CSC-325
Artificial Intelligence
Dr. Adam Wyner

Lab 2

Feb 24, 2023.
Last day to sign-off: Mar 3, 2023.

The practical has one question with three parts (A)-(C) below, all of which must be completed to complete the practical.

Your work, solution, and code should be your own. If you discuss how to address the problem, use words and narratives.

It is strongly recommended that you read through the entire lab information before beginning to work. The hints are there for a reason.

To get signed off, you should:

- Show your code and solutions to a lecturer/demonstrator;
- Explain your code and solutions to a lecturers/ demonstrators.

```
% Code for an Expert System Shell:

% See Bratko pages 353-356.
%
% The code below is for the expert system shell. This is the shell that you need to use
% to write your own expert system; that is, your task will be to write the rules
% which use the shell. Get the shell running in Prolog and test using Bratko's
% examples.
%
% Interaction with user and why and how explanation

% Operators for easy to read rules.

:-  op( 800, fx, if).
:-  op( 700, xfx, then).
:-  op( 300, xfy, or).
:-  op( 200, xfy, and).
:-  op( 800, xfx, <=).

%%%%
% Adam Wyner
% Added the dynamic fact predicate as there were otherwise errors.
% p. 351 in Bratko book.

:- dynamic( fact/1).

% is_true( P, Proof): Proof is a proof that P is true

is_true( P, Proof)  :-
  explore( P, Proof, []).

%  explore( P, Proof, Trace):
%     Proof is an explanation for P, Trace is a chain of rules between P's ancestor goals

explore( P, P, _)  :-
  fact( P).                                          %  P is a fact

explore( P1 and P2, Proof1 and Proof2, Trace)  :-  !,
  explore( P1, Proof1, Trace),
  explore( P2, Proof2, Trace).

explore( P1 or P2, Proof, Trace)  :-  !,
  (
    explore( P1, Proof, Trace)
```

```
    ;
    explore( P2, Proof, Trace)
  ).

explore( P, P <= CondProof, Trace)  :-
  if Cond then P,                                   %  A rule relevant to P
  explore( Cond, CondProof, [ if Cond then P | Trace]).

explore( P, Proof, Trace)  :-
  askable( P),                                      % P may be asked of user
  \+ fact( P),                                      % P not already known fact
  \+ already_asked( P),                             % P not yet asked of user
  ask_user( P, Proof, Trace).

ask_user( P, Proof, Trace)  :-
  nl, write( 'Is it true:'), write( P), write(?), nl, write( 'Please answer yes, no, or why'), nl,
  read( Answer),
  process_answer( Answer, P, Proof, Trace).         % Process user's answer

process_answer( yes, P, P  <= was_told, _)  :-      % User told P is true
  asserta( fact(P)),
  asserta( already_asked( P)).

process_answer( no, P, _, _)  :-
  asserta( already_asked( P)),                      % Make sure not to ask again about P
  fail.                                             % User told P is not true

process_answer( why, P, Proof, Trace)  :-           % User requested why-explanation
  display_rule_chain( Trace, 0), nl,
  ask_user( P, Proof, Trace).                       % Ask about P again

display_rule_chain( [], _).

display_rule_chain( [if C then P | Rules], Indent)  :-
  nl, write( 'To explore whether '), write( P), write(' is true, using rule:'),
  nl, write( if C then P),
  NextIndent is Indent + 2,
  display_rule_chain(  Rules, NextIndent).

:- dynamic already_asked/1.
```

PROBLEM:

Write rules in Prolog for an interactive expert which does a consultation in relation to the coronavirus. Then write queries for the rules which provide explanations.

Expressed in natural language, the rules are:
1. You are healthy if you have self-isolated for two weeks and you have had no symptoms for two weeks.
2. You may be infected if you went to a large party and a person at the party tested positive.
3. You may not be immune if you are not vaccinated or you have not previously been ill with coronavirus.
4. You should get tested if you may be infected and you may not be immune and you have symptoms.

Do the following:

A. Write in Prolog the rules for the natural language expressions above so that you can write queries for the interactive expert system and get explanations. Note: You do not need a full and detailed representation of the natural language expressions, just predicates; in other words, you need to consider what aspects of natural language are relevant to represent in your rules. For example, from a sentence such as "You may be happy", we might have the Prolog predicate may_be_happy, as the predicate applies generically. If you want it to apply specifically to individuals, it might be may_be_happy(you). See Bratko for examples.
B. Note that you cannot use negation-as-failure, though there are several negated expressions. This means that negated expressions in natural language will need to be included in the Prolog predicate; for example "You are not happy" might be the Prolog predicate not_happy. You should explain why negation-as-failure will not work in this context.
C. Write queries and report the interactions for the following, giving alternative responses (yes, no, why) to the queries so that at least in one run the explanation is generated (using How), in another a response to 'why' is provided, and in another run, the answer is 'false'.
    a. Are you healthy?
    b. Should you get tested?

Hints:

1. There is syntactic sugar. Rather than (a), we use (b). That is the point of the redefinition of operators. Recall:
    a. leak_in_bathroom :- hall_wet, kitchen_dry.
    b. if hall_wet and kitchen_dry then leak_in_bathroom.
2. We don't use predicates with terms, but only atomic propositions. So, for example, rather than (a), we use (b). For example:
    a. if hall_wet(X) and kitchen_dry(X) then leak_in_bathroom(X).
    b. if hall_wet and kitchen_dry then leak_in_bathroom.
3. You will need to introduce 'askables' as discussed in the book chapter.