**Name - Anirban Dey**

**Roll - 002111001108**

**Question 1**

Implement the animal identification game (simple expert system).

Write a small and simple prolog program to create an efficient rule-based system that attempts to identify 10 different animals based on their characteristics.

Source Code :-

| *% Name - Anirban Dey* *% Roll - 002111001108* *%* *%* *% Question :-* *%* *% Implement the animal identification game (simple expert system).* *% Write a small and simple prolog program to create an efficient rule-based system that attempts to identify 10 different animals based on their characteristics.* *%* *% Solution :-*  *% Knowledge base with animals and their characteristics as lists* animal(lion, [carnivore, fur, mammals, 4]). animal(boar, [omnivore, fur, mammals, 4]). animal(elephant, [herbivore, rough\_skin, mammals, 4]). animal(kangaroo, [herbivore, fur, mammals, 2]). animal(whale, [herbivore, smooth\_skin, mammals, 0]). animal(hawk, [carnivore, feathers, birds, 2]). animal(tiger, [carnivore, fur, mammals, 4]). animal(butterfly, [herbivore, scales, insects, 6]). animal(spider, [carnivore, smooth\_skin, insects, 8]). animal(parrot, [herbivore, feathers, birds, 2]).  *% Predicate to ask the user for a feature value* ask\_feature\_value(Feature, Options, Value) :-  write('Enter a value for '), write(Feature), write(Options), write(': '),  read(Value).  ask\_feature\_value('no\_of\_legs', '', NoOfLegs) :-  write('Enter the number of legs: '),  read(NoOfLegs).  *% Predicate to check if an animal has a specified feature* animal\_has\_feature(Animal, Feature) :-  animal(Animal, Features),  member(Feature, Features).  *% Predicate to filter valid animals based on feature* filter\_valid\_animals([], \_, []). filter\_valid\_animals([Animal|Rest], Feature, ValidAnimals) :-  (animal\_has\_feature(Animal, Feature) ->  ValidAnimals = [Animal|NewValidAnimals]  ;  ValidAnimals = NewValidAnimals  ),  filter\_valid\_animals(Rest, Feature, NewValidAnimals).  *% Main predicate to interact with the user and update the list of valid animals* main :-  *% Initialize the list of valid animals with all animals*  findall(Animal, animal(Animal, \_), AllAnimals),  writeln('Initial list of valid animals: '),  writeln(AllAnimals), nl,    *% Ask for feature values and update the list*  ask\_feature\_value('diet', ' [carnivore/herbivore]', Diet),  filter\_valid\_animals(AllAnimals, Diet, UpdatedAnimals1),  writeln('Current list of valid animals: '),  writeln(UpdatedAnimals1), nl,   ask\_feature\_value('skin\_type', ' [fur/rough\_skin/smooth\_skin/feathers/scales]', SkinType),  filter\_valid\_animals(UpdatedAnimals1, SkinType, UpdatedAnimals2),  writeln('Current list of valid animals: '),  writeln(UpdatedAnimals2), nl,   ask\_feature\_value('animal\_class', ' [mammals/birds/insects]', AnimalClass), *% Ask for animal class*  filter\_valid\_animals(UpdatedAnimals2, AnimalClass, UpdatedAnimals3),  writeln('Current list of valid animals: '),  writeln(UpdatedAnimals3), nl,   ask\_feature\_value('no\_of\_legs', '', NoOfLegs), *% Ask for number of legs*  filter\_valid\_animals(UpdatedAnimals3, NoOfLegs, FinalValidAnimals),   *% Display the final list of valid animals*  writeln('Final list of valid animals: '),  writeln(FinalValidAnimals). |
| --- |

**Sample Input Output**

| % swipl Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4) SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software. Please run ?- license. for legal details.  For online **help** **and** background, visit https://www.swi-prolog.org **For** built-**in** **help**, **use** ?- **help**(Topic). **or** ?- apropos(Word).  ?- consult('assignment\_part1\_animals\_classication.pl'). true.  ?- main. **Initial** **list** **of** valid animals:  [lion,boar,elephant,kangaroo,whale,hawk,tiger,butterfly,spider,parrot]  Enter a **value** **for** diet [carnivore/herbivore]: herbivore. **Current** **list** **of** valid animals:  [elephant,kangaroo,whale,butterfly,parrot]  Enter a **value** **for** skin\_type [fur/rough\_skin/smooth\_skin/feathers/scales]: |: fur. **Current** **list** **of** valid animals:  [kangaroo]  Enter a **value** **for** animal\_class [mammals/birds/insects]: |: mammals. **Current** **list** **of** valid animals:  [kangaroo]  Enter a **value** **for** no\_of\_legs: |: 2. **Final** **list** **of** valid animals:  [kangaroo] true . |
| --- |

**Explanation :-**

The provided Prolog program is an implementation of a simple expert system that plays an animal identification game. Users are asked a series of questions about the characteristics of an animal, and the system filters the list of possible animals based on their features to make a guess. Here's a documentation of the program:

1. Knowledge Base

The program starts with a knowledge base that defines ten different animals and their characteristics. Each animal is represented as a predicate with a list of features, including their diet, skin type, animal class, and the number of legs.

2. User Interaction

The `main` predicate is the main entry point for interacting with the user and updating the list of valid animals. It follows these steps:

- Initialization: The list of valid animals is initialized with all ten animals.

- Asking for Diet: The user is prompted to enter the diet of the animal (carnivore or herbivore). Based on their response, the list of valid animals is filtered.

- Asking for Skin Type: The user is prompted to enter the skin type of the animal (fur, rough\_skin, smooth\_skin, feathers, or scales). Again, the list of valid animals is updated based on the response.

- Asking for Animal Class: The user is prompted to enter the animal class (mammals, birds, or insects). The list of valid animals is further filtered based on this input.

- Asking for the Number of Legs: Finally, the user is asked about the number of legs of the animal. This further narrows down the list of valid animals.

3. Filtering Valid Animals

The `filter\_valid\_animals` predicate takes a list of animals, a feature, and filters the list to include only animals that have that feature. It's used at each step to update the list of valid animals based on the user's responses.

4. Asking for Feature Values

The `ask\_feature\_value` predicate is responsible for prompting the user for specific feature values, such as diet, skin type, animal class, and the number of legs. It then reads the user's input and stores it in the respective variables.

5. Checking Animal Features

The `animal\_has\_feature` predicate checks whether a given animal has a specific feature. It is used in the `filter\_valid\_animals` predicate to determine if a particular animal matches the user's input.

6. Displaying Results

The program displays the initial list of valid animals, the list of valid animals after each feature question, and the final list of valid animals based on all the user's responses.

7. Game Flow

The program iteratively narrows down the list of possible animals based on the user's input, attempting to identify the correct animal. After the final question, it presents the best guess based on the remaining valid animals.

8. Running the Program

To play the game, you can run the `main` predicate in your Prolog environment. The program will interact with the user and provide a final list of animals that match the specified features, making an educated guess about the animal's identity.

Overall, this Prolog program serves as a simple expert system for animal identification and demonstrates the use of rule-based reasoning in Prolog.

**Question 2**

Patients seek medical help for determination (or diagnosis) and treatment of various health problems. Sometimes a combination of the patient’s history and a clinical examination by a physician are enough to make diagnoses and decide whether medical treatment is needed, and what treatment should be given. However, often laboratory investigations or diagnostic imaging procedures are required to confirm a clinically suspected diagnosis or to obtain more accurate information.

For example, malaria may be suspected by the presence of fever and by excluding other causes of fever on history and physical examination, but a firm diagnosis is made on microscopic examination of a blood slide.

Implement an Expert System for medical disease diagnosis facilities with several human diseases using methodology of rule-based systems; Knowledge-based systems; Intelligent agent (IA); Database methodology; Inference engine; and System-user interaction.

You can use Python and Prolog language as per your requirements.

Source Code:-

assignment\_part2.py

| **from** pyswip **import** Prolog  *# Initialize Prolog* prolog = Prolog()  *# Load your Prolog code file (assuming the Prolog code is saved in a file named 'diagnosis.pl')* prolog.consult('medical\_data.pl')  number\_of\_times\_diagnosis = 2  **for** i **in** range(number\_of\_times\_diagnosis):   *# Run the diagnosis*  **for** result **in** prolog.query("start\_diagnosis."):  **pass** |
| --- |

medical\_data.pl

| *% Define symptoms for various diseases* symptom(fever). symptom(cough). symptom(headache). symptom(rash). symptom(fatigue). symptom(joint\_pain). symptom(shortness\_of\_breath). symptom(chest\_pain). symptom(sore\_throat). symptom(abdominal\_pain). symptom(vomiting). symptom(diarrhea). symptom(nausea). symptom(blurred\_vision). symptom(frequent\_urination). symptom(weight\_loss). symptom(swelling). symptom(persistent\_cough). symptom(bruising). symptom(blood\_in\_urine).  *% Define diseases and their associated symptoms* disease(common\_cold, [fever, cough, headache]). disease(influenza, [fever, cough, fatigue, sore\_throat]). disease(measles, [fever, cough, rash]). disease(asthma, [shortness\_of\_breath, chest\_pain, cough]). disease(bronchitis, [cough, fatigue, chest\_pain]). disease(strep\_throat, [fever, sore\_throat]). disease(pneumonia, [fever, cough, shortness\_of\_breath]). disease(appendicitis, [abdominal\_pain, vomiting, nausea]). disease(kidney\_stones, [abdominal\_pain, blood\_in\_urine, nausea]). disease(food\_poisoning, [vomiting, diarrhea, abdominal\_pain]). disease(diabetes, [frequent\_urination, fatigue, blurred\_vision]). disease(hypertension, [headache, blurred\_vision, fatigue, chest\_pain]). disease(lymphoma, [weight\_loss, swelling, persistent\_cough, fever]). disease(leukemia, [fatigue, bruising, weight\_loss, fever]).  *% Rules for diagnosis* diagnose(Symptoms, Disease, MatchedSymptoms) :-  disease(Disease, RequiredSymptoms),  intersection(RequiredSymptoms, Symptoms, CommonSymptoms),  length(CommonSymptoms, MatchedSymptoms).  *% Find the disease with the most matched symptoms* find\_most\_likely\_disease(Symptoms, MostLikelyDisease) :-  findall(MatchedSymptoms-Disease, diagnose(Symptoms, Disease, MatchedSymptoms), DiseaseMatches),  max\_member(Max-MostLikelyDisease, DiseaseMatches),  Max >= 2.  *% Main diagnosis function* diagnose\_disease :-  write('Enter your symptoms (space-separated, e.g., fever cough fatigue): '),  read\_line\_to\_string(user\_input, SymptomsInput),  atomic\_list\_concat(SymptomsList, ' ', SymptomsInput),  list\_to\_set(SymptomsList, Symptoms),  (find\_most\_likely\_disease(Symptoms, MostLikelyDisease) ->  format('Based on your symptoms, you may have: ~w~n', [MostLikelyDisease]);  write('Based on your symptoms, no specific disease could be determined.'), nl  ).  *% Interaction with the user* start\_diagnosis :-  write('Welcome to the Medical Diagnosis Expert System.'), nl,  write('Let\'s diagnose your disease based on your symptoms.'), nl,  diagnose\_disease. |
| --- |

**Sample Input Output**

| python3 assignment\_part2.py  Welcome **to** **the** Medical Diagnosis Expert System. Let's diagnose your disease based **on** your symptoms. Enter your symptoms (space-separated, e.g., fever cough fatigue): fever cough Based **on** your symptoms, you may have: pneumonia Welcome **to** **the** Medical Diagnosis Expert System. Let's diagnose your disease based **on** your symptoms. Enter your symptoms (space-separated, e.g., fever cough fatigue): fever fatigue  Based **on** your symptoms, you may have: leukemia |
| --- |

**Explanation**

Introduction

The provided code is an implementation of a medical disease diagnosis expert system. It uses rule-based systems and knowledge-based systems to diagnose diseases based on a set of symptoms. The system interacts with the user to input symptoms and then suggests potential diseases based on those symptoms.

System Components

The expert system is implemented in two files: `assignment\_part2.py` (Python) and `medical\_data.pl` (Prolog). Below are the key components and how they work together:

1. Python Script (`assignment\_part2.py`)

- This Python script acts as the main interface for user interaction and manages the interaction with the Prolog knowledge base.

- It uses the `pyswip` library to interface with the Prolog knowledge base.

- The main functionality of the Python script is to ask the user for symptoms, query the Prolog knowledge base for a diagnosis, and display the result.

2. Prolog Knowledge Base (`medical\_data.pl`)

- The Prolog knowledge base defines various components of the expert system, such as symptoms, diseases, and rules for diagnosis.

- It includes a list of symptoms that can be associated with different diseases.

- It defines diseases and their associated symptoms.

- Rules for diagnosis are also defined, including how to determine if a set of symptoms matches a disease.

- The main diagnosis function in Prolog finds the disease with the most matched symptoms.

- The Prolog code is consulted by the Python script to access the knowledge base.

How to Run the System

1. Ensure you have both Python and SWI-Prolog installed on your system.

2. Run the Python script `assignment\_part2.py` to start the expert system.

3. The system will welcome the user and prompt for symptoms (e.g., "fever cough fatigue").

4. Enter a space-separated list of symptoms and press Enter.

5. The system will query the knowledge base to identify the most likely disease based on the entered symptoms.

6. The result will be displayed, suggesting a potential disease.

Example Usage

- User Input: "fever cough fatigue"

- System Output: "Based on your symptoms, you may have: influenza"

Code Structure

The Python script handles user interaction and integration with the Prolog knowledge base. The Prolog knowledge base defines the symptoms, diseases, and rules for diagnosis.

- `symptom`: List of common symptoms.

- `disease`: Defines diseases and their associated symptoms.

- `diagnose`: A rule to determine if a set of symptoms matches a disease.

- `find\_most\_likely\_disease`: Finds the disease with the most matched symptoms.

- `diagnose\_disease`: Main function for user interaction and diagnosis.

- `start\_diagnosis`: Initiates the diagnosis process.

Extending the System

To extend the system, you can modify the `medical\_data.pl` file to add more diseases and their associated symptoms. You can also add more complex rules for diagnosis.

The Python script can be enhanced by providing a more user-friendly interface and error handling for input.

Conclusion

This expert system combines Python and Prolog to diagnose diseases based on symptoms. It demonstrates the use of rule-based and knowledge-based systems for medical diagnosis. The system can be extended and improved to handle a broader range of diseases and symptoms, making it a valuable tool for medical professionals and patients.