NANYANG
TECHNOLOGICAL
UNIVERSITY

CZ3005 – Artificial Intelligence
Gr. TSP4

An Assignement in Prolog – 23/11/17

De Bodinat, Jean
N1701748F

1. Problem Comprehension & Overview.

I chose the Assignment number 4, because it had an added difficulty, which made it more interesting. Indeed, not only the doctor i.e. the program needs to find the correct disease that our patient, i.e. user has, but it must adapt its questions formulation according to other questions non-related to the disease itself, but to the patient's state of mind (one could argue that they are related).

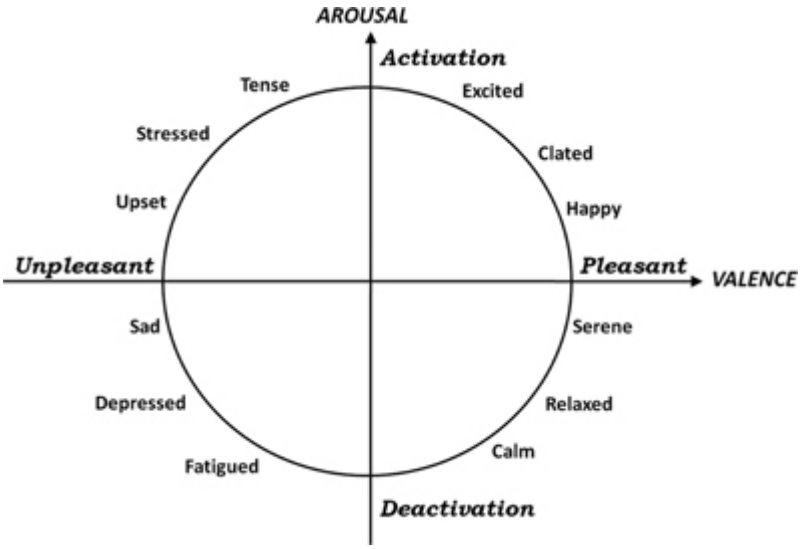
Therefore, I chose to build a simple set of 7 diseases that have from 2 to 8 symptoms, a pain level varying from 0 to 4 and 8 different mood states. The doctor will first ask some questions about the mood, then the pain and finally about symptoms. According to the answers about symptoms, the doctor will be able to diagnosis a disease, or not.

I tried very hard to make my program as moldable as possible, to make it easily maintained and upgraded. I found this task particularly hard in prolog. I'm aware that the same result could have been obtained with a simpler program, but it would've been less of a "good" coding style.

To execute the code, load the file 'doctor.pl' and write **consult.**

2. A realistic approach.

In artificial intelligence, and more specifically computational neuroscience, psychology plays a very important role. Therefore, to have a more realistic approach of what would really be implemented in a more sophisticated program, I decided to use the below quadrant to define emotions from James Russel. Emotions in my "doctor.pl" are ranked by arousal and then valence. This way, we can assign a realistic value to emotion which helps us a lot when trying to find the patient's emotion.



For simplicity matters, I decided to take the average between the pain and our realistic emotions index (the rank of the emotion) to define a state index for the patient. According to this index, we will choose a different prefix for the following questions about symptoms.

3. Different searching techniques: Sequential, Dichotomy, Memoization.

When trying to find a pain level or symptoms for example, by asking questions, I realized that a lot of different techniques to define which question to ask could be used. Those techniques had different advantage and disadvantage:

- Sequential:

All my diseases rules (l.87) are asked sequentially, by their order in the code. It means that the doctor will not choose the questions about symptoms according to the result of previous questions: It is a static order. This is very simple, and has the advantage to be easily modified (add or remove diseases and symptoms), but it is very "stupid", indeed a question about a symptom will be asked every time it appears in a disease.

- Memoization:

To avoid repeating questions, I used a dynamic programming technique where we search the disease tree "top-down" using memoization: We memorize the result of a question while going sequentially through the different diseases.
At first, I wanted to use a table to memorize the results, but after some research and failed experiences I felt like it was not suited for prolog, therefore I used a built-in function called assert/1 (l.32) which memorize predicates.
Then, whenever a question result is sought, we first check if it has already been asked, and return the previous result.
Implementing this logic made me face a lot of different difficulties, mainly in keeping my code simple, factorized and so, moldable.
The issue with this sequential analysis is that it will return the first disease rule that is true. If a disease A which has symptoms *a* and *b* is positioned before a disease B that has symptoms *a*, *b*, and *c*, then the disease B will never be found.

- Dichotomy:

The mood and the pain have assigned values, therefore, it is possible to find them using dichotomy. Here, I used a binary tree of If/else-like statements (l.185) that seek efficiently the right value of the patient's mood and pain. This technique is every efficient as we never have to ask a question twice, and the search is in O(lgn), but it is hard to add new values, as the decision search tree is more complex.
Because moods are classified in 2D, the same questions had to be written down 2 times (l.186, l.194) even if it is not asked twice.

Top-Down memoization and Dichotomy using a decision tree are two very different ways to find what our user has in mind, and they are both relevant in different type of search, depending on the possible results. Let's see more in detail how they have been merged with different logics in my prolog program.

4. Predicates Documentation.

Predicates in this prolog program are mainly related to the emotion and pain logic.

- ***Mood (rank, name):***

Mood establishes a relation between a mood (name) and a number (rank), therefore creating moods and pre-ranking them. I used this technique since the first lab, as I found it easy to implement and manage.
L.52

- ***Pain (rank, name):***

Very similar to the mood, but this time for pain Levels.
L.66

- ***Question (rank, string):***

Again, very similar to mood and pain.
L.74

5. Rules Documentation.

- ***Interface (Question, Input, Symptom):***

Question is the question that will be ask through the interface, Input is supposed to be a "Boolean" value that indicate whether an answer is expected and Symptom is only used when we need to keep the answer in memory.
This rule uses the JPL library to implement and Java interface that allows us to interact with the user.
It will display a window asking a question or giving some information and will register the answer given to the question.
L.15

- ***Interface_w_ans (Question) :***

Simple call of the above rule with predefined Input and Symptom. It will ask for an answer but register the answer on an obsolete predicate.
L.39

- ***Interface_n_asw (Phrase):***

Simple call of the above rule with predefined Input and Symptom. It will only display the Phrase.
L.44

- ***Symptoms_q (Prefix, Symptom):***

Prefix is only the prefix that will be added to the question, Symptom is the symptom the question will be asking on.
This rule first creates a question and is true, if the user answers yes or false if he answers no.
It also verifies that the questions, if it has, then it will take the value of the previous answer.
L.140

- ***Disease (Prefix, Name):***

Simple rule, the Prefix is the prefix that will be added to the question and Name is the name of the disease rule.
Each one of these rules rare true if every question about related symptoms is true. It is a very efficient way to build this relationship and query about a symptom, we don't even need to have symptoms or disease predicates.
L.92

- ***Consult:***

The sequence of rules we need check the value in order to implement the doctor interaction. **It is the instruction to launch the program.**
L.158

- ***Undo:***

Deletes the memorized value of Symptom.
L.177

- ***Prefix (X, Y):***

X is the pain level, Y the mood level, we compute the average and assign the string of the related question to Prefix. Since questions are unique for a given number, prefix will be unique.
L.182

- ***Get_mood (X):***

Dichotomy search of the patient's mood.
L.188

- ***Get_pain(Y):***

Dichotomy search of the patient's pain level.
L.209

6. Conclusion

By combining different search techniques, psychology, medicine and prolog logic, we can make a simple program which, by using a lot more data, might be able to really find a patient's disease. But this approach is very limited, because this algorithm could be improved indefinitely without ever really having a very accurate diagnosis, and the goal of a sympathetic doctor is a hard problem: on one side, it has to behave as a human, and on the other side it has to understand humans' emotions, which implies a certain empathy.