```
/* 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).

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

% Amintesc acest predicat pentru calculul inversei unei liste:

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)))).

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

/* Desigur, ca in predicatul listaValBool, putem instantia o lista de variabile cu
elementele oricarei liste de constante, date mai jos in argumentul ListaValori: */

listaVal(L,ListaValori) :- instantiazaLista(L,ListaValori), write(L), nl.

instantiazaLista([],_).
instantiazaLista([H|T],LVal) :- member(H,LVal), instantiazaLista(T,LVal).

/* Interogati:
?- listaVal([U,V,W],[a,b,c,d]).
```

```
si dati ; (in Prolog-ul desktop) /Next/100 (in Prolog-ul online) pentru a obtine toate cele
4x4x4=64 solutii.
Desigur, afisarea tuturor solutiilor interogarii anterioare poate fi obtinuta astfel:
?- listaVal([U,V,W],[a,b,c,d]), fail.
Sau astfel, cu tot cu numararea acestor solutii:
?- setof([U,V,W], listaVal([U,V,W],[a,b,c,d]), L), length(L,N).
*/

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

/* 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)))).

% 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).
```

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

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

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

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

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

demcharparticomplem :- not((listaValBool([A,B]), not(charparticomplem(A,B)))).

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

/* Prin metoda de mai sus am demonstrat semantic (i.e. prin calcul cu valori de adevar)
proprietati ale operatiilor si relatiilor intre multimi. Cum putem calcula rezultatele unor
astfel de operatii pentru multimi finite, reprezentate prin liste in Prolog? Desigur, nu
putem efectua in Prolog astfel de calcule pentru multimi infinite. */

% Reuniunea, ca lista fara duplicate:

reuniune(A,B,R) :- append(A,B,C), elimdupl(C,R).

% Intersectia (rezulta fara duplicate daca prima lista e fara duplicate):

inters([],_,[]).
inters([H|T],B,[H|L]) :- member(H,B), !, inters(T,B,L).
```

```
inters([_|T],B,L) :- inters(T,B,L).

% Intersectia, ca lista fara duplicate:

intersectie(A,B,I) :- inters(A,B,J), elimdupl(J,I).

/* Diferenta, prin recurenta dupa primul termen (daca acesta e fara duplicate, atunci
rezultatul e fara duplicate): */

dif([],_,[]).
dif([H|T],B,L) :- member(H,B), !, dif(T,B,L).
dif([H|T],B,[H|L]) :- dif(T,B,L).

% Diferenta, ca lista fara duplicate:

diferenta(A,B,D) :- dif(A,B,E), elimdupl(E,D).

/* Diferenta, prin recurenta dupa al doilea termen (daca primul termen e fara duplicate,
atunci rezultatul e fara duplicate): */

difer(A,[],A).
difer(A,[H|T],L) :- sterge(H,A,M), difer(M,T,L).

/* Diferenta, ca lista fara duplicate, cu a doua metoda de mai sus pentru obtinerea
diferentei inainte de eliminarea duplicatelor: */

diferen(A,B,D) :- difer(A,B,E), elimdupl(E,D).
```

```
/* Diferenta simetrica, ca lista fara duplicate, cu fiecare dintre cele doua metode de mai
sus pentru obtinerea diferentei: */

difsim(A,B,D) :- dif(A,B,AminusB), dif(B,A,BminusA), reuniune(AminusB,BminusA,D).

difersim(A,B,D) :- diferenta(A,B,AminusB), diferenta(B,A,BminusA),
                        append(AminusB,BminusA,D).

/* Interogati:
?- reuniune([1,2,3,4,5],[0,2,4,6],Cat).
?- reuniune([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- intersectie([1,2,3,4,5],[0,2,4,6],Cat).
?- inters([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- intersectie([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- dif([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- diferenta([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- difer([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- diferen([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- difsim([1,2,2,4,5,5],[0,2,4,4,6],Cat).
?- difersim([1,2,2,4,5,5],[0,2,4,4,6],Cat).
*/

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

/* Desigur, in listele fara duplicate din Prolog conteaza ordinea elementelor, spre
deosebire de multimi. Doua liste fara duplicate sunt egale ca multimi ddaca fiecare dintre
```

```
ele este permutare a celeilalte: */

egalmult(A,B) :- permutare(A,B).

/* Predicat care sterge o singura aparitie a unui element intr-o lista, dar de pe o pozitie
arbitrara: */

stergeuna(_,[],_) :- fail.
stergeuna(H,[H|T],T).
stergeuna(X,[H|T],[H|L]) :- stergeuna(X,T,L).

/* Interogati:
?- stergeuna(a,[a,1,2,a,3,a,4,a],L).
si dati ;/Next pentru a obtine toate cele patru solutii.
Predicatul de mai sus poate fi folosit pentru a adauga un element la o lista pe o pozitie
arbitrara. Interogati:
?- stergeuna(a,DinCeLista,[1,2,3]).
si dati ;/Next pentru a obtine toate cele patru solutii.
De aceea, predicatul de mai sus poate fi folosit pentru a genera permutarile unei liste, in
aceasta maniera recursiva: */

permutare([],[]).
permutare([H|T],P) :- permutare(T,Q), stergeuna(H,P,Q).

/* Predicatul de mai sus poate fi folosit sub forma:
?- permutare([1,2,3],P).
dand ;/Next pentru a obtine toate cele 6 solutii, precum si sub forma:
```

```
?- permutare(P,[1,2,3]).
cu conditia ca, dupa a sasea solutie, anume P=[3,2,1] (inversa listei [1,2,3]), sa punem
punct, nemaicerand inca om solutie; altfel, dupa aceasta ultima solutie, predicatul
cicleaza. */

% Multimea permutarilor unei liste:

permutarile(L,LP) :- setof(P, permutare(L,P), LP).

% Afisarea unei liste cu fiecare element pe alta linie:

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

/* Interogati:
?- permutarile([1,2,3,4],L), afislista(L), length(L,N).
Numarul permutarilor listei [1,2,3,4] este N=24=4!. */

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

/* Determinarea sublistelor unei liste; daca lista e fara duplicate, reprezentand, asadar, o
multime, atunci obtinem partile (i.e. submultimile) multimii respective: */

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; daca lista e fara
duplicate, obtinem multimea partilor multimii respective: */

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

/* Putem testa daca o multime e submultime a alteia, indiferent de ordinea elementelor
listelor care le reprezinta, astfel: */

submultime(S,L) :- permutare(L,P), sublista(S,P).

/* La o interogare de forma:
?- submultime([4,2,1],[1,2,3,4,5]).
trebuie sa punem punct dupa primul true obtinut, altfel Prolog-ul va raspunde true de 20 de
ori, cate o data pentru fiecare permutare a listei [1,2,3,4,5] care contine elementele 4,2,1
in aceasta ordine (nu neaparat pe pozitii consecutive). Intr-adevar, daca numaram solutiile
interogarii de mai sus:
?- findall(true, submultime([4,2,1],[1,2,3,4,5]), L), length(L,N).
obtinem N=20. Putem afisa cele 20 de permutari P ale listei [1,2,3,4,5] in care elementele
4,2,1 apar in aceasta ordine (nu neaparat pe pozitii consecutive), astfel:
?- findall(P, (permutare([1,2,3,4,5],P), sublista([4,2,1],P)), L), afislista(L),
length(L,N).
*/

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

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

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

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

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

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

/* Interogati:
?- relatiibinare([a,b],[1,2,3],L), afislista(L), length(L,N).
?- relbinare([a,b],[1,2,3],L), afislista(L), length(L,N).
Avem N=64=2**6=2**(2x3) relatii binare de la multimea {a,b} la multimea {1,2,3}. */

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

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

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

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

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

functionala(F,A) :-  not((member(X,A), member((X,B),F), member((X,C),F), B\=C)).

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

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

/* Determinarea relatiilor totale de la A la B, cu doua metode de testare a totalitatii,
i.e. a definirii peste tot, adica pentru toate elementele lui A: */

reltot(R,A,B) :- relbin(R,A,B), tot(R,A,B).

tot(R,A,B) :- not((member(X,A), not((member(Y,B), member((X,Y),R))))).

reltotala(R,A,B) :- relbin(R,A,B), totala(R,A).

totala(R,A) :- not((member(X,A), not(member((X,_),R)))).

/* Interogati:
?- functiile([a,b],[1,2,3],L), afislista(L), length(L,N).
?- setof(F, (fctpart(F,[a,b],[1,2,3]), tot(F,[a,b],[1,2,3])), L), afislista(L), length(L,N).
```

```
?- setof(F, (fctpart(F,[a,b],[1,2,3]), totala(F,[a,b])), L), afislista(L), length(L,N).
Trebuie sa obtinem aceeasi multime de 3**2=9 functii, i.e. functii partiale totale, i.e.
relatii binare functionale totale, la fiecare dintre cele trei interogari anterioare. */

% Inversa unei relatii binare, calculata in doua moduri:

invrel([],[]).
invrel([(X,Y)|T],[(Y,X)|L]) :- invrel(T,L).

inversarel(R,Q) :- setof((Y,X), member((X,Y),R), Q), !.
inversarel(_,[]).

/* Interogati:
?- invrel([(a,1),(a,2),(b,1),(c,2),(c,3)],Inversa).
?- inversarel([(a,1),(a,2),(b,1),(c,2),(c,3)],Inversa).
Predicatul setof face si o sortare a solutiilor celui de-al doilea argument al sau; de aceea
listele obtinute la cele doua interogari de mai sus nu coincid, dar sunt egale ca multimi.
*/

/* Determinarea relatiilor binare injective de la A la B, cu doua metode de testare a
injectivitatii, prima direct cu definitia, in doua moduri, iar a doua folosind fiecare
dintre predicatele de mai sus pentru determinarea inversei unei relatii binare: */

relinj(R,A,B) :- relbin(R,A,B), inj(R,B).

inj(R,B) :-  not((member(Y,B), member((A,Y),R), member((U,Y),R), A\=U)).
```

```
injectiv(R) :- not((member((U,Y),R), member((X,Y),R), U\=X)).

relinjectiv(R,A,B) :- relbin(R,A,B), injectiv(R).

relinjectiva(R,A,B) :- relbin(R,A,B), injectiva(R,B).

injectiva(R,B) :- invrel(R,I), functionala(I,B).

relinject(R,A,B) :- relbin(R,A,B), inject(R,B).

inject(R,B) :- inversarel(R,I), functionala(I,B).

/* Determinarea relatiilor binare surjective de la A la B, cu doua metode de testare a
surjectivitatii, prima direct cu definitia, in doua moduri (cu un predicat cu trei
argumente, respectiv unul cu doua argumente), a doua folosind cate un predicat de mai sus
pentru determinarea inversei unei relatii binare impreuna cu unul dintre cele pentru
testarea totalitatii unei relatii binare: */

relsurj(R,A,B) :- relbin(R,A,B), surj(R,A,B).

surj(R,A,B) :- not((member(Y,B), not((member(X,A), member((X,Y),R))))).

relsurjectiv(R,A,B) :- relbin(R,A,B), surjectiv(R,B).

surjectiv(R,B) :- not((member(Y,B), not(member((_,Y),R)))).

relsurjectiva(R,A,B) :- relbin(R,A,B), surjectiva(R,B).
```

```
surjectiva(R,B) :- invrel(R,I), totala(I,B).

relsurject(R,A,B) :- relbin(R,A,B), surject(R,A,B).

surject(R,A,B) :- inversarel(R,I), tot(I,B,A).

% Cateva modalitati de determinare a functiilor bijective de la A la B:

functiebij(F,A,B) :- functie(F,A,B), inj(F,B), surj(F,A,B).

functiebijectiva(F,A,B) :- functie(F,A,B), injectiva(F,B), surjectiva(F,B).

functiebiject(F,A,B) :- functie(F,A,B), inject(F,B), surject(F,A,B).

functiebijectiv(F,A,B) :- functie(F,A,B), injectiv(F), surjectiv(F,B).

/* Interogati:
?- setof(F, (functie(F,[a,b],[1,2,3]), inj(F,[1,2,3])), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b],[1,2,3]), injectiv(F)), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b],[1,2,3]), injectiva(F,[1,2,3])), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b],[1,2,3]), inject(F,[1,2,3])), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b],[1,2,3]), surj(F,[a,b],[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b],[1,2,3]), surjectiva(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b],[1,2,3]), surjectiv(F,[1,2,3])), L), afislista(L), length(L,N).
```

```
?- setof(F, (functie(F,[a,b],[1,2,3]), surject(F,[a,b],[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, functiebij(F,[a,b],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebijectiva(F,[a,b],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebiject(F,[a,b],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebijectiv(F,[a,b],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebij(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebijectiva(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebiject(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebijectiv(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), inj(F,[1,2,3])), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), injectiv(F)), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), injectiva(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), inject(F,[1,2,3])), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), surj(F,[a,b,c],[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), surjectiva(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), surjectiv(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c],[1,2,3]), surject(F,[a,b,c],[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), inj(F,[1,2,3])), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), injectiv(F)), L), afislista(L), length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), injectiva(F,[1,2,3])), L), afislista(L),
length(L,N).
```

```
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), inject(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), surj(F,[a,b,c,d],[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), surjectiva(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), surjectiv(F,[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, (functie(F,[a,b,c,d],[1,2,3]), surject(F,[a,b,c,d],[1,2,3])), L), afislista(L),
length(L,N).
?- setof(F, functiebij(F,[a,b,c,d],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebijectiva(F,[a,b,c,d],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebiject(F,[a,b,c,d],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebijectiv(F,[a,b,c,d],[1,2,3]), L), afislista(L), length(L,N).
Putem interoga astfel, pentru a obtine lista vida in loc de false:
?- (setof(F, functiebijectiva(F,[a,b,c,d],[1,2,3]), L), !; L=[]), afislista(L), length(L,N).
*/

/* Putem determina functiile de la A la B prin listele valorilor acestora, ca in urmatorul
predicat, in definitia caruia nu conteaza elementele lui A, ci doar numarul acestora, iar
ordinea elementelor lui A conteaza pentru a recupera definitia functiei: */

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

injFunctieListaVal(F) :- elimdupl(F,F).
```

```
injectFunctieListaVal(F) :- faradupl(F).

% Predicat care testeaza daca o lista nu are duplicate:

faradupl([]).
faradupl([H|T]) :- not(member(H,T)), faradupl(T).

surjFunctieListaVal(F,B) :- elimdupl(F,G), elimdupl(B,C),
                            egalmult(G,C).

functieBijListaVal(F,A,B) :- functieListaVal(F,A,B),
               injFunctieListaVal(F), surjFunctieListaVal(F,B).

functieBijectListaVal(F,A,B) :- functieListaVal(F,A,B),
                 injectFunctieListaVal(F), surjFunctieListaVal(F,B).

/* Interogati:
?- setof(F, functiebij(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functiebij(F,[1,2,3],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functieBijListaVal(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functieBijListaVal(F,[1,2,3],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functieBijectListaVal(F,[a,b,c],[1,2,3]), L), afislista(L), length(L,N).
?- setof(F, functieBijectListaVal(F,[1,2,3],[1,2,3]), L), afislista(L), length(L,N).
?- permutarile([1,2,3],L), afislista(L), length(L,N).
Desigur, ultimele cinci interogari au acelasi rezultat: lista permutarilor listei [1,2,3].
*/
```