x 4))
          b))
```

Приложения на специалната форма lambda

Специалната форма lambda може да се използва:

б) като оператор в комбинация

Пример:

Допустима е комбинацията

((lambda (<var$_1$> <var$_2$> ... <var$_n$>) <тяло>) <expr$_1$> <expr$_2$>... <expr$_n$>)

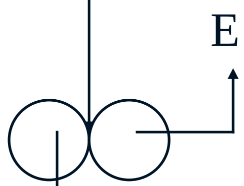
Оценяването на комбинацията:

((lambda (<var_1> <var_2> ... <var_n>) <тяло>) <$expr_1$> <$expr_2$>... <$expr_n$>)

в средата Е се осъществява по апликативния модел (основното правило).

1. Ламбда изрази

((lambda (<var$_1$> <var$_2$> ... <var$_n$>) <тяло>) <expr$_1$> <expr$_2$> ... <expr$_n$>)



параметри: <var$_1$> <var$_2$> ... <var$_n$>

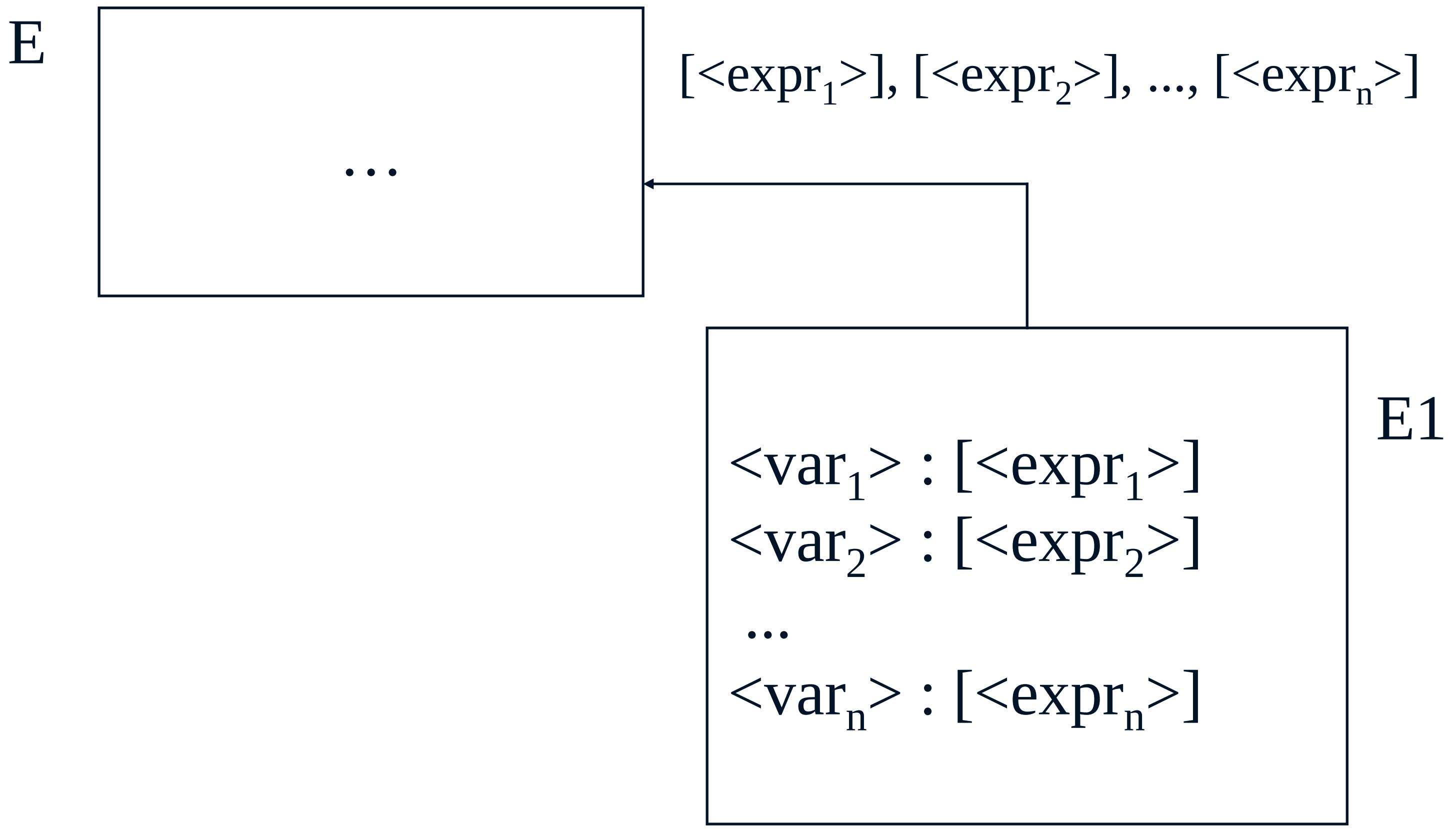
тяло: <тяло>

[<expr$_1$>] [<expr$_2$>] … [<expr$_n$>]

<тяло> в E1

1. Ламбда изрази

((lambda (<var$_1$> <var$_2$> ... <var$_n$>) <тяло>) <expr$_1$> <expr$_2$> ... <expr$_n$>)



2. Локални дефиниции

Забележка. Ламбда изразите са най-общата конструкция на езика, чрез която може да бъде изразена всяка друга.

Примери:

```
(define zero (lambda (f) (lambda (x) x)))

(define (1+ n)
  (lambda (f) (lambda (x) (f ((n f) x)))))
```

Числа на Чърч

1 -> (1+ zero); 2 -> (1+ (1+ zero)) и т.н.

```
(define one (lambda (f) (lambda (x) (f x))))

(define two (lambda (f) (lambda (x) (f (f x)))))
```

2. Локални дефиниции

Събиране на естествени числа

```
(define (add a b)
   (lambda (f)
      (lambda (x) ((a f) ((b f) x)))))
```

2. Локални дефиниции

Използването на ламбда-израз като операция в комбинация се прилага много често. Затова е създадена специална форма **let**, която я прави по-удобна за използване.

Let е друго средство за абстракция в езика.

2. Локални дефиниции

Специална форма let

Синтаксис:

(let ((<var$_1$> <expr$_1$>)
        (<var$_2$> <expr$_2$>)
                ...
        (<var$_n$> <expr$_n$>))
    <тяло>)

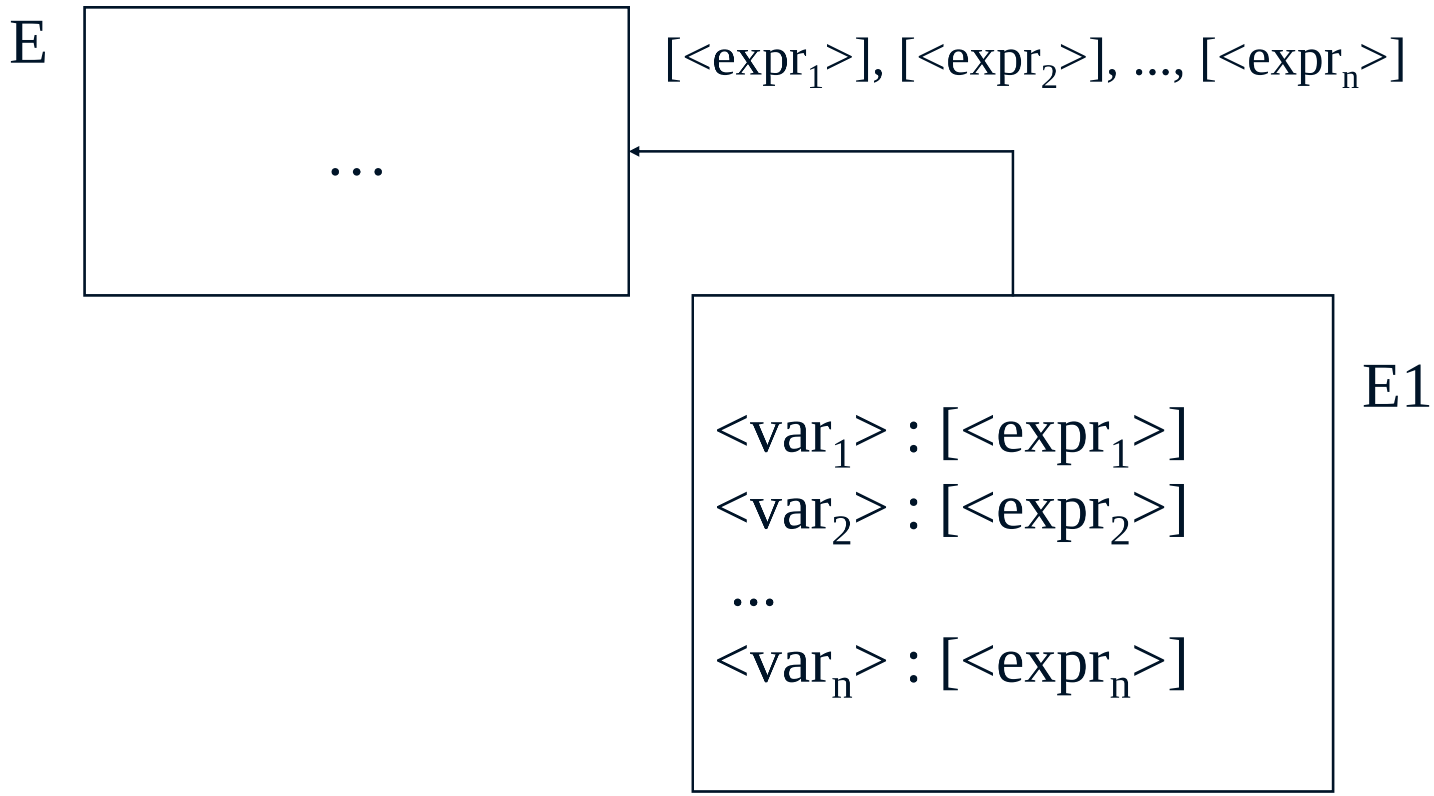
където

- <var$_1$>, <var$_2$>, ..., <var$_n$> са символи;
- <expr$_1$>, <expr$_2$>, ..., <expr$_n$> са произволни изрази;
- <тяло> е израз или редица от изрази.

Семантика:

Нека let-изразът се оценява в средата E. Изразите <$expr_1$>, <$expr_2$>, ... и <$expr_n$> се оценяват в E. Създава се нова среда E1, която е разширение на E. В нея <var_1> се свързва с оценката на <$expr_1$>, <var_2> се свързва с оценката на <$expr_2$>, ... , <var_n> се свързва с оценката на <$expr_n$>, т.е.

2. Локални дефиниции



и в средата E1 се оценява <тяло> на let-израза.

2. Локални дефиниции

Забележка. Оценката на ***let***-израза

$(\text{let }((<\text{var}_1>\ <\text{expr}_1>)$

$(<\text{var}_2>\ <\text{expr}_2>)$

...

$(<\text{var}_n>\ <\text{expr}_n>))$

$<\text{тяло}>)$

или

(let [(<var1> <expr1>)

(<var2> <expr2>)

...

(<varn> <exprn>)]

<тяло>)

в средата E е еквивалентна на оценката на

$((\text{lambda }(<\text{var}_1>\ <\text{var}_2>\ \ldots\ <\text{var}_n>)\ <\text{тяло}>)\ \ <\text{expr}_1>\ \ <\text{expr}_2>\ \ \ldots\ \ <\text{expr}_n>)$

в средата E.

2. Локални дефиниции

Примери:

```
>(let ((x 2))
    (+ x 10))
12
```

```
>(let [(x 5)
       (y 10)
       (z 15)]
    (* x y z))
750
```

```
>(let  ((f -))
    (f 10 5))
5
```

```
>(let ((f +)
       (x 2)
       (y 3))
    (f x y))
5
```

```
>(let ((+ *))
      (+ 2 3))
6
```

2. Локални дефиниции

Задача. Да се дефинира процедура, която пресмята стойността на двуаргументната функция f, дефинирана по следния начин:

$f(x,y) = x(1\text{-}xy)^2(1+y)+ y(1+y)^2 + (1\text{-}xy)(1+y)^3$.

Ако положим:

a = 1-xy

b = 1+y,

то f(x, y) може да бъде записано като

$f(x,y) = xa^2b + yb^2 + ab^3$

Ако

a = 1-xy

b = 1+y,

то

$f(x,y) = xa^2b + yb^2 + ab^3$

Процедурата

```
(define (f x y)
    (let ((a (- 1 (* x y)))
          (b (+ 1 y)))
      (+ (* x (square a) b) (* y (square b)) (* a (cube b)))))
```

решава задачата.

Забележки:

1) Областта на вложените в ***define***–израз процедури е блокът, а областта на дефинираните променливи в let-израз е тялото на ***let***.

Пример:

```
(define (f ... )
   (define (g ... ) ( ...h... ))
   (define (h ... ) ( ... g ... ))
   ( ... ))
```

Област на g и h

е допустима дефиниция.

2. Локални дефиниции

Областта на вложените в *define*–израз процедури е блокът, а областта на дефинираните променливи в let-израз е тялото на ***let***.

Пример:

(define (f ...)

(let ((g ...) ...) (...))

(let ((h ...) ...) (... g ...))

(...))

не се допуска, освен ако няма дефинирано име **g** в обхващащата let (съдържаща обръщението към ***let***) среда.

2) В let-израз, символите <var_1>, <var_2>, ..., <var_n> се свързват с оценки едновременно.

Пример: Оценката на

```
> (define x 2)
> (let ((x 3)
        (y (+ x 2)))
    (* x y))
```

е 12, а не 15.

Специална форма let*

Използва се за реализиране на вложени ***let***-изрази.

Синтаксис:

(let* (($\langle var_1 \rangle$ $\langle expr_1 \rangle$)
($\langle var_2 \rangle$ $\langle expr_2 \rangle$)
...
($\langle var_n \rangle$ $\langle expr_n \rangle$))
<тяло>)

където

- $\langle var_1 \rangle$, $\langle var_2 \rangle$, ..., $\langle var_n \rangle$ са символи;
- $\langle expr_1 \rangle$, $\langle expr_2 \rangle$, ..., $\langle expr_n \rangle$ са произволни изрази;
- <тяло> е израз или редица от изрази.

2. Локални дефиниции

Семантика (неточна): Оценяването на let* израза в средата E се осъществява по следния начин:

Оценява се <expr$_1$> в средата E; <var$_1$> се свързва с получената оценка.

Оценява се <expr$_2$> в средата E; <var$_2$> се свързва с получената оценка. Ако при оценяването на <expr$_2$> в средата E, в него се среща символът <var$_1$>, използва се свързаната вече с <var$_1$> оценка на <expr$_1$> и т.н.

Оценява се <expr$_n$> в средата E; <var$_n$> се свързва с получената оценка. Ако при оценяването на <expr$_n$> в средата E, в него се среща някой от символите <var$_1$>, <var$_2$>, ..., <var$_{n-1}$> използват се свързаните вече с тях оценки.

При тези свързвания се оценява <тяло>.

2. Локални дефиниции

По-точно ***let*-изразът***

(let* ((<var$_1$> <expr$_1$>)
        (<var$_2$> <expr$_2$>)
                ...
        (<var$_n$> <expr$_n$>))
    <тяло>)

е еквивалентен на

(let ((<var$_1$> <expr$_1$>))
  (let ((<var$_2$> <expr$_2$>))
    . . .
    (let ((<var$_n$> <expr$_n$>))
      <тяло>)...))

Оценката на израза

(let ((<var$_1$> <expr$_1$>))

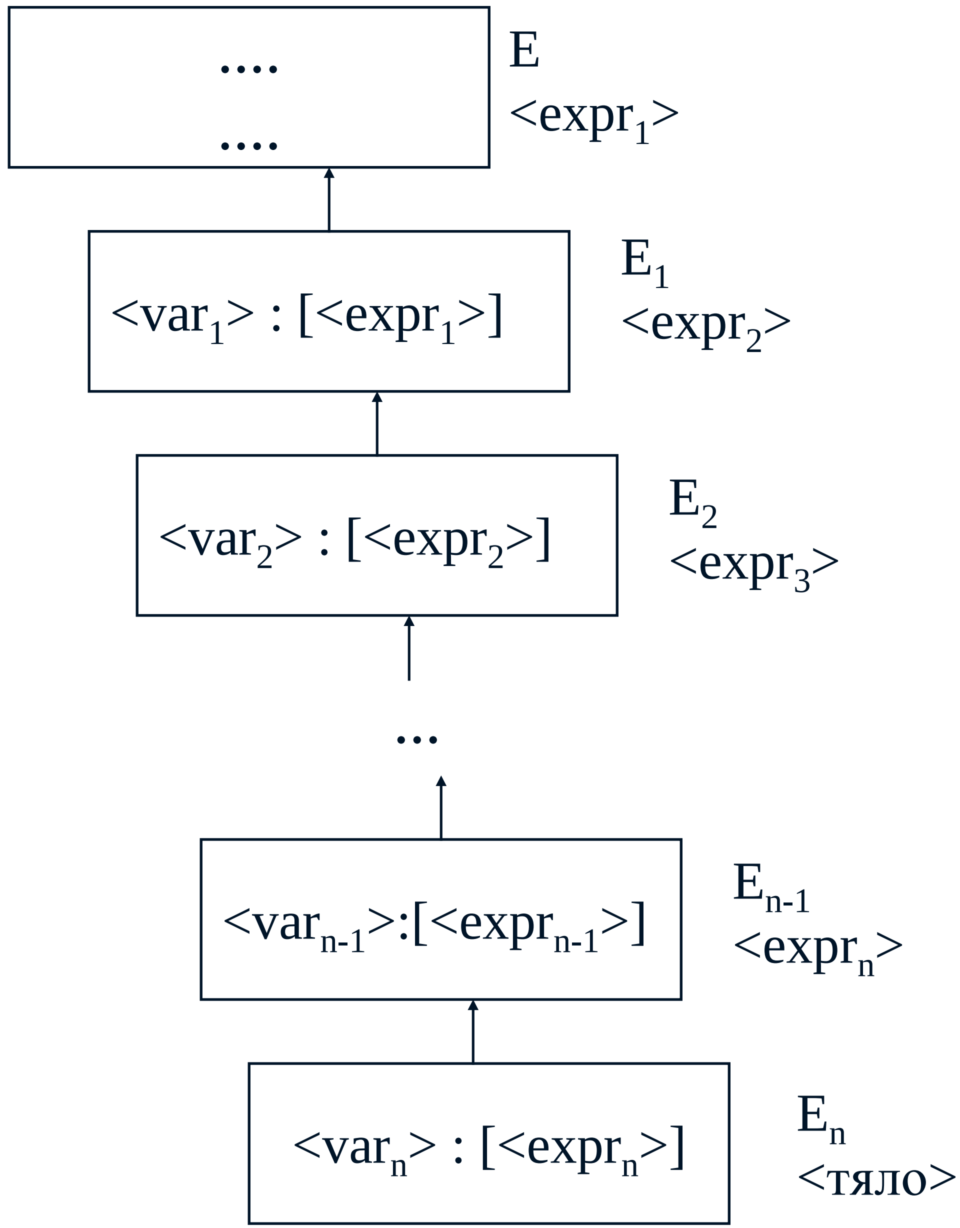
  (let ((<var$_2$> <expr$_2$>))

    . . .

    (let ((<var$_n$> <expr$_n$>))

      <тяло>)...))

в средата E, има вида



Пример: Оценката на let*-израза

```
(define x 2)
(let* ((x 3)
       (y (+ x 2)))
  (* x y))
```

е 15.

2. Локални дефиниции

Специална форма letrec

В ***let*** израза

(let ((<var$_1$> <expr$_1$>)

(<var$_2$> <expr$_2$>)

...

(<var$_n$> <expr$_n$>))

<тяло>)

всички променливи, които се срещат в изразите <expr$_i$>, i = 1, 2, ..., n, трябва да бъдат свързани със стойности извън ***let***-израза, т.е. в обхващащата (съдържащата) го среда.

Изразът

```
(let ((fact (lambda (n) (if (= n 0) 1 (* n (fact (- n 1)))))))
  (fact 4))
```

е некоректен, тъй като подчертаното включване на fact в него не е свързано извън *let*-израза.

2. Локални дефиниции

Проблемът, показан в горния пример може да се преодолее с помощта на специалната форма ***letrec***.

При тази специална форма се извършват локални свързвания, при които рекурсията е допустима.

2. Локални дефиниции

Специална форма letrec (рекурсивен let)

Синтаксис:

(letrec ((<var$_1$> <expr$_1$>)
(<var$_2$> <expr$_2$>)
...
(<var$_n$> <expr$_n$>))
<тяло>)

област на
<var$_1$>, <var$_2$>, ..., <var$_n$>

където

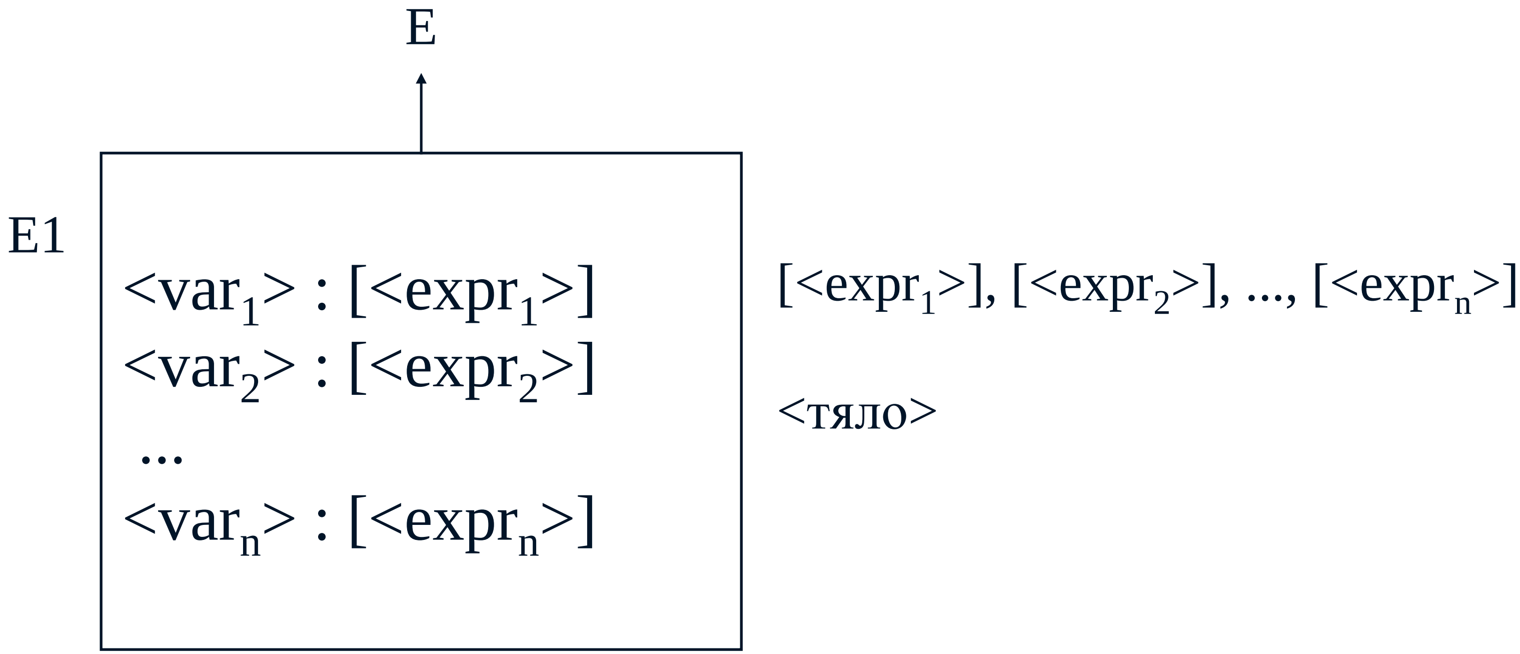
- <var$_1$>, <var$_2$>, ..., <var$_n$> са различни символи;
- <expr$_1$>, <expr$_2$>, ..., <expr$_n$> са изрази, във всеки от които може да се използва всяка от локалните променливи <var$_1$>, <var$_2$>, ..., <var$_n$>;
- <тяло> е израз или редица от изрази.

2. Локални дефиниции

Специална форма letrec

Семантика:

Подобна е на *let*. Различава се от *let* по това, че изразите <expr$_1$>, <expr$_2$>, ..., <expr$_n$> и <тяло> се оценяват в една и съща среда.



Пример. Локално дефиниране на функцията fact чрез letrec.

```
(letrec ((fact (lambda (n) (if (= n 0) 1 (* n (fact (- n 1)))))))
        (fact 4))
```

Пример. Описание на косвена рекурсия с помощта на ***letrec***

```
(letrec ((even? (lambda (n)  (if  (= n 0)  #t  (odd? (- n 1)))))
         (odd?  (lambda (n)  (if  (= n 0)  #f  (even? (- n 1))))) )
  (even? 4) )
```

3. Процедурите като върнати оценки

Задача.

Известно е, че производната на диференцируема функция е функция. Да се дефинира процедура, която намира производната на диференцируемата функция f.

Производната на една диференцируема функция може да се разглежда като оператор

$$\boldsymbol{D: f \to Df}$$

където f и Df са функции.

3. Процедурите като върнати оценки

За да се намери производната на функцията f, може да се разсъждава по следния начин: Нека dx е произволно, достатъчно малко число, тогава производната Df на f е функция, чиято стойност за произволно число x може да се получи по формулата

$$Df(x) = \frac{f(x + dx) - f(x)}{dx}$$

3. Процедурите като върнати оценки

Derive: f, dx $\rightarrow$ ***Df***

$$Df(x) = \frac{f(x + dx) - f(x)}{dx}$$

```
(define (derive f dx)
   (lambda (x)
        (/ (- (f (+ x dx))
              (f x))
        dx)))
```

3. Процедурите като върнати оценки

***Примери*:**

```
> ((derive cube 0.001) 5)
75.01500100002545

> ((derive cube 0.00001) 5)
75.00014999664018
```

3. Процедурите като върнати оценки

Задача.

Да се дефинира функция от по-висок ред, която връща като резултат функция, представляваща приближение на *n*-тата производна на функцията *f* при дадено нарастване на аргумента *dx*.

```
(define (derive f dx)
    (lambda (x)
        (/  (- (f (+ x dx))  (f x))  dx)))


(define (Df f n dx)
    (if (= n 1) (derive f dx)
        (Df (derive f dx) (- n 1) dx)))
```

3. Процедурите като върнати оценки

Пример: Ако

```
(define (f x)
     (- (* 3 x x x) (* 7 x x) 10))

> ((derive f 0.001) 2)
8.011003000000017
> ((Df f 2 0.001) 2)
22.0179999923537
> ((Df f 3 0.0001) 1)
17.999823853642738
>((Df sin 4 0.001) 1.0)
0.8426592756904938
```

3. Процедурите като върнати оценки

Задача.

Ако *f* е едноаргументна <u>числова</u> функция и *n* е естествено число, *n*-кратното прилагане на *f* се дефинира като функция, чиято стойност в дадена точка *x* е равна на $\underbrace{f(f(\ldots (f(x)) \ldots))}_{n\text{ - пъти}}$.

Да се дефинира процедура от по-висок ред, която намира *n*-кратното прилагане на *f*.

```
(define (repeated f n)
  (lambda (x)
    (if (= n 1) (f x)
      (f ((repeated f (- n 1)) x)))))
```

Тест 2 (Ламбда изрази)

Задача. Оценете изразите в глобалната среда

а)

```
((lambda (x) (x 12)) (lambda (y) (/ 108 y)))
```

б)

```
((lambda (x y) (x (y 9))) sqrt (lambda (y) (* y 9)))
```

в)

```
(let* ((x (lambda (x) (* x x x)))
          (y (x 3)))
     (* 2 (quotient y 9)))
```