Seminar 4 – Backtracking in Prolog

Backtracking – the facility of Prolog to determine all solutions for a problem.

In this case, we will have predicates with more than one solution => **non-deterministic predicates.**

Until now, all our predicates were **deterministic predicates** – predicate with just **one** solution.

To collect all solutions of a predicate, we have the built-in predicate **findall**.

The built-in predicate findall collect all solutions of a predicate and put them in a single list.

findall (ResPredicatePartial, PredicatePartial (InitialList, ResPredicatePartial), FinalResult).

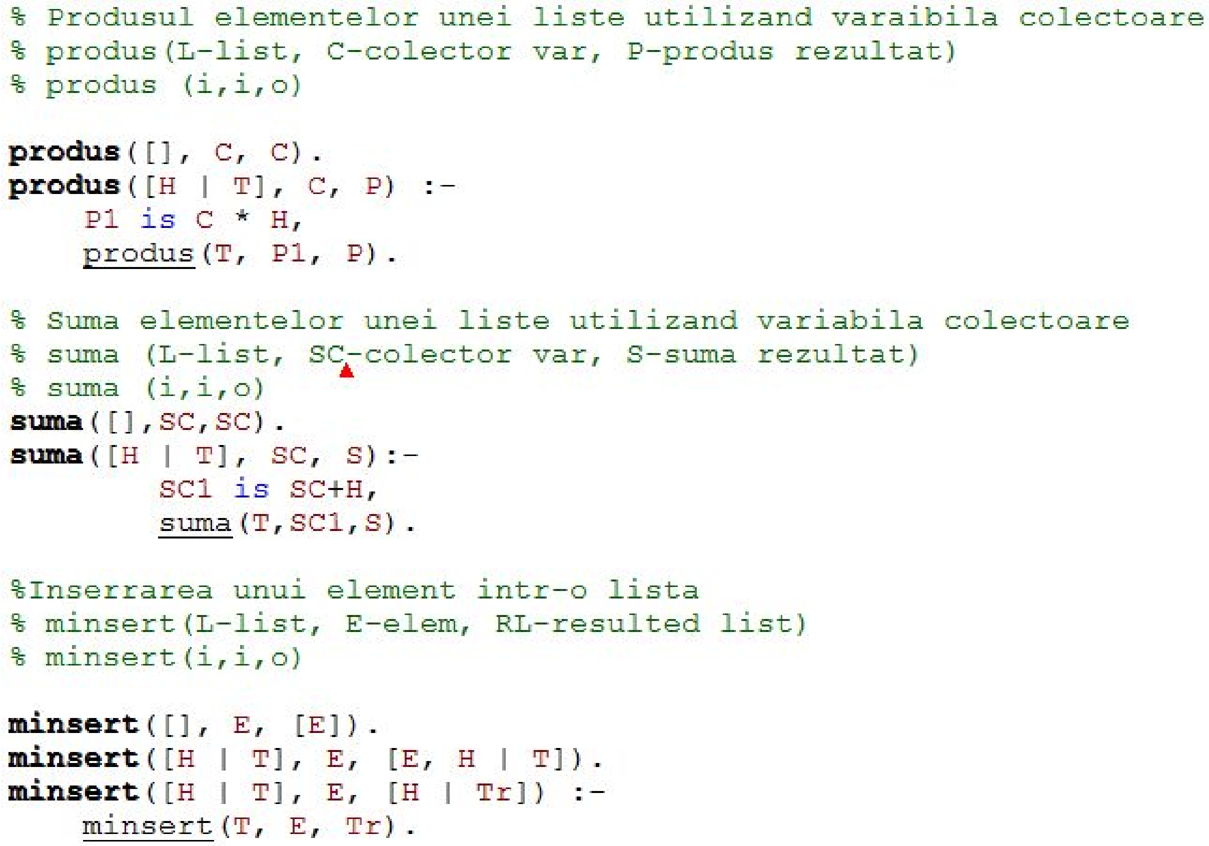
Eg. If we already have a predicate for onesolution (L, ROS), *L - initial list, ROS - resulted list and flow model (i, o),* then collecting all solution in a **main predicate** called allsolutions(L, RALL), *L - initial list, RALL – resulted list and flow model (i, o),* we will use **findall** as follows:

allsolutions (L, RALL) :- findall (ROS, onesolution (L, ROS), RALL).

Probleme:

Se da o lista L. Sa se genereze lista tuturor aranjamentelor de K elemente din lista care au produsul P si suma S.

Ex. L=[1,2,3,10], K=2, P=30, S=13 atunci rezultatul este R = [[3, 10], [10, 3]].



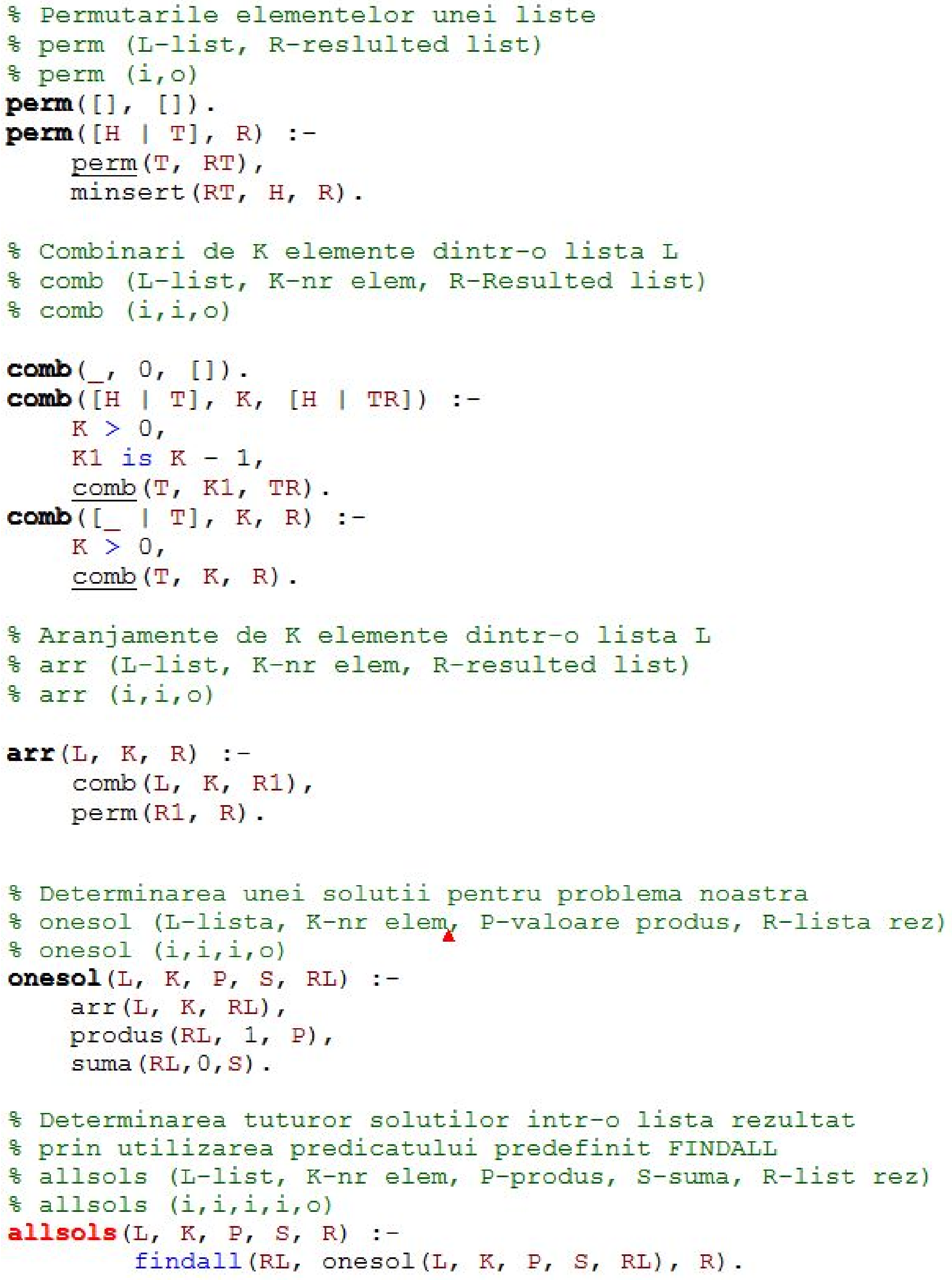
```
% Produsul elementelor unei liste utilizand varaibila colectoare
% produs(L-list, C-colector var, P-produs rezultat)
% produs (i,i,o)

produs([], C, C).
produs([H | T], C, P) :-
    P1 is C * H,
    produs(T, P1, P).

% Suma elementelor unei liste utilizand variabila colectoare
% suma (L-list, SC-colector var, S-suma rezultat)
% suma (i,i,o)
suma([],SC,SC).
suma([H | T], SC, S):-
        SC1 is SC+H,
        suma(T,SC1,S).

%Inserrarea unui element intr-o lista
% minsert(L-list, E-elem, RL-resulted list)
% minsert(i,i,o)

minsert([], E, [E]).
minsert([H | T], E, [E, H | T]).
minsert([H | T], E, [H | Tr]) :-
    minsert(T, E, Tr).
```



```
% Permutarile elementelor unei liste
% perm (L-list, R-reslulted list)
% perm (i,o)
perm([], []).
perm([H | T], R) :-
    perm(T, RT),
    minsert(RT, H, R).

% Combinari de K elemente dintr-o lista L
% comb (L-list, K-nr elem, R-Resulted list)
% comb (i,i,o)

comb(_, 0, []).
comb([H | T], K, [H | TR]) :-
    K > 0,
    K1 is K - 1,
    comb(T, K1, TR).
comb([_ | T], K, R) :-
    K > 0,
    comb(T, K, R).

% Aranjamente de K elemente dintr-o lista L
% arr (L-list, K-nr elem, R-resulted list)
% arr (i,i,o)

arr(L, K, R) :-
    comb(L, K, R1),
    perm(R1, R).


% Determinarea unei solutii pentru problema noastra
% onesol (L-lista, K-nr elem, P-valoare produs, R-lista rez)
% onesol (i,i,i,o)
onesol(L, K, P, S, RL) :-
    arr(L, K, RL),
    produs(RL, 1, P),
    suma(RL,0,S).

% Determinarea tuturor solutilor intr-o lista rezultat
% prin utilizarea predicatului predefinit FINDALL
% allsols (L-list, K-nr elem, P-produs, S-suma, R-list rez)
% allsols (i,i,i,i,o)
allsols(L, K, P, S, R) :-
        findall(RL, onesol(L, K, P, S, RL), R).
```