**Batch: C - 3 Roll No.: 16010122096**

**Experiment / assignment / tutorial No. 2**

|  |
| --- |
| **Title: Implementation of condition-action rules based agent using PROLOG** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** Developing a basic level agent program that runs on condition-action rules

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **Course Outcome** | **After successful completion of the course students should be able to** |
| **CO1** | Understand the history & various application of AI and choose appropriate agent architecture to solve the given problem. |
| **CO3** | Represent and formulate the knowledge to solve the problems using various reasoning techniques |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **https://www.csupomona.edu/~jrfisher/www/prolog\_tutorial/contents.html**
2. **http://www.csupomona.edu/~jrfisher/www/prolog\_tutorial/pt\_framer.html**
3. **http://www.doc.gold.ac.uk/~mas02gw/prolog\_tutorial/prologpages/**
4. **“Artificial Intelligence: a Modern Approach” by Russell and Nerving, Pearson education Publications**
5. **“Artificial Intelligence” By Rich and knight, Tata McGraw Hill Publications**
6. **“Prolog: Programming for Artificial Intelligence” by Ivan Bratko, Pearson education Publications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:** Intelligent Agent, Agent Architectures, Rule base Vs Knowledgebase approach

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:** Agent programs for simple applications need not be very complicated. They can be based on condition-action rules and still they give better results, though not always rational. The family tree program makes use of similar concept.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:**

Defining rules, using and programming with PROLOG

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

A simple agent program can be defined mathematically as an agent function which maps every possible percepts sequence to a possible action the agent can perform or to a coefficient, feedback element, function or constant that affects eventual actions:

*F*: *P* \* − >*A*

**Algorithm for ‘Condition-Action Rule Table’ Agent function:**

**function**SIMPLE-REFLEX-AGENT (percept) **returns** an action

**Static:** *rules,* a set of condition-action rules

*State*:- INTERPRET-INPUT (percept)

*Rule*:- *RULE-MATCH (state, rules)*

*Action*:- *RULE-ACTION [rule]*

**Returnaction**

This approach follows a table for lookup of condition-action pairs defining all possible condition-action rules necessary to interact in an environment.

**Example Family Tree/disease-symptom mapping/ City map with their distances between them:**

**Problem Statements: (Choose Any 1)**

**1) Family tree problem:**

Write a prolog program model and query a family tree. The family tree should represent familial relationships. It should allow users to query and infer relationships using defined rules. The example family tree taken should have enough breadth and depth (number of children in one generation and number of generations)

**2) Medieval Kingdom problem:**

Write a program to model a medieval kingdom, including its hierarchy, territories, alliances, conflicts, economy, and trade. Write a` program in Prolog to represent all these relationships.

**3) Medical Diagnosis problem:**

Write a prolog program to represent common medical conditions, symptoms, diagnostic tests, and treatment protocols. The system will be capable of suggesting possible diagnoses, generating a list of potential conditions, and offering treatment options based on the available data.

**Knowledgebase:**

**% Conditions**

condition(flu).

condition(cold).

condition(covid19).

condition(asthma).

condition(diabetes).

**% Symptoms for Flu**

symptom(flu, fever).

symptom(flu, cough).

symptom(flu, body\_ache).

**% Symptoms for Cold**

symptom(cold, cough).

symptom(cold, runny\_nose).

**% Symptoms for COVID-19**

symptom(covid19, fever).

symptom(covid19, cough).

symptom(covid19, loss\_of\_taste\_smell).

**% Symptoms for Asthma**

symptom(asthma, shortness\_of\_breath).

symptom(asthma, wheezing).

symptom(asthma, coughing).

**% Symptoms for Diabetes**

symptom(diabetes, excessive\_thirst).

symptom(diabetes, frequent\_urination).

symptom(diabetes, fatigue).

**% Diagnostic Tests**

test(flu, rapid\_flu\_test).

test(cold, none).

test(covid19, pcr\_test).

test(asthma, spirometry).

test(asthma, chest\_xray).

test(diabetes, blood\_sugar\_test).

**% Treatments**

treatment(flu, rest\_and\_fluids).

treatment(flu, antiviral\_medication).

treatment(cold, rest\_and\_fluids).

treatment(covid19, isolation).

treatment(covid19, antiviral\_medication).

treatment(asthma, inhalers).

treatment(asthma, corticosteroids).

treatment(diabetes, insulin).

treatment(diabetes, lifestyle\_changes).

**Rules:**

**% Recursive Rule to Process a List of Symptoms**

diagnose\_symptoms([], []). % Base case: Empty list of symptoms, no conditions.

diagnose\_symptoms([Symptom | RestSymptoms], Conditions) :-

findall(Condition, symptom(Condition, Symptom), MatchingConditions),

diagnose\_symptoms(RestSymptoms, OtherConditions),

append(MatchingConditions, OtherConditions, AllConditions),

sort(AllConditions, Conditions). % Remove duplicates and sort.

**% Recursive Rule to Suggest Tests for a List of Conditions**

suggest\_tests([], []). % Base case: No conditions, no tests.

suggest\_tests([Condition | RestConditions], Tests) :-

findall(Test, test(Condition, Test), ConditionTests),

suggest\_tests(RestConditions, OtherTests),

append(ConditionTests, OtherTests, Tests).

**% Recursive Rule to Suggest Treatments for a List of Conditions**

suggest\_treatments([], []). % Base case: No conditions, no treatments.

suggest\_treatments([Condition | RestConditions], Treatments) :-

findall(Treatment, treatment(Condition, Treatment), ConditionTreatments),

suggest\_treatments(RestConditions, OtherTreatments),

append(ConditionTreatments, OtherTreatments, Treatments).

**% Main Diagnosis Flow**

diagnosis\_flow\_recursive(Symptoms) :-

write('Analyzing symptoms: '), write(Symptoms), nl,

diagnose\_symptoms(Symptoms, Conditions),

write('Possible conditions: '), write(Conditions), nl,

suggest\_tests(Conditions, Tests),

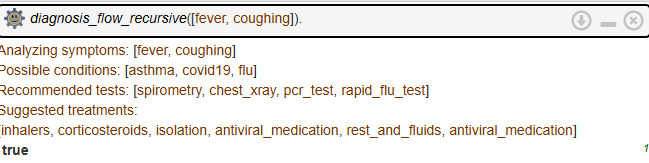
write('Recommended tests: '), write(Tests), nl,

suggest\_treatments(Conditions, Treatments),

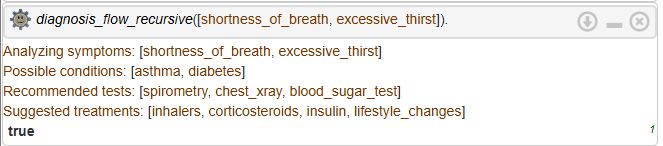
write('Suggested treatments: '), write(Treatments), nl.

**Some Sample queries and Outputs:**

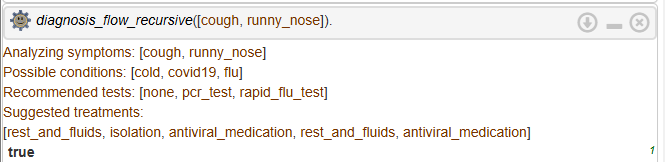
diagnosis\_flow\_recursive([fever, coughing]).

****

diagnosis\_flow\_recursive([shortness\_of\_breath, excessive\_thirst]).

****

diagnosis\_flow\_recursive([cough, runny\_nose]).

****

diagnose\_symptoms([rash], Conditions).



suggest\_tests([common\_cold], Tests).



symptom(diabetes, fever).

****

test(flu, spirometry).

****

**Post Lab Objective Questions**

1. **The PROLOG suit is based on**
   1. Interpreter
   2. Compiler
   3. None of the above

**Answer: a] Interpreter**

1. **State true of false**

There must be at least one fact pertaining to each predicate written in the PROLOG program.

**Answer: False**

1. **State true of false**

In PROLOG program the variable declaration is a compulsory part.

**Answer: False**

**Post Lab Subjective Questions**

1. **Differentiate between a fact and a predicate with syntax.**

 **Fact**: A fact in PROLOG represents a basic assertion or statement that is unconditionally true. It is a fundamental building block of a PROLOG knowledge base.

* **Syntax**: fact\_name(argument1, argument2, ...).  
  Example: parent(john, mary). This fact states that "John is a parent of Mary."

 **Predicate**: A predicate represents a relation or property that can be true or false based on the facts or rules in the knowledge base. A predicate is a more general term, and it may or may not have facts associated with it.

* **Syntax**: predicate\_name(argument1, argument2, ...). Example: parent(X, Y). This is a predicate that expresses the relationship between X and Y being a parent-child relationship.

1. **Differentiate between knowledgebase and Rule base approach.**

 **Knowledge Base**: The knowledge base in PROLOG contains facts and rules, which define the information that is known about the domain being modeled. The facts are unconditional assertions, while the rules define how facts relate to each other.

* Example: parent(X, Y) :- father(X, Y).

 **Rule Base**: A rule base consists specifically of rules (or logical implications) that define relationships and infer new facts from existing facts. Rules are more focused on how knowledge is derived, while the knowledge base stores both the knowledge and the inference mechanisms.

* Example: father(X, Y) :- man(X), parent(X, Y).

1. **Differentiate between database and knowledgebase.**

 **Database**: A database is a structured collection of data stored in a systematic manner, typically in tables or records. The data in a database is often static and does not include inferencing or reasoning. A database is primarily used to store and retrieve information.

* Example: A relational database with tables like employees, departments, etc.

 **Knowledge Base**: A knowledge base is a collection of structured facts and rules designed for reasoning and inference. It is not only about storing information but also about drawing conclusions and making decisions based on that information.

* Example: In AI or expert systems, a knowledge base contains facts and rules that help the system answer questions or solve problems by reasoning.

1. **What is a ‘free variable’? Explain with an example.**

A **free variable** is a variable in a logical expression (such as a rule or a query in PROLOG) that is not bound to any specific value. It is not assigned a value within the scope of a fact or rule and can represent any value within a query or goal.

* **Example**: In the rule parent(X, Y) :- mother(X, Y)., if X and Y are not bound (i.e., not assigned any specific value), they are free variables.
* In a query, for example, parent(john, X)., X is a free variable because it's not bound to any value, and PROLOG will try to find all X that satisfy the rule or facts for the query.