Представление и обработка списков

Список - динамическая структура с хотя бы одним элементом (головой) и остальное- хвост. Синтаксически список в термах.

“.” ( a, “.”( b, “.” ( c, []))) //как бы составной терм

[] – пустой список, но вообще атом

// фото – 14.00 – представление списка примера выше

[a, b, c]

[a,b,c,d] = [a, b | [ c, d] ] – начало из a, b, а остаток обязательно список = [ a | [b, c, d]] – внутри остатка еще список быть не может, только одинарный = [a , b , c | [d] ] = [a, b, c , d | [] ]

Любой элемент списка, как и в лисп, может быть список внутри. Кроме этого, также могут быть внутри переменные, именованные могут хранить значения, анонимные сохраняет только на один шаг обработки значение.

{ X = [a], Y = [b , c] }

[ X, Y] => [ [a] | [b, c] ] = 1 случай [ [a] , b , c ],

2 случай

Примеры:

1. [ L ] = [ ] – унификация невозможна, так как конкретизация двух разных объектов  
   [ L ] = [a | [] ] => { L = a } – унификация успешна  
   [ L ] = [ a, b, c ]   
   [ L, b ] = [ a, b ] => { L = a }  
   [ [L], b] = [a, b] – унификация невозможна, так как у первого голова это список, а у второго – атом.
2. [ a, L ] = [ a, b, c] – невозможно, так как разная арность  
   [ a | [L] ] = [ a | [b, c] ] => { L = [b, c] }  
   [ a, L ] = [ a, b | [c] ] => { L = [b, c] }

Формирование предикатов(процедура, описывающая знания)

Проверка является ли аргумент списком.

List (L)

Конкретизированную переменную L мы можем считать списком тогда, когда аргумент является пустым списком или непустым с разбитыми головой и хвостом. // *а я уже ждала подвох*

List (L) :- L = []. //проверка, является ли список пустым  
List (L) :- L = [ H | T ] , list ( T ). //проверка, есть ли голова и хвост

List ( [ ] ).  
List ( [ \_ | T ]) :- list ( T ). //аналогично примерам выше

Доказательство состоит из шагов, которые нумеруются римскими цифрами.

List ( [ ] ).  
List ( [ \_ | T ]) :- list ( T ).

Goal

List ( [ a, b, c ] )

TP: list ( [ a, b, c] )  
шаг 1 ТЦ: list ( [ a, b, c] )  
 ПРI : [ ] = [ a, b, c ] => униф невозм  
 ПРII : [ \_ | T1 ] = [ a, b, c ] => { T1 = [b, c] }  
 TP: list ( [ b, c] )  
шаг 2 ТЦ: list ( [ b, c] )  
 ПРI : [ ] = [ b, c ] => униф невозм  
 ПРII: [ \_ | T2] = [ b, c ] => { T2 = [c] }  
 TP: list ( [c] )  
шаг 3 ТЦ : list ( [c] )  
 ПРI : [] = [c] => униф невозм  
 ПРII : [ \_ | T3 ] = [c] => {T3 = [ ] }  
 ТР: list ( [ ] )  
шаг 4 ТЦ: list ( [ ] )  
 ПРI : [ ] = [ ] =>успешно  
 ТР – пусто, откат  
 ТЦ: list ( [ ] )  
{ПРII [ \_ | T4 ] = [ ] => по завершении работы}

Member – первый арг – элемент проверки, второй – список

member ( X, [ H | T ] ) :- X = H.  
member ( X, [ \_ | T ] ) :- member ( X, T ). //аналогично мембер выше

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

Goal  
 member ( a, [ b, a, c] ) //ему подойдет второе правило.

Что мы должны сделать для тренировки – подсчитать длину списка на 1 уровне(советует использовать одноуровневые списки пока что), найти произведение элементов числового списка, организовать объединение двух списков, научиться удалять элементы из списка, создавать список их элементов, стоящих на четных/нечетных местах, найти максимум/минимум.