```
/* Directiva pentru includerea uneia sau a mai multor
baze de cunostinte in cea curenta: */

:- [lab3lmc4].

xor(P,Q) :- P, not(Q) ; Q, not(P).

/* Reuniunea, intersectia, diferenta si diferenta simetrica intre multimi, calculate cu
setof: */

reuniunea(A,B,R) :- setof(X, member(X,A);member(X,B), R), !.
reuniunea(_,_,[]).

intersectia(A,B,I) :- setof(X, (member(X,A),member(X,B)), I), !.
intersectia(_,_,[]).

diferentamult(A,B,D) :- setof(X, (member(X,A),not(member(X,B))), D), !.
diferentamult(_,_,[]).

diferentasimetrica(A,B,D) :- setof(X, xor(member(X,A),member(X,B)), D), !.
diferentasimetrica(_,_,[]).

/* Intrucat am folosit setof, rezultatul va fi intors fara duplicate chiar daca dam liste cu
duplicate in interogari:
?- reuniunea([0,1,0,2,2,3,4,5,0,5],[1,2,2,1,5,1,5,6,7,7],M).
?- intersectia([0,1,0,2,2,3,4,5,0,5],[1,2,2,1,5,1,5,6,7,7],M).
?- diferentamult([0,1,0,2,2,3,4,5,0,5],[1,2,2,1,5,1,5,6,7,7],M).
```

```
?- diferentasimetrica([0,1,0,2,2,3,4,5,0,5],[1,2,2,1,5,1,5,6,7,7],M).
*/

% Testarea incluziunii intre multimi:

inclusa([],_).
inclusa([H|T],L) :- member(H,L), inclusa(T,L).

% Egalitatea de multimi, definita prin dubla incluziune:

egaldemult(L,M) :- inclusa(L,M), inclusa(M,L).

/* Urmatoarele liste sunt egale ca multimi:
?- egaldemult([4,3,0,1,4,2,0,1],[1,3,2,2,4,0]).
Iar urmatoarele difera; niciuna dintre ele nu e inclusa in cealalta:
?- egaldemult([1,2,2,3,4],[1,2,4,5]).
?- inclusa([1,2,2,3,4],[1,2,4,5]).
?- inclusa([1,2,4,5],[1,2,2,3,4]).
*/

% Desigur, putem defini incluziunea intre multimi si astfel:

incluziune(A,B) :- not((member(X,A), not(member(X,B)))).

% Si egalitatea de multimi folosind aceasta incluziune:

egalecamult(A,B) :- incluziune(A,B), incluziune(B,A).
```

```
/* Interogati:
?- egalecamult([4,3,0,1,4,2,0,1],[1,3,2,2,4,0]).
?- egalecamult([1,2,2,3,4],[1,2,4,5]).
?- incluziune([1,2,2,3,4],[1,2,4,5]).
?- incluziune([1,2,4,5],[1,2,2,3,4]).
*/

/* Toate predicatele pe care le-am scris sub forma: nu exista astfel de termeni care sa nu
satisfaca aceasta conditie, ca predicatul binar incluziune de mai sus, pot fi scrise
printr-o recursie care defineste proprietatea: fiecare astfel de termen satisface aceasta
conditie, ca predicatul binar inclusa de mai sus. Uneori aceasta recursie e mai dificil de
scris. De exemplu, pentru demonstrarea primei legi a lui De Morgan pentru disjunctie si
conjunctie:
   non(p sau q) <=> [non p si non q]
sau a distributivitatii la stanga a implicatiei fata de conjunctie:
   [p => (q si r)] <=> [(p => q) si (p => r)],
cel mai simplu este sa procedam astfel: */

deMorgan :- not((listaValBool([P,Q]), not(echiv(not(P;Q), (not(P),not(Q)))))).

distribimplicconj :- not((listaValBool([P,Q,R]),
        not(echiv(implica(P,(Q,R)), (implica(P,Q), implica(P,R)))))).

/* Implicatia definita sub forma urmatoare are o singura satisfacere pentru    P=false si
Q=true: */
```

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

/* Predicatele implica si echiv vor fi suprascrise. Prologul foloseste aceste noi versiuni
ale lor si nu versiunile din lab3lmc4.pl. */

% Varianta de recursie pentru fiecare dintre demonstratiile de mai sus:

v1deMorgan :- setof([P,Q], listaBool([P,Q]), L), auxv1deMorgan(L).

auxv1deMorgan([]).
auxv1deMorgan([[P,Q]|T]) :- echiv(not(P;Q), (not(P),not(Q))),
                            write((P,Q)), nl, auxv1deMorgan(T).

v1distribimplicconj :- setof([P,Q,R], listaBool([P,Q,R]), L),
                       auxv1distribimplicconj(L).

auxv1distribimplicconj([]).
auxv1distribimplicconj([[P,Q,R]|T]) :-
                echiv(implica(P,(Q,R)), (implica(P,Q), implica(P,R))),
                write((P,Q,R)), nl, auxv1distribimplicconj(T).

% Exemplu de proprietate care nu e satisfacuta de toate valorile booleene:

implicatia :- setof([P,Q], listaBool([P,Q]), L), auximplicatia(L).

auximplicatia([]).
```

```
auximplicatia([[P,Q]|T]) :- implica(P,Q),
                        write((P,Q)), nl, auximplicatia(T).

/* Prologul vede produsul cartezian asociativ la stanga, dar nu la dreapta:
?- (a,(b,c)) = (a,b,c).
?- ((a,b),c) = (a,b,c).
?- (a,(b,(c,(d,e)))) = (a,b,c,d,e).
?- (a,((b,(c,d)),e)) = (a,b,c,d,e).
*/

% Produsul cartezian a N multimi:

prodNmult([L],L) :- !.
prodNmult([Lista|ListaListe],Prod) :- prodNmult(ListaListe,P),
                                      prodcartmult(Lista,P,Prod).

/* Interogati:
?- prodNmult([],P).
?- prodNmult([[a,b,c]],P).
?- prodNmult([[a,b,c],[1,2]],P).
?- prodNmult([[a,b,c],[1,2],[x,y]],P), write(P).
?- prodNmult([[a,b,c],[1,2],[x,y],[0,10]],P), write(P).
?- prodNmult([[a,b,c],[1,2],[x,y],[0,10]],P), afislista(P).
*/

% Puterile naturale nenule ale unei multimi:
```

```
puteremult(A,1,A).
puteremult(A,N,AlaN) :- N>1, PN is N-1, puteremult(A,PN,AlaPN),
                        prodcartmult(A,AlaPN,AlaN).

% Varianta pentru puterile naturale nenule ale unei multimi:

listaNcopiiMult(_,0,[]).
listaNcopiiMult(A,N,[A|L]) :- N>0, PN is N-1, listaNcopiiMult(A,PN,L).

putereaNmult(A,N,P) :- listaNcopiiMult(A,N,L), prodNmult(L,P).

/* Interogati:
?- puteremult([a,b,c],1,P).
?- puteremult([a,b,c],2,P), write(P).
?- puteremult([a,b,c],3,P), write(P).
?- puteremult([a,b,c],4,P), write(P).
?- puteremult([a,b,c],4,P), afislista(P), length(P,Card).
?- putereaNmult([a,b,c],1,P).
?- putereaNmult([a,b,c],2,P), write(P).
?- putereaNmult([a,b,c],3,P), write(P).
?- putereaNmult([a,b,c],4,P), write(P).
?- putereaNmult([a,b,c],4,P), afislista(P), length(P,Card).
*/

% Alta varianta de recursie pentru fiecare dintre demonstratiile de mai sus:

v2deMorgan :- puteremult([false,true],2,PerCtBool), auxv2deMorgan(PerCtBool).
```

```
auxv2deMorgan([]).
auxv2deMorgan([(P,Q)|T]) :- echiv(not(P;Q), (not(P),not(Q))),
                        write((P,Q)), nl, auxv2deMorgan(T).

v2distribimplicconj :- puteremult([false,true],3,TripleteCtBool),
                        auxv2distribimplicconj(TripleteCtBool).

auxv2distribimplicconj([]).
auxv2distribimplicconj([(P,Q,R)|T]) :-
                echiv(implica(P,(Q,R)), (implica(P,Q), implica(P,R))),
                write((P,Q,R)), nl, auxv2distribimplicconj(T).

% Sau, cu al doilea predicat pentru calculul puterilor unei multimi:

v3deMorgan :- putereaNmult([false,true],2,PerCtBool), auxv2deMorgan(PerCtBool).

v3distribimplicconj :- putereaNmult([false,true],3,TripleteCtBool),
                         auxv2distribimplicconj(TripleteCtBool).

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

/* Fie M si N multimi, R<=MxN o relatie binara de la M la N, A<=M si B<=N.
Generalizand definitia imaginii si a preimaginii printr-o functie la relatii binare
arbitrare obtinem definitiile din sistemul axiomatic din primul set de cursuri (amintesc ca
aceasta parte din curs este facultativa), care, aplicate in cazul particular al relatiilor
functionale totale, coincid cu definitiile imaginii, respectiv preimaginii printr-o functie:
```

```
imaginea lui A prin R este {y in N | x in A, (x,y) in R}, = R(A) daca R este functie;
preimaginea lui B prin R este {x in M | y in B, (x,y) in R}, = R^-1(B) daca R este functie.
Prima varianta, folosind setof: */

im(R,A,Im) :- setof(Y, X^(member(X,A), member((X,Y),R)), Im), !.
im(_,_,[]).

preim(R,B,PreIm) :- setof(X, Y^(member(Y,B), member((X,Y),R)), PreIm), !.
preim(_,_,[]).

% A doua varianta, recursiv:

imag([],_,[]).
imag([(X,Y)|T],A,[Y|L]) :- member(X,A), !, imag(T,A,L).
imag([_|T],A,L) :- imag(T,A,L).

imaginea(R,A,I) :- imag(R,A,J), elimdupl(J,I).

preimag([],_,[]).
preimag([(X,Y)|T],B,[X|L]) :- member(Y,B), !, preimag(T,B,L).
preimag([_|T],B,L) :- preimag(T,B,L).

preimaginea(R,B,P) :- preimag(R,B,Q), elimdupl(Q,P).

/* Interogati:
?- im([(a,1),(b,1),(c,2),(c,3),(d,4)],[a,b,d],Im).
?- imaginea([(a,1),(b,1),(c,2),(c,3),(d,4)],[a,b,d],Im).
```

```
?- preim([(a,1),(b,1),(c,2),(c,3),(d,4)],[1,2,3],Im).
?- preimaginea([(a,1),(b,1),(c,2),(c,3),(d,4)],[1,2,3],Im).
*/

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

/* Fie T o multime, A<=T, B<=T si f:P(T)->P(A)xP(B), definita prin:
pentru orice X<=T, f(X)=(X^A,X^B). Atunci:
   f este injectiva <=> AUB=T;
   f este surjectiva <=> A^B=0.
Sa verificam proprietatile de mai sus pentru multimi finite date T: */

egalperechimult((A,B),(C,D)) :- egaldemult(A,C), egaldemult(B,D).

/* Amintesc ca predicatele pentru testarea proprietatilor relatiilor binare pe care le-am
scris pana acum functioneaza doar pentru relatii binare intre multimi date ca liste de
constante.
Pentru f, al carei domeniu este o lista de liste si al carei codomeniu este o lista de
perechi de liste, vom folosi urmatoarele predicate care testeaza injectivitatea, respectiv
surjectivitatea: */

finjectiva(F) :- not((member((X,(Y,Z)),F), member((U,(V,W)),F),
                egalperechimult((Y,Z),(V,W)), not(egaldemult(X,U)))).

fsurjectiva(F,Codom) :- not((member(Per,Codom),
         not((member((_,AltaPer),F), egalperechimult(Per,AltaPer))))).
```

```
/* Sa construim functia f, ca si pana acum, sub forma relatiei binare date de graficul sau:
f = {(X,f(X)) | X in P(T)} = {(X,(X^A,X^B)) | X in P(T)}: relatie binara functionala totala
de la P(T) la P(A)xP(B): */

functiaF(F,T,A,B) :- setof((X,(XintersA,XintersB)), (sublista(X,T),
        intersectia(X,A,XintersA), intersectia(X,B,XintersB)), F).

/* Urmatorul predicat intoarce true pentru o lista T ddaca, pentru orice submultimi A si B
ale multimii date de lista T, functia f definita ca mai sus satisface proprietatile de mai
sus, afisand fiecare pereche (A,B) de submultimi ale lui T in timp testeaza functia f pentru
perechea (A,B): */

verificare(T) :- not((sublista(A,T), sublista(B,T), functiaF(F,T,A,B),
        write(A), tab(1), write(B), nl,
        sublistele(A,PA), sublistele(B,PB), prodcart(PA,PB,P),
 not((echiv(finjectiva(F), (reuniunea(A,B,R), egaldemult(R,T))),
        echiv(fsurjectiva(F,P), (intersectia(A,B,I), I=[]))))).

/* Daca scriam direct intersectia(A,B,[]) intorcea true intotdeauna, pentru ca unifica acest
scop cu faptul din definitia predicatului intersectia. Trebuie sa calculam intersectia
folosind variabila I, apoi sa verificam ca aceasta este vida, ceea ce putem testa direct
prin unificare cu constanta [].
Poate ca e mai elegant sa calculam reuniunea si intersectia inainte de testarea acelei
echivalente, chiar daca nu se face decat o singura verificare: */

verific(T) :- not((sublista(A,T), sublista(B,T), functiaF(F,T,A,B),
        write(A), tab(1), write(B), nl, reuniunea(A,B,R), intersectia(A,B,I),
```

```
        sublistele(A,PA), sublistele(B,PB), prodcart(PA,PB,P),
 not((echiv(finjectiva(F), egaldemult(R,T)), echiv(fsurjectiva(F,P), I=[])))).

% Putem afisa si functia f pe parcursul acestei verificari:

afisfctf([]).
afisfctf([(X,(XintersA,XintersB))|T]) :- write('f('), write(X), write(') = '),
        write('('), write((XintersA,XintersB)), write(')'), nl, afisfctf(T).

verificarefct(T) :- not((sublista(A,T), sublista(B,T), functiaF(F,T,A,B),
        write(A), tab(1), write(B), nl, afisfctf(F), nl,
        reuniunea(A,B,R), intersectia(A,B,I),
        sublistele(A,PA), sublistele(B,PB), prodcart(PA,PB,P),
 not((echiv(finjectiva(F), egaldemult(R,T)),
        echiv(fsurjectiva(F,P), I=])))).

% Si cu a doua varianta de verificare de mai sus:

verificfct(T) :- not((sublista(A,T), sublista(B,T), functiaF(F,T,A,B),
        write(A), tab(1), write(B), nl, afisfctf(F), nl,
        reuniunea(A,B,R), intersectia(A,B,I),
        sublistele(A,PA), sublistele(B,PB), prodcart(PA,PB,P),
 not((echiv(finjectiva(F), egaldemult(R,T)), echiv(fsurjectiva(F,P), I=])))).

/* Interogati:
?- verificare([a,b,c]).
?- verific([a,b,c]).
```

```
?- verificarefct([a,b,c]).
?- verificfct([a,b,c]).
*/

/* Putem testa si astfel surjectivitatea lui f, pentru a nu mai calcula produsul cartezian:
*/

fsurj(F,A,B) :- not((sublista(SA,A), sublista(SB,B),
                not((member((_,Per),F), egalperechimult(Per,(SA,SB)))))).

verif(T) :- not((sublista(A,T), sublista(B,T), functiaF(F,T,A,B),
        write(A), tab(1), write(B), nl, reuniunea(A,B,R), intersectia(A,B,I),
 not((echiv(finjectiva(F), egaldemult(R,T)), echiv(fsurj(F,A,B), I=[])))).

veriffct(T) :- not((sublista(A,T), sublista(B,T), functiaF(F,T,A,B),
        write(A), tab(1), write(B), nl, afisfctf(F), nl,
        reuniunea(A,B,R), intersectia(A,B,I),
 not((echiv(finjectiva(F), egaldemult(R,T)), echiv(fsurj(F,A,B), I=[])))).

/* Interogati:
?- verif([a,b,c]).
?- veriffct([a,b,c]).
*/

/* Putem reprezenta o multime T={x1,x2,...,xn}, de cardinal n, prin multimea indicilor
{1,2,...,n}, aflata in bijectie cu T, si astfel putem efectua verificarea de mai sus pentru
orice multime T de cardinal pana la un N natural dat: */
```

```
lista(0,[]).
lista(N,[N|L]) :- N>0, PN is N-1, lista(PN,L).

% Variante de recursie:

verifPanaLaCard(N) :- (N>0, !, PN is N-1, verifPanaLaCard(PN) ; N>=0),
        lista(N,T), write('T='), write(T), nl, verif(T),
                        write('----------------------------------------'), nl.

verifpanalaCard(-1).
verifpanalaCard(N) :- N>=0, !, PN is N-1, verifpanalaCard(PN),
        lista(N,T), write('T='), write(T), nl, verif(T),
                        write('----------------------------------------'), nl.

/* Interogati:
?- verifPanaLaCard(5).
?- verifpanalaCard(5).
Interogati cu urmatorul predicat predefinit:
?- atom_concat(exemplu,deConcatenareDeAtomi,A).
?- atom_concat(exemplu,' de concatenare de atomi',A).
?- atom_concat(exemplul,3,A).
E dificil de urmarit in fereastra interpretorului Prolog-ului tot acest output, asadar sa
efectuam afisarea intr-un fisier, folosind predicatele predefinite: tell, care creeaza un
fisier si il deschide pentru scriere, si told, care inchide fisierul astfel creat: */

cale('d:/tempwork/').
```

```
numeFis('verificare.txt').

verifpanalacard(N) :- cale(C), numeFis(Fis), atom_concat(C,Fis,Fisierul),
                tell(Fisierul), verifpanalaCard(N), told.

% Sau, mai bine, in cate un fisier diferit pentru fiecare cardinal:

verifpanalacardinalul(-1).
verifpanalacardinalul(N) :- N>=0, !, PN is N-1, verifpanalacardinalul(PN),
lista(N,T), cale(C), atom_concat('verifptcardinalul',N,Fis),
        atom_concat(Fis,'.txt',Fisier), atom_concat(C,Fisier,Fisierul),
        tell(Fisierul), write('T='), write(T), nl, verif(T), told.

/* Interogati:
?- verifpanalacard(5).
?- verifpanalacardinalul(5).
Desigur, putem construi multimea T chiar sub forma {x1,x2,...,xn}, folosind predicatul
predefinit pentru concatenare de atomi pe care l-am folosit mai sus pentru construirea
numelor fisierelor: */

listax(0,[]).
listax(N,[Atom|L]) :- N>0, atom_concat(x,N,Atom), PN is N-1, listax(PN,L).

verifpanalacardinalulx(-1).
verifpanalacardinalulx(N) :- N>=0, !, PN is N-1, verifpanalacardinalulx(PN),
listax(N,T), cale(C), atom_concat('verifptcardinalcux',N,Fis),
        atom_concat(Fis,'.txt',Fisier), atom_concat(C,Fisier,Fisierul),
```

```
        tell(Fisierul), write('T='), write(T), nl, verif(T), told.

/* Putem scrie recursia si astfel, pentru cazul in care dam de la tastatura un numar care nu
e neaparat natural: */

verifpanalacardcux(N) :- N<0, !.
verifpanalacardcux(N) :- PN is N-1, verifpanalacardcux(PN), listax(N,T),
        cale(C), atom_concat('verifptcardcux',N,Fis),
        atom_concat(Fis,'.txt',Fisier), atom_concat(C,Fisier,Fisierul),
        tell(Fisierul), write('T='), write(T), nl, verif(T), told.

/* Interogati:
?- verifpanalacardinalulx(5).
?- verifpanalacardcux(5).
Pentru un numar mare in locul cardinalului (maxim) 5, este posibil ca output-ul sa nu incapa
in fisiere .txt. Putem furniza extensia fisierelor de output: */

numeFisier('ptcardcux').

verifpanalacardinal(-1,_).
verifpanalacardinal(N,Extensie) :- N>=0, !, PN is N-1,
        verifpanalacardinal(PN,Extensie), listax(N,T),
        cale(Cale), numeFisier(Fi), atom_concat(Fi,N,Fis),
        atom_concat(Fis,Extensie,Fisier), atom_concat(Cale,Fisier,Fisierul),
        tell(Fisierul), write('T='), write(T), nl, verif(T), told.

/* Interogati:
```

```
?- verifpanalacardinal(5,'.txt').
?- verifpanalacardinal(10,'.docx').
*/

/* Sa vedem si dimensiunea output-ului: numarul de perechi de multimi (A,B) pentru fiecare
cardinal al multimii T: */

verifmult(T) :- sublistele(T,PT), prodcart(PT,PT,Prod), auxverifmult(T,Prod,0).

auxverifmult(_,[],_).
auxverifmult(T,[(A,B)|Coada],N) :- functiaF(F,T,A,B),
        write(A), tab(1), write(B), tab(1), SN is N+1, write(SN), nl,
        sublistele(A,PA), sublistele(B,PB), prodcart(PA,PB,P),
        echiv(finjectiva(F), (reuniune(A,B,R), egalmult(R,T))),
        echiv(fsurjectiva(F,P), (intersectie(A,B,I), I=[])),
        auxverifmult(T,Coada,SN).

verifpanalacardcunr(-1).
verifpanalacardcunr(N) :- N>=0, !, PN is N-1, verifpanalacardcunr(PN),
listax(N,T), cale(C), atom_concat('verifptcardcux',N,Fis),
        atom_concat(Fis,'.txt',Fisier), atom_concat(C,Fisier,Fisierul),
        tell(Fisierul), write('T='), write(T), nl, verifmult(T), told.

/* Interogati:
?- verifpanalacardcunr(5).
*/
```

```
/* Sau, cu o doua varianta de testare a surjectivitatii, reuniunea si intersectia calculate
inainte de testarea echivalentelor, iar recursia tinand seama de posibilitatea unui input
eronat: */

verifm(T) :- sublistele(T,PT), prodcart(PT,PT,Prod), auxverifm(T,Prod,0).

auxverifm(_,[],_).
auxverifm(T,[(A,B)|Coada],N) :- functiaF(F,T,A,B),
        write(A), tab(1), write(B), tab(1), SN is N+1, write(SN), nl,
        reuniune(A,B,R), intersectie(A,B,I),
        echiv(finjectiva(F), egalmult(R,T)),
        echiv(fsurj(F,A,B), I=[]),
        auxverifm(T,Coada,SN).

verifcardcunr(N) :- N<0, !.
verifcardcunr(N) :- PN is N-1, verifcardcunr(PN),
        listax(N,T), cale(C), atom_concat('verifcardcux',N,Fis),
        atom_concat(Fis,'.txt',Fisier), atom_concat(C,Fisier,Fisierul),
        tell(Fisierul), write('T='), write(T), nl, verifm(T), told.

/* Interogati:
?- verifcardcunr(5).
*/
```