Prolog - Introduction

Prolog as the name itself suggests, is the short form of LOGical PROgramming. It is a logical and declarative programming language. Before diving deep into the concepts of Prolog, let us first understand what exactly logical programming is.

Logic Programming is one of the Computer Programming Paradigm, in which the program statements express the facts and rules about different problems within a system of formal logic. Here, the rules are written in the form of logical clauses, where head and body are present. For example, H is head and B1, B2, B3 are the elements of the body. Now if we state that “H is true, when B1, B2, B3 all are true”, this is a rule. On the other hand, facts are like the rules, but without any body. So, an example of fact is “H is true”.

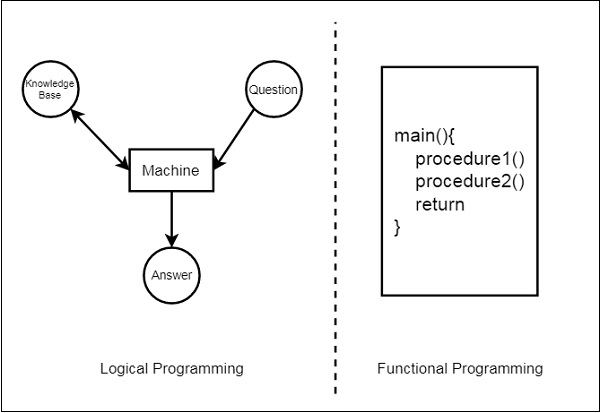
Some logic programming languages like Datalog or ASP (Answer Set Programming) are known as purely declarative languages. These languages allow statements about what the program should accomplish. There is no such step-by-step instruction on how to perform the task. However, other languages like Prolog, have declarative and also imperative properties. This may also include procedural statements like “To solve the problem H, perform B1, B2 and B3”.

Some logic programming languages are given below −

* ALF (algebraic logic functional programming language).
* ASP (Answer Set Programming)
* CycL
* Datalog
* FuzzyCLIPS
* Janus
* Parlog
* Prolog
* Prolog++
* ROOP

## **Logic and Functional Programming**

We will discuss about the differences between Logic programming and the traditional functional programming languages. We can illustrate these two using the below diagram −



From this illustration, we can see that in Functional Programming, we have to define the procedures, and the rule how the procedures work. These procedures work step by step to solve one specific problem based on the algorithm. On the other hand, for the Logic Programming, we will provide knowledge base. Using this knowledge base, the machine can find answers to the given questions, which is totally different from functional programming.

In functional programming, we have to mention how one problem can be solved, but in logic programming we have to specify for which problem we actually want the solution. Then the logic programming automatically finds a suitable solution that will help us solve that specific problem.

Now let us see some more differences below −

|  |  |
| --- | --- |
| **Functional Programming** | **Logic Programming** |
| Functional Programming follows the Von-Neumann Architecture, or uses the sequential steps. | Logic Programming uses abstract model, or deals with objects and their relationships. |
| The syntax is actually the sequence of statements like (a, s, I). | The syntax is basically the logic formulae (Horn Clauses). |
| The computation takes part by executing the statements sequentially. | It computes by deducting the clauses. |
| Logic and controls are mixed together. | Logics and controls can be separated. |

## **What is Prolog?**

Prolog or **PRO**gramming in **LOG**ics is a logical and declarative programming language. It is one major example of the fourth generation language that supports the declarative programming paradigm. This is particularly suitable for programs that involve **symbolic** or **non-numeric computation**. This is the main reason to use Prolog as the programming language in **Artificial Intelligence**, where **symbol manipulation** and **inference manipulation** are the fundamental tasks.

In Prolog, we need not mention the way how one problem can be solved, we just need to mention what the problem is, so that Prolog automatically solves it. However, in Prolog we are supposed to give clues as the **solution method**.

Prolog language basically has three different elements −

**Facts** − The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

**Rules** − Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as −

grandfather(X, Y) :- father(X, Z), parent(Z, Y)

This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

**Questions** − And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

## **History of Prolog**

The heritage of prolog includes the research on theorem provers and some other automated deduction system that were developed in 1960s and 1970s. The Inference mechanism of the Prolog is based on Robinson’s Resolution Principle, that was proposed in 1965, and Answer extracting mechanism by Green (1968). These ideas came together forcefully with the advent of linear resolution procedures.

The explicit goal-directed linear resolution procedures, gave impetus to the development of a general purpose logic programming system. The **first** Prolog was the **Marseille Prolog** based on the work by **Colmerauer** in the year 1970. The manual of this Marseille Prolog interpreter (Roussel, 1975) was the first detailed description of the Prolog language.

Prolog is also considered as a fourth generation programming language supporting the declarative programming paradigm. The well-known Japanese Fifth-Generation Computer Project, that was announced in 1981, adopted Prolog as a development language, and thereby grabbed considerable attention on the language and its capabilities.

## **Some Applications of Prolog**

Prolog is used in various domains. It plays a vital role in automation system. Following are some other important fields where Prolog is used −

* Intelligent Database Retrieval
* Natural Language Understanding
* Specification Language
* Machine Learning
* Robot Planning
* Automation System
* Problem Solving

# Prolog - Basics

In this chapter, we will gain some basic knowledge about Prolog. So we will move on to the first step of our Prolog Programming.

The different topics that will be covered in this chapter are −

**Knowledge Base** − This is one of the fundamental parts of Logic Programming. We will see in detail about the Knowledge Base, and how it helps in logic programming.

**Facts, Rules and Queries** − These are the building blocks of logic programming. We will get some detailed knowledge about facts and rules, and also see some kind of queries that will be used in logic programming.

Here, we will discuss about the essential building blocks of logic programming. These building blocks are **Facts, Rules and the Queries**.

## **Facts**

We can define fact as an explicit relationship between objects, and properties these objects might have. So facts are unconditionally true in nature. Suppose we have some facts as given below −

* Tom is a cat
* Kunal loves to eat Pasta
* Hair is black
* Nawaz loves to play games
* Pratyusha is lazy.

So these are some facts, that are unconditionally true. These are actually statements, that we have to consider as true.

Following are some guidelines to write facts −

* Names of properties/relationships begin with lower case letters.
* The relationship name appears as the first term.
* Objects appear as comma-separated arguments within parentheses.
* A period "." must end a fact.
* Objects also begin with lower case letters. They also can begin with digits (like 1234), and can be strings of characters enclosed in quotes e.g. color(penink, ‘red’).
* phoneno(agnibha, 1122334455). is also called a predicate or clause.

### **Syntax**

The syntax for facts is as follows −

relation(object1,object2...).

### **Example**

Following is an example of the above concept −

cat(tom).

loves\_to\_eat(kunal,pasta).

of\_color(hair,black).

loves\_to\_play\_games(nawaz).

lazy(pratyusha).

## **Rules**

We can define rule as an implicit relationship between objects. So facts are conditionally true. So when one associated condition is true, then the predicate is also true. Suppose we have some rules as given below −

* Lili is happy if she dances.
* Tom is hungry if he is searching for food.
* Jack and Bili are friends if both of them love to play cricket.
* will go to play if school is closed, and he is free.

So these are some rules that are **conditionally** true, so when the right hand side is true, then the left hand side is also true.

Here the symbol ( :- ) will be pronounced as “If”, or “is implied by”. This is also known as neck symbol, the LHS of this symbol is called the Head, and right hand side is called Body. Here we can use comma (,) which is known as conjunction, and we can also use semicolon, that is known as disjunction.

### **Syntax**

rule\_name(object1, object2, ...) :- fact/rule(object1,

object2, ...)

Suppose a clause is like :

P :- Q;R.

This can also be written as

P :- Q.

P :- R.

If one clause is like :

P :- Q,R;S,T,U.

Is understood as

P :- (Q,R);(S,T,U).

Or can also be written as:

P :- Q,R.

P :- S,T,U.

### **Example**

happy(lili) :- dances(lili).

hungry(tom) :- search\_for\_food(tom).

friends(jack, bili) :- lovesCricket(jack), lovesCricket(bili).

goToPlay(ryan) :- isClosed(school), free(ryan).

## **Queries**

Queries are some questions on the relationships between objects and object properties. So question can be anything, as given below −

* Is tom a cat?
* Does Kunal love to eat pasta?
* Is Lili happy?
* Will Ryan go to play?

So according to these queries, Logic programming language can find the answer and return them.

## **Knowledge Base in Logic Programming**

In this section, we will see what knowledge base in logic programming is.

Well, as we know there are three main components in logic programming − **Facts, Rules** and **Queries**. Among these three if we collect the facts and rules as a whole then that forms a **Knowledge Base**. So we can say that the **knowledge base** is a **collection of facts and rules**.

Now, we will see how to write some knowledge bases. Suppose we have our very first knowledge base called KB1. Here in the KB1, we have some facts. The facts are used to state things, that are unconditionally true of the domain of interest.

### **Knowledge Base 1**

Suppose we have some knowledge, that Priya, Tiyasha, and Jaya are three girls, among them, Priya can cook. Let’s try to write these facts in a more generic way as shown below −

girl(priya).

girl(tiyasha).

girl(jaya).

can\_cook(priya).

**Note** − Here we have written the name in lowercase letters, because in Prolog, a string starting with uppercase letter indicates a **variable**.

Now we can use this knowledge base by posing some queries. “Is priya a girl?”, it will reply “yes”, “is jamini a girl?” then it will answer “No”, because it does not know who jamini is. Our next question is “Can Priya cook?”, it will say “yes”, but if we ask the same question for Jaya, it will say “No”.

### **Output**

GNU Prolog 1.4.5 (64 bits)

Compiled Jul 14 2018, 13:19:42 with x86\_64-w64-mingw32-gcc

By Daniel Diaz

Copyright (C) 1999-2018 Daniel Diaz

| ?- change\_directory('D:/TP Prolog/Sample\_Codes').

yes

| ?- [kb1]

.

compiling D:/TP Prolog/Sample\_Codes/kb1.pl for byte code...

D:/TP Prolog/Sample\_Codes/kb1.pl compiled, 3 lines read - 489 bytes written, 10 ms

yes

| ?- girl(priya)

.

yes

| ?- girl(jamini).

no

| ?- can\_cook(priya).

yes

| ?- can\_cook(jaya).

no

| ?-

Let us see another knowledge base, where we have some rules. Rules contain some information that are conditionally true about the domain of interest. Suppose our knowledge base is as follows −

sing\_a\_song(ananya).

listens\_to\_music(rohit).

listens\_to\_music(ananya) :- sing\_a\_song(ananya).

happy(ananya) :- sing\_a\_song(ananya).

happy(rohit) :- listens\_to\_music(rohit).

playes\_guitar(rohit) :- listens\_to\_music(rohit).

So there are some facts and rules given above. The first two are facts, but the rest are rules. As we know that Ananya sings a song, this implies she also listens to music. So if we ask “Does Ananya listen to music?”, the answer will be true. Similarly, “is Rohit happy?”, this will also be true because he listens to music. But if our question is “does Ananya play guitar?”, then according to the knowledge base, it will say “No”. So these are some examples of queries based on this Knowledge base.

### **Output**

| ?- [kb2].

compiling D:/TP Prolog/Sample\_Codes/kb2.pl for byte code...

D:/TP Prolog/Sample\_Codes/kb2.pl compiled, 6 lines read - 1066 bytes written, 15 ms

yes

| ?- happy(rohit).

yes

| ?- sing\_a\_song(rohit).

no

| ?- sing\_a\_song(ananya).

yes

| ?- playes\_guitar(rohit).

yes

| ?- playes\_guitar(ananya).

no

| ?- listens\_to\_music(ananya).

yes

| ?-

### **Knowledge Base 3**

The facts and rules of Knowledge Base 3 are as follows −

can\_cook(priya).

can\_cook(jaya).

can\_cook(tiyasha).

likes(priya,jaya) :- can\_cook(jaya).

likes(priya,tiyasha) :- can\_cook(tiyasha).

Suppose we want to see the members who can cook, we can use one **variable** in our query. The variables should start with uppercase letters. In the result, it will show one by one. If we press enter, then it will come out, otherwise if we press semicolon (;), then it will show the next result.

Let us see one practical demonstration output to understand how it works.

### **Output**

| ?- [kb3].

compiling D:/TP Prolog/Sample\_Codes/kb3.pl for byte code...

D:/TP Prolog/Sample\_Codes/kb3.pl compiled, 5 lines read - 737 bytes written, 22 ms

warning: D:/TP Prolog/Sample\_Codes/kb3.pl:1: redefining procedure can\_cook/1

D:/TP Prolog/Sample\_Codes/kb1.pl:4: previous definition

yes

| ?- can\_cook(X).

X = priya ? ;

X = jaya ? ;

X = tiyasha

yes

| ?- likes(priya,X).

X = jaya ? ;

X = tiyasha

yes

| ?-