Nanyang Technological University

CZ3005
Artificial Intelligence

Question 4
Patient with a Sympathetic Doctor

Poon Wing Sze

N1904283E

Group TS5

15 April 2020

Description: Patient with a sympathetic doctor

Assume that the prolog script is a sympathetic doctor, conversing with a patient who can answer only yes or no. The doctor should be able to diagnose the patient's condition while asking question sensitively depending upon patient's pain level and mood level. You can choose 5 or more different mood considerations (calm, angry, etc.) and 5 or more levels of pain. Five or more diseases should be diagnosable, each disease characterized by 5 or more symptoms.

Interpretation of Requirements

1. Other than the patient's pain and mood, the doctor would ask if the patient has certain symptoms for making the diagnosis.
2. The patient's mood and pain level are unlikely to change during the diagnosis.
    a. It is only assessed once during the diagnosis.
3. According to the hints, there are some response gestures given by the doctor.
    a. The doctor would express his gesture when he asks a question.
    b. The type of gestures would changes based on the patient's response.
4. The patient may not have any illnesses.
5. Dynamic database should be used to record the response of the patient.

Platform

- SWI-Prolog

Program Flow

```
%The command to start the session.
entry() :- reset(), moodAssessment(), painAssessment(), matchPainLevel(),
    mood(M), pain(A,B),
    format("So you feel ~w and have ~w pain which has a pain level of ~w.", [M,A,B]),
    nl(), diagnose().
```

Upon entry of the patient, the doctor would clear the memory of his previous sessions. The doctor starts by assessing the mood and pain level. The pain level will be interpreted and match with various illnesses. The doctor would give a summary of the assessment before starting the diagnosis. The doctor would tell the result at the end of the diagnosis.

Diagnosable Diseases

```
/*
    There are a total of 6 diagnosable diseases.
    Predicate name:  Name of the disease
    First argument:  Serious level of the disease. Affect the gesture of doctor
    Second argument: Standard pain level of the disease
    Third argument:  List of symptoms of the disease
*/
coronavirus(1,0,[body_ache,breathlessness,cough,diarrhea,headache,high_temperature]).
food_poison(0,3,[body_ache,dehydration,diarrhea,nausea,rapid_heart_beat]).
injury(1,4,[bleeding,body_ache,bruises,dizziness,headache,nausea]).
fever(0,1,[body_ache,high_temperature,shivering,sweating,rapid_heart_beat]).
cold(-1,0,[cough,dehydration,headache,high_temperature,sneeze,sweating,runny_nose]).
low_blood_sugar(-1,0,[dehydration,headache,nausea,shivering,rapid_heart_beat]).
```

There is a total of 6 diagnosable diseases which are coronavirus, food poisoning, physical injury, fever, cold and low blood sugar level.
Details of each disease is listed as arguments. Refer to the comment block for the meaning of each argument.

Counter Variables

```
%Counters used in the program
%Percepted serious level of the illness. Affect gesture of the doctor
seriousLevel(0).
%Likelihood of different illnesses. Differentiated bt first two character
cv_count(0). %Coronavirus
fp_count(0). %Food Poisoning
ij_count(0). %Injury
fv_count(0). %Fever
cd_count(0). %Cold
lb_count(0). %Low Blood Sugar
```

Predicates are used to mimic counter variables with the help of dynamic scope.
There are one counter to determine the gesture of the doctor and one counter for each diagnosable disease.
All the counters are initialized to have value 0.

```
%Rules to update counters
adjSerious(N) :- seriousLevel(X), Y is X+N, retract(seriousLevel(X)), assert(seriousLevel(Y)),!.
cv_adj(N) :- cv_count(X), Y is X+N, retract(cv_count(X)), assert(cv_count(Y)),!.
fp_adj(N) :- fp_count(X), Y is X+N, retract(fp_count(X)), assert(fp_count(Y)),!.
ij_adj(N) :- ij_count(X), Y is X+N, retract(ij_count(X)), assert(ij_count(Y)),!.
fv_adj(N) :- fv_count(X), Y is X+N, retract(fv_count(X)), assert(fv_count(Y)),!.
cd_adj(N) :- cd_count(X), Y is X+N, retract(cd_count(X)), assert(cd_count(Y)),!.
lb_adj(N) :- lb_count(X), Y is X+N, retract(lb_count(X)), assert(lb_count(Y)),!.
```

Implementation of updating counters. Same implementation is used for all.
Given the increment value **`N`** and current value **`X`** of the counter, the new value of the counter should be **`Y`** (**`Y is X+N`**). The fact with the old value is removed from the knowledge base(retract). The fact with the new value is inserted to the knowledge base(assert).

Reset Counter Variables

```
% Remove all records of the counters and insert new record with their default value
reset() :- retractall(seriousLevel(_)), assert(seriousLevel(0)),
    retractall(symptom(_,_)), assert(symptom(nothing, n)),
    retractall(cv_count(_)), assert(cv_count(0)), retractall(fp_count(_)), assert(fp_count(0)),
    retractall(ij_count(_)), assert(ij_count(0)), retractall(fv_count(_)), assert(fv_count(0)),
    retractall(cd_count(_)), assert(cd_count(0)), retractall(lb_count(_)), assert(lb_count(0)).
```

Mood Assessment

The Mood Library

```
/*
    The mood_library contains a list of all possible mood of the patient
    For each element in the list, first argument describes the mood
        and second argument is the influence on the doctor's gesture
*/
mood_library([[peaceful|-2], [calm|-1], [content|-1], [angry|3], [weepy|1],
    [stressed|1], [irritated|2], [cranky|2]]).
```

Here is the list of possible mood of the patient for mood assessment.

Query Order

Among all the possible moods, the doctor would pick a random one from the list that has not been asked before. If the patient rejected all the choices, the doctor would restart the assessment.

Implementation

```
/*  Mood Assessment
    A random sequence of question will be generated */
moodAssessment() :- retractall(mood(_)), % Have to clear the memory before it starts
    mood_library(Lib), random_permutation(Lib, MoodSeq), askMood(MoodSeq).
```

Upon the start, a random permutation of the **`mood_library`** will be generated. The question order is determined by the random sequence. The random sequence **`MoodSeq`** is passed to **`askMood()`** which would perform the query.

```
/*  Given a list of mood, ask about the first element and remove from the list
    M: Description of the mood
    X: Effect on the serious level of the mood
    T: Remaining options    */
askMood(L) :- expressGesture(), L = [[M|X]|T],
    format("Do you feel ~w ? y/n/q: ", [M]), read(Ans),
    (Ans==q -> abort; Ans==y -> assert(mood(M)), adjSerious(X); % If answers yes, update knowledge base
        (T=[] -> print("You are not sure about your mood. Let me ask again."), nl(), moodAssessment();
        %If answers no and no more choices, restart the assesment
            askMood(T))). % If answers no, move on to the next choice
```

Doctor expresses his gesture before he asks the question about his mood.
Given the list of potential moods, the doctor would ask about the first choice in the list. If the patient confirms, the knowledge base will be updated to store his mood and the doctor's serious level is changed. If the patient denies, the doctor would move on to the next option in the list. If no more option is available, the mood assessment restarts.

Pain Assessment

The Pain Library

```
/*
    The pain_library contains a list of all possible pain levels
    For each element in the list, first argument describes the pain level
        and second argument is the influence on the doctor's gesture
*/
pain_library([[worst_possible|5], [unbearable|5], [severe|4], [intense|4],
    [lot_of|3], [moderate|2], [discomforting|2], [manageable|1], [mild|1]]).
```

Here is the list of possible pain levels for pain assessment.

Query Order

Different from mood assessment, the doctor starts by asking if the patient experiences any pain. If he experiences pain, the doctor would start asking from the middle of the list. When the patient rejects the choice, the doctor would ask if the pain is more or less intense compares to the suggested level. A binary search is performed for the pain level assessment.

Implementation

```
/*  Pain Assessment
    First ask whether the patient have any pain */
painAssessment() :- expressGesture(), retractall(pain(_,_)), % Clear previous memory about pain
    print("Do you feel any pain? y/n/q:"), read(Ans), (Ans==q -> abort;
    Ans==y -> print("From mild pain to worst possible pain, please tell me your pain level."), nl(),
        pain_library(P), askPain(P);        % Answer yes, start to ask for the pain level
    assert(pain(no,0)), adjSerious(-3)).    % Answer no, patient do not have pain
```

The doctor would ask if the patient experiences any pain. The binary search would start if answer is ‘yes’ and the list of pain levels **`P`** is passed to **`askPain()`**.

```
/*  Given a list of pain levels, perform binary search */
askPain(L) :- getMid(L,[M|X]), % Select the middle choice from the list
    format("Do you have ~w pain? y/n/q: " [M]), read(Ans), (Ans==q -> abort;
        Ans==y -> assert(pain(M,X)), adjSerious(X);     % Answer yes, record the pain level
        length(L,Len), Len>1 -> comparePain(L,[M|X]);   % Answer no, ask for comparision
            reassess()).        % Answer no and no more options, re-assess the pain level
```

Given the list of possible pain levels, ask about the mid-intensity using **`getMid()`**. Depending on the answers, the doctor will (1)record the pain level, (2)ask for comparison with his experienced pain level or (3)re-assess the pain level.

```
comparePain(L,[M|X]) :- format("Is it more intense than ~w pain? y/n/q: ", [M]), read(Ans),
    (Ans==q -> abort;
        Ans==y -> getFront(L,[M|X],N), askPain(N);  % N: Options with more intense pain
        getBehind(L,[M|X],N), askPain(N)).          % N: Options with less intense pain

reassess() :- print("You are not sure about your pain level. Let me ask again."), nl(),
    pain_library(P), askPain(P).
```

`comparePain()` uses the helper functions **`getFront()`** and **`getBehind()`** to remove half of the list. **`reassess()`** is called when restart is needed.

Helper Functions

```
/*  Obtain the middle element from the list
    Recursion: Remove one element at the front and back each time
    [_|L1] contains the list without the last element
    L1 contains the list without the first and last element
    Perform recursion to trim the list until it has 1 or 2 elements
*/
getMid([L],L).      % Only one element in the list
getMid([L,_|[]],L). % Only two elements in the list
getMid(L,M) :- removeLast(L,[_|L1]), getMid(L1,M), !.

/*  Remove the last element from the list
    Recursion: If there are more than one element in the list,
    Copy the first element to the front of the buffer.
    When the function returns, the previous element would be insert
    before the current element to preserve the order of the list
*/
removeLast([_],[]). % Only one the last element remains, remove and return empty list
removeLast([A|B],[A|C]) :- removeLast(B,C).
```

Functions used to obtain the middle element.

```
/*  Obtain the list of elements before element M.
    Append the first element to the buffer when
    the first element does not match with M.
    Return the buffer list N.   */
getFront([H|T],H,[]) :- !.
getFront([H|T],M,[H|N]) :- getFront(T,M,N), !.

/*  Obtain the list of elements after element M.
    Remove the first element of the list,
    until the first element matches with M.
    Return the remaining list N.    */
getBehind([H|T],H,T) :- !.
getBehind([_|T],M,N) :- getBehind(T,M,N), !.
```

Functions used to remove half of the list in binary search.

Diagnose using Reported Pain Level

```
% Compare the experienced pain level with the standard pain level of each disease.
matchPainLevel() :- pain(_,P), check_cv(P),check_lb(P),
    check_cd(P), check_fv(P), check_ij(P), check_fp(P), !.

/*  Increase the likelihood of disease diagnosis by 1
        if the pain level is similar to the standard
    P: Pain level of patient
    S: Standard pain level of the disease   */
check_cv(P) :- coronavirus(_,S,_), abs(P-S,X), X =< 1 -> cv_adj(1); true.
check_fp(P) :- food_poison(_,S,_), abs(P-S,X), X =< 1 -> fp_adj(1); true.
check_ij(P) :- injury(_,S,_), abs(P-S,X), X =< 1 -> ij_adj(1); true.
check_fv(P) :- fever(_,S,_), abs(P-S,X), X =< 1 -> fv_adj(1); true.
check_cd(P) :- cold(_,S,_), abs(P-S,X), X =< 1 -> cd_adj(1); true.
check_lb(P) :- low_blood_sugar(_,S,_), abs(P-S,X), X =< 1 -> lb_adj(1); true.
```

The doctor would compare the reported pain level with the standard pain level of each disease. If the pain levels are similar, the doctor believes the patient is more likely to have such disease. A difference of 1 level is allowed as people have different perception of pain intensity. Using **`abs()`**, it allows the comparison to include the range of ± 1. The counter of the disease is increased by 1 if the pain level matches.

Give Diagnosis Doctor's Gesture/Serious Level

```
/*  List of gestures performed by doctor at different attitudes.
    Normal gesture by default.
    Calming gesture if doctor gets serious.
    Polite gesture if doctor relaxes (less serious).    */
polite_gesture([look_concerned, mellow_voice, light_touch, faint_smile, smile]).
normal_gesture([broad_smile, joke, beaming_voice, writes_on_paper, adjust_mask]).
calming_gesture([greet, look_composed, look_attentive, pat_on_shoulder, deep_breath]).

/*  Doctor get serious if serious level > 2.
    Becomes relaxed if serious level < -2.
    A random gesture that fits his attitude is performed.   */
expressGesture() :- seriousLevel(X), (X < -2 -> polite_gesture(L);
    (X > 2 -> calming_gesture(L); normal_gesture(L))),
    random_member(E, L), format("(~w) ",[E]).
```

The gesture of the doctor is determined by his serious level. It increases when he thinks the patient might get a horrible disease and decreases if he thinks the disease can be easily treated. The serious level will be adjusted after each question.

`expressGesture()` would show the doctor's gesture.

Further Diagnosis

Initialization

```
possibleSymptoms(16). % Total number of possible symptoms
symptom(nothing, n).  % No known symptom initially
```

The rule about the number of possible symptoms is not used.

`symptom()` would record each query about the patient's symptoms. The first argument is the description of the symptom and the second argument signifies whether the patient have the symptom or not (**`'y'`** or **`'n'`**).

Helper Function

```
% Obtain values of counter variable
status([A,B,C,D,E,F],S) :- cv_count(A), fp_count(B), ij_count(C),
    fv_count(D), cd_count(E), lb_count(F), seriousLevel(S), !.
```

Give Appropriate Diagnosis

```
% Perform actions if sufficient information is obtained
% Otherwise, would ask for symptoms to gather more information
diagnose() :- expressGesture(), sufficientInfo(), !.
diagnose() :- askSymptom().
```

Doctor expresses his gesture before he asks queries about symptoms.

If sufficient information is obtained, the doctor would give his final judgement.

Otherwise, the doctor would continue to ask for certain symptoms.

```
% Arrive to a decision if the likelihood level reach the threshold,
%   when all diseases have very low likelihood,
%   or when all symptoms are asked and no decision is made
sufficientInfo() :- status(X,Y), max_list(X,Max), %print(X), print(Y),
    not(between(-1, 4, Max)) -> giveResult(Max); % Likelihood level is too high or too low
    queryHistory(H), length(H, Hs), Hs = 17 -> giveResult(0). % Exhausted all symptoms
```

A decision is reached, and the doctor would give his result upon 3 conditions:

- Any disease has accumulated a high enough likelihood (5)
- All the diseases have a very low likelihood (-2)
- All the possible symptoms were asked, and it does not match any disease

Incorporate Response to Knowledge Base

```
% Ask if the patient has a specific symptom S
% Record his answer and adjust the assessment base on the answer
askSymptom() :- mostProbableSymptom(S),
    format("Do you have ~w? y/n/q: ",[S]), read(Ans),
    (Ans==q -> abort; Ans==y -> assert(symptom(S,y)), matchDisease(S,1);
        assert(symptom(S,n)), matchDisease(S,-1)),
    diagnose(). % Continue the diagnosis after asking a question
```

If the patient confirmed that he has the symptom **S**, an entry **`symptom(S,y)`** would be created and for diseases with such symptom would increase their likelihood level by 1. If the patient denies, an entry **`symptom(S,n)`** would be created instead and likelihood level for respective diseases would be decreases by 1.

```
% Diseases with symptom S would adjust its likelihood level by N.
%   L_: List of symptom for respective diseases, used for finding matches with S
%   X_: The serious level of each disease,
%       doctor would changes his response as the likelihood for certain disease changes
matchDisease(S,N) :- coronavirus(X1,_,L1), (member(S,L1) -> adjSerious(X1*N), cv_adj(N); true),
    food_poison(X2,_,L2), (member(S,L2) -> adjSerious(X2*N), fp_adj(N); true),
    injury(X3,_,L3), (member(S,L3) -> adjSerious(X3*N), ij_adj(N); true),
    fever(X4,_,L4), (member(S,L4) -> adjSerious(X4*N), fv_adj(N); true),
    cold(X5,_,L5), (member(S,L5) -> adjSerious(X5*N), cd_adj(N); true),
    low_blood_sugar(X6,_,L6), (member(S,L6) -> adjSerious(X6*N), lb_adj(N); true).
```

Based on the response by the patient, the likelihood of getting certain disease is adjusted and the perceived serious level of the doctor would change too. Doctor is able to shift his attitude based on the response of patients.

Generate Related Symptoms

```
/*  Randomly pick one symptom from the symptom list of most likely diagnosis.   */
mostProbableSymptom(S) :- status(X,_), max_list(X,Max), %Max: The highest likelihood level
    % Lx would store the symptom list of each disease if they have the highest likelihood
    % Otherwise, Lx would be empty
    cv_count(C1), (C1 = Max -> coronavirus(_,_,L1); L1=[]),
    fp_count(C2), (C2 = Max -> food_poison(_,_,L2); L2=[]),
    ij_count(C3), (C3 = Max -> injury(_,_,L3); L3=[]),
    fv_count(C4), (C4 = Max -> fever(_,_,L4); L4=[]),
    cd_count(C5), (C5 = Max -> cold(_,_,L5); L5=[]),
    lb_count(C6), (C6 = Max -> low_blood_sugar(_,_,L6); L6=[]),
    % L123456 is the concatanation of L1-L6.
    % L is the set which eliminates duplicate entries in L123456
    append(L1,L2,L12), append(L12,L3,L123), append(L123,L4,L1234),
    append(L1234,L5,L12345), append(L12345,L6,L123456), list_to_set(L123456,L),
    queryHistory(H),    % H is a list of symptom that has been asked
    subtract(L,H,P),    % P is the remaining possible symptoms
    % If P is empty, all symptoms of the highly likely diseases were asked
    % Should eliminate these options
    (P == [] -> removeChoices(Max), mostProbableSymptom(S);
    % If P is not empty, pick a random item from P
    random_member(S,P)).
```

Based on information in the knowledge base, it would generate a most probable symptom for query. If multiple options are available, it would give a random item among the list of highly probable symptoms.

First, it obtains the highest level of likelihood **`Max`** among the 6 possible diseases. For each disease, if they have a likelihood level equals to **`Max`**, the list of symptoms for the respective disease would be retrieved. **`L`** is the set of symptoms for these highly top suspect symptoms. Duplicate entries in **`L`** are removed.

Previous query history **`H`** is obtained. Possible symptoms **`P`** is formed by removing entries in **`H`** from **`L`** to prevent asking the same question twice. A random item from the possible symptom list **`P`** would be selected for the query.

If there are no more choices, it implies that all symptoms of the top suspect disease are exhausted, and it is unlikely that the patient have such disease. These disease would be removed from the top suspect list by **`removeChoices()`**. A new list of probable symptoms would be generated again.

```
% Generate a list of symptoms based on the query history
queryHistory(H) :- findall(X,symptom(X,_),H).
```

Irrespective of the answer from the patient, all symptoms that was asked before would be included in the query history.
As the total number of symptoms are limited, we do not need to limit on the length of the list.

```
% Significantly reduce the likelihood for diseases with specific likelihood level.
removeChoices(N) :- (cv_count(N) -> cv_adj(-20); true),
    (fp_count(N) -> fp_adj(-20); true),
    (ij_count(N) -> ij_adj(-20); true),
    (fv_count(N) -> fv_adj(-20); true),
    (cd_count(N) -> cd_adj(-20); true),
    (lb_count(N) -> lb_adj(-20); true).
```

For diseases with a given likelihood level **`N`**, its likelihood level would be significantly reduced. It is unlikely that these diseases would be considered again.

Result Announcement

```
% Announce the result.
% N is the highest likelihood level among the diseases
% N>=5 represents a disease is diagnosed, patient can be diagnosed with multiple diseases
% Any smaller values represents that they do not have the disease
giveResult(N) :- N >= 5 ->
    (cv_count(N) -> print("You have COVID-19. Please go to the hospital for quarantine."), nl(); true),
    (fp_count(N) -> print("You have food poisioning. Please have some rest."), nl(); true),
    (ij_count(N) -> print("You are injuried. The nurse would treat your wound."), nl(); true),
    (fv_count(N) -> print("You have fever. Take these pills."), nl(); true),
    (cd_count(N) -> print("You have a cold. Take these pills."), nl(); true),
    (lb_count(N) -> print("You have low blood sugar level. Have some food and have a rest."), nl(); true);
    print("You do not have any illness. You can go home now."). % Can go home if they do not have any disease
```

It corresponds to the 3 condition which the doctor give diagnosis:

1. Any disease has accumulated a high enough likelihood (5)
   - **N** would be larger or equals to 5, any diseases with such likelihood level would be diagnosed.
   - Multiple diseases can be diagnosed
2. All the diseases have a very low likelihood (-2)
   - **N** would be -2, it would not match the first condition and the doctor would announce that they do not have any disease
3. All the possible symptoms were asked, and it does not match any disease
   - **N** would be 0, similarly, it does not match the first condition and patient can go home

Steps to execute the program

1. Import source code from “sympatheticDoc.pl”
2. Enable dynamic scope for counter variables. Input:
   - `dynamic seriousLevel/1, symptom/2, cv_count/1, cd_count/1, fp_count/1, fv_count/1, ij_count/1, lb_count/1.`
3. Use function `entry.` To initiate the procedure.

```
% c:/Users/pwc02/Desktop/CZ3005/sympatheticDoc.pl compiled 0.03 sec, 62 clauses
?- dynamic seriousLevel/1, symptom/2, cv_count/1, cd_count/1, fp_count/1, fv_cou
nt/1, ij_count/1, lb_count/1.
true.

?- entry.
(beaming_voice) Do you feel calm ? y/n/q: y.
(adjust_mask) "Do you feel any pain? y/n/q:"|: y.
"From mild pain to worst possible pain, please tell me your pain level."
Do you have lot_of pain? y/n/q: |: n.
Is it more intense than lot_of pain? y/n/q: |: n.
Do you have discomforting pain? y/n/q: |: y.
So you feel calm and have discomforting pain which has a pain level of 2.
(writes_on_paper) Do you have body_ache? y/n/q: |: n.
(joke) Do you have shivering? y/n/q: |: n.
(broad_smile) Do you have nausea? y/n/q: |: n.
(adjust_mask) Do you have dehydration? y/n/q: |: y.
(beaming_voice) Do you have sneeze? y/n/q: |: y.
(mellow_voice) Do you have headache? y/n/q: |: y.
(smile) Do you have sweating? y/n/q: |: y.
(mellow_voice) Do you have runny_nose? y/n/q: |: y.
(smile) "You have a cold. Take these pills."
true.
```