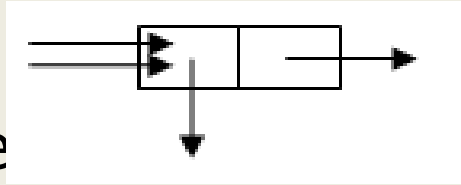
1.8. Внутреннее представление s-выражений

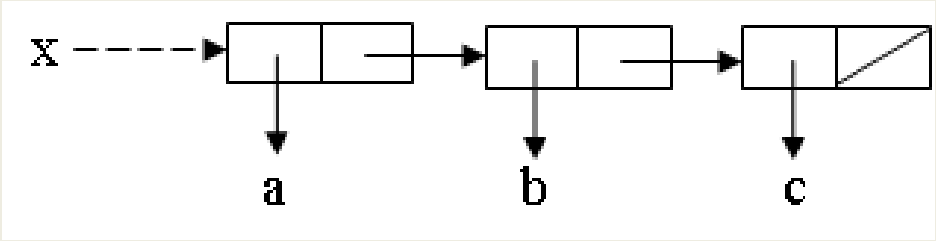
Атом → информационная ячейка→
список свойств атома

Оперативная память логически разбивается на списочные ячейки, состоящие из двух полей с указателями



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

Пример 1:
(**SETQ** x '(a b c)) →(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, а функция **CDR** – значение правой списочной ячейки.

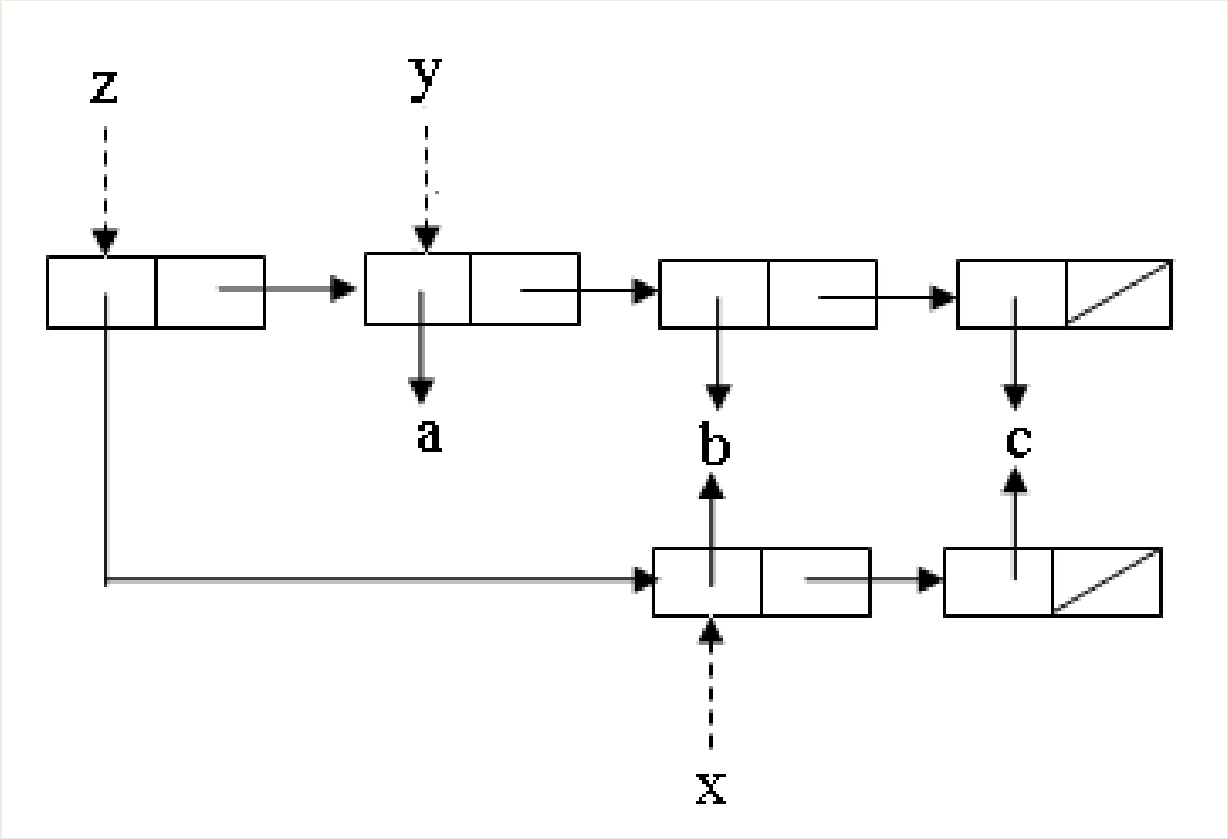
Функция **CONS** создает новую списочную ячейку, содержимое левого поля которого – это указатель на первый аргумент функции, а содержимое правого поля – это указатель на второй аргумент функции.

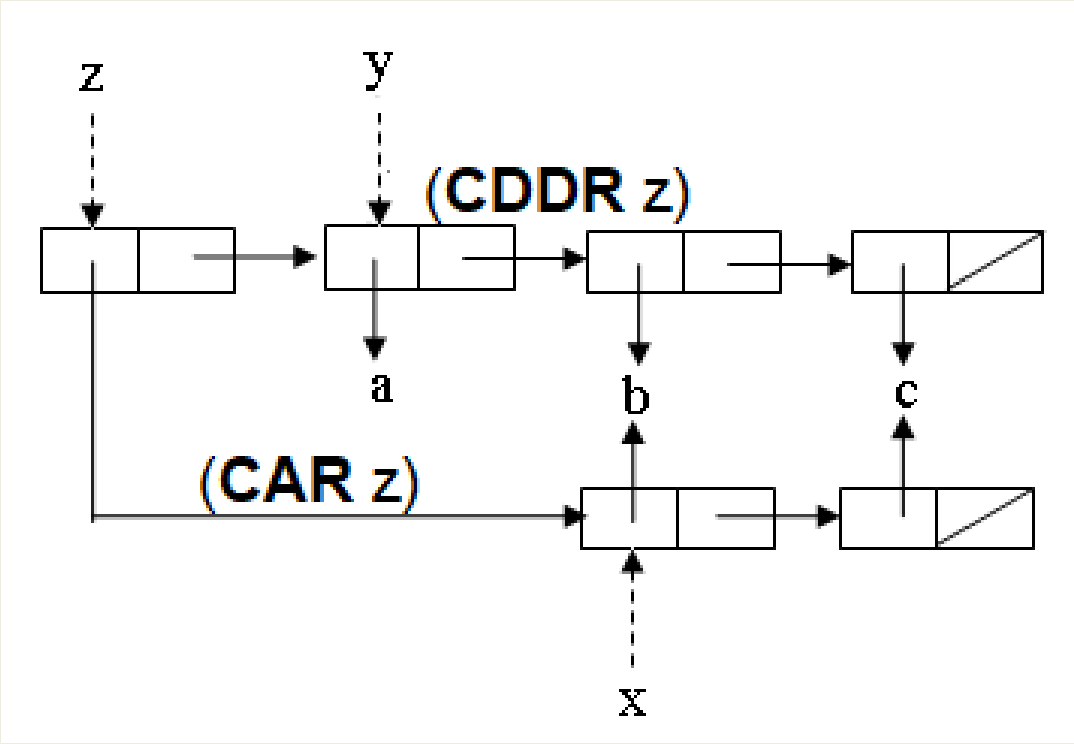
Пример 2:

(**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. Однако, логически идентичные списки могут быть представлены различными списочными ячейками.

(**CAR** z) $\rightarrow$ (b c)

(**CDDR** z) $\rightarrow$ (b c)

В результате вычислений в памяти могут возникнуть структуры, на которые нельзя сослаться. Такие структуры называются *мусором*. Образование мусора происходит в тех случаях, когда вычисленная структура не сохраняется с помощью **SETQ** или когда теряется ссылка на старое значение в результате побочного эффекта нового вызова **SETQ** или другой функции.

Примеры образования «мусора»:

1. (**SETQ** x '((a b) c d))

   (**SETQ** x (**CDR** x))

Списочные ячейки, связанные с атомами a и b, стали мусором.

2. (**CONS** 'a (**LIST** 'b))

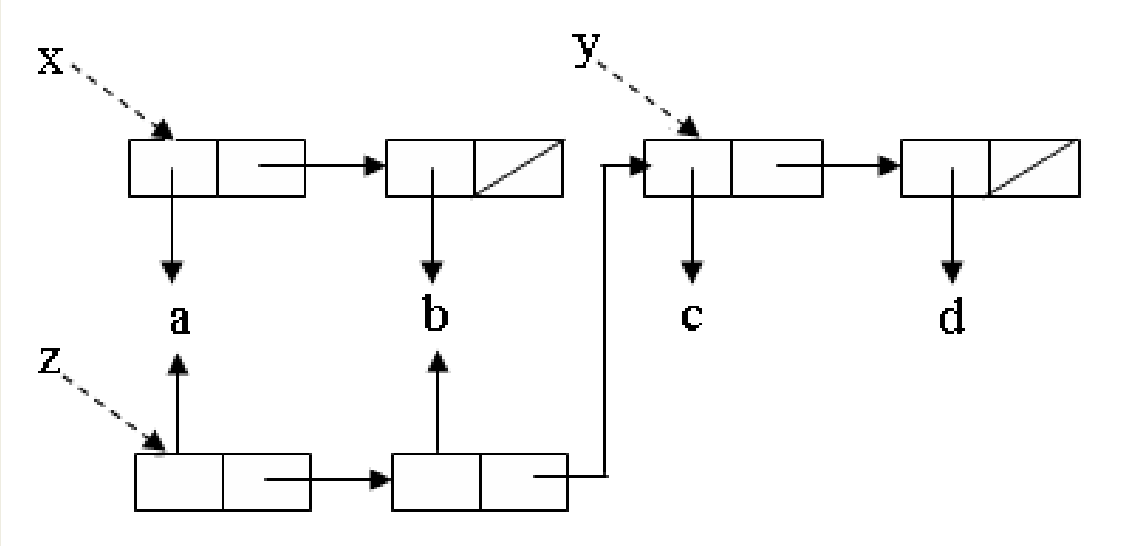
В Лиспе есть специальные функции, которые изменяют внутреннюю структуру списков - структуроразрушающие функции.

Пример 3 (работа функции **APPEND**):

(**SETQ** x '(a b))→(a b)

(**SETQ** y '(c d))→(a b c d)

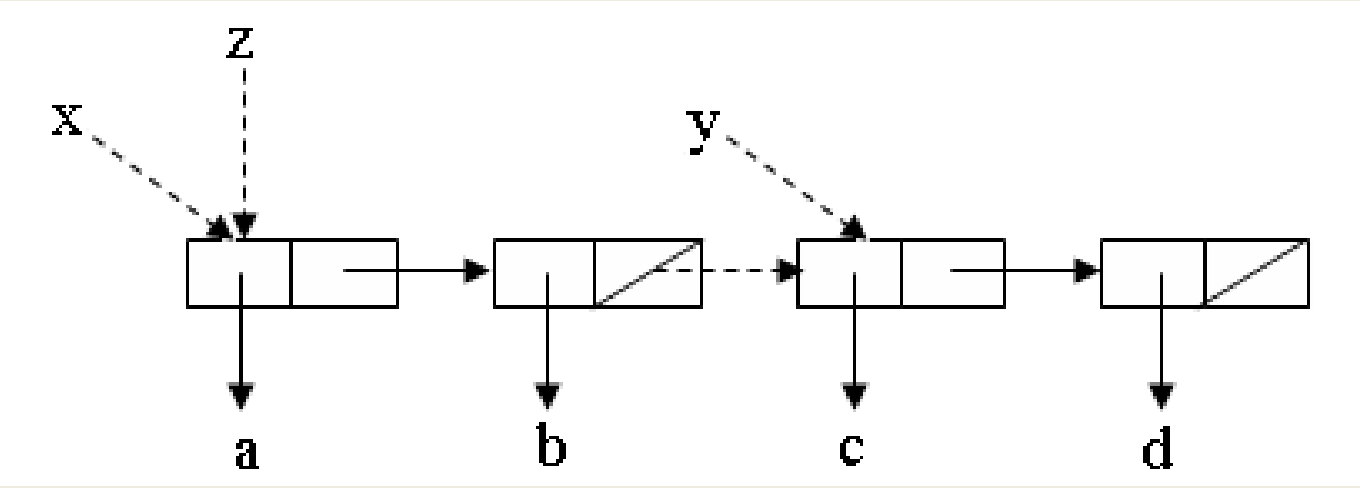
(**SETQ** z (**APPEND** x y))→(a b c d)



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**.

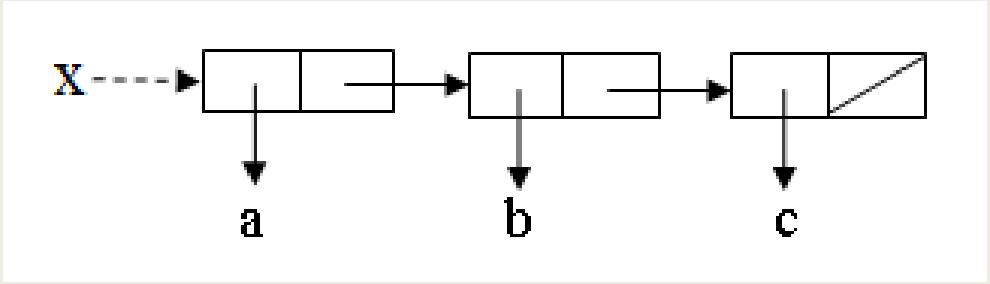
(**SETQ** z (**NCONC** x y))→(a b c d)



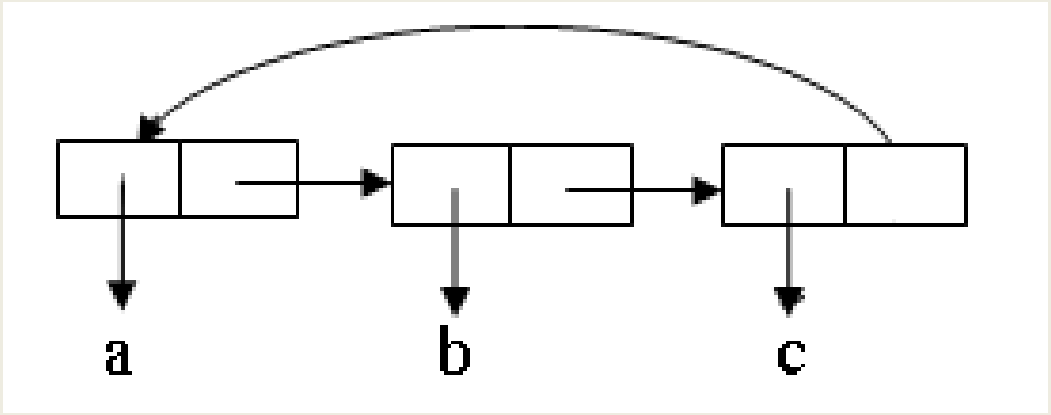
NCONC может создавать циклические структуры

Пример 4:

(**SETQ** x '(a b c))→(a b c)



(**NCONC** x x)



(a b c a b c a b c ……)

Еще 2 функции, изменяющие структуру своих аргументов:

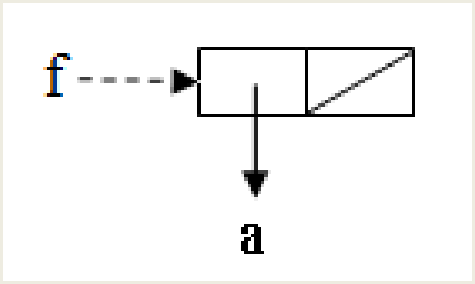
RPLACA - replace the car (заменяет указатель на голову списка)

RPLACD - replace the cdr (заменяет указатель на хвост списка)

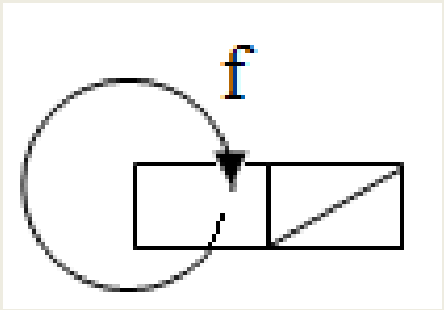
RPLACA (список s-выражение)

Пример 5:

(**SETQ** f '(a))–> (a)



(**RPLACA** f f)

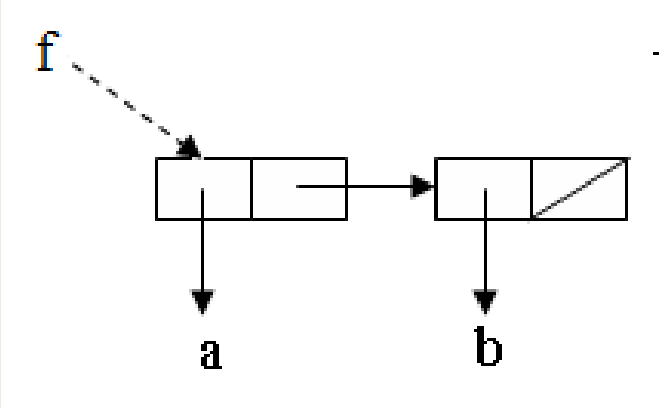


(((((((.....)))))))

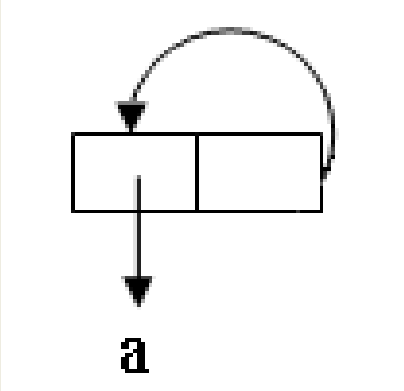
RPLACD (список s-выражение)

Пример 6:

(**SETQ** f '(a b))



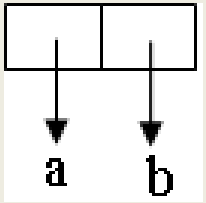
(**RPLACD** f f)



(a a ……)

1.9 Точечная пара

(**CONS** '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 7:

(a b (c d) e)$\rightarrow$(a . (b . ((c .(d . nil)) . (e . nil))))

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 8:

(a . ((b . nil) . (c . nil)))$\rightarrow$(a (b) c)

(a . (b . c)) $\Leftrightarrow$ (a b . c)

1.10 Функционалы

В Лиспе функции могут выступать в качестве аргументов (аргументом функции может быть определяющее функцию лямбда-выражение или имя другой функции). Такой аргумент называется *функциональным*, а функция, имеющая функциональный аргумент, называется *функционалом*.

1.10.1 Аппликативные (применяющие) функционалы

Применяющим функционалом называется функционал, который применяет функциональный аргумент к остальным параметрам.

(**APPLY** fn sp)

Вычисляет значение функционального аргумента (функции от n переменных) для фактических параметров, которые являются элементами списка.

Пример 9:

Написать функциональный предикат **ALL**, который возвращает t в том и только в том случае, если функциональный аргумент истинен для каждого элемента списка.

```
(DEFUN ALL (p l)
 (COND
      ((NULL l) t)
      ((APPLY p (LIST (CAR l))) (ALL p (CDR l)))
      (t nil)
 )
)
```

(**ALL** (**LAMBDA** (x) (**<=** x 0)) '(-1 -3 -4 0)) → t

(**ALL** ‘**SYMBOLP**) ‘(a s d 1 f)) → nil

(**FUNCALL** fn v_1 v_2 … v_n)

Работает аналогично **APPLY**, но аргументы функционального аргумента (функции от n переменных) задаются не списком, а как аргументы **FUNCALL**, начиная со второго.

Пример 10:

Написать функцию сортировки списка методом вставки в виде функционала **SORT1**, у которого функциональный аргумент будет задавать порядок сортировки.

```
(DEFUN  SORT1 (l p)
  (COND
      ((NULL l) l)
      (t (ADD_ORD (CAR l) (SORT1 (CDR l) p) p))
 ))
(DEFUN ADD_ORD (x l p)
  (COND
     ((NULL l) (LIST x))
     ((FUNCALL p x (CAR l)) (CONS x l))
     (t (CONS (CAR l) (ADD_ORD x (CDR l) p)))
 ))
```

(**SORT1** '(5 1 3 2 4) ‘<) → (1 2 3 4 5)
(**SORT1** '(5 1 3 1 2 4 2) (**LAMBDA** (x y) (>= x y))) →
(5 4 3 2 2 1 1)
(**SORT1** '(ab sc aefg srt) 'string>) → (srt sc aefd ab)

1.10.2 Отображающие функционалы или MAP-функции

Отображающие функционалы с помощью функционального аргумента преобразуют список в новый список или порождают побочный эффект, связанный с этим списком. Такие функционалы начинаются на MAP.

(**MAPCAR** fn sp_1 sp_2 … sp_n)

Возвращает список, состоящий из результатов последовательного применения функционального аргумента (функции n переменных) к соответствующим элементам n списков. Число аргументов-списков должно быть равно числу аргументов функционального аргумента.

Пример 11:

Заменить в списке все числа на пару (<число> *).

```
(MAPCAR (LAMBDA(x)
                (COND
                  ((NUMBERP x)(LIST x '*))
                   (t x)
                )) '(a 1 d f 3 4))
```

→(a (1 *) d f (3 *) (4 *))

Пример 12:

Функция **SUM3** вычисляет сумму кубов элементов числового списка.

```
(DEFUN SUM3 (l)
 (EVAL (CONS '+ (MAPCAR '* l l l))))
```

или

```
(DEFUN SUM3 (l)
 ( APPLY '+ (MAPCAR '* l l l)))
```

(**SUM3** '(1 2 3))→36

(**MAPLIST** fn sp_1 sp_2 … sp_n)

Отображающий функционал **MAPLIST** действует подобно **MAPCAR**, но действия осуществляются не над элементами списков, а над последовательными хвостами этих списков, начиная с самих списков.

Пример 13:

(**MAPLIST** 'reverse '(a b c)) → ((c b a) (c b) (c))

(**MAPLIST** (**LAMBDA** (x) (**EVAL**(**CONS** '+ x))) '(1 2 3 4))→ (10 8 5 4)

Объединяющие функционалы MAPCAN и MAPCON

Работа их аналогична соответственно **MAPCAR** и **MAPLIST**. Различие заключается в способе построения результирующего списка. Если функционалы **MAPCAR** и **MAPLIST** строят новый список из результатов применения функционального аргумента с помощью функции **LIST**, то функционалы **MAPCAN** и **MAPCON** для построения нового списка используют структуроразрушающую псевдофункцию **NCONC**, которая делает на внешнем уровне то же самое, что и функция **APPEND.** Функционалы **MAPCAN** и **MAPCON** удобно использовать в качестве фильтров для удаления нежелательных элементов из списка.

(MAPCAN fn sp_1 sp_2 … sp_n)

Пример 14:

Удалить из числового списка все элементы, кроме отрицательных.

```
(MAPCAN (LAMBDA (x)
                (COND
                   ((MINUSP x) (LIST x))
                    (t nil)
                )) '(-3 1 4 -5 0))
```

→ (-3 -5)

(**MAPCON** fn sp_1 sp_2 … sp_n)

Пример 15:

Преобразовать одноуровневый список во множество.

```
(MAPCON (LAMBDA (l)
                (COND
                   ((MEMBER (CAR l) (CDR l)) nil)
                    (t (LIST (CAR l)))
                )) '(1 2 3 1 1 4 2 3))
```

→ (1 4 2 3)