# UNIT VI

# LOGIC PROGRAMMING LANGUAGES

## Introduction to logic programming:

- Programming that uses a form of symbolic logic as a programming language is often called logic programming
- Languages based on symbolic logic are called logic programming languages, or declarative languages.
- In this case, the approach is to express programs in a form of symbolic logic and use a logical inferencing process to produce results.
- Programs in logic programming languages are collections of facts and rules.
- The semantics of logic programs also bears little resemblance to that of imperative-language programs. These observations should lead the reader to some curiosity about the nature of logic programming and declarative languages.
- A **proposition** can be thought of as a logical statement that may or may not be true. It consists of objects and the relationships among objects.
- **Symbolic logic** can be used for the three basic needs of formal logic:
    - To express propositions,
    - To express the relationships between propositions
    - To describe how new propositions can be inferred from other propositions that are assumed to be true.
- Particular form of symbolic logic used for logic programming called predicate calculus.
- **Object Representation :** Objects in propositions are represented by simple terms: either constants or variables
    - •Constant: a symbol that represents an object.
    - •Variable: a symbol that can represent different objects at different times.
- The simplest propositions, which are called atomic propositions, consist of compound terms.
- Compound term: one element of a mathematical relation, written like a mathematical function.
- Mathematical function is a mapping, which can be represented either as an expression or as a table or list of tuples.
- Compound term composed of two parts
    - Functor: function symbol that names the relationship
    - Ordered list of parameters (tuple)
- A compound term with a single parameter is a 1-tuple; one with two parameters is a 2-tuple, and so forth.

  Examples: student(jon) , like(seth, OSX) ,like(nick, windows)
  like(jim, linux)

- Propositions can be stated in two forms:
  - Fact: proposition is assumed to be true
  - Query: truth of proposition is to be determined
- Compound propositions have two or more atomic propositions, which are connected by logical connectors, or operators.
- The names, symbols, and meanings of the predicate calculus logical connectors are as follows:

**Logical Operators**

| Name | Symbol | Example | Meaning |
|---|---|---|---|
| negation | ¬ | ¬ a | not a |
| conjunction | ∩ | a ∩ b | a and b |
| disjunction | ∪ | a ∪ b | a or b |
| equivalence | ≡ | a ≡ b | a is equivalent to b |
| implication | ⊃<br>⊂ | a ⊃ b<br>a ⊂ b | a implies b<br>b implies a |

- The following are examples of compound propositions:

  aΠbΠ c

  a Π¬ b ΠdThe ¬ operator has the highest precedence.

  The ¬ operator has the highest precedence. The operators ∩, ∪, and ≡ all have higher precedence than ⊃ and ⊂ So, the second example is equivalent to

  (a Π (¬ b))Πd

- Variables can appear in propositions but only when introduced by special symbols called quantifiers.
- Predicate calculus includes two quantifiers, as described below, where X is a variable and P is a proposition.

**Quantifiers**

| Name | Example | Meaning |
|---|---|---|
| universal | ∀X.P | For all X, P is true |
| existential | ∃X.P | There exists a value of X such that P is true |

- The period between X and P simply separates the variable from the proposition.

For example, consider the following:

$$\forall X.(\text{woman}(X) \supset \text{human}(X))$$
$$\exists X.(\text{mother}(\text{mary}, X) \cap \text{male}(X))$$

The first of these propositions means that for any value of X, if X is a woman, then X is a human. The second means that there exists a value of X such that mary is the mother of X and X is a male; in other words, mary has a son.

**Clausal Form:**

- It is a relatively simple form of propositions. All propositions can be expressed in clausal form.
- A proposition in clausal form has the following general syntax:

$$B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$$

in which the A's and B's are terms. The meaning of this clausal form proposition is as follows: If all of the A's are true, then at least one B is true.

- The primary characteristics of clausal form propositions are the following:

Existential quantifiers are not required. Universal quantifiers are implicit in the use of variables in the atomic propositions; and no operators other than conjunction and disjunction are required.

**Predicate Calculus and Proving Theorems:**

- A use of propositions is to discover new theorems that can be inferred from known axioms and theorems.
- Resolution is an inference rule that allows inferred propositions to be computed from given propositions.

Suppose there are two propositions with the forms:

P1 $\subset$ P2

Q1 $\subset$ Q2

Their meaning is that P2 implies P1 and Q2 implies Q1. Furthermore, suppose that P1 is identical to Q2, so that we could rename P1 and Q2 as T. Then, we could rewrite the two propositions as

T $\subset$ P2

Q1 $\subset$ T

Now, because P2 implies T and T implies Q1, it is logically obvious that P2 implies Q1, which we could write as

Q1 $\subset$ P2

The process of inferring this proposition from the original two propositions is resolution.

- Unification: finding values for variables in propositions that allows matching process to succeed.

- Instantiation: assigning temporary values to variables to allow unification to succeed.
- After instantiating a variable with a value, if matching fails, may need to backtrack and instantiate with a different value.

**Theorem Proving:**

- Basis for logic programming.
- When propositions used for resolution, only restricted form can be used.
- Horn clause - can have only two forms
  –Headed: single atomic proposition on left side.
  –Headless: empty left side (used to state facts).
- Most propositions can be stated as Horn clauses.

## Programming with Prolog

**Origins of Prolog:**

- PROgramming in LOGic.
- It is the most widely used logic programming language
- Its development started in 1970 and it was result of the collaboration between researchers from University of Aix-Marseille (Natural language processing) and University of Edinburgh (Automated theorem proving).
- Main applications:
  - Artificial intelligence: expert systems, natural language processing, …
  - Databases: query languages, data mining,…
  - Mathematics: theorem proving, symbolic packages,…

**Term:**

- Prolog programs consist of collections of statements. There are only a few kinds of statements in Prolog, but they can be complex.
- All Prolog statement, as well as Prolog data, are constructed from terms.
- A Prolog term is a **constant**, **a variable**, or **a structure**.
- A **constant** is either an atom or an integer.
  - Atoms are the symbolic values of Prolog and are similar to their counterparts in LISP. In particular, an atom is either a string of letters, digits, and underscores that begins with a lowercase letter or a string of any printable ASCII characters delimited by apostrophes.
- A **variable** is any string of letters, digits, and underscores that begins with an uppercase letter or an underscore ( _ ).The binding of a value, and thus a type, to a variable is called an**instantiation.** Instantiation occurs only in the resolution process. A variable that has not been assigned a value is called **uninstantiated**. Instantiations last only as long as it

takes to satisfy one complete goal, which involves the proof or disproof of one proposition.

- **Structures** represent the atomic propositions of predicate calculus, and their general form is the same: functor(parameter list)

**Fact Statements:**

- The statements used to construct the hypotheses, or database of assumed information—the statements from which new information can be inferred.
- Prolog has two basic statement forms; these correspond to the headless and headed Horn clauses of predicate calculus.
- Every Prolog statement is terminated by a period

female(shelley).
male(bill).
female(mary).
male(jake).
father(bill, jake).
father(bill, shelley).
mother(mary, jake).
mother(mary, shelley).

These simple structures state certain facts about jake, shelley, bill, andmary.For example, the first states that shelley is a female. The last four connect their two parameters with a relationship that is named in the functor atom;

For example, the fifth proposition might be interpreted to mean thatbill is the father of jake. Note that these Prolog propositions, like those of predicate calculus, have no intrinsic semantics. They mean whatever the programmer wants them to mean.

For example, the proposition father(bill, jake).could mean bill and jake have the same father or that jake is the father of bill.

**Rule Statements:**

- Used for the hypotheses.
- Headed Horn clause.
- Right side: antecedent (if part)–May be single term or conjunction.
- Left side: consequent (then part)–Must be single term.
- If the antecedent of a Prolog statement is true, then the consequent of the statement must also be true.
- Conjunction: multiple terms separated by logical AND operations (implied).

**Example Rules:**

ancestor(mary, shelley) :- mother(mary, shelley).

states that if mary is the mother of shelley, then mary is an ancestor of shelley. Headed Horn clauses are called rules.

The following demonstrates the use of variables in Prolog statements:

parent(X, Y) :- mother(X, Y).

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

grandparent(X, Z) :- parent(X, Y) , parent(Y, Z).

**Goal Statements:**

- For theorem proving, theorem is in form of proposition that we want system to prove or disprove – goal statement.
- The syntactic form of Prolog goal statements is identical to that of headless Horn clauses. For example, we could have

  man(fred).

  to which the system will respond either yes or no. The answer yes means that the system has proved the goal was true under the given database of facts and relationships. The answer no means that either the goal was determined to be false or the system was simply unable to prove it.
- Conjunctive propositions and propositions with variables are also legal goals.

**Inferencing Process of Prolog:**

- Queries are called goals. When a goal is a compound proposition, each of the facts (structures) is called a subgoal.
- To prove a goal is true, must find a chain of inference rules and/or facts.
  For goal Q:
  B :- A
  C :- B
  …
  Q :- P
- Process of proving a subgoal called matching, satisfying, or resolution.
- **Approaches**
  - Bottom-up resolution, forward chaining
    –Begin with facts and rules of database and attempt to find sequence that leads to goal.
    –Works well with a large set of possibly correct answers
  - Top-down resolution, backward chaining
    –Begin with goal and attempt to find sequence that leads to set of facts in database.
    –Works well with a small set of possibly correct answers.
  - Prolog implementations use backward chaining.

**Subgoal Strategies**

- When goal has more than one subgoal, can use either

  –Depth-first search: find a complete proof for the first subgoal before working on others

  –Breadth-first search: work on all subgoals in parallel

- Prolog uses depth-first search - Can be done with fewer computer resources

**Backtracking**

- With a goal with multiple subgoals, if fail to show truth of one of subgoals, reconsider previous subgoal to find an alternative solution: backtracking
- Begin search where previous search left off.
- Can take lots of time and space because may find all possible proofs to every subgoal.

To illustrate backtracking, consider the following example database and

traced compound goal:

likes(jake, chocolate).

likes(jake, apricots).

likes(darcie, licorice).

likes(darcie, apricots).

trace.

likes(jake, X), likes(darcie, X).

(1) 1 Call: likes(jake, _0)?

(1) 1 Exit: likes(jake, chocolate)

(2) 1 Call: likes(darcie, chocolate)?

(2) 1 Fail: likes(darcie, chocolate)

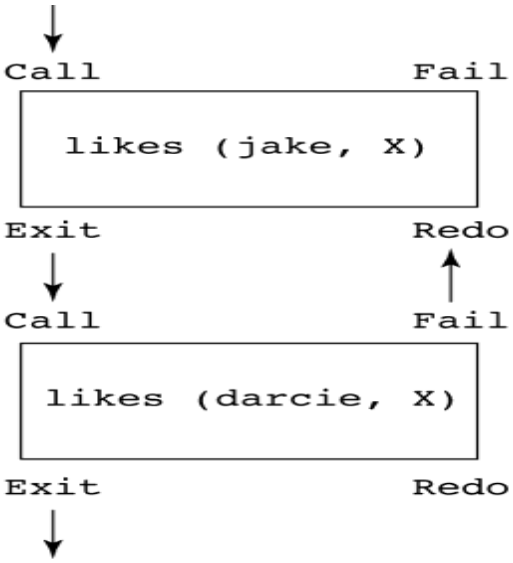
(1) 1 Redo: likes(jake, _0)?

(1) 1 Exit: likes(jake, apricots)

(3) 1 Call: likes(darcie, apricots)?

(3) 1 Exit: likes(darcie, apricots)

X = apricots




**Control flow model for the goal likes (jake, X), likes (darcie, X)**

**List Structures:**

- Atomic propositions, which are also called structures, are actually a form of records.
- The other basic data structure supported is the list. Lists are sequences of any number of elements, where the elements can be atoms, atomic propositions, or any other terms, including other lists.
- Prolog uses the syntax of ML and Haskell to specify lists. The list elements are separated by commas, and the entire list is delimited by square brackets, as in

  [apple, watermelon, grape, orange]
  [] (empty list)
  [X | Y] (head X and tail Y)

  **Append Example**

```
append([], List, List).
append([Head | List_1], List_2, [Head | List_3]) :-
append (List_1, List_2, List_3).
```

  **Reverse Example**

```
reverse([], []).
reverse([Head | Tail], List) :-
reverse (Tail, Result),
append (Result, [Head], List).
```

**Deficiencies of Prolog:**

•Resolution order control

•The closed-world assumption

•The negation problem

•Intrinsic limitations

**Applications of Logic Programming:**

•Relational database management systems

•Expert systems

•Natural language processing

## Multi - Paradigm Languages

**Concurrent programming** – have language constructs for concurrency, these may involve multi-threading, support for distributed computing, message passing, shared resources (including shared memory), or futures

**Actor programming** – concurrent computation with actors that make local decisions in response to the environment (capable of selfish or competitive behavior)

**Constraint programming** – relations between variables are expressed as constraints (or constraint networks), directing allowable solutions (uses constraint satisfaction or simplex algorithm)

**Dataflow programming** – forced recalculation of formulas when data values change (e.g. spreadsheets)

**Declarative programming** – describes actions (e.g. HTML describes a page but not how to actually display it)

**Distributed programming** – have support for multiple autonomous computers that communicate via computer networks

**Functional programming** – uses evaluation of mathematical functions and avoids state and mutable data

**Generic programming** – uses algorithms written in terms of to-be-specified-later types that are then instantiated as needed for specific types provided as parameters

**Imperative programming** – explicit statements that change a program state

Logic programming – uses explicit mathematical logic for programming

**Metaprogramming** – writing programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime

**Template metaprogramming** – metaprogramming methods in which templates are used by a compiler to generate temporary source code, which is merged by the compiler with the rest of the source code and then compiled

**Reflective programming** – metaprogramming methods in which a program modifies or extends itself

**Object-oriented programming** – uses data structures consisting of data fields and methods together with their interactions (objects) to design programs

Class-based – object-oriented programming in which inheritance is achieved by defining classes of objects, versus the objects themselves

Prototype-based – object-oriented programming that avoids classes and implements inheritance via cloning of instances

**Pipeline programming** – a simple syntax change to add syntax to nest function calls to language originally designed with none

**Rule-based programming** – a network of rules of thumb that comprise a knowledge base and can be used for expert systems and problem deduction & resolution

**Visual programming** – manipulating program elements graphically rather than by specifying them textually (e.g. Simulink); also termed diagrammatic programming.

## Important Questions

1.) Discuss about basic elements of Prolog? (2017 SET-1 8M)
2.) Explain different types of propositions in logic programming? (2017 SET-1 8M)
3.) Discuss Terms and Goal statements and List structures in Prolog? (2016 SET-4 16M, 2017 SET-2 8M)
4.) What are the syntactic form and usage of Fact and Ruled statements in Prolog? (2016 SET-2 8M2017 SET-2,3 8M)
5.) Discuss in detail about Logic Programming Language and compare it with Functional Programming Language? (2017 SET-3 8M)
6.) Mention various applications of multi paradigm languages? (2016 2017 SET-3 8M)
7.) Explain Prolog interfacing process in detail? (2016 SET-1 10M, 2017 SET-4 8M)
8.) Explain in detail about Prolog Programming in detail? (2017 SET-4 8M)
9.) Explain how backtracking works in Prolog? (2016 SET-2 8M)