K. J. Somaiya College of Engineering, Mumbai-77

Batch: A-4 **Roll No.:** 16010122151
Experiment / assignment / tutorial No. 2
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

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:

functionSIMPLE-REFLEX-AGENT (percept) **returns** an action **Static:** *rules,* a set of condition-action rules *State*:- nINTERPRET-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:

```
% Facts
parent(john, mary).
parent(john, james).
parent(mary, sophia).

male(john).
female(mary).
female(sophia).

% Rules
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.
```

```
% Queries
?- father(X, mary).
?- sibling(mary, Sibling).
```

Base Knowledgebase:

```
% Hierarchy
kingdom(kingdom_of_valeria).
ruler(king, valerian, kingdom_of_valeria).
ruler(queen, elenora, kingdom_of_valeria).

noble(duke, alric, kingdom_of_valeria).
noble(duchess, lysandra, kingdom_of_valeria).
noble(baron, edric, kingdom_of_valeria).

% Territories
territory(kingdom_of_valeria, northshire).
territory(kingdom_of_valeria, eastwood).
territory(kingdom_of_valeria, southport).

% Alliances
alliance(kingdom_of_valeria, kingdom_of_altheris).
alliance(kingdom_of_valeria, kingdom_of_eldoria).

% Conflicts
conflict(kingdom_of_valeria, kingdom_of_darkmoor).
conflict(kingdom_of_valeria, kingdom_of_noor).

% Economy
trade_goods(northshire, wheat).
trade_goods(eastwood, timber).
trade_goods(southport, fish).

% Trade
trade_route(northshire, eastwood, wheat, timber).
trade_route(southport, northshire, fish, wheat).
```

Rules:

```
% Rules

% Check if a person is a ruler of a given kingdom.
is_ruler(Title, Name, Kingdom) :-
    ruler(Title, Name, Kingdom).


% Check if a person is a noble of a given kingdom.
is_noble(Name, Kingdom) :-
    noble(_, Name, Kingdom).

% Check if two kingdoms are allies.
are_allies(Kingdom1, Kingdom2) :-
    alliance(Kingdom1, Kingdom2);
    alliance(Kingdom2, Kingdom1).

% Check if two kingdoms are enemies.
are_enemies(Kingdom1, Kingdom2) :-
    conflict(Kingdom1, Kingdom2);
    conflict(Kingdom2, Kingdom1).

% Check if a territory belongs to a given kingdom.
belongs_to_kingdom(Territory, Kingdom) :-
    territory(Kingdom, Territory).

% Check if a trade route exists between two territories.
has_trade_route(Source, Destination) :-
    trade_route(Source, Destination, _, _);
    trade_route(Destination, Source, _, _).

% Find goods exported from a specific territory.
exports_goods(Territory, Goods, To) :-
    trade_route(Territory, To, Goods, _).

% Find goods imported into a specific territory.
imports_goods(Territory, Goods, From) :-
    trade_route(From, Territory, _, Goods).

% List all territories producing a specific resource.
```

produces(Territory, Resource) :-
trade_goods(Territory, Resource).

Some Sample queries and Outputs:

is_ruler(Title, Name, kingdom_of_valeria).

Name = valerian,
Title = king

Name = elenora,
Title = queen

```
?- is_ruler(Title, Name, kingdom_of_valeria).
```

are_allies(kingdom_of_valeria, Ally).

Ally = kingdom_of_altheris

Ally = kingdom_of_eldoria

```
?- are_allies(kingdom_of_valeria, Ally).
```

```
are_enemies(kingdom_of_valeria, Enemy).
Enemy = kingdom_of_darkmoor
Enemy = kingdom_of_noor

?- are_enemies(kingdom_of_valeria, Enemy).
```

.

```
exports_goods(northshire, Goods, To).
Goods = wheat,
To = eastwood

?- exports_goods(northshire, Goods, To).
```

```
imports_goods(eastwood, Goods, From).
From = northshire,
Goods = timber

?- imports_goods(eastwood, Goods, From).
```

```
produces(Territory, fish).

Territory = southport

?- produces(Territory, fish).
```

Post Lab Objective Questions

1. **The PROLOG suit is based on**
    - **a.** Interpreter
    - **b.** Compiler
    - c. None of the above

Answer: Interpreter(A)

2. **State true of false**
   There must be at least one fact pertaining to each predicate written in the PROLOG program.

Answer: False

3. **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	Predicate
A fact is a basic assertion that states something unconditionally true in the system.	A predicate is used to express a relationship or function between objects and can be true or false based on the context.
<predicate>(<arguments>).	<predicate>(<arguments>) :- <conditions>.
likes(john, pizza).	likes(john, X) :- food(X), tasty(X).

2. Differentiate between knowledgebase and Rule base approach.

Knowledgebase Approach	Rule-based Approach
Stores facts and rules about the domain in a declarative form.	Focuses on deriving new facts or decisions by applying logical rules to known facts.
Knowledge representation.	Logical inference and reasoning.
Facts: is_a(dog, mammal).	Rule: mammal (X) :- is_a(X, mammal).
Used to organize and represent information about the system.	Used to infer new knowledge or automate decision-making.

3. Differentiate between database and knowledgebase.

Database	Knowledgebase
A structured collection of data, typically stored in tables.	A system designed to store, retrieve, and manipulate knowledge represented in a symbolic form.
Data storage and retrieval.	Logical reasoning and inference.
Relational tables, records, and fields.	Rules, facts, and logic.
Table: Student(Name, Age, Grade).	Fact: student(john). Rule: passed(X) :- student(A) , grade (X , A).

4. What is a 'free variable'? Explain with an example.

A free variable in logic programming (like in PROLOG) is a variable that is not yet bound to any specific value or object. It can represent any value and gets instantiated during execution or unification.

example : likes(john, X).

Here, X is a free variable. It can match any value (e.g., pizza, burger) depending on the available facts in the knowledge base.

If the fact likes(john, pizza). exists, querying likes(john, X). will instantiate X to pizza.