Lists in Prolog

Brief introduction

Atoms

An atom is either:

1. A string of characters made up of upper-case letters, lower-case letters, digits, and the underscore character, that begins with a lower-case letter.
2. An arbitrary sequence of characters enclosed in single quotes.
3. A number (s). Real numbers aren't particularly important in typical Prolog applications. So although most Prolog implementations do support floating point numbers or floats (that is, representations of real numbers such as 1657.3087)

Variables

A variable is a string of upper-case letters, lower-case letters, digits and underscore characters that starts either with an upper-case letter or with an underscore.

For example, X , Y ,Variable , _tag , X_526 , List , List24 , _head , Tail , _input and Output are all Prolog variables.

The variable _ (that is, a single underscore character) is rather special. It's called the anonymous variable.

Facts are complex terms which are followed by a full stop.

Rules are of the form *Head* :- *Body*. => if Body is true, then the Head is also true!

Arithmetic operations:

The built-in predicate **is** means the asignments: Var2 is Var1+1

Arithmetic examples	Prolog Notation
x < y	X < Y.
x ≤ y	X =< Y.
x = y	X =:= Y.
x ⁄= y	X =\= Y.
x ≥ y	X >= Y
x > y	X > Y

Lists

Intuitively: sequences or enumerations of things;

In Prolog: a list is a special kind of data structure, i.e., special kinds of Prolog terms.

Prolog lists either look like this:

the empty list: []

non-empty lists: [2,3,4,a,bv,45c,[3,5,c]]

Prolog has a special built-in operator | which can be used to decompose a list into its head and tail. It is important to get to know how to use | , for it is a key tool for writing Prolog list manipulation programs.

Any non-empty list can be splitted using |.

eg: L=[2,3,4,a,bv,45c,[3,5,c]] => H=2, T=[3,4,a,bv,45c,[3,5,c]]

L1=[20] => H=20, T=[]

The standard representation of a list in Prolog is:

- *the empty list*: [], which is an atomic element
- *the list containing at least 1 element*: [H|T], where H is the first element in the list (the *head*), while T represents the rest of the list (its *tail*). Thus, H can be any Prolog object, while T is a list.

Prolog tools: SWI Prolog or SWISH Prolog

Prolog is case-sensitive!

1. Multiply elements of a list with a constant value

```
%mmul(k-integer,L-initial list, R-Resulted list)
%flow model: (i i i), (i i o)
mmul(_,[],[]).
mmul(K,[H|T],[HR|TR]):-
        HR is K*H,
        mmul(K,T,TR).
%mmul(2,[2,3],R).
```

2. Add an element at the end of a list.

```
% addE(L-list of elements, E-elements to be added in list; R-resulted list)

addE([],E,[E]).
addE([H|T],E,[H|R]):-
        addE(T,E,R).
```

3. Compute number of occurrences of an element in a list.

```
%nOcc(L-list of elements, E-element, N-number of occurences)
%flow model (i i i), (i i o)
nOcc([],_,0).
nOcc([H|T],E,N):-
        H=E,
        nOcc(T,E,N1),
        N is N1+1.
nOcc([H|T],E,N):-
        H\=E,
        nOcc(T,E,N).

%nOcc([2,3,4,2],2,N).
```

```
%For a list of elements, compute the sum.
%sum(L-list, S-result, integer)
%flow model: sum(i,o)

suma([],0).
suma([H|T],S):-
        suma(T,ST),
        S is H+ST.
```

trace, *suma*([2,4],S).

Call: *suma*([2, 4], _4030)
Call: *suma*([4], _4306)
Call: *suma*([], _4306)
Exit: *suma*([], 0)
Call: _4310 is 4+0
Exit: 4 is 4+0
Exit: *suma*([4], 4)
Call: _4030 is 2+4
Exit: 6 is 2+4
Exit: *suma*([2, 4], 6)

S = 6

```
%product of even numbers from a list
%mypro(L-list, R-result, integer)
%mypro(i,o) - flow model

mypro([],1).
mypro([H|T],R):-
    mod(H,2)=:=0,
    mypro(T,RT),
    R is RT*H.

mypro([H|T],R):-
    mod(H,2)=\=0,
    mypro(T,RT),
    R is RT.
```

trace, *mypro*([2,3,4],R).

Call: *mypro*([2, 3, 4], _4716)
Call: 2 mod 2=:=0
Exit: 2 mod 2=:=0
Call: *mypro*([3, 4], _4998)
Call: 3 mod 2=:=0
Fail: 3 mod 2=:=0
Redo: *mypro*([3, 4], _5002)
Call: 3 mod 2=\=0
Exit: 3 mod 2=\=0
Call: *mypro*([4], _5004)
Call: 4 mod 2=:=0
Exit: 4 mod 2=:=0
Call: *mypro*([], _5010)
Exit: *mypro*([], 1)
Call: _5014 is 1*4
Exit: 4 is 1*4
Exit: *mypro*([4], 4)
Call: _5014 is 4
Exit: 4 is 4
Exit: *mypro*([3, 4], 4)
Call: _4716 is 4*2
Exit: 8 is 4*2
Exit: *mypro*([2, 3, 4], 8)

R = 8

```
%Insert an element E on a position P.
%First position is considered 1
%ins(L-list,E-elem, P-insertion position,
%    CP-current position, R-resulted list)
%flow model (i,i,i,i,o)
%ins([10,11,12],1000,2,1,R).
%R = [10, 1000, 11, 12]

ins([],E,IP,1,[E]).
ins([],E,IP,CP,[]).
ins(L,E,IP,CP,R):-
        CP=:=IP,
        NewCP is CP+1,
        ins(L,E,IP,NewCP,RT),
        R=[E|RT].
ins([H|T],E,IP,CP,R):-
        NewCP is CP+1,
        ins(T,E,IP,NewCP,RT),
        R=[H|RT].
```

```
%Particular insertion:
%In a list, on all positions multiple of M,
%insert an element E
%First position is considered 1
%L=[10,11,12], E=1000,M=2
% => R = [10, 1000, 11, 1000, 12]
%L2=[10,11,12,13,14], E=1000,M=3
% => R = [10, 11, 1000, 12, 13, 1000, 14]

ins([],E,CP,[],M).
ins([],E,CP,[E],M):-mod(CP,M)=:=0.
ins(L,E,CP,R,M):-mod(CP,M)=:=0,
        NewCP is CP+1,
        ins(L,E,NewCP,RT,M),
        R=[E|RT].
ins([H|T],E,CP,R,M):-NewCP is CP+1,
        ins(T,E,NewCP,RT,M),
        R=[H|RT].
```

```
%Delete an element E from a list L
delete1(_, [], []).
delete1(E, [H|T], R):-
            H=:=E,
            R=T.
delete1(E, [H|T], R):-
            H=\=E,
            delete1(E, T, RT),
            R=[H|RT].

%%Delete all the occurrences of a
% element E from a list

delete_all(_, [], []).
delete_all(E, [H|T], R):-
            H=:=E,
            delete_all(E, T, RT),
            R=RT.
delete_all(E, [H|T], R):-
            delete_all(E, T, RT),
            R=[H|RT].
```


```
%Se da o lista de numere intregi.
%Sa se gaseasca cmmdc al elems din lista,
%si sa se stearga cmmdc din lista de pe poz pare.

% ex: L = (5, 10, 15, 5, 25, 5, 30)
% c = 5, LR = (5,10,15,25,30)

%gcd(M-first number,N-second number,R-result)
%gcd(i,i,o)
gcd(M, N, R) :-
    M =:= N,
    R is M.
gcd(M, N, R) :-
    N>M,
    Y is N-M,
    gcd(M, Y, R).
gcd(M, N, R) :-
    N<M,
    Y is M-N,
    gcd(Y, N, R).
```

```
%listGCD(L-list,R-res, integer)
%listGCD(i,o)-flow model
listGCD([E], E).
listGCD([H|T], R) :-
    listGCD(T, R1),
    gcd(H, R1, R).


%removeEl(L-list,E-removed elem,CP-currentpos,RL-reslist).
%removeEl(i,i,i,o) - flow model
removeEl([], _, _, []).
removeEl([H|T], E, CurrentPos, [H|R]) :-
    H =\= E,
    CP1 is CurrentPos + 1,
    removeEl(T, E, CP1, R).

removeEl([_|T], E, CurrentPos, R) :-
    CurrentPos mod 2 =:= 0,
    CP1 is CurrentPos + 1,
    removeEl(T, E, CP1, R).

removeEl([H|T], E, CurrentPos, [H|R]) :-
    CP1 is CurrentPos + 1,
    removeEl(T, E, CP1, R).

%main(L-list,R-resultedlist)
%main(i,o) flow model
main(L, R) :-
    listCmmdc(L, R1),
    removeEl(L, R1, 1, R).
```