Yachay Tech University

Mathematical and Computational Logic

Prolog Lab 3: AI Logic Playground – Intelligent Maze Advisor

1. Introduction

This lab brings together the main Prolog concepts learned so far: facts, rules, recursion, lists, graph traversal, and reasoning. The objective is to build an intelligent system that can navigate a maze, find a path, and explain its reasoning in human-readable form.

2. Objectives

- Represent a maze as a graph.
- Use reasoning rules to decide moves.
- Apply recursion to find a path.
- Generate explanations as the program runs.

3. Background

3.1 Graphs in Prolog

A maze can be modeled as a graph where rooms are nodes and doors are edges.

Example:

```
edge(entrance, a).
edge(a, b).
edge(a, c).
edge(b, exit).
edge(c, b).
```

3.2 Reasoning Rules

Rules allow us to infer actions or preferences. For example:

```
can_move(X, Y) :- edge(X, Y), \+ blocked(X, Y).
```

4. Lab Setup

Create a new file named `maze_advisor.pl` and add the following base facts:

```
edge(entrance, a).
edge(a, b).
edge(a, c).
edge(b, exit).
```

```
edge(c, b).
blocked(a, c).  % Door is blocked from a to c.
```

5. Reasoning Predicates

Define movement reasoning rules:

```
can_move(X, Y) :- edge(X, Y), \+ blocked(X, Y).
reason(X, Y, 'path is open') :- can_move(X, Y).
reason(X, Y, 'path is blocked') :- blocked(X, Y).
```

6. Recursive Traversal with Explanation

We now extend traversal to print reasoning at each step:

```
move(X, Y, Visited, [Y|Visited]) :-
   can_move(X, Y),
   format('Moving from ~w to ~w.~n', [X, Y]).
move(X, Y, Visited, Path) :-
   can_move(X, Z),
   \+ member(Z, Visited),
   format('Exploring from ~w to ~w...~n', [X, Z]),
   move(Z, Y, [Z|Visited], Path).
```

7. Main Predicate

```
find_path(X, Y, Path) :-
   move(X, Y, [X], RevPath),
   reverse(RevPath, Path).
```

8. Example Execution

```
?- find_path(entrance, exit, Path).
Moving from the entrance to a.
Exploring from a to b...
Moving from b to exit.
Path = [entrance, a, b, exit].
```

9. Lab Exercises

Part 1 – Maze Representation

1. Represent the maze as a graph using edge/2 facts.
2. Add some blocked paths using blocked/2 facts.

3. Test with queries like:
   ?- can_move(a, b).
   ?- can_move(a, c).

Part 2 – Reasoning Rules

1. Define rules for can_move/2 and reason/3.
2. Extend reason/3 with additional explanations such as:
   reason(X, Y, 'destination reached') :- Y == exit.

Part 3 – Recursive Traversal

1. Implement move/4 using recursion and format/2 for explanations.
2. Ensure you prevent loops using a visited list.
3. Test your implementation with various start and end nodes.

Part 4 – Putting It All Together

1. Combine traversal and reasoning into find_path/3.
2. Execute your program with queries like:
   ?- find_path(entrance, exit, Path).
3. Observe the printed reasoning trace.

Part 5 – Extensions (Optional)

1. Add reasoning for alternative decisions, e.g., prefer left paths.
2. Add a predicate why(X, Y) that explains the reasoning.
3. Add performance tracking (number of steps, visited nodes).

10. Deliverables

- A .pl file with your full solution.
- Screenshot showing at least one successful reasoning trace.
- One paragraph (in comments) summarizing how reasoning was implemented.

- All of these deliverables should be in your GitHub repository for the class

11. Summary

In this lab, you applied all major Prolog concepts learned so far to build an intelligent agent. You used recursion for traversal, logical rules for reasoning, and predicates to explain behavior — a complete example of symbolic AI in action.