Artificial Intelligence

Lab Sheet No: 1

Introduction to Prolog

IOE, Thapathali Campus

December, 2025

Introduction

Prolog is a very important tool in the programming of artificial intelligence applications and in the development of expert systems. By allowing the programmer to model the logical relationships among objects and processes, complex problems are inherently easier to solve, and the resulting program is easier to maintain through its lifecycle. With Visual Prolog, applications such as customized knowledge bases, expert systems, natural language interfaces, and smart information management systems are easy to develop.

Prolog is what is known as a declarative language. This means that given the necessary facts and rules, Prolog will use deductive reasoning to solve the programming problems. This is in contrast to traditional computer languages, such as C, BASIC and Pascal, which are procedural languages. We can also use prolog as any other programming language in a procedural manner.

So prolog can be viewed as a tool to solve problems in the field of artificial intelligence or it can be very well used as a general programming language.

Prolog enforces a different problem solving paradigm complementary to traditional programming languages so it is believed that a student of computer science should learn programming in prolog.

Along with prolog, we are going to introduce the python based syntax for the different applications where necessary in this laboratory work.

Data Types in Prolog

- Atoms and numbers
- Variables
- Structures

Atoms and Numbers

Atoms can be constructed in three different ways:

1. Strings of letters, digits, and the underscore character '_' starting with a lower case letter.

   For example: `man, ram, comp_students, pc_ct_059`

2. Strings of special characters

   For example: `<-------->, ::::::::::::`

   Care should be taken not to use the character combination that may have some built in meaning.

3. Strings of characters enclosed in quotes

   For example: `'Ram'`, `'Bird'`

Numbers used in prolog are integers and real numbers.

Variables

Variables are strings of letters, digits and underscore that start with an underscore or an upper-case letter. The scope of a variable is one clause only. So the same variable used in different clauses means different thing.

For example: `X, Ram, _weight`, etc.

Note: Here that Ram is a variable unlike the earlier use 'Ram' where it was a constant, an atom.

An underscore '_' also known as anonymous variable is used in clauses when a variable need not be inferred more than once.

Structures

Structures are objects that have different components. The components can be atoms or yet some other structures. A functor is used to construct a structure as follows:

```
family(father, mother, children)
```

Here family is a structure that has father, mother and the children as its elements. The father and mother may be atoms while the children may be yet another structure or a list of atoms. List is a special built-in structure in prolog.

Lists

A list is a built-in structure in prolog. It can be thought of as a sequence of elements ordered linearly however it is internally represented as a binary tree.

For example: `[ram, shyam, hari, sita]`

The list as such can be broken down into two parts: the **HEAD** and the **TAIL**. The head is the first element of the list and the tail is the remaining list. The above list can be broken down as:

```
[H|T]
```

Where $H =$ `ram` and $T =$ `[shyam, hari, sita]`
List is one of the most useful structures in prolog.

Writing Programs in Prolog

All prolog programs start from the goal. It then uses the facts and clauses to break down the goal to sub-goals and tries to prove the subgoals. A clause is said to have succeeded if there is a combination of facts and clause(s) that holds true.

Prolog has built-in backtracking mechanism i.e. whenever there are multiple paths, it chooses one, tries to prove it, and comes back to the other choices whether the first one succeeds or fails.

Program 1: A Typical Prolog Program

```
PREDICATES
bigger(integer,integer,integer)

CLAUSES
bigger(X,Y,Z):-
  X>Y,Z=X.
bigger(X,Y,Z):-
  X<Y,Z=Y.

GOAL
bigger(5,7,X).
```

In program 1 we have defined a predicate that gives us the larger of the two given integers. The predicate `bigger` is defined in the PREDICATES section and the relation is defined in the CLAUSES section. It is customary to put together the predicate definition and clause definition for each predicate but it is not necessary. However all the clauses for a predicate have to be written together.

The predicates section defines what types of relations or facts we are going to use. It is somewhat like declaring functions in programming languages like C. The clauses section holds the relation and facts and can be compared to function definition.

Program 2: A Program Combining Facts and Rules

```
PREDICATES
husband(STRING,STRING)
```

```
father(STRING,STRING)
mother(STRING,STRING)
son(STRING,STRING)

CLAUSES
mother("Kaushalya","Ram").
mother("Kaikai","Bharat").
mother("Sumitra","Laxman").
mother("Sumitra","Satrughan").
husband("Dasarath","Kaushalya").
husband("Dasarath","Kaikai").
husband("Dasarath","Sumitra").

son(A,C):-mother(C,A).
son(A,C):-husband(C,B),mother(B,A).
father(A,B):-husband(A,C),mother(C,B).

GOAL
son(X,"Kaikai").
```

Here the predicate `mother` is used to assert facts. There are four facts in that section. Similarly the predicate `husband` is used to assert more facts. The predicates `son` and `father` are used to define rules. We could have defined rules even with mother and husband but we bet that conciseness for program clarity at this stage.

The goal can be checked with `son("Ram",X)` or `father(X,"Ram")`.

Program 3: A Program to Find the Length of a List

```
DOMAINS
int_list=integer*

PREDICATES
length(int_list,integer)

CLAUSES
length([],0).
length([H|T],L):-
  length(T,L1),
  L=L1+1.

GOAL
length([1,2,5,2,1,6,7],X).
```

Program 4: A Program to Read Integers into a List and Display Them

```
DOMAINS
list=integer*

PREDICATES
start
read_a_list(list)
insert(integer,list,list)
display(list)

CLAUSES
start:-
  write("enter␣the␣numbers"),
  nl,
  write("enter␣0␣to␣stop"),nl,
  read_a_list([]).

read_a_list(Y):-
  readint(X),
  insert(X,Y,Z),
  read_a_list(Z).

insert(0,Y,_):-
  write("these␣were␣the␣elements␣you␣inserted"),
  nl,
  write("["),
  display(Y).

insert(X,Y,[X|Y]).

display([ ]):-
  write("␣]"),nl.

display([H|T]):-
  write(H),write("\t"),display(T).

GOAL
start.
```

A Vacuum Cleaner as a Simple Reflex Agent

A simple reflex agent (vacuum cleaner) operates in a two-room environment:

- **Room A** and **Room B**
- Each room can be either **Dirty** (D) or **Clean** (C)
- The agent starts in **Room A**

1 State Space

With 2 rooms and 2 states per room (dirty/clean), we have $2^2 = 4$ possible environment states. Combined with the agent location (Room A or Room B), we get $4 \times 2 = 8$ possible states.

1.1 Complete State List

State ID	Room A	Room B	Agent Location
S_1	D	D	A
S_2	D	D	B
S_3	D	C	A
S_4	D	C	B
S_5	C	D	A
S_6	C	D	B
S_7	C	C	A
S_8	C	C	B

2 Reflex Agent Rules

The simple reflex agent operates using the following rules:

Rule	Action
IF current location is dirty	THEN suck
IF current location is clean	THEN move to next location

3 State Transition Diagram

4 Percept-Action Mapping

The reflex agent makes decisions based on:

$$\text{Action} = f(\text{Current Location, Room Cleanliness}) \tag{1}$$

Current Location	Current Status	Action
Room A	Dirty	Suck
Room A	Clean	Move to Room B
Room B	Dirty	Suck
Room B	Clean	Move to Room A

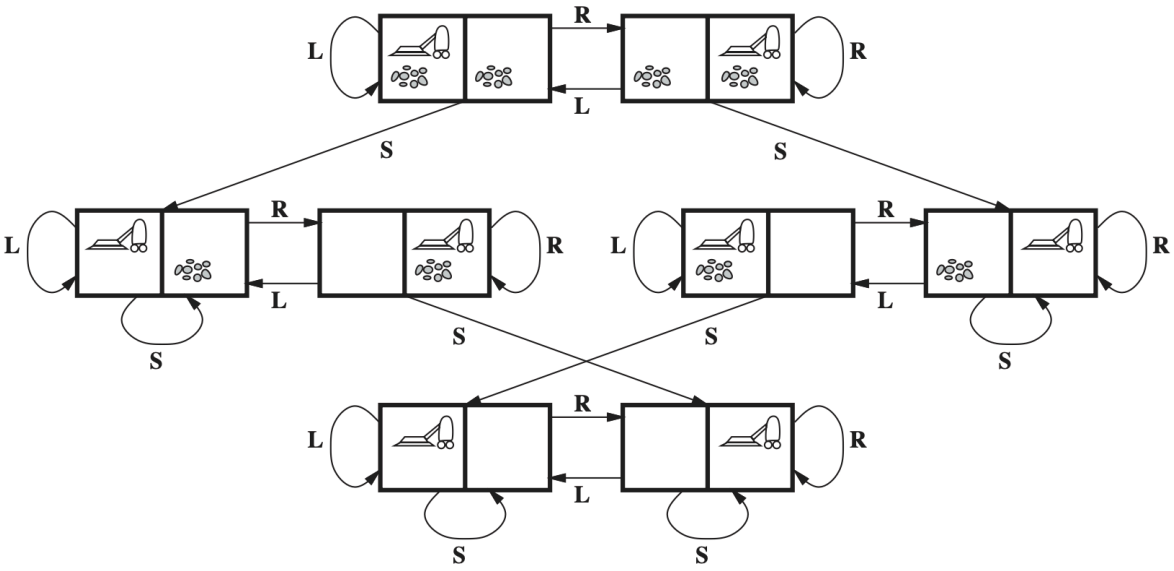

Figure 1: The state space for the vacuum world. Links denote actions: L = Left, R = Right, S = Suck.

5 Example Execution Sequence

Starting from S_1 (Both rooms dirty, agent in Room A):

1. State S_1: Room A is dirty → **Action: Suck** → State S_5
2. State S_5: Room A is clean, Room B is dirty → **Action: Move** → State S_6
3. State S_6: Room B is dirty → **Action: Suck** → State S_8
4. State S_8: Both rooms clean → **Action: Move** → State S_7
5. State S_7: Both rooms clean → **Action: Move** → State S_8
6. (Cycle continues between S_7 and S_8 as environment is fully clean)

6 Characteristics of This Reflex Agent

- **No Memory:** Decisions are based only on current percepts, not history
- **Reactive:** Responds immediately to environment changes
- **Deterministic:** Same percept always produces same action
- **Simple Rules:** Uses basic IF-THEN conditions
- **Limited:** Works well in small, predictable environments
- **Oscillation:** May oscillate between clean rooms when fully clean

Assignments

1. Write a program to find the HCF of two numbers.
   Hint: use a predicate like `hcf(no1, no2, result)`.

2. Write a program of your choice. Give some facts and use some rules to make a few deductions.

3. Write a program to add the content of an integer list and display it.

4. Write a program to find the length of a list.

5. Write a program to append two lists.

6. Write a program which takes a list of integers and displays only 1s and 2s. (If the input is `[1,2,4,5,2,4,5,1,1]` the solution list should be `[1,2,2,1,1]`).

7. Write a program to delete a given element from the list.

8. Generate a python based code to execute the Vacuum Cleaner, A Simple Reflex Agent.