Функциональное и логическое программирование

Лекция 3

1.7.4 Параллельная рекурсия

Рекурсия называется *параллельной*, если рекурсивный вызов встречается одновременно в нескольких аргументах функции. Такая рекурсия встречается обычно при обработке вложенных списков. В операторном программировании параллельная рекурсия соответствует следующим друг за другом (текстуально) циклам.

Параллельность рекурсии не временная, а текстуальная. При выполнении тела функции в глубину идет сначала левый вызов (рекурсия «в глубину»), а потом правый (рекурсия «в ширину»).

Пример 1: Определим функцию **COPY_ALL**, копирующую список на всех уровнях.

```
(defun copy_all(L)
 (cond
   ;((null L) L) Можно убрать
   ((atom L) L)
   (t(cons(copy_all(car L))(copy_all(cdr L))))
 )
)
```

(copy_all '(((1) 2) 3 ((4)))) → '(((1) 2) 3 ((4)))

Пример 2: Определим функцию **IN_ONE**, преобразующую список в одноуровневый (удаление вложенных скобок).

```
(defun in_one(L)
 (cond
    ((null L) L)
    ((atom L) (list L))
   (t(append(in_one(car L))(in_one(cdr L))))
 )
)
```

(in_one '(((1) 2) 3 ((4)))) → (1 2 3 4)

Пример 3: Определим функцию **MAX_IN_LIST**, находящую максимальный элемент в числовом списке, содержащем подсписки.

```
(defun max_in_list(L)
 (cond
  ((atom L) L)
  ((null (cdr L))(max_in_list(car L)))
  (t(max(max_in_list(car L))(max_in_list(cdr L))))
 )
)
```

(max_in_list '(((10) 20) 3 ((4)))) → 20

1.8 Интерпретатор языка Лисп EVAL

Интерпретатор Лиспа называется **EVAL** и его можно так же, как и другие функции вызывать из программы.

«Лишний» вызов интерпретатора может, например, снять эффект блокировки вычисления от функции **QUOTE** или найти значение значения выражения, т.е. осуществить двойное вычисление.

(**EVAL** s-выражение)

Возвращает значение значения аргумента.

Примеры:

(setq x '(a b c)) → (a b c)

(eval 'x) → (a b c)

(eval x) → ошибка, начинает вычисляться функция a

(setq x 'y) → y

(setq y 'z) → z

(setq z '*) → *

(eval (eval x)) → *

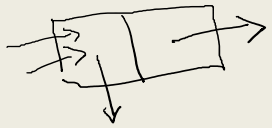
Используя **EVAL**, мы можем выполнить «оператор», который создан Лисп-программой и который может меняться в процессе выполнения программы.

Лисп позволяет с помощью одних функций формировать определения других функций, программно анализировать и редактировать эти определения как s-выражения, а затем, используя функцию **EVAL**, исполнять их.

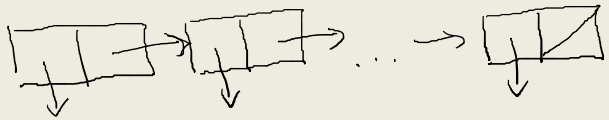
1.9. Внутреннее представление s-выражений

Атом → информационная ячейка (ячейка памяти).
Атом заменяется во внутреннем представлении на адрес информационной ячейки.
Через информационную ячейку можно получить доступ к списку свойств атома, среди которых содержится как внешнее представление, так и указатель на значение.

Оперативная память логически разбивается на списочные ячейки, состоящие из двух полей с указателями. Каждый указатель может ссылаться на другую списочную ячейку или объект Лиспа:

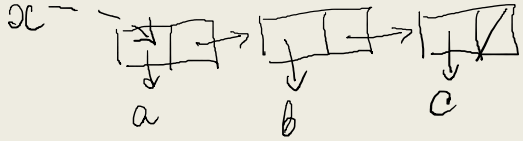


Список – последовательность списочных ячеек, связанных через указатели в правой части.



Пример 1:

Рассмотрим функцию (**SETQ** x ‘(a b c))



Пунктиром выделен побочный эффект.

Исходя из графического представления становится понятной работа функций **CAR**, **CDR**, **CONS**.

Функция **CAR** возвращает значение левой списочной ячейки.

Функция **CDR** возвращает значение правой списочной ячейки.

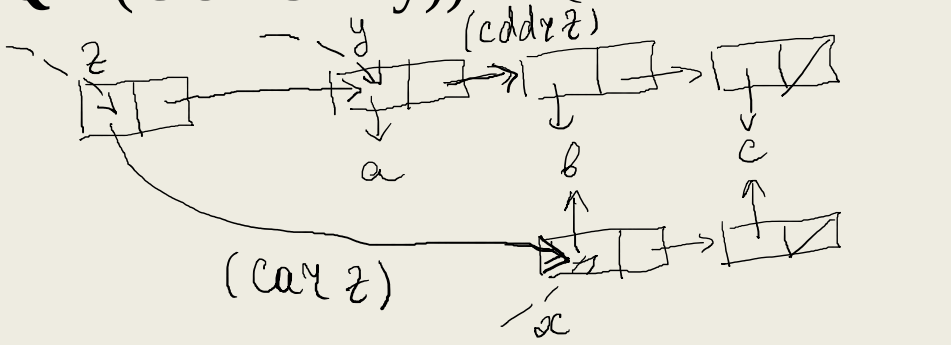
Функция **CONS** создает новую списочную ячейку, содержимое левого поля которого – это указатель на первый аргумент функции, а содержимое правого поля – это указатель на второй аргумент функции.

Пример 2:

(SETQ y ‘(a b c)) → (a b c)

(SETQ x ‘(b c)) → (b c)

(SETQ z (**CONS** x y)) → ((b c) a b c)



Идентичные атомы содержатся в структуре один раз. Логически идентичные списки могут быть представлены различными списочными ячейками.

(CAR z) → (b c)

(CDDR z) → (b c)

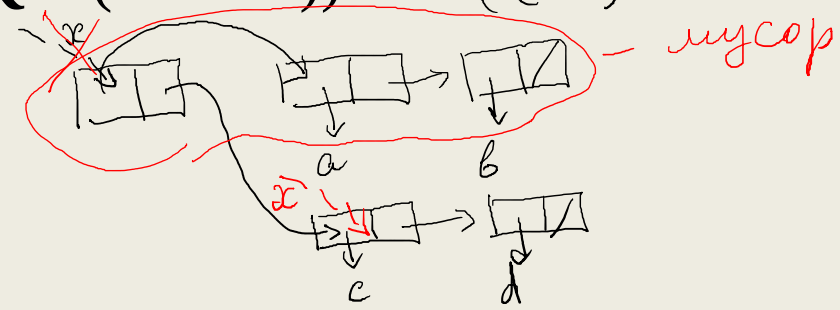
(equal (car z) (cddr z)) → t

(eq (car z) (cddr z)) → nil

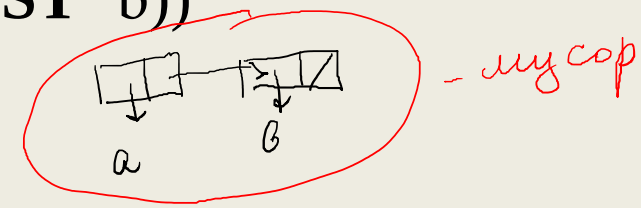
В результате вычислений в памяти могут возникнуть структуры, на которые нельзя сослаться. Такие структуры называются *мусором*.

Примеры образования «мусора»:

1. (**SETQ** x ‘((a b) c d)) → ((a b) c d)
   (**SETQ** x (**CDR** x)) → (c d)



2. (**CONS** ‘a (**LIST** ‘b))



Для повторного использования ставшей мусором памяти в Лиспе предусмотрен специальный сборщик мусора, который автоматически запускается, когда в памяти остается мало свободного места.

Все рассмотренные до сих пор функции манипулировали выражениями, не меняя существующие структуры.

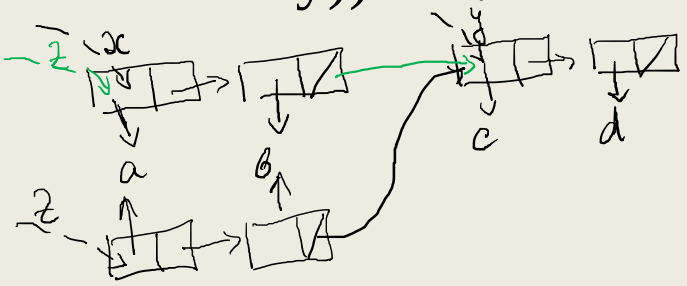
В Лиспе есть специальные функции, которые изменяют внутреннюю структуру списков - структуроразрушающие функции.

Пример 3 (работа функции **APPEND**):

(**SETQ** x ‘(a b)) → (a b)

(**SETQ** y ‘(c d)) → (c d)

(**SETQ** z (**APPEND** x y)) → (a b c d)



x = (a b c d) это не верно
append так не работает
append создает копию
первого аргумента

Очевидно, что если первый аргумент функции **APPEND** является списком из 1000 элементов, а второй – списком из одного элемента, то будет создано 1000 новых ячеек, хотя нужно добавить всего лишь один элемент к 1000 имеющимся.

Если нам не важно, что значение переменной x может измениться, то можно использовать соединение списков с помощью структуроразрушающей функции **NCONC**:

(NCONC sp_1 sp_2 … sp_n)

(SETQ z **(NCONC** x y))

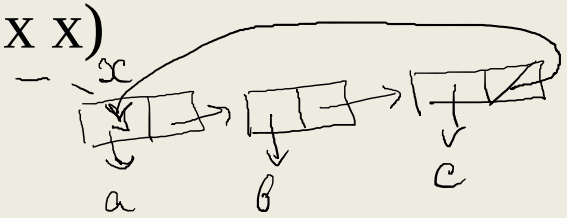
NCONC работает как указано зелёным на предыдущем слайде

NCONC может создавать циклические структуры

Пример 4:

(**SETQ** x ‘(a b c))

(**NCONC** x x)



(a b c a b c a b c ……)

Еще 2 функции, изменяющие структуру своих аргументов.

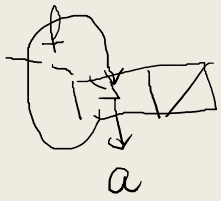
RPLACA (список s-выражение)

Заменяет указатель на голову списка на значение s-выражения.

Пример 5:

(**SETQ** f ‘(a))

(**RPLACA** f f)



(((((((……..……)))))))

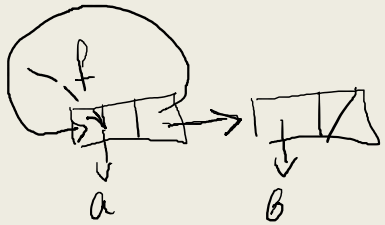
RPLACD (список s-выражение)

Заменяет указатель на хвост списка на значение s-выражения.

Пример 6:

(**SETQ** f ‘(a b))

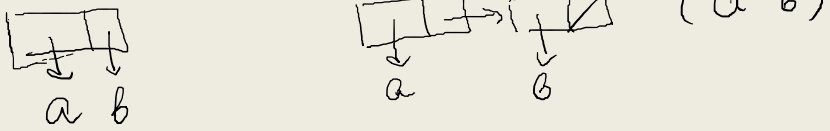
(**RPLACD** f f)



(a a ……..)

1.10 Точечная пара

(**CONS** ‘a ‘b) → (a . b)



Любой список можно представить в точечной нотации. Преобразования можно осуществить следующим образом: каждый пробел заменяется точкой, за которой ставится открывающаяся скобка. Соответствующая закрывающаяся скобка ставится непосредственно перед ближайшей справа от этого пробела закрывающейся скобкой, не имеющей парной открывающей скобки также справа от пробела. После каждого последнего элемента списка добавляется .nil.

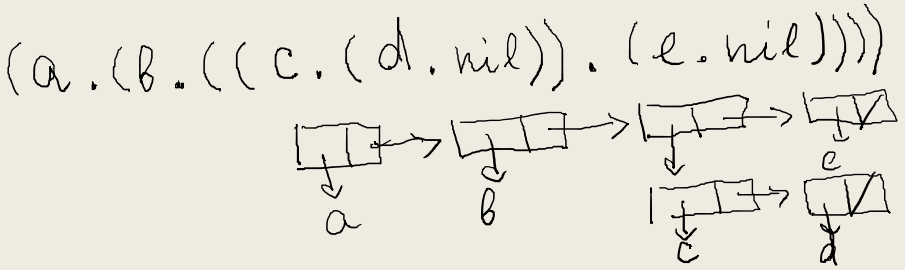
Переход к точечной нотации:

$(a_1\ a_2 \ldots a_n) \Leftrightarrow (a_1 . (a_2 . \ldots (a_n . \text{nil}) \ldots))$

Пример 1:

(a b (c d) e)

Эквивалентное представление с помощью точечных пар:



Записанное в точечной нотации выражение можно частично или полностью привести к списочной нотации.

Переход к списочной записи осуществляется по следующему правилу: если точка стоит перед открывающейся скобкой, то она заменяется пробелом и одновременно убирается соответствующая закрывающаяся скобка. Это же правило позволяет избавиться и от лишних nil, если помнить, что nil эквивалентен ().

Пример 2:

(a . ((b . nil) . (c . nil)))

Эквивалентное представление с помощью списка:

(a␣(b␣)␣c␣) <=> (a (b) c)

(a . (b . c))

Эквивалентное представление с помощью списка:

(a␣b . c) <=> (a b . c)