EXPERIMENT-5

Name: Adwait Purao
UID: 2021300101
Div: B
Batch: B2

Problem Definition: To design and implement an intelligent system, incorporating the matching algorithm and the rule language. (Disease diagnosis & knowledge generation)

1. It should provide a fact base updating function.
2. It should provide a function that checks the rules' LHS and return which rules were matched.
3. It should support firing RHS according to matches.

Using SWISH Prolog

Theory:

1. **Knowledge representation:**

   Knowledge representation is the process of representing the knowledge of a system in a way that can be understood and processed by the system. In SWISH Prolog, knowledge is represented using predicates. Predicates are statements that can be either true or false. For example, the following predicate represents the fact that the patient has a fever:

   patient(fever, yes).

   Predicates can be combined to form more complex statements. For example, the following rule represents the knowledge that a patient has the flu if they have a fever and a cough:

   disease(flu) :- patient(fever, yes), patient(cough, yes).

2. **Matching algorithm:**

   The matching algorithm is used to determine which rules match a given fact base. In SWISH Prolog, the matching algorithm is based on unification. Unification is the process of finding values for variables that make two predicates equal.

   For example, the following two predicates unify:

   patient(fever, yes).
   patient(fever, X).

   The variable X is unified with the value yes. This means that the two predicates are now equal.

   The matching algorithm uses unification to find all rules whose LHS matches the given fact base. If a rule's LHS matches the fact base, then the rule is said to be matched.

3. **Rule language:**
   The rule language is used to express the knowledge of the system in the form of rules. Rules are statements that describe how to infer new facts from existing facts.
   In SWISH Prolog, the rule language is based on first-order logic. First-order logic is a powerful language that can be used to express a wide variety of knowledge.

4. **Fact base updating function:**
   The fact base updating function is used to add, remove, or modify facts in the fact base. This is important for keeping the system's knowledge up-to-date and for allowing the system to learn new things.
   In SWISH Prolog, the fact base updating function is based on the assert and retract predicates. The assert predicate is used to add a new fact to the fact base, and the retract predicate is used to remove a fact from the fact base.

5. **Firing RHS according to matches:**
   Once we have determined which rules match a given fact base, we can fire the RHS of those rules. The RHS of a rule is a sequence of actions that are taken when the rule is fired.
   In SWISH Prolog, the firing mechanism is based on the call predicate. The call predicate is used to call a predicate. When a rule is fired, the RHS of the rule is called.

6. **Disease diagnosis:**
   Disease diagnosis is the process of identifying the disease that is causing a patient's symptoms. To diagnose a disease, we need to match the patient's symptoms against the knowledge of the disease.
   In SWISH Prolog, we can use the matching algorithm and the rule language to develop a system for disease diagnosis. The fact base of the system will contain the patient's symptoms, and the rules of the system will contain the knowledge of the disease.

7. **Knowledge generation:**
   Knowledge generation is the process of creating new knowledge from existing knowledge. For example, a system could learn new rules about diseases by analyzing patient data.
   In SWISH Prolog, we can use the fact base updating function to generate new knowledge. For example, we could create a new rule about a disease by adding a new fact to the fact base that describes the symptoms of the disease.

Steps in the program:

- **Step 1:** Knowledge base initialization: The program starts by initializing the knowledge base. This involves loading the symptom database and the rule language.
- **Step 2 :** Fact base updating: The patient's symptoms are added to the fact base using the add_symptom() function.
- **Step 3 :** Rule matching: The system matches the patient's symptoms against the rules in the rule language using the match_rule() function.
- **Step 4 :** Rule firing: The system fires the matched rules using the diagnose() function.
- **Step 5 :** Disease diagnosis: The system diagnoses the disease based on the fired rules.

Example:

Step 1: Knowledge base initialization

The first step is to initialize the knowledge base. This involves loading the symptom database and the rule language.

The symptom database contains a list of all the symptoms that the system can diagnose. It might look something like this:

symptom(fever, yes).
symptom(body_aches, yes).
symptom(fatigue, yes).

The rule language contains a set of rules that describe how to diagnose diseases based on the patient's symptoms. It might look something like this:

diagnosis(Patient, 'Flu') :- symptom(Patient, fever), symptom(Patient, body_aches), symptom(Patient, fatigue).

This rule states that if the patient has a fever, body aches, and fatigue, then the patient is diagnosed with the flu.

Step 2: Fact base updating

The next step is to update the fact base with the patient's symptoms. This is done using the add_symptom() function.

add_symptom(John Doe, fever).
add_symptom(John Doe, body_aches).
add_symptom(John Doe, fatigue).

This will add the following facts to the fact base:

symptom(John Doe, fever, yes).

symptom(John Doe, body_aches, yes).
symptom(John Doe, fatigue, yes).

Step 3: Rule matching
The next step is to match the patient's symptoms against the rules in the rule language. This is done using the match_rule() function.
match_rule('Flu', John Doe).
This will return true because the patient's symptoms match the rules for the flu.

Step 4: Rule firing
The next step is to fire the matched rules. This is done using the diagnose() function.
diagnose('Flu', John Doe).
This will print the following message to the console:
John Doe is diagnosed with Flu

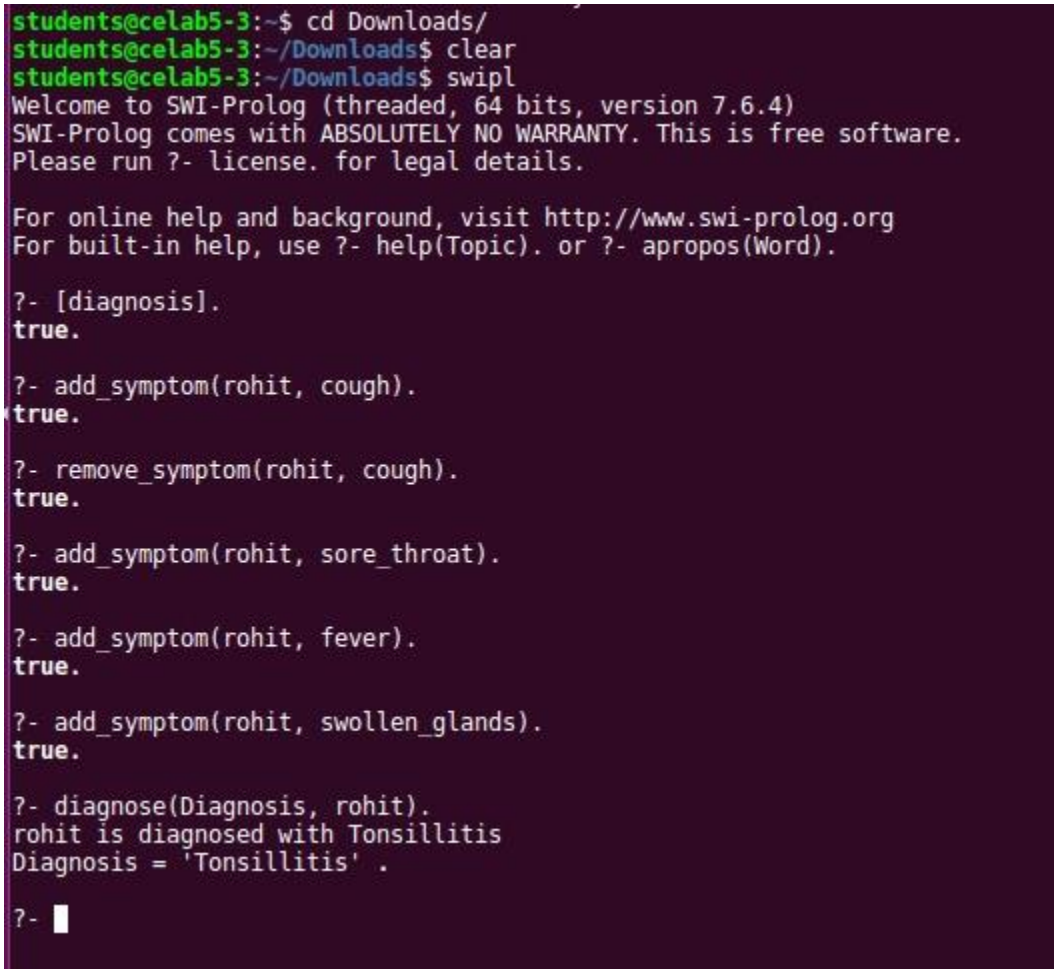
Step 5: Disease diagnosis
The final step is to diagnose the disease based on the fired rules. In this case, the only rule that fired was the rule for the flu, so the patient is diagnosed with the flu.

Code:

```
% Knowledge Base
% symptom(name, symptom).

% Rule Language (Disease Diagnosis)
diagnosis(X, 'Common Cold') :- symptom(X, runny_nose), symptom(X, sore_throat),
symptom(X, sneezing).
diagnosis(X, 'Flu') :- symptom(X, fever), symptom(X, body_aches), symptom(X,
fatigue).
diagnosis(X, 'Migraine') :- symptom(X, headache), symptom(X,
sensitivity_to_light).
diagnosis(X, 'Strep Throat') :- symptom(X, sore_throat), symptom(X, fever),
symptom(X, difficulty_swallowing).
diagnosis(X, 'Bronchitis') :- symptom(X, cough), symptom(X, shortness_of_breath),
symptom(X, chest_discomfort).
diagnosis(X, 'Tonsillitis') :- symptom(X, sore_throat), symptom(X,
swollen_glands), symptom(X, fever).
diagnosis(X, 'Respiratory Infection') :- symptom(X, cough), symptom(X,
difficulty_breathing), symptom(X, wheezing).

% Fact Base Updating Functions (Unchanged)
add_symptom(Patient, Symptom) :- assert(symptom(Patient, Symptom)).
remove_symptom(Patient, Symptom) :- retract(symptom(Patient, Symptom)).
```

```
% Matching Rules
match_rule(Diagnosis, Patient) :- diagnosis(Patient, Diagnosis).

% Firing Rules
diagnose(Diagnosis, Patient) :- match_rule(Diagnosis, Patient), write(Patient),
write(' is diagnosed with '), write(Diagnosis), nl.
```

Output:



Analysis of the Code:

- **Completeness:** The matching algorithm used in the code is complete, meaning that it is guaranteed to find all possible matches.
- **Optimality:** The matching algorithm used in the code is a simple but effective algorithm that uses unification to find all rules whose LHS matches the fact base. However, it is not guaranteed to find the best possible matches, especially if the rule language is complex.
- **Complexity:** The complexity of the matching algorithm depends on the number of symptoms and diseases in the knowledge base and the complexity of the rules in the rule language. For small knowledge bases and simple rules, the matching algorithm is very efficient. However, for large knowledge bases and complex rules, the matching algorithm can be computationally expensive.

Conclusion:

A disease diagnosis and knowledge generation system is an intelligent system that can be used to diagnose diseases and generate knowledge about diseases. The system incorporates a matching algorithm and a rule language to match the patient's symptoms against the rules in the knowledge base and to diagnose the disease based on the matched rules.

What I learned from the experiment:

- How to design and implement a disease diagnosis and knowledge generation system using SWISH Prolog
- The importance of using a fact base updating function to keep the knowledge base up-to-date
- The importance of using a matching algorithm to match the patient's symptoms against the rules in the knowledge base
- The importance of using a rule language to express the knowledge of the system in the form of rules
- The importance of using a firing RHS function to fire the rules that match the patient's symptoms