Project 2 KRR - Iordachescu Anca Mihaela

1. Define my own Knowledge Base:

- If a person ***travels on planes*** and is vaccinated at least two times, then he arrives in the USA.
- If a person is at least 14 years old and ***has a visa***, he ***travels on planes***.
- If a person does not intend to stay in a country more than 90 days and the purpose of the journey is tourism or business, he ***has a visa***.

Questions:

1. How many days does the person intend to stay in a country?
2. Is the person's purpose of the journey tourism/business?
3. How old is the person?
4. How many times has the person been vaccinated?

The goal is to decide if a person arrives in the USA.

The knowledge base expressed as positive Horn propositional clauses in the form, where n(x) means the negation of x:

KB : [[n(travel_plane), n(vacinnated_2_times), arrive_USA], [n(age_14), n(visa), travel_plane], [n(not_stay_more_90_days), n(jorney_t_b), visa]]

Based on the KB and the answer that the user provided to the system, it should say whether or not the person arrives in the USA can be logically entailed.

I have implemented both backward and forward chaining algorithms, the KB is read from an input file and the output is written in the console.

2. The rules are:

- If the blood pressure level is high, the cardiovascular risk chance is high.
- If the total cholesterol level is moderate or the blood pressure level is moderate, the cardiovascular risk chance is moderate.

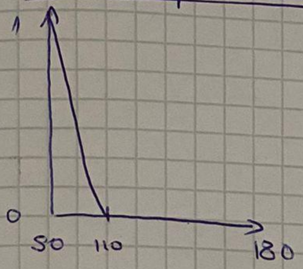
- If the total cholesterol level is low and the blood pressure level is low, the cardiovascular risk chance is low.

Rules:
[(or, [blood/high], risk/high),
(or,[blood/moderate,cholesterol/moderate],risk/moderate),
(or, [blood/low, cholesterol/low], risk/low)]

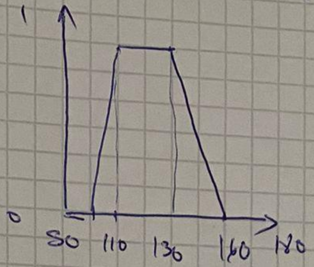
Premises:
[(blood/high, [(90,0), (110,0), (160,1), (180,1)]),
(blood/moderate, [(90,0), (100,0), (110,1), (130,1), (160,0), (180,0)]),
(blood/low, [(90,1), (110, 0), (180,0)]),
(cholesterol/moderate, [(160, 0), (190, 1), (210,1), (220, 0), (240,0)]),
(cholesterol/low, [(160,1), (200,0), (240,0)]),
(risk/high, [(0,0),(50,0), (80,1), (100,1)]),
(risk/moderate, [(0,0),(20,0), (40,1),(60,0), (100,0)]),
(risk/low,[(0,1), (30,0), (100,0)])]

Degree curves

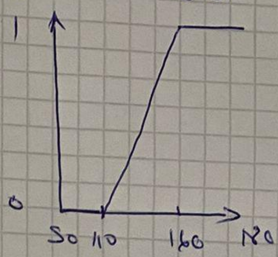
Blood pressure - values between 80 and 180



low

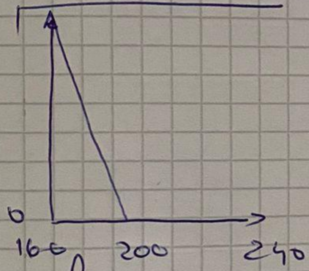


moderate

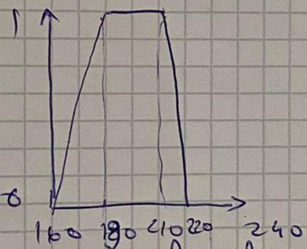


high

Cholesterol level - values between 160 and 240

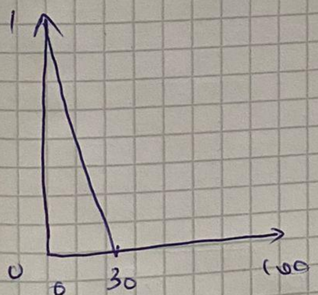


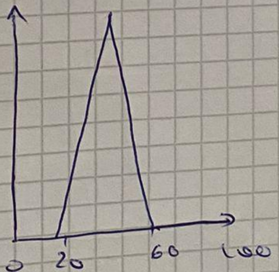
low

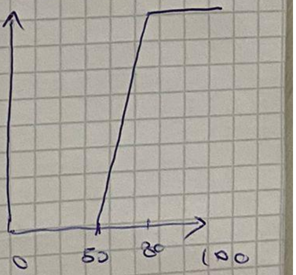


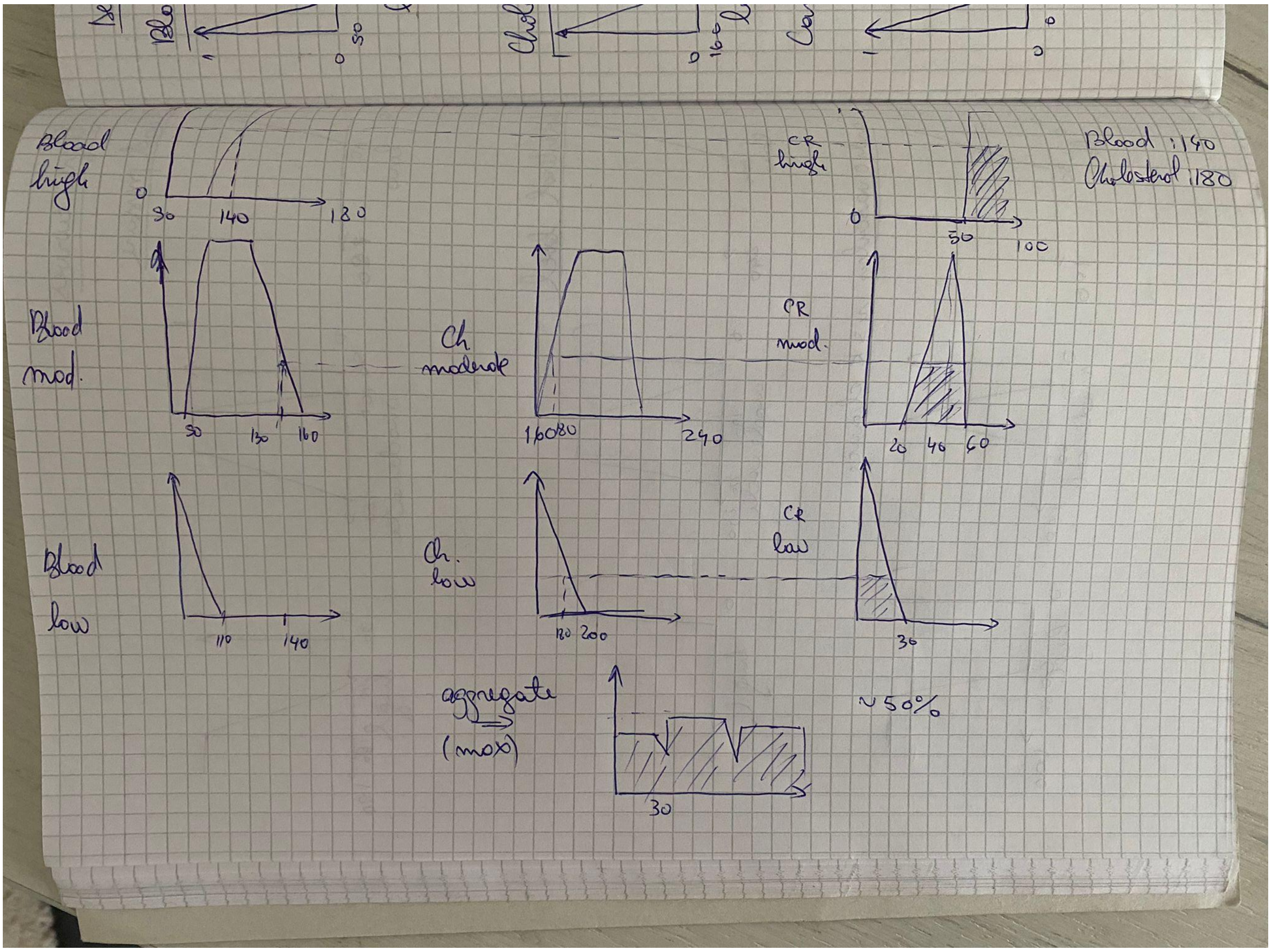
moderate

Cardiovascular ~~level~~ risk chance - values between 1 and 100









Code for ex.1:

```
delete_neg([], []).
delete_neg([n(H)|T], [H|T1]):- delete_neg(T, T1), !.

append([], X, X).
append([H|T], X, [H|T1]) :-
   append(T, X, T1).

number_pos_literals([], 0).
number_pos_literals([n(_)|T], C) :- number_pos_literals(T, C),!.
number_pos_literals([_|T], C) :- number_pos_literals(T, Cnew), C is Cnew+1.

get_pos_literal([],0).
get_pos_literal([n(_)|T], Lit) :- get_pos_literal(T,Lit),!.
get_pos_literal([H|_], H).
```

```
backward_chaining_f([], _, 'YES').
backward_chaining_f([Goal|Goals], KB, Ans) :- member(Cl, KB),
member(Goal, Cl),
  delete(Cl, Goal, Cl1), delete_neg(Cl1, NewCl),append(Goals, NewCl,
NewGoals), backward_chaining_f(NewGoals, KB, Ans), !.
backward_chaining_f(_, _, 'NO') :-!.

delete1(Val, [Val|X], X).
delete1(Val, [X|T1], [X|T2]):-
  delete1(Val, T1, T2).

perm([], []).
perm([X|T1], Ans):-perm(T1, Rez), delete1(X, Ans, Rez).


forward_chaining_f(Goals, _, S, 'YES') :- intersection(Goals, S, Int), perm(Int,
Goals).
forward_chaining_f(Goals, KB,S, Ans) :- member(Cl, KB), member(Goal, Cl),
not(Goal=n(_)), not(member(Goal, S)), delete(Cl, Goal, Cl1),
                                                            delete_neg(Cl1,
NewCl), intersection(NewCl, S, S1), perm(S1, NewCl),
                                   forward_chaining_f(Goals, KB, [Goal|S], Ans),
!.
forward_chaining_f(_, _, _, 'NO') :- !.


days_q(Pred) :-
 repeat,
 writeln('How many days does the person intend to stay in a country? the
answer is a number'),
 read(Days),nl,
 (Days > 90 ->
  writeln('The person intends to stay in a country more than 90 days.'), Pred =
n(not_stay_more_90_days), !
 ; writeln('The person does not intend to stay in a country more than 90
days.'), Pred = not_stay_more_90_days,
  !
 ).

age_q(Pred) :-
```

```
  repeat,
  writeln('How old is te person? the answer is a number'),
  read(Age),nl,
  (Age >= 14 ->
    writeln('The person is at least 14 years old.'), Pred = age_14, !
  ; writeln('The person is at most 14 years old.'), Pred = n(age_14),
    !
  ).

purpose_q(Pred) :-
  repeat,
  writeln('Is the person purpose of the journey tourism/business?(the answer is
yes/no)'),
  read(Choice),nl,
  (Choice == 'yes' ->
    writeln('The purpose of the journey is tourism or business.'), Pred =
jorney_t_b, !
  ; writeln('The purpose of the journey is not tourism or business,'), Pred =
n(jorney_t_b),
    !
  ).


vaccinated_q(Pred) :-
  repeat,
  writeln('How many times has the person been vaccinated? the answer is a
number'),
  read(Vacc),nl,
  (Vacc >= 2 ->
    writeln('The person has been vaccinated at least 2 times.'), Pred =
vacinnated_2_times, !
  ; writeln('The person has been vaccinated at most 2 times.'), Pred =
n(vacinnated_2_times),
    !
  ).

questions([[P1],[P2],[P3],[P4]]) :- days_q(P1), age_q(P2), purpose_q(P3),
vaccinated_q(P4).
```

```
main1(KB):- repeat, questions(X),
                 append(KB, X, NewKB), backward_chaining_f([arrive_USA],
NewKB, Ans), write('Backward chaining answer: '), writeln(Ans),
                 forward_chaining_f([arrive_USA], NewKB,[], Ans1),
write('Forward chaining answer: '), writeln(Ans1),
        writeln('Write stop - stop the execution; Write anything else - continue.'),
read(Choice), nl, (Choice == 'stop' ->
   writeln('You selected to stop the execution of the program.'),!; writeln('You
selected to continue.'), fail).
```

```
read_clauses(S,[]) :- at_end_of_stream(S).
read_clauses(S,[H|T]) :- not(at_end_of_stream(S)), read(S,H),
read_clauses(S,T).

main:- open('own_kb.txt', read, S), read(S, H), close(S), main1(H).
```

Code for ex. 2:

```
get_coordonates([],[],[]).
get_coordonates([(X,Y)|T], [X|T1], [Y|T2]) :- get_coordonates(T, T1, T2).

my_max([], V, V).
my_max([H|T], Max, Ans):- H >  Max, my_max(T, H, Ans).
my_max([H|T], Max, Ans):- H =< Max, my_max(T, Max, Ans).
my_max([H|T], Ans):- my_max(T, H, Ans).

my_min([], V, V).
my_min([H|T], Min, Ans):- H =<  Min, my_min(T, H, Ans).
my_min([H|T], Min, Ans):- H > Min, my_min(T, Min, Ans).
my_min([H|T], Ans):- my_min(T, H, Ans).

intervals([_], []).
intervals([X,Y|T], [(X,Y)|I]) :- intervals([Y|T], I).


get_function_aux(X1,X1,_,_,_,_).
get_function_aux(X1,X2,Y1,Y2,C1,C2) :- A is Y2-Y1, B is X2-X1, C1 is A/B,
C2 is -1*X2*C1+Y2.

functions([_],[_],[]).
functions([X1,X2|T1], [Y1,Y2|T2], [(M,N)|T3]) :-
get_function_aux(X1,X2,Y1,Y2,M,N),

functions([X2|T1], [Y2|T2], T3).

obtain_functions_from_points(P,F) :- get_coordonates(P,X,Y),
functions(X,Y,F).

calculate_y(M,N,X,Y) :- Y is M*X+N.

calculate_membership([(M,N)|_], [X1,X2|_], X, V) :- X1 =< X, X =< X2,
calculate_y(M,N,X,V).
calculate_membership([(_,_)|F], [_,X2|T], X, V) :-
calculate_membership(F,[X2|T],X,V).


return_curve(A/B, [(A/B,Y)|_], Y):- !.
return_curve(A/B, [(_/_,_)|T], X) :- return_curve(A/B, T, X).

get_val_predicate(X, [X/N|_], N) :- !.
get_val_predicate(V, [_/_|T], A) :- get_val_predicate(V,T, A).

get_value_aux(P,Val,Rez) :- get_coordonates(P,X,Y), functions(X,Y,F),
calculate_membership(F,X,Val, Rez).
get_value(A/B, Val, Curves, Rez) :- return_curve(A/B, Curves, P),
get_value_aux(P, Val, Rez).

get_degrees([],_,_,[]).
get_degrees([A/B|Predicates], Val, Curves, [Ans|Rez]) :- get_val_predicate(A,
Val, V), get_value(A/B, V, Curves, Ans), get_degrees(Predicates, Val, Curves,
Rez).

calculate_intersection(M, N, Y, X) :- A is Y-N, X is A/M.

f_intersection(X1,Y1,X2,Y2,Y,[(X1,Y1)]) :- Y1 =< Y, Y2 =< Y, !.
f_intersection(X1,Y1,X2,Y2,Y, [(X1,Y)]) :- Y =< Y1, Y =< Y2, !.
f_intersection(X1,Y1,X2,Y2,Y, [(X1,Y1), (Int,Y)]) :- Y1 < Y, Y < Y2,
get_function_aux(X1,X2,Y1,Y2,M,N), calculate_intersection(M,N,Y,Int), !.
f_intersection(X1,Y1,X2,Y2,Y, [(X1,Y), (Int,Y)]) :- Y1 > Y, Y > Y2,
get_function_aux(X1,X2,Y1,Y2,M,N), calculate_intersection(M,N,Y,Int), !.

get_intervals([(X0,Y0)], Y, [(X0,Y0)]) :- Y0 =< Y, !.
get_intervals([(X0,Y0)], Y, [(X0,Y)]) :- Y =< Y0, !.
get_intervals([(X1,Y1), (X2,Y2)|T], Y, Int) :- get_intervals([(X2,Y2)|T], Y, Int1),
f_intersection(X1,Y1,X2,Y2,Y, NewInt),append(NewInt, Int1, Int).



degree_connector((Connector,Premises,Conc), Val, Curves, X) :-
get_degrees(Premises, Val, Curves, D), Connector==or, my_max(D, Max),
return_curve(Conc, Curves, ConcC), get_intervals(ConcC, Max, X), !.
degree_connector((Connector,Premises,Conc), Val, Curves, X) :-
get_degrees(Premises, Val, Curves, D), Connector==and, my_min(D, Min),
return_curve(Conc, Curves, ConcC), get_intervals(ConcC, Min, X), !.

calculate_intersection(M1,M2,N1,N2,XI, YI) :- A is N2-N1, B is M1-M2, XI is
A/B, YA is M1*XI, YI is YA+N1.

aggregate_curve([(A,C)], [(_,D)], [(A,C)]) :- D =< C,!.
aggregate_curve([(_,C)], [(B,D)], [(B,D)]) :- C =< D,!.
aggregate_curve([(X1,Y1),(X2,Y2)|T1], [(A1,B1),(A2,B2)|T2], [(X1, Y1)|List]) :-
B1 =< Y1, B2 =< Y2, C is min(X2, A2),
        (C == A2 -> get_function_aux(X1,X2,Y1,Y2,M,N),
calculate_y(M,N,A2,Y), (C ==X2, Y == Y2  ->  aggregate_curve([(X2,Y2)|T1],
[(A2,B2)|T2], List);aggregate_curve( [(C,Y),(X2,Y2)|T1], [(A2,B2)|T2], List));
        get_function_aux(A1,A2,B1,B2,M,N), calculate_y(M,N,X2,Y), (C == A2,
Y2 == B2 ->  aggregate_curve([(X2,Y2)|T1], [(A2,B2)|T2],
List);aggregate_curve([(X2,Y2)|T1], [(C,Y2),(A2,B2)|T2], List))), !.

aggregate_curve([(X1,Y1),(X2,Y2)|T1], [(A1,B1),(A2,B2)|T2], List) :- Y1 < B1,
Y2 < B2, aggregate_curve([(A1,B1),(A2,B2)|T2],[(X1,Y1),(X2,Y2)|T1], List), !.
aggregate_curve([(X1,Y1),(X2,Y2)|T1], [(A1,B1),(A2,B2)|T2], [(X1, Y1),
(XI,YI)|List]) :- B1 =< Y1, Y2 =< B2,
                        get_function_aux(X1,X2,Y1,Y2,M1,N1),
get_function_aux(A1,A2,B1,B2,M2,N2),
calculate_intersection(M1,M2,N1,N2,XI, YI),
                        aggregate_curve([(X2,Y2)|T1], [(A2,B2)|T2], List), !.
aggregate_curve([(X1,Y1),(X2,Y2)|T1], [(A1,B1),(A2,B2)|T2], List) :- B1 < Y1,
B2 < Y2,  aggregate_curve([(A1,B1),(A2,B2)|T2],[(X1,Y1),(X2,Y2)|T1], List), !.


aggregate_all([Rule1, Rule2|[]],[Val1, Val2|[]], Curves, X) :-
degree_connector(Rule1, Val1, Curves,X1), degree_connector(Rule2, Val2,
Curves,X2),

                              aggregate_curve(X1, X2, X), !.
aggregate_all([Rule|T1], [Val|T2], Curves, Ag) :- degree_connector(Rule, Val,
Curves,X), aggregate_all(T1, T2, Curves, NewAg), aggregate_curve(X,
NewAg, Ag).


premises_values([],_,_,[]) :- !.
premises_values([blood/_|R],S,F, [blood/S|T]) :- premises_values(R,S,F,T), !.
premises_values([cholesterol/_|R],S,F,[cholesterol/F|T]) :-
premises_values(R,S,F,T), !.

premises_for_all_rules([],_, _,[]).
premises_for_all_rules([(_,Premises,_)|T], S,F, [R|T1]) :-
premises_values(Premises,S,F, R), premises_for_all_rules(T,S,F,T1).


sum_prod_aux([],[],0) :- !.
sum_prod_aux([A1|A],[B1|B], V) :- sum_prod_aux(A,B,C), Aux is A1*B1, V is
Aux+C.

sum_aux([], 0) :- !.
sum_aux([A|A1], V) :- sum_aux(A1, V1), V is V1+A.


centroid(X, Rez) :- get_coordonates(X,A,B), sum_prod_aux(A, B, C),
sum_aux(B, D), Rez is C/D.

blood_q(Value) :-
  writeln('The value of blood pressure:'),
  read(Value).

cholesterol_q(Value) :-
```

```
  writeln('The value of cholesterol:'),
  read(Value).


main_aux(Rules, Curves) :-
  writeln('Welcome.'),
  repeat,
        blood_q(S), cholesterol_q(F), premises_for_all_rules(Rules, S, F, P),
aggregate_all(Rules, P, Curves, X), writeln(X), centroid(X,C), writeln(C),
  writeln('Write stop to terminate the execution or anything else to continue'),
  read(Ans),nl,
  (Ans == 'stop' -> writeln('Stop'),!;  fail
  ).

main:- open('rules.txt', read, S), read(S, Rules), close(S), open('curves.txt',
read, S1), read(S1, Curves), close(S1), main_aux(Rules, Curves).
```