Описание функций в Лиспе.

Лекция 3.

Специальности : 230105, 010501

Лямбда-исчисление как основа определения функций.

Определение функций и их вычисление в Лиспе основано на лямбда-исчислении Черча. В предложенном Черчем исчислении функция записывается в следующем виде :

lambda(x1, x2, … ,xn).fn

В Лиспе исчисление Черча заимствовано для определение вычислений и описания параметров функций :

(lambda (*x1 x2 … xn*) *fn*)

Символ lambda означает, что мы имеем дело с определением производимых функцией действий. Символы *xi* являются формальными параметрами определения, которые именуют аргументы в описывающем вычисления теле функции *fn*. Телом функции является произвольная форма, значение которой может вычислить интерпретатор Лиспа.

Формальность **параметров означает, что их можно заменить на любые другие символы, и это не отразится на определяемых функцией действиях.**

Неименованные функции Лиспа.

Определение 1. **Лямбда-выражение – это определение вычислений и параметров функций в чистом виде без фактических параметров :**

(lambda (<список формальных параметров>) <тело функции>).

Для применения описанной таким образом функции к некоторым аргументам *a1, … ,an* необходимо в вызове функции поставить лямбда-выражение на место имени функции :

(< лямбда-выражение > *a1 a2 … an*).

Такую форму вызова называют лямбда-вызовом.

Если вычислять значения аргументов не нужно, то такую функцию нужно определять с помощью nlambda – выражения (от англ. No-spread lambda) :

(nlambda (<список формальных параметров>) <тело функции>)

nlambda – функции применяются при разработке новых синтаксических форм для расширения языка, а также при реализации интерпретаторов проблемно-ориентированных языков с лиспоподобной структурой.

Порядок вычисления лямбда-вызовов.

1. **Вычисляются значения фактических параметров.**
2. **Выполняется связывание параметров : первый формальный параметр связывается с вычисленным значением первого фактического, второй формальный параметр связывается с вычисленным значением второго фактического и т.д.**
3. **Выполняется тело лямбда-выражения. Полученное значение возвращается в качестве значения всего лямбда-вызова.**
4. **Разрывается связь между формальными и фактическими параметрами.**

Примечание. LAMBDA и NLAMBDA – вызовы могут быть вложенными.

Примеры неименованных функций muLISP’a.

((lambda (x y) ((atom x) y) ((atom y) x) (cons x y)) (car ‘((a b c) d))(cadr ‘(1 2 3 4)))	((nlambda (x y) ((atom x) y) ((atom y) x) (cons x y)) (car ‘((a b c) d))(cadr ‘(1 2 3 4)))

X→(a b c) Y→2 Результат : (a b c)	X →(car ‘((a b c) d)) Y →(cadr ‘(1 2 3 4)) Результат : (car ‘((a b c) d)) cadr ‘(1 2 3 4))

Вложенные лямбда-вызовы.

```
((lambda (x)
  ((lambda (y)
     (cons x y)
   )
   (cdr '(a b c))
  )
 )
 (car '((d e f)))
)
```

```
((nlambda (x)
  ((lambda (y)
     (cons x y)
   )
   (cdr '(a b c))
  )
 )
 (car '((d e f)))
)
```

X→(d e f)

Y→(b c)

Результат : ((d e f) b c)

X→ (car '((d e f)))

Y → (cdr '(a b c))

Результат : ((car '((d e f))) cdr '(a b c))

Описание неименованных функций в newLISP-tk.

В общих чертах синтаксис описания неименованных функций в newLISP-tk сходен с описанием аналогичных конструкций в muLISP'e. Основное отличие касается описания функций в целом и заключается в обязательности объявления управляющих структур if и cond в случае наличия разветвлений в структуре преобразования (вычисления).

<u>Пример.</u> Описать функцию, которая возвращает в качестве результата y в случае атомарного x и список '(x y) - в противном случае .

Реализация на muLISP’e :

```
((lambda (x y)
    ((atom (car x)) y)
    (cons x y))
  (car '((f) g h)) '(r t y))
```

Реализация на newLISP-tk :

```
((lambda (x y)
   (if (atom? (first x)) y)
     (cons x y))
  (first '((f) g h)) '(r t y))
```

Именованные функции.

Лямбда-выражение есть не имеющий имени механизм, соответствующий используемому в алгоритмических языках определению функции. Для многократного вызова одной и той же функции, но с различными фактическими параметрами функцию необходимо именовать подобно именованию данных посредством функции SET. Дать имя и определить новую функцию в muLISP'е можно с помощью функции DEFUN :

(defun <*имя*> <*лямбда-выражение*>)

Вызов именованной функции :

(<*имя*> <*список фактических параметров*>)

Пример описания и вызова именованной функции в muLISP'е :

$$f(x,y)=\begin{cases} y, \text{ если } x - \text{атом}, \\ x, \text{ если } y - \text{атом}, \\ (x\ y) - \text{иначе} \end{cases}$$

```
(defun f1 (list1 list2)
 ((lambda (x y)
  ((atom x) y)
  ((atom y) x)
  (cons x y))
  (car list1)(cadr list2)))
```

Вызов функции :

(f1 '(1 2) '(3 4))

Результат : 4

Современная сокращенная нотация записи именованных функций.

Функция DEFUN в muLISP'е соединяет символ имени функции с лямбда-выражением, после чего символ начинает представлять (именовать) определенные этим лямбда-выражением вычисления.

В современной нотации для удобства исключены внешние скобки лямбда-выражения и сам символ LAMBDA :

(defun <*имя*> (<*список формальных параметров*>)

<*тело функции*>)

Пример (сокращенная запись для f1) :

```
(defun f2 (list1 list2)
 ((atom (car list1))(cadr list2))
 ((atom (cadr list2)) (car list1))
 (cons (car list1) (cadr list2)))
```

Описание именованных функций в newLISP-tk.

В общих чертах синтаксис именованной функции имеет следующий вид :

(define (<имя функции> <список формальных параметров>)

<тело функции>)

Пример записи функций f1 и f2 в newLISP-tk :

```
(define (f1_new list1 list2)
  ((lambda (x y)
     (cond
       ((atom? x) y)
       ((atom? y) x)
       (true (cons x y))
     )
   )
   (first list1)(first (rest list2))
  )
)
```

```
(define (f2_new list1 list2)
   (cond
      ((atom? (first list1)) (first (rest list2)) )
      ((atom? (first (rest list2))) (first list1))
      (true (cons (first list1) (first (rest list2))
))
   )
)
```

Функции и ветвление.

Для разветвления вычислений в Лиспе служит функция COND. Синтаксическая форма COND позволяет управлять вычислениями на основе определяемых предикатами условий.

Структура условной формы в muLISP'e :

(cond (*p1 a1*)

(*p2 a2*)

. . .

(*pN aN*)

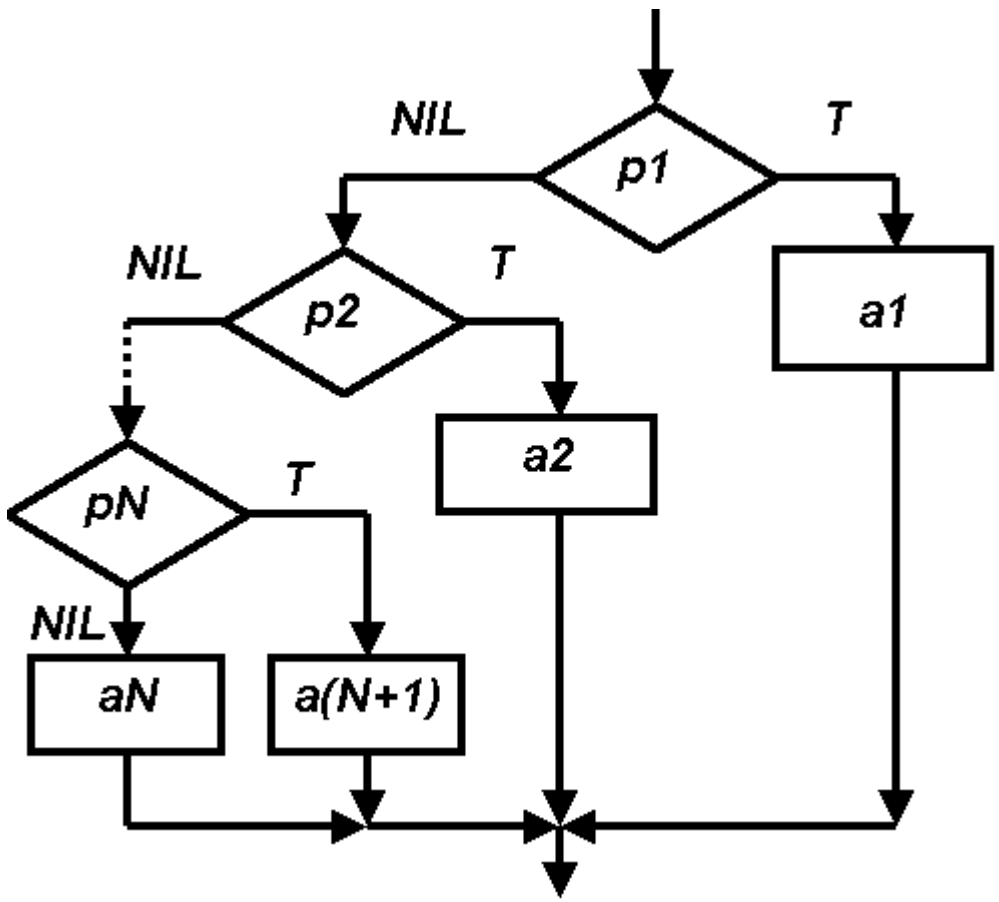
a(N+1)

)

***Примечание.**

Предикатами *pi* и результирующими выражениями *ai* могут быть произвольные формы.

Алгоритмический аналог :



Порядок вычисления условной формы.

1. **Предикативные выражения *pi* последовательно вычисляются сверху вниз до тех пор, пока результатом вычисления одного из *pi* не будет *T*.**
2. **Вычисляется результирующее выражение ai и полученное значение возвращается в качестве значения всего COND-предложения.**
3. **Если среди значений *pi* нет истинных и в теле условия отсутствует результирующее выражение, соответствующее ветви “иначе”, то значением COND будет NIL.**

Пример описания на muLISP’е функции, содержащей ветвление.

Опишем на muLISP’е функцию f2 из предыдущего примера, но с применением ветвления .

$$f(x,y)=\begin{cases} y, \text{если } x - \text{атом}, \\ x, \text{если } y - \text{атом}, \\ (x\ y) - \text{иначе} \end{cases}$$

В нашем примере :

X⇔(car list1),

Y⇔(cadr list2)

Вызов функции :

(f3 ‘(1 2) ‘(3 4))

Результат : 4

Описание на Лиспе :

```
(defun f3 (list1 list2)
 (cond
  ((atom (car list1)) (cadr list2))
  ((atom (cadr list2)) (car list1))
  (cons (car list1)(cadr list2))
 )
)
```