Лекция № 4

Списки в языке PROLOG

Lists in the PROLOG language

Списки в языке Prolog

Список – это упорядоченный набор объектов любого типа. Объекты списка называются его **элементами**.

Список может иметь неограниченное число элементов (границы определяются лишь размером оперативной памяти).

Для обозначения списка используются квадратные скобки, в которых через запятую перечисляются элементы списка.

Примеры списков:

[1,2,3]

[a,b,c]

[[a,b,c], [c,d,e]]

Список, не содержащий элементов, называется пустым и обозначается **[ ].**

Создание списка

Список может задаваться перечислением своих элементов:

L = [first, second, third, fourth, fifth].

Для порождения списка также может использоваться встроенный предикат *findall/3:*

findall (VarName, Predicate (VarName), List).

Здесь **VarName** – переменная, указывающая на аргумент предиката **Predicate**, все значения которого будут собраны в результирующий список **List**.

Создание списка

Города Новосибирской области

city ("Барабинск", 50000).
city ("Бердск", 80000).
city ("Искитим", 60000).
city ("Куйбышев", 40000).
city ("Новосибирск", 2300000).

Получить список городов Новосибирской области можно, указав в цели предикат

findall (X, city (X,_), L).

Получим **L = ["Барабинск", "Бердск", "Искитим", "Куйбышев", "Новосибирск"].**

Работа со списками

Непустой **список** можно рассматривать как состоящий из двух частей: **головы** (Head) от **хвоста** (Tail).

Голова списка – его первый элемент. Хвост – остальная часть списка, получаемая после удаления головы.

Голова – это элемент списка, а хвост – новый список.

Внимание! Пустой список нельзя разделить на голову и хвост.

Основной метод работы со списками в языке Prolog:

отделение Головы (Head) от Хвоста (Tail) списка.

Операция деления списка на голову и хвост обозначается при помощи вертикальной черты "|": **[Head | Tail]**

В списке целых [1, 2, 3]: Head = 1, Tail = [2, 3].
В списке [2,3]: Head = 2, Tail = [3].
В списке [3]: Head = 3, Tail = [].

У списка можно взять несколько “голов” сразу:

[X, Y, Z | Tail]

Работа со списками

Печать списка

Вид предиката *print_list/1* = **print_list (L).**

Реализация

```
print_list ( [ ]).
print_list ([ Head | Tail ]) :- write (Head), nl,
                                print_list (Tail).
?- print_list ([1, 2, 3]).
```

Коротко семантику утверждений предиката **print_list** можно описать следующим образом:

1. Если список пустой, то ничего не делать.
2. Иначе печатать его первый элемент, а затем применить правило к хвосту.

Операции со списками

Принадлежность элемента списку

Вид предиката *member/2* = **member (Elem, List).**

Предикат успешный, если **Elem** содержится в списке **List**.

Реализация

```
member (X, [X|T]).
member (X, [H|T]) :- member(X,T).
```

?- member (3, [1, 2, 3, 4, 5]).

Более простой вариант

```
member (X, [X | _]).
member (X, [_|T]) :- member (X,T).
```

Операции со списками

Подсчет количества элементов списка

Вид предиката *length_list/2* = **length_list (List, Length).**

Реализация

```
length_list ( [ ] , 0).
length_list ([ _ | Tail], Length ) :-
        length_list (Tail, Tail_Length),
        Length is Tail_Length + 1.
```

?- **length_list ([1, 2, 3, 4, 5] , X).**

Семантика:

1. Длина пустого списка [ ] равна 0.
2. Длина непустого списка равна длине его хвоста плюс 1.

Операции со списками

Подсчет суммы элементов списка

Вид предиката *sum_list/2* = **sum_list (List, Sum).**

Реализация

```
sum_list ([ ], 0).
sum_list ([Head | Tail], Sum) :- sum_list (Tail, TailSum),
                                 Sum is TailSum + Head.
```

Вычисление цели

?- sum_list ([1, 2, 3, 4, 5] , S).

Даст результат **S = 15.**

Операции со списками

Поиск максимального элемента списка

Вид предиката *max_list/2* = **max_list (List, Max).**

Реализация

```
max_list ([ X ], X ).
max_list ([X | Tail], Max ) :- max_list ( Tail, MaxTail ),
                               max ( X, MaxTail, Max ).
max ( X, Y, Z ):- X >= Y, Z = X;
                  Y > X,  Z = Y.
```

Вычисление цели

?- max_list ([1, 7, 5, 9, 5], M).

даст результат **M = 9.**

Операции со списками

Замечание. В языке Prolog невозможно непосредственно изменить заданный список. Это можно сделать, только породив новый список с требуемыми свойствами. В связи с этим предикаты, реализующие операции, изменяющие списки, должны включать не менее двух аргументов (исходный и результирующий список).

Наиболее простой способ добавить элемент X в список L – это вставить его в самое начало так, чтобы он стал его головой:

[X | L].

Таким образом операция "|" может использоваться не только для отделения головы от списка, но и для присоединения нового элемента к списку.

Добавление элемента в список

add (X, L, L1) :- L1 = [X | L].

или **add (X, L,[X | L]).**

Вычисление цели: **add ("я", ["ты", "он", "она"], L).**

Даст результат **L = ["я", "ты", "он", "она"].**

Операции со списками

Удаление элемента из списка
Вид предиката *delete/3* = **delete(X, List, List1).**

Реализация

```
delete(Head, [Head | List], List).
delete(Head, [Head1 | List], [Head1 | List1]) :-
                                        delete(Head, List, List1).
```

Вычисление цели

?- delete(5, [1, 3, 5, 7, 9], L).

даст результат **L = [1, 3, 7, 9].**

Операции со списками

Соединение двух списков

Вид предиката *append /2* = **append (List1, List2, List3).**

(*Смысл операции:* List3 = List1 + List2)

Реализация

append ([ ], L, L).
append ([N | L1], L2, [N | L3]):- append (L1, L2, L3).

Вычисление цели

?- append ([1, 2], [3, 4, 5], L3).

даст результат **L3 = [1, 2, 3, 4, 5].**

Операции со списками

Семантика операции соединения списков

Сначала Пролог попытается удовлетворить первое правило

append ([ ], L, L).

Но так как первый список не пуст, перейдет ко второму правилу.

append ([N | L1], L2, [N | L3]):- append (L1, L2, L3).

Второе правило будет рекурсивно применяться до тех пор, пока первый список (**L1**) не станет пустым. При этом элементы первого списка будут последовательно пересылаться в стек.

Когда первый список станет пустым, применится первое правило **append ([ ], L, L).**
При этом третий список инициализируется вторым (т.е. L3 станет равным L2). Цель успешно вычислится, и после этого начнется сворачивание рекурсивных вызовов второго правила.

Операции со списками

Сворачивание рекурсивных вызовов второго правила:

append ([N | L1], L2, [N | L3]):- append (L1, L2, L3).

Извлекаемые при этом из стека элементы (промежуточные значения переменной **N**) помещаются один за другим в качестве головы в первый и третий списки. Этот процесс будет продолжаться до полного исчерпания стека.

В результате список **L3** будет содержать все элементы обоих входных списков.

Напомним, что из стека элементы извлекаются в обратном порядке (поскольку это стек!), поэтому в результирующем списке они будут иметь тот же порядок, что и в исходном.

Операции со списками

Сортировка списка

Рассмотрим *сортировку методом вставок*, как наиболее удобную и эффективно реализуемую на Прологе.

Идея данного метода состоит в том, что сортировка осуществляется при помощи многократной вставки элементов в список, пока он не будет упорядочен.

Вид предиката *insert_sort/2* = **insert_sort (List1, List2).**

Здесь List1 – исходный список, List2 – отсортированный список.

Операции со списками

Реализация

```
insert_sort([ ], [ ]).
insert_sort([ X|Tail ], Sorted_list):- insert_sort(Tail, Sorted_Tail),
                                       insert(X, Sorted_Tail, Sorted_list).
insert(X, [Y|Sorted_list], [Y|Sorted_list1]):- X>Y, !,
                                        insert(X, Sorted_list, Sorted_list1).
insert(X, Sorted_list, [X|Sorted_list]).
```

Вычисление цели

?- insert_sort ([1, 3, 2, 5, 2, 4], L).

даст результат **L3 = [1, 2, 2, 3, 4, 5].**

Операции со списками

Взятие последнего элемента списка

```
last(X, List):- append(_, [X], List).

append ([ ], L, L).
append ([ N | L1], L2, [N | L3 ]):- append ( L1, L2, L3).
```

Обращение списка

“Наивная” реализация:

Рекурсивно обратить хвост исходного списка и затем добавить первый элемент в конец обращенного списка.

```
reverse ([ ], [ ]).
reverse ([ X | Tail], List):- reverse (Tail, List1),
                              append ( List1, [X], List).
```