Списки в языке Пролог

Определение

Список — упорядоченное множество объектов одинакового типа.

Формально это определение соответствует определению массива в традиционных языках программирования. Однако в списке не оговаривается ни размерность, ни число элементов.

Список - упорядоченная последовательность элементов произвольной длины.

Список задается перечислением элементов списка через запятую в квадратных скобках.

Примеры записи списков

[monday, tuesday, wednesday, thursday, friday, saturday, sunday] — список, элементами которого являются английские названия дней недели;

["понедельник", "вторник", "среда", "четверг", "пятница", "суббота", "воскресенье"] —элементами списка являются русские названия дней недели;

[1, 2, 3, 4, 5, 6, 7] —элементами списка являются номера дней недели;

['п', 'в', 'с', 'ч', 'п', 'с', 'в'] —элементами списка являются первые символы русских названий дней недели;

[ ] - пустой список;

[1 , 7, 3 , 50] – список целых чисел;

[‘1’ , ‘7’, ‘3’ , ‘d’] – список символов.

Примеры записи списков

Элементы списка могут быть ***любыми***, в том числе и составными объектами. В частности, элементы списка сами могут быть списками.

Например,

[[1,3,7],[],[5,2,94],[–5,13]]

Описание списков в программе

В разделе описания доменов списки описываются следующим образом:

DOMAINS
<имя спискового домена>=<имя домена элементов списка>*

Звездочка после имени домена указывает на то, что описывается список, состоящий из объектов соответствующего типа.

Примеры описания списков

domains

listI = integer* /* список целых чисел */
listR = real* /*список вещественных чисел*/
listC = char* /* список символов */
lists = string* /* список строк */
listL = listI*
/* список, элементами которого являются списки целых чисел */

В классическом Прологе элементы списка могут принадлежать разным доменам, например: **[monday, 1, "понедельник"].**

В Турбо Прологе, в связи со строгой типизацией, все элементы списка должны принадлежать одному домену. Однако можно разместить в одном списке объекты разной природы, используя домен с соответствующими альтернативами.

Пример записи списка с объектами разной природы

DOMAINS
element = i(integer); c(char); s(string)
listE = element*

Данное описание позволит работать со списками вида:

[i(–15),s("Мама"),c('А'),s("мыла"),c('+'),
s("раму"), i(48),c('!')]

Рекурсивное определение списка

Список — это структура данных, определяемая следующим образом:

пустой список **[ ]** является списком;
структура вида **[H|T]** является списком, если **H** — первый элемент списка (или несколько первых элементов списка, перечисленных через запятую), а **T** — список, состоящий из оставшихся элементов исходного списка.

Рекурсивное определение списка

H - голова списка,
T — хвост списка.

По-английски голова — **Head**, а хвост — **Tail**.

Фактически операция "|" позволяет разделить список на хвост и голову или, наоборот, приписать объект (объекты) к началу списка.

Рекурсивное определение позволяет организовывать рекурсивную обработку списков, разделяя непустой список на голову и хвост. Хвост, в свою очередь, также является списком, содержащим меньшее количество элементов, чем исходный список. Если хвост не пуст, его также можно разбить на голову и хвост. И так до тех пор, пока не будет пустого списка, у которого нет головы.

Примеры записей списков

[1, 2, 3] = [1|[2, 3]],
т.е. в списке [1, 2, 3] элемент 1 является головой, а список [2, 3] — хвостом.
Хвост этого списка **[2, 3]**, также может быть представлен в виде головы **2** и хвоста **[3]**, а список **[3]** можно рассматривать в виде головы **3** и хвоста **[]**. Пустой список далее не разделяется. Таким образом,
[1, 2, 3] = [1|[2, 3]],
[1|[2, 3]]= [1|[2|[3]]],
[1|[2|[3]]]=[1|[2|[3|[ ]]]].

Примеры записей списков

В списке **[1, 2, 3]** можно выделить два первых элемента и хвост из третьего элемента **[1,2|[3]]**.

Возможен вариант разбиения на голову из трех первых элементов и пустой хвост: **[1, 2, 3|[]]**.

Чтобы организовать обработку списка, в соответствии с рекурсивным определением, достаточно задать предложение (правило или факт, определяющее, что нужно делать с пустым списком), которое будет базисом рекурсии, а также рекурсивное правило, устанавливающее порядок перехода от обработки всего непустого списка к обработке его хвоста. Иногда базис рекурсии записывается не для пустого, а для одно- или двухэлементного списка.

Обработка списков

Пример 1. Вычислить длину списка (количество элементов в списке).

Идея решения:

1) в пустом списке элементов нет;

2) непустой список представляется в виде объединения первого элемента и хвоста;

3) количество элементов непустого списка равно количеству элементов хвоста, увеличенному на единицу.

```
length([], 0).        /* в пустом списке элементов нет */
length([_|T], L):– length(T, L_T), L = L_T + 1.
 /* L_T — количество элементов в хвосте , L — количество элементов исходного списка */
```

Пример 2. Проверить принадлежность элемента списку.

Идея решения:

1) предикат будет иметь два аргумента: первый — искомое значение, второй — список, в котором производится поиск.

2) объект принадлежит списку, если он либо является первым элементом списка, либо элементом хвоста.

member(X,[X|_]). /* X — первый элемент списка */

member(X,[_|T]) :– member(X,T).

/* X принадлежит хвосту T*/

Описанный предикат можно использовать:
1) для проверки, имеется ли в списке конкретное значение. Например, принадлежит ли двойка списку [1, 2, 3]:

Goal: member(2, [1, 2, 3]).
True

Или принадлежит ли 4 списку [1,2, 3]:

Goal: member(4, [1, 2, 3]).
False

2) получение по списку его элементов.
Для этого нужно в качестве первого аргумента предиката указать свободную переменную.
Например:
Goal: member(X, [1, 2, 3]).

В качестве результата получим список всех элементов списка:
X=1
X=2
X=3

3) получить по элементу варианты списков, которые могут его содержать.

Теперь свободную переменную запишем вторым аргументом предиката, а первым — конкретное значение. Например,

Goal: member(1, X).

Вначале Пролог-система выдаст предупреждение о том, что переменная X не связана в первом предложении. При нажатии кнопки **Esc** происходит отказ от генерации списков, содержащих единицу в качестве элемента. При нажатии **F10** продолжается выполнение цели. При этом Пролог-система начнет выдавать варианты списков, содержащих единицу:

X=[1|_] /* единица — первый элемент списка */

X=[_,1|_] /* единица — второй элемент списка */

X=[_,_,1|_] /* единица — третий элемент списка */ и т.д.

Этот процесс будет продолжаться до тех пор, пока не будет нажата комбинация клавиш **Ctrl+Break**.

Пример 3. Объединить два списка.

Идея решения:

1) Первые два аргумента предиката будут представлять соединяемые списки, а третий — результат соединения.

2) В качестве основы для решения этой задачи возьмем рекурсию по первому списку. Базисом рекурсии будет факт, устанавливающий, что если присоединить к списку пустой список, в результате получим исходный список. Шаг рекурсии позволит создать правило, определяющее, что для того, чтобы приписать элементы списка, состоящего из головы и хвоста, ко второму списку, нужно соединить хвост и второй список, а затем к результату приписать спереди первый элемент первого списка.

Решение:

conc([ ], L, L).
/* при соединении пустого списка с L получим список L */
conc([H|T], L, [H|T1]) :– conc(T,L,T1).
/* соединяем хвост и список L, получаем хвост результата */

Варианты решения задач

1) для соединения списков.

Например,
Goal: conc([1, 2, 3], [4, 5], X)

то получим в результате
X= [1, 2, 3, 4, 5]

Варианты решения задач

2) получится ли при объединении двух списков третий.
Например,

Goal: conc([1, 2, 3], [4, 5], [1, 2, 5]).

False

Варианты решения задач

3) для разбиения списка на подсписки.

Например,

Goal: conc([1, 2], Y, [1, 2, 3]).

Y=[3]

Goal: conc(X, [3], [1, 2, 3]).

X=[1, 2]

Goal: conc(X, Y, [1, 2, 3]).

X=[], Y=[1, 2, 3]

X=[1], Y=[2, 3]

X=[1, 2], Y=[3]

X=[1, 2, 3], Y=[]

Варианты решения задач

4) для поиска элементов, находящихся левее и правее заданного элемента.
Например, какие элементы находятся левее и правее числа 2:

Goal: conc(L, [2|R], [1, 2, 3, 2, 4]).

L=[1], R=[3, 2, 4].
L=[1, 2, 3], R=[4]

Варианты решения задач

5) на основе предиката **conc** создать предикат, находящий последний элемент списка:

last(L,X):– conc(_,[X],L).

Этот предикат можно реализовать и без использования предиката **conc**:

last2([X],X).

/* последний элемент одноэлементного списка — этот элемент */

last2([_|L],X):– last2(L,X).

/* последний элемент списка совпадает с последним элементом хвоста */

Варианты решения задач

6) проверить принадлежность элемента списку, используя предикат **conc**.

Идея решения: если элемент принадлежит списку, то список может быть разбит на два подсписка так, что искомый элемент является головой второго подсписка:

member4(X,L):– conc(_,[X|_],L).

Варианты решения задач

7) по двум значениям и списку проверить, являются ли эти значения соседними элементами списка (использовать предикат, объединяющий списки).

Предикат будет иметь три параметра: первые два — значения, третий — список.

Идея решения : если два элемента оказались соседними в списке, значит, этот список можно разложить на два подсписка, причем голова второго подсписка содержит два данных элемента в нужном порядке. Например:

sosed(X,Y,L):– conc(_,[X,Y|_],L).

/* список L получается путем объединения некоторого списка со списком, голову которого составляют элементы X и Y */

Пример 4. Удалить все вхождения заданного значения из списка

Идея решения:
Предикат будет зависеть от трех параметров. Первый параметр будет соответствовать *удаляемому списку*, второй — *исходному значению*, а третий — *результату удаления* из первого параметра всех вхождений второго параметра.

Пример 4. Удалить все вхождения заданного значения из списка

Идея решения:

Базис рекурсии - если первый элемент окажется удаляемым, то нужно перейти к удалению заданного значения из хвоста списка. Результатом в данном случае должен стать список, полученный путем удаления всех вхождений искомого значения из хвоста первоначального списка. Шаг рекурсии будет основан на том, что если первый элемент списка не совпадает с тем, который нужно удалять, то он должен остаться первым элементом результата, и нужно переходить к удалению заданного значения из хвоста исходного списка. Полученный в результате этих удалений список должен войти в ответ в качестве хвоста.

Решение

```
delete_all(_,[],[]).

delete_all(X,[X|L],L1):– delete_all (X,L,L1).

delete_all (X,[Y|L],[Y|L1]):– X<>Y,
                              delete_all (X,L,L1).
```