Рекурсивные структуры данных

Списки в **Prolog**

Список как частный вид структуры

Определение. Под списком понимается упорядоченная последовательность **однотипных** элементов, которая может иметь произвольную длину.

Признак ***упорядоченный*** указывает на то,что порядок элементов в последовательности является существенным и список [1,2,3] не эквивалентен списку [3,2,1].

Элементами списка могут быть константы, переменные, структуры, последние могут включать в себя другие списки.

В Прологе списки -один из частных видов структур. Список -это либо пустой список, не содержащий ни одного элемента, либо структура, имеющая два компонента : голову и хвост. Конец списка представляется как хвост, который является пустым списком.

Способы представления списков

Функторный, графический и скобочный

При использовании функторной формы записи голова и хвост являются компонентами функтора, обозначаемого точкой «.».

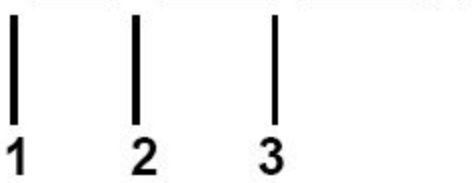
При использовании графической формы записи список представляется как специального вида дерево, «растущее» слева направо, причем ветви направлены вниз («виноградная гроздь»).

При использовании скобочной формы записи последовательность элементов списка, разделенных запятыми, заключается в квадратные скобки.

Пример : список из трех чисел.

Функторная форма : .(1,.(2,.(3,[ ])))

Графическая форма :



Скобочная форма : [1,2,3]

Описание списков в **Prolog**

Графическая форма записи списка в виде «виноградной лозы» удобна для записи списков на бумаге, но в Пролог-программах не используется. Функторная форма записи оказывается неудобной для записи сложных списков. В Прологе для записи списков используется скобочная форма записи.

Работа со списками основана на расщеплении на *голову* и *хвост*: [Head| Tail].

Примеры :
[1,2,3] -Head :1, Tail : [2,3] ; [[1], [2]] -Head : [1], Tail : [[2]].

[1 | [2]]. Здесь Head : 1, Tail : [2].

Применение списков в программе

Домен списка описывается в разделе domains, а работающий со списком предикат - в разделе predicates. Сам список задается в программе либо в разделе clauses, либо в разделе goal.

Пример :

```
    domains
s_list=symbol*
    predicates
print_list(s_list)
```

Правила сопоставления списков

Сопоставление списков - конкретизация переменных

Список 1	Список 2	Результат
[X,Y,Z]	[cat,dog,mouse]	X=cat Y=dog Z=mouse
[dog]	[X|Y]	X=dog Y=[]
[X,Z|Y]	[cat,dog,mouse]	X=cat Z=dog Y=[mouse]
[[b,c]|Z]	[[X,Y]|[w,b]]	X=b Y=c Z=[[w,b]]
[X,Y|Z,W]	Синтаксически некорректная конструкция списка	
[white,cat]	[cat,X]	Сопоставление невозможно

Печать элементов списка

print_list(Список)

```
domains n=integer*
predicates print_list(n)

clauses
```

```
print_list([]):- ! .
print_list([Head|Tail]) :-
write(Head), nl, print_list(Tail).
```

```
goal print_list([ 1, 2, 3,
4]).
```

Определение длины списка

length(Список, Количество)

```
domains n=symbol*
predicates
length(n, integer)
clauses
length([],0):-!.
length([_|T],N) :-
    length(T,N1),
    N = N1+1.
```

Goal length([a, b, c], N),
write(N).

Предикат length не может использоваться для генерации списка переменных заданной длины

- **Упражнение:** создать предикат **для** генерации списка заданной длины.

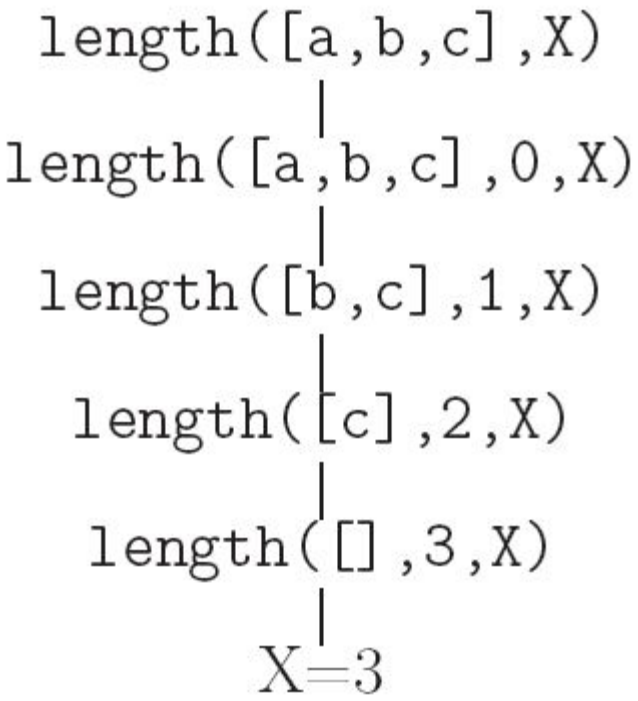
Определение длины списка

length(Список, Количество)

За счет введения дополнительного аргумента (аккумулятора) удается проводить все вычисления до рекурсивного вызова.

```
domains n=symbol*
predicates length(n,
integer)
length(n,integer,integer)
clauses
  length(L,N) :- length(L,0,N).

  length([_|T],S,N) :-
      S1 = S+1,
  length(T,S1,N).
  length([],N,N).
```




```
Goal length([a, b, c], X),
write(X).
```

Определение суммы элементов списка

sum(Список, Сумма)

```
sum([],S,S):-!.
sum([X|T],S,R) :-
S1=S+X,
    sum(T,S1,R).
```

```
domains n=integer*
predicates
sum(n, integer, integer)
```

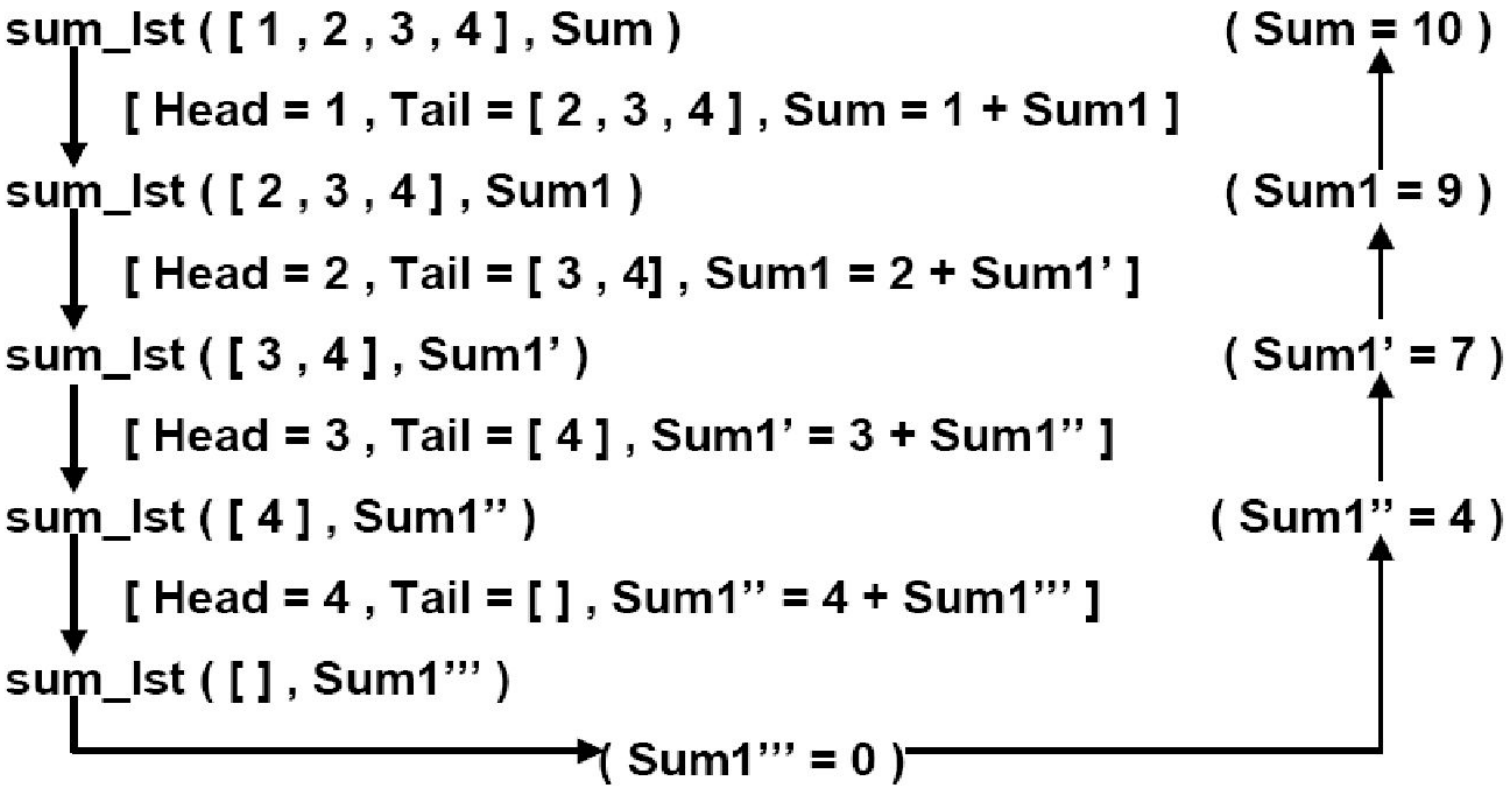
```
sum([X],X):-!.
sum([X|T],S) :-
    sum(T,S1),
    S = S1+X.
```

```
domains n=integer*
predicates
sum(n, integer)
```

Этапы решения задачи «Нахождение суммы элементов списка»

```
sum_lst([], 0):-!.
sum_lst([Head|Tail], Sum) :-
    sum_lst(Tail, Sum1),
    Sum = Head + Sum1.
```

Goal sum_lst([1, 2, 3, 4], S), write(S).



Принадлежность элемента списку

member(Элемент, Список)

```
Goal X=[1,2,3,4], member(7,X),
write(yes);write(no).
Goal  member(1,[]),
write(yes);write(no).
```

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

```
domains n=integer*
predicates
member(integer,n)
```

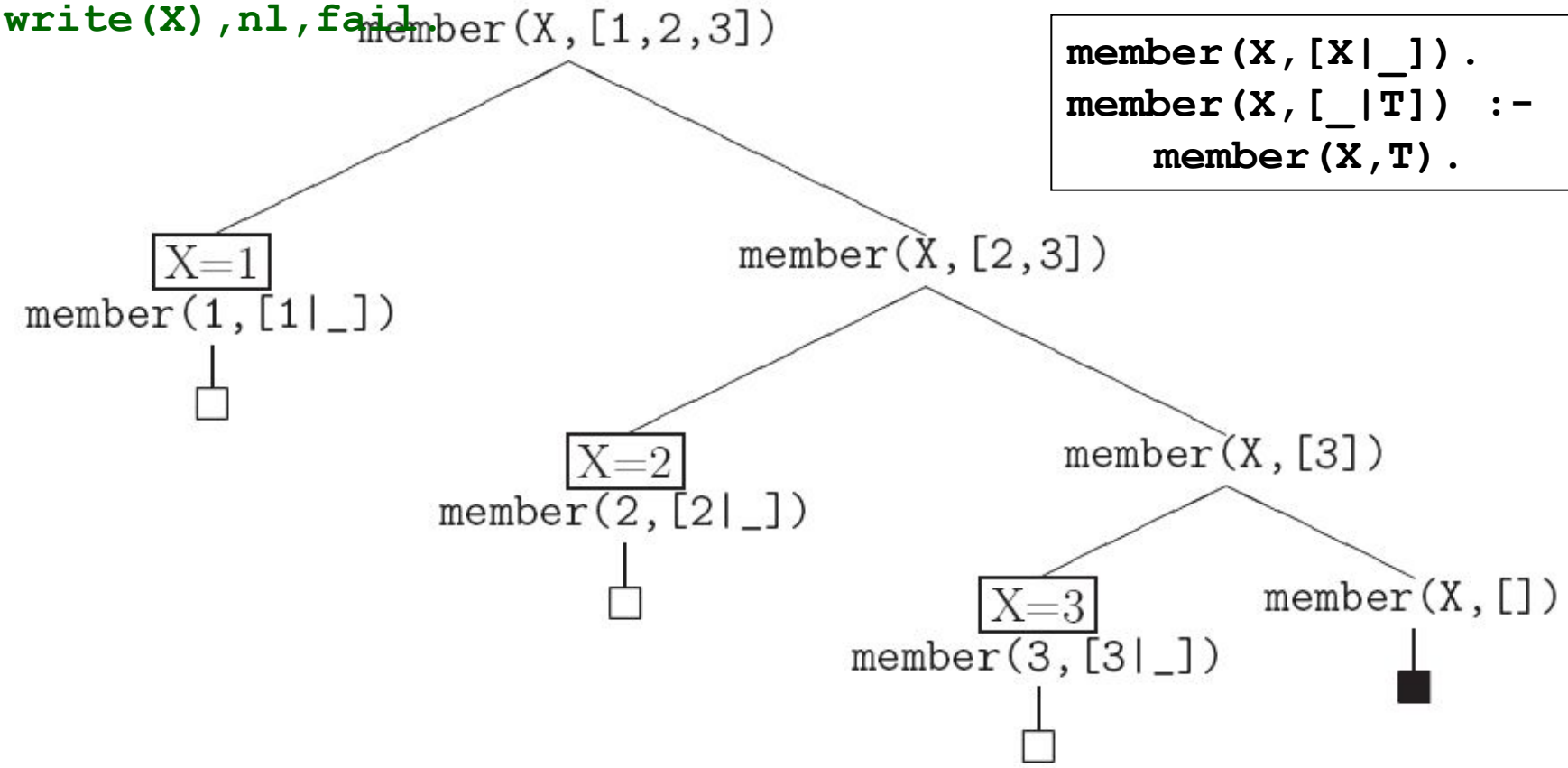
- member используется не только для проверки, но и для генерации последовательности значений

```
Goal member(X,[1,2,3]), write(X), nl, fail.
```

member для генерации значений

```
Goal member(X,[1, 2, 3]),
write(X),nl,fail.
```

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



Удаление элемента из списка

remove(Элемент, Список,Результат)

```
remove(X,[X|T],T):-!.
remove(X,[H|T],[H|R]):-
    remove(X,T,R).
```

```
domains n=integer*
predicates
remove(integer,n,n)
```

Goal **remove(1,[2,1,4,7,1,1],R),write(R).**

- Удаляется лишь одно вхождение элемента.
- **Упражнение:** построИТЬ предикат remove, который в случае, если элемента нет в списке, возвращает исходный список.
- **Упражнение:** построИТЬ предикат remove_all удаления всех вхождений элемента из списка.

Конкатенация (объединение) списков

append(Список, Список, Результат)

Постановка задачи. Требуется построить список – результат присоединения одного списка к другому.

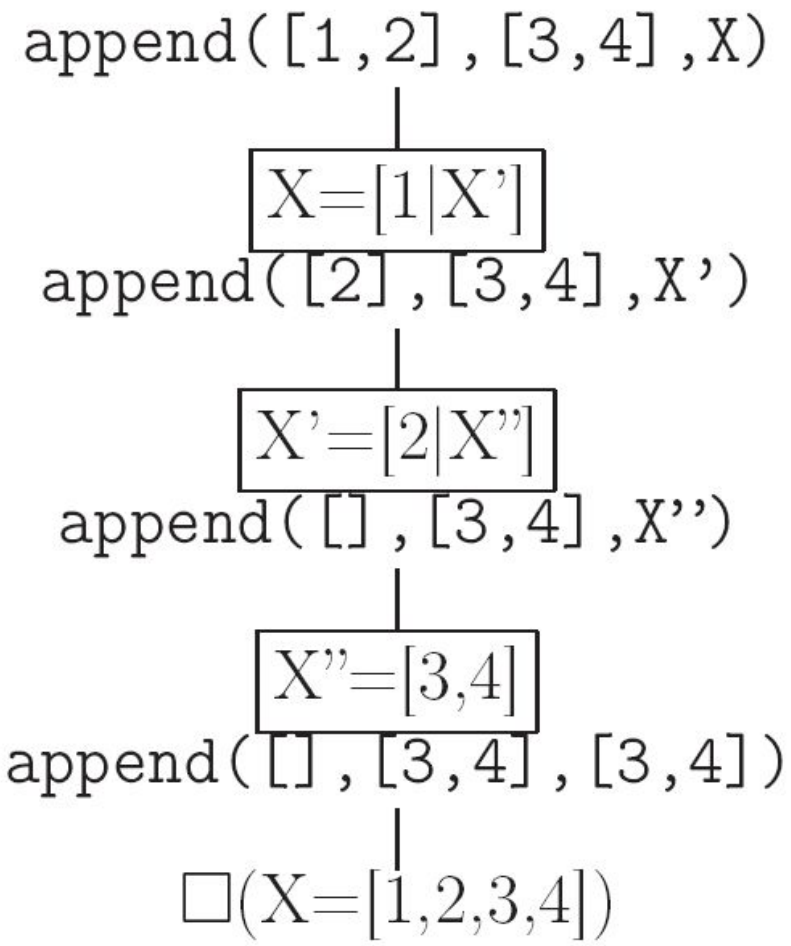
```
append([],L,L):-!.
append([X|T],L,[X|R]):-
    append(T,L,R).
```

```
goal append([1, 2],[3,
4],L),write(L).
```

goal

```
Y=[3],append(X,Y,[1,2,3]),write(X),nl,write(Y).
```

append для конкатенации




```
append([],L,L):-!.
append([X|T],L,[X|R]):-
    append(T,L,R).
```

Обращение (реверсирование) списка

```
reverse(L,R) :-
reverse(L,[],R).
reverse([],R,R).
reverse([X|T],L,R):-
reverse(T,[X|L],R).
```

```
domains n=integer*
predicates
reverse(n,n)
reverse(n,n,n)
```

```
goal
reverse([1,2,3,4,5],Y),write(Y).
```

```
reverse([],[]).
reverse([X|T],L) :-
    reverse(T,R),
    append(R,[X],L).
```

```
domains n=integer*
predicates append(n,n,n)
           reverse(n,n)
```

```
goal
reverse([1,2,3,4,5],Y),write(Y).
```