```
/* Interogati:
?- setof(X, member((X,Y,Z),[(1,a,V),(2,a,W),(2,b,U),(2,b,W),(3,a,V),(3,b,U),(3,c,V)]), L).
?- bagof(X, member((X,Y,Z),[(1,a,V),(2,a,W),(2,b,U),(2,b,W),(3,a,V),(3,b,U),(3,c,V)]), L).
Exemplu de cuantificare existentiala pentru mai multe variabile in setof si bagof:
?- setof(X,
(Y,Z)^member((X,Y,Z),[(1,a,V),(2,a,W),(2,b,U),(2,b,W),(3,a,V),(3,b,U),(3,c,V)]), L).
?- bagof(X,
(Y,Z)^member((X,Y,Z),[(1,a,V),(2,a,W),(2,b,U),(2,b,W),(3,a,V),(3,b,U),(3,c,V)]), L).
?- findall(X, member((X,Y,Z),[(1,a,V),(2,a,W),(2,b,U),(2,b,W),(3,a,V),(3,b,U),(3,c,V)]),
L).
Ultimele doua interogari intorc acelasi raspuns.
*/

% Produsul cartezian (de multimi, i.e. generat fara duplicate), cu setof:

prodmult(L,M,LxM) :- setof((X,Y), (member(X,L), member(Y,M)), LxM), !.
prodmult(_,_,[]).

% Produsul cartezian de liste, cu bagof, respectiv findall:

prodlist(L,M,LxM) :- bagof((X,Y), (member(X,L), member(Y,M)), LxM), !.
prodlist(_,_,[]).

prodliste(L,M,LxM) :- findall((X,Y), (member(X,L), member(Y,M)), LxM).

% Produsul cartezian de liste, definit recursiv, fara metapredicate:

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

```
prodcart(L,[H|T],P) :- prodsgl(L,H,Q), prodcart(L,T,R), append(Q,R,P).

prodsgl([],_,[]).
prodsgl([H|T],X,[(H,X)|U]) :- prodsgl(T,X,U).

% Produsul cartezian (de multimi, i.e.) fara duplicate:

prodcartmult(L,M,P) :- prodcart(L,M,Q), elimdupl(Q,P).

/* Eliminarea duplicatelor dintr-o lista, cu pastrarea
primei aparitii a fiecarui element: */

elimdupl([],[]).
elimdupl([H|T],[H|L]) :- sterge(H,T,U), elimdupl(U,L).

sterge(_,[],[]).
sterge(H,[H|T],L) :- sterge(H,T,L), !.
sterge(X,[H|T],[H|L]) :- sterge(X,T,L).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inversa([],[]).
inversa([H|T],L) :- inversa(T,U), append(U,[H],L).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

implica(P,Q) :- not(P) ; Q.
echiv(P,Q) :- implica(P,Q), implica(Q,P).
```

```
/* Fie A,B,C multimi arbitrare si x arbitrar.
Variabilelor (booleene, adica menite a lua ca valori expresii booleene, destinate spre a
fi unificate cu expresii booleene; pentru demonstratii ca mai jos, cu constantele booleene
false sau true) A,B,C le dam ca valori urmatoarele enunturi:
A: x apartine multimii A
B: x apartine multimii B
C: x apartine multimii C
*/

/* Predicatul listaValBool trebuie apelat cu argumentul dat de o lista de variabile.
Cand este apelat cu o lista L de N variabile distincte (i.e. o lista L de lungime N
continand variabile doua cate doua distincte), acest predicat intoarce 2**N solutii, anume
listele de N valori booleene, pe care le si afiseaza pe ecran, urmate de cate o trecere la
linie noua. */

listaValBool(L) :- listaBool(L), write(L), nl.

listaBool([]).
listaBool([H|T]) :- member(H,[false,true]), listaBool(T).

/* Folosind predicatul de mai sus, putem instantia oricate variabile cu constante booleene
false sau true, pentru acest gen de demonstratii: */

% Sa demonstram ca: (A<=C si B<=C) <=> AUB<=C.

ms2multincla3a(A,B,C) :- implica(A,C), implica(B,C).
md2multincla3a(A,B,C) :- implica(A;B, C).
```

```
propr2multincla3a(A,B,C) :- echiv(ms2multincla3a(A,B,C),md2multincla3a(A,B,C)).

dem2multincla3a :- not((listaValBool([A,B,C]), not(propr2multincla3a(A,B,C)))).

% Sa demonstram ca: A\B=A\(A^B).

difinters(A,B) :- echiv((A,not(B)), (A,not((A,B)))).

demdifinters :- not((listaValBool([A,B]), not(difinters(A,B)))).

/* Fie A,B,C multimi arbitrare. Fie x arbitrar.
Notam cu variabilele booleene _a,_b,_c urmatoarele enunturi:
_a: x apartine lui A
_b: x apartine lui B
_c: x apartine lui C
Sa demonstram ca: (A<=B si A<=C) <=> A<=B^C.
*/

incl2multvsinters(_a,_b,_c) :-
        echiv((implica(_a,_b), implica(_a,_c)), implica(_a,(_b,_c))).

demincl2multvsinters :- not((listaValBool([_a,_b,_c]),
                        not(incl2multvsinters(_a,_b,_c)))).

% Sa demonstram ca: A\B=A\(A^B), la fel ca mai sus, dar cu aceste nume de variabile.

difsufinters(_a,_b) :- echiv((_a,not(_b)), (_a,not((_a,_b)))).
```

```
demdifsufinters :- not((listaValBool([_a,_b]), not(difsufinters(_a,_b)))).

/* Fie T,A,B multimi a.i. A<=T si B<=T. => A^T=A si B^T=B.
   Notez:
cuantificatorul universal cu -V;
pentru orice multime M cu M<=T, cu -M=T\M;
pentru orice x si orice multime M, cu "x in M" faptul ca x apartine lui M.
   A=B <=> A^T=B^T <=> (-Vx)(x in A^T <=> x in B^T)
<=> (-Vx)[(x in A si x in T) <=> (x in B si x in T)]
<=> (-Vx)[x in T => (x in A <=> x in B)]
<=> (-Vx in T)(x in A <=> x in B)
   A<=B <=> A^T<=B^T <=> (-Vx)(x in A^T => x in B^T)
<=> (-Vx)[(x in A si x in T) => (x in B si x in T)]
<=> (-Vx)[x in T => (x in A => x in B)]
<=> (-Vx in T)(x in A => x in B)
   Fie x in T, arbitrar, fixat.
   Pentru orice multime M cu M<=T, avem, intrucat x in T e adevarata:
x in -M <=> x in T\M <=> [x in T si not(x in M)] <=> not(x in M).
   Notam cu variabilele booleene A,B enunturile:
A: x apartine lui A
B: x apartine lui B
   Sa demonstram ca: A\B=A^-B.
*/

difeinterscomplem(A,B) :- echiv((A, not(B)), (A, not(B))). % triviala

demdifeinterscomplem :- not((listaValBool([A,B]), not(difeinterscomplem(A,B)))).
```

```prolog
% Sa demonstram ca: --A=A.

complemcomplem(A) :- echiv(not(not(A)), A).

demcomplemcomplem :- not((listaValBool([A]), not(complemcomplem(A)))).

% Sa demonstram prima lege a lui De Morgan: -(AUB) = -A ^ -B.

deMorgan1(A,B) :- echiv(not(A;B), (not(A), not(B))).

demdeMorgan1 :- not((listaValBool([A,B]), not(deMorgan1(A,B)))).

% Sa demonstram a doua lege a lui De Morgan: -(A^B) = -A U -B.

deMorgan2(A,B) :- echiv(not((A,B)), not(A);not(B)).

demdeMorgan2 :- not((listaValBool([A,B]), not(deMorgan2(A,B)))).

/* Cand sunt A si B parti complementare ale lui T: sa demonstram ca:
(AUB=T si A^B=0) <=> A=-B <=> B=-A. */

reunemulttot(A,B) :- echiv(A;B,true).

multdisj(A,B) :- echiv((A,B),false).

msechiv1particomplem(A,B) :- reunemulttot(A,B), multdisj(A,B).
```

```
egalcucomplem(A,B) :- echiv(A,not(B)).

echiv1particomplem(A,B) :- echiv(msechiv1particomplem(A,B), egalcucomplem(A,B)).

echiv2particomplem(A,B) :- echiv(egalcucomplem(A,B),egalcucomplem(B,A)).

particomplem(A,B) :- echiv1particomplem(A,B), echiv2particomplem(A,B).

demparticomplem :- not((listaValBool([A,B]), not(particomplem(A,B)))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Determinarea sublistelor unei liste:

sublista([],_).
sublista([H|T],[H|L]) :- sublista(T,L).
sublista([H|T],[_|L]) :- sublista([H|T],L). % fara aceasta regula rezulta prefixele

% Multimea (i.e. lista fara duplicate a) sublistelor unei liste:

sublistele(L,LS) :- setof(S, sublista(S,L), LS).

% Afisarea unei liste cu fiecare element pe alta linie:

afislista([]).
afislista([H|T]) :- write(H), nl, afislista(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Determinarea relatiilor binare R de la A la B:

relbin(A,B,R) :- prodcartmult(A,B,P), sublista(R,P).

% Determinarea multimii LR a relatiilor binare de la A la B, doua variante:

listarelbin(A,B,LR) :- setof(R, relbin(A,B,R), LR).

listrelbin(A,B,LR) :- prodcartmult(A,B,P), sublistele(P,LR).

/* Identificam orice functie cu graficul ei, astfel ca o functie de la A la B va fi
o relatie binara functionala totala de la A la B:
   f:A->B <=> f = {(a,f(a)) | a in A} <= AxB, unde (-Va in A)(f(a) in B).
Amintesc ca, pentru orice a in A si b in B:
   a f b <=> (a,b) in f <=> b=f(a).
*/

/* Predicatul functie(+listaA,+listaB,-Functie) determina functiile Functie de la
multimea (i.e. lista fara duplicate) listaA la multimea listaB: */

functie([],_,[]).
functie([H|T],B,[(H,FH)|L]) :- member(FH,B), functie(T,B,L).

% Determinarea multimii LF a functiilor de la A la B:

functiile(A,B,LF) :- setof(F, functie(A,B,F), LF), !.
functiile(_,_,[]).
```

```
% Testarea functionalitatii unei relatii binare R de la multimea A la o alta multime:

functionala(A,R) :- not((member(X,A), member((X,B1),R), member((X,B2),R), B1\=B2)).

/* Determinarea relatiilor binare functionale
(i.e. a functiilor partiale) R de la A la B: */

fctpart(A,B,R) :- relbin(A,B,R), functionala(A,R).

/* Determinarea multimii relatiilor binare functionale
(i.e. a functiilor partiale) R de la A la B: */

functiipart(A,B,LF) :- setof(F, fctpart(A,B,F), LF).
```