**Laborator 1**

**ex2:**

father\_of(F,C):- male(F), parent(F,C).

mother\_of(M,C):- female(M),parent(M,C).

grandmother\_of(G,C):- mother\_of(G,P), parent(P,C).

grandfather\_of(G,C):- father\_of(G,P), parent(P,C).

sister\_of(S,P) :- parent(Q,S), parent(Q,P), female(S), S \= P.

brother\_of(B,P):- parent(Q,B), parent(Q,P), male(B), B \= P.

aunt\_of(A,P) :- sister\_of(A,B), parent(B,P).

uncle\_of(A,B) :- brother\_of(A,C), parent(C,B).

**ex3:**

% not\_parent(X,Y) :- not(parent(X,Y)).

not\_parent(X,Y) :- (male(X); female(X)), (male(Y); female(Y)), X\=Y, not(parent(X,Y)).

female(mary).

female(sandra).

female(juliet).

female(lisa).

male(peter).

male(paul).

male(dony).

male(bob).

male(harry).

parent(bob, lisa).

parent(bob, paul).

parent(bob, mary).

parent(juliet, lisa).

parent(juliet, paul).

parent(juliet, mary).

parent(peter, harry).

parent(lisa, harry).

parent(mary, dony).

parent(mary, sandra).

**Laborator 2**

**ex1:**

Definit¸i un predicat distance/3 pentru a calcula distant¸a dintre dou˘a

puncte din plan. Punctele sunt date ca perechi de coordonate.

distance((A,B),(C,D),X) :- X is sqrt((C-A)\*\*2 + (D-B)\*\*2).

**ex2:**

Scriet¸i un predicat fib/2 pentru a calcula, pentru orice n, num˘arul de pe

pozit¸ia n din ¸sirul Fibonacci. Secvent¸a de numere Fibonacci este definit˘a

prin: F0 := 1, F1 := 1, iar, pentru orice n ≥ 2, Fn := Fn−1 + Fn−2.

fib(0,1).

fib(1,1).

fib(N,X) :- 2 =< N, M is N - 1, fib(M, Y), P is N - 2, fib(P, Z), X is Y + Z.

fibo(0,0,1).

fibo(1,1,1).

fibo(N,Z,X) :- 2 =< N, M is N-1, fibo(M,Y,Z), X is Y + Z.

fibg(N,X) :- fibo(N,\_,X).

**ex3:**

Scriet¸i un program ˆın Prolog pentru a afi¸sa un p˘atrat de n × n caractere

pe ecran.

Denumit¸i predicatul square/2. Primul argument este un num˘ar natural

diferit de 0, iar al doilea un caracter care trebuie afi¸sat

line(0,\_).

line(X,C):- X>0, Y is X-1, write(C), line(Y,C).

rectangle(0,\_,\_):-nl.

rectangle(X,Z,C):- X>0, Y is X-1, line(Z,C), nl, rectangle(Y,Z,C).

square(X,C) :- rectangle(X,X,C).

**ex4:**

Definit¸i un predicat all a/1 care prime¸ste ca argument o list˘a ¸si care

verific˘a dac˘a argumentul s˘au este format doar din a-uri.

all\_a([]).

all\_a([a|X]):-all\_a(X).

Scrieti un predicat trans a b/2 care traduce o list˘a de a-uri ˆıntr-o

list˘a de b-uri. trans a b(X,Y) trebuie s˘a fie adev˘arat dac˘a X este o list˘a

de a-uri ¸si Y este o list˘a de b-uri, iar cele dou˘a liste au lungimi egale.

trans\_a\_b([],[]).

trans\_a\_b([a|X],[b|Y]):-trans\_a\_b(X,Y).

**ex5:**

Scriet¸i un predicat scalarMult/3, al c˘arui prim argument este un

ˆıntreg, al doilea argument este o list˘a de ˆıntregi, iar al treilea argument

este rezultatul ˆınmult¸irii cu scalari al celui de-al doilea argument cu

primul.

scalarMult(\_,[],[]).

scalarMult(N,[H|T],[X|Y]) :- X is N \* H, scalarMult(N,T,Y).

dot([],[],0).

dot([H|T],[X|Y],M) :- dot(T,Y,N), M is N + H \* X.

max([],0).

max([H|T],X) :- max(T,Y), maxim(H,Y,X).

maxim(A,B,B) :- B>A.

maxim(A,B,A) :- A>=B.

**Laborator 3:**

**ex1:**

Definit¸i un predicat palindrome/1 care este adev˘arat dac˘a lista primit˘a

ca argument este palindrom (lista citit˘a de la stˆanga la dreapta este

identic˘a cu lista citit˘a de la dreapta la stˆanga).

rev([],[]).

rev([H|T], L) :- rev(T,N), append(N,[H],L).

palindrome(L) :- rev(L,L).

**ex2:**

Definit¸i un predicat remove duplicates/2 care ¸sterge toate duplicatele

din lista dat˘a ca prim argument ¸si ˆıntoarce rezultatul ˆın al doilea

argument.

remove\_duplicates([],[]).

remove\_duplicates([H|L],M) :- remove\_duplicates(L,M), member(H,M).

remove\_duplicates([H|L],[H|M]) :- remove\_duplicates(L,M), not(member(H,M)).

**ex3:**

Definit¸i un predicat atimes/3 care s˘a fie adev˘arat exact atunci cˆand

elementul din primul argument apare ˆın lista din al doilea argument de

num˘arul de ori precizat ˆın al treilea argument.

atimes(\_,[],0).

atimes(N,[N|T],X) :- atimes(N,T,Y), X is Y + 1.

atimes(N,[H|T],X) :- atimes(N,T,X), H \= N.

**ex4:**

Scriet¸i regulile care definesc comportamentul predicatului

ajut˘ator insert/3.

insertsort([],[]).

insertsort([H|T],L) :- insertsort(T,L1), insert(H,L1,L).

insert(X,[],[X]).

insert(X,[H|T],[X|[H|T]]) :- X < H.

insert(X,[H|T],[H|L]) :- X >= H, insert(X,T,L).

**ex5:**

Scriet¸i regulile care definesc comportamentul predicatului

ajut˘ator split/4.

quicksort([],[]).

quicksort([H|T],L) :- split(H,T,A,B), quicksort(A,M), quicksort(B,N),

append(M,[H|N],L).

split(\_,[],[],[]).

split(X,[H|T],[H|A],B) :- H < X, split(X,T,A,B).

split(X,[H|T],A,[H|B]) :- H >= X, split(X,T,A,B).

**Laborator 4**

**ex1:**

Definit¸i un predicat listaNelem/3 astfel ˆıncˆat, pentru orice L, N, M,

listaNelem(L,N,M) este adev˘arat exact atunci cˆand M este o list˘a cu N

elemente care sunt toate elemente ale lui L (cu eventuale repetit¸ii).

listaNelem(\_,0,[]).

listaNelem(L,N,[H|T]) :- N > 0, P is N - 1, member(H,L), listaNelem(L,P,T).

**ex2:**

Definit¸i un predicat crosswd/6 care calculeaz˘a toate variantele ˆın care

putet¸i completa grila

word(abalone,a,b,a,l,o,n,e).

word(abandon,a,b,a,n,d,o,n).

word(enhance,e,n,h,a,n,c,e).

word(anagram,a,n,a,g,r,a,m).

word(connect,c,o,n,n,e,c,t).

word(elegant,e,l,e,g,a,n,t).

crosswd(V1,V2,V3,H1,H2,H3) :-

word(V1,\_,A,\_,B,\_,C,\_),

word(V2,\_,D,\_,E,\_,F,\_),

word(V3,\_,G,\_,H,\_,I,\_),

word(H1,\_,A,\_,D,\_,G,\_),

word(H2,\_,B,\_,E,\_,H,\_),

word(H3,\_,C,\_,F,\_,I,\_).

**ex3:**

Ad˘augat¸i un predicat path/3 care indic˘a dac˘a dintr-un punct se poate

ajunge ˆıntr-un alt punct, ˆın mai mult¸i pa¸si, cel de-al treilea argument

reprezentˆand lista pa¸silor. Pe baza lui, construit¸i un predicat pathc/2

care spune doar dac˘a dintr-un punct se poate ajunge ˆıntr-un alt punct

path(X,X,[X]).

path(X,Y,[X|L]) :- connected(X,Z), path(Z,Y,L).

pathc(X,Y) :- path(X,Y,\_).

**ex4:**

word\_letters(X,Y) :- atom\_chars(X,Y).

liminus([C|L],C,L).

liminus([D|L],C,[D|M]) :- D\==C, liminus(L,C,M).

cover([],\_).

cover([H|T],L) :- liminus(L,H,M), cover(T,M).

solution(Letters, Word, Len) :- word(Word), word\_letters(Word,WordLetters), length(WordLetters,Len), cover(WordLetters, Letters).

search\_solution(\_,'no solution',0).

search\_solution(ListLetters,Word,X) :- X > 0, solution(ListLetters,Word,X).

search\_solution(ListLetters,Word,X) :- X > 0, not(solution(ListLetters,Word,X)), Y is X-1, search\_solution(ListLetters,Word,Y).

topsolution(ListLetters,Word) :- length(ListLetters, MaxScore), search\_solution(ListLetters,Word,MaxScore).

**Laborator 5**

**ex1:**

Definit¸i un predicat vars/2 care este adev˘arat exact atunci cˆand primul

argument este o formul˘a, iar al doilea argument este lista care reprezint˘a

mult¸imea variabilelor care apar ˆın ea.

vars(V,[V]) :- atom(V).

vars(non(X),S) :- vars(X,S).

vars(si(X,Y),S) :- vars(X,T), vars(Y,U), union(T,U,S).

vars(sau(X,Y),S) :- vars(X,T), vars(Y,U), union(T,U,S).

vars(imp(X,Y),S) :- vars(X,T), vars(Y,U), union(T,U,S).

**ex2:**

Definit¸i un predicat val/3, astfel ˆıncˆat, pentru orice variabil˘a V ¸si orice

evaluare E, avem c˘a, pentru orice A, val(V,E,A) este adev˘arat exact

atunci cˆand A este ”E(V)”.

val(V,[(V,A)|\_],A).

val(V,[\_|T],A) :- val(V,T,A).

%Solutie alternativa:

val(V,E,A) :- member((V,A),E).

**ex3:**

Definit¸i predicate bnon/2, bsi/3, bsau/3, bimp/3 care implementeaz˘a

operat¸iile ¬, ∧, ∨, → pe mult¸imea {0, 1}.

bnon(0,1). bnon(1,0).

bsi(0,0,0). bsi(0,1,0). bsi(1,0,0). bsi(1,1,1).

bsau(0,0,0). bsau(0,1,1). bsau(1,0,1). bsau(1,1,1).

% X -> Y = (non X) sau Y

bimp(X,Y,Z) :- bnon(X,NX), bsau(NX,Y,Z).

**ex4:**

Definit¸i un predicat eval/3, astfel ˆıncˆat, pentru orice formul˘a X ¸si orice

evaluare E, avem c˘a, pentru orice A, eval(X,E,A) este adev˘arat exact

atunci cˆand A este ”

E

+(X)”.

eval(V,E,A) :- atom(V), val(V,E,A).

eval(non(X),E,A) :- eval(X,E,B), bnon(B,A).

eval(si(X,Y),E,A) :- eval(X,E,B), eval(Y,E,C), bsi(B,C,A).

eval(sau(X,Y),E,A) :- eval(X,E,B), eval(Y,E,C), bsau(B,C,A).

eval(imp(X,Y),E,A) :- eval(X,E,B), eval(Y,E,C), bimp(B,C,A).

**ex5:**

Definit¸i un predicat evals/3, astfel ˆıncˆat, pentru orice formul˘a X ¸si orice

list˘a de evalu˘ari Es, avem c˘a, pentru orice As, evals(X,Es,As) este

adev˘arat exact atunci cˆand As este lista rezultatelor evalu˘arii lui X ˆın

fiecare dintre elementele lui Es.

evals(\_,[],[]).

evals(X,[E|Es],[A|As]) :- eval(X,E,A), evals(X,Es,As).

**ex6:**

Definit¸i un predicat evs/2, astfel ˆıncˆat, pentru orice list˘a de variabile S,

avem c˘a, pentru orice Es, evs(S,Es) este adev˘arat exact atunci cˆand Es

este lista evalu˘arilor definite pe S.

evs([],[[]]).

evs([V|T],Es) :- evs(T,Esp), adauga(V,Esp,Es).

adauga(\_,[],[]).

adauga(V,[E|T], [[(V,0)|E],[(V,1)|E]|Es]) :- adauga(V,T,Es).

**ex7:**

Definit¸i un predicat all evals/2, astfel ˆıncˆat, pentru orice formul˘a X,

avem c˘a, pentru orice As, all evals(X,As) este adev˘arat exact atunci

cˆand As este lista rezultatelor evalu˘arii lui X ˆın fiecare dintre elementele

listei evalu˘arilor definite pe Var(X).

all\_evals(X,As) :- vars(X,S), evs(S,Es), evals(X,Es,As).

**ex8:**

Definit¸i un predicat taut/1, astfel ˆıncˆat, pentru orice formul˘a X, avem c˘a

taut(X) este adev˘arat exact atunci cˆand X este tautologie.

all\_ones([]).

all\_ones([1|T]) :- all\_ones(T).

taut(X) :- all\_evals(X,As), all\_ones(As).

**Model Test Laborator**

**ex1:**

Definit¸i un predicat lista\_puncte/3 care, pentru o list˘a L format˘a din puncte reprezentate sub forma punct(coordX, coordY) ¸si un num˘ar Val, calculeaz˘a lista acelor puncte din L care au a doua coordonat˘a mai mare decˆat Val.

lista\_puncte([],\_,[]).

lista\_puncte([punct(X,Y)|T1], Val, [punct(X,Y)|Lrez]):- Y>Val, lista\_puncte(T1, Val, Lrez).

lista\_puncte([punct(\_,Y)|T1], Val, Lrez):- Y=<Val, lista\_puncte(T1, Val, Lrez).

**ex2:**

Definit¸i un predicat dropN/3, astfel ˆıncˆat, pentru orice liste L, R ¸si num˘ar natural N, dropN(L, R, N) este adev˘arat dac˘a ¸si numai dac˘a R este lista care rezult˘a din eliminarea ultimelor N elemente ale lui L. Predicatul va fi fals ˆın cazul ˆın care N este mai mare decˆat lungimea lui L.

dropN(L, R, N) :- append(R, L1, L), length(L1, N).

**ex3:**

Consider˘am ˆın continuare reprezentarea formulelor logicii propozit¸ionale folosit˘a ˆın laboratorul 5. Scriet¸i un predicat rmdn/2 cu proprietatea c˘a rmdn(Phi, Psi) este adev˘arat dac˘a ¸si numai dac˘a Psi este rezultatul elimin˘arii tuturor negat¸iilor duble din Phi.

rmdn(Phi, Phi) :- atom(Phi).

rmdn(non(Phi), non(Phi)) :- atom(Phi).

rmdn(non(non(Phi)), Psi) :- rmdn(Phi, Psi).

rmdn(non(si(Phi,Psi)),non(A)) :- rmdn(si(Phi,Psi),A).

rmdn(non(sau(Phi,Psi)),non(A)) :- rmdn(sau(Phi,Psi),A).

rmdn(non(imp(Phi,Psi)),non(A)) :- rmdn(imp(Phi,Psi),A).

rmdn(si(Phi, Psi), si(Phi1, Psi1)) :- rmdn(Phi, Phi1), rmdn(Psi, Psi1).

rmdn(sau(Phi, Psi), sau(Phi1, Psi1)) :- rmdn(Phi, Phi1), rmdn(Psi, Psi1).

rmdn(imp(Phi, Psi), imp(Phi1, Psi1)) :- rmdn(Phi, Phi1), rmdn(Psi, Psi1).