ECII/ECSI 3206:
Artificial Intelligence [and expert systems]
Topic 10:A.I programming languages[Prolog]

By: Edgar Otieno

Introduction to Prolog

Logic Programming is a declarative language and very different from actual conventional programming like C, C++ or Java

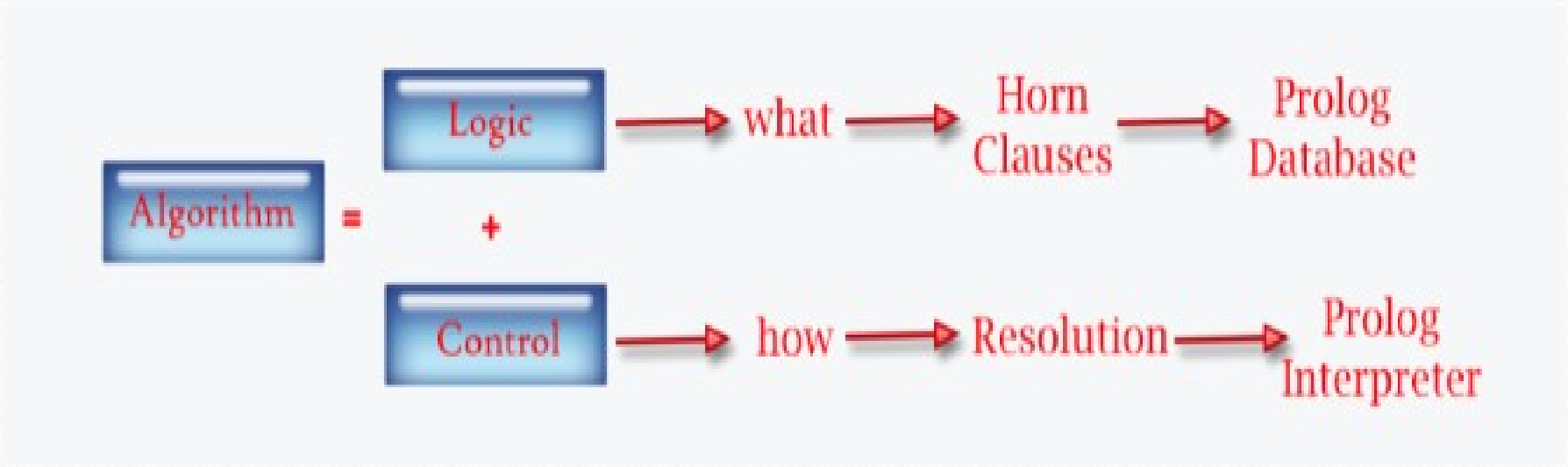
In Logic programming user only needs to specify the problem statements and the logic programming system finds the solution to that problem based on the provided information. In other words, we define *what* part of the problem and system finds *how* to get solution to that problem.

In declarative languages, instead of writing the problem in exact sequence of steps; our main focus is to define correct relationship between the different parts of the problem. we provide information and declare relationships between objects and after supplying the query, the logic programming system deduce the result by tracing through given information.

- Though there are many logic programming languages but **Prolog** is widely used logic programming language in artificial intelligence research..
- Prolog - PROgramming in LOGic, was first developed by Alain Colmerauer and Philippe Roussel in 1972 - A Logic Programming language based on the predicate logic.
- In prolog, clauses are actually descriptive statements that specifies *what* is true about the problem and because of that Prolog is also known as declarative language or rule-based language.

- In Prolog program we write the program statements in terms of facts and rules. The system reads in the program and stores it. Upon asking the questions (known as queries) the system gives the answer by searching through the possible solution(s).
- Prolog is also widely used for AI programs especially experts systems.
- Prolog is also very useful in some problem areas, such as natural language processing, databases.

CONNECTION BETWEEN LOGIC PROGRAMMING & PROLOG



Basic features of Prolog include:

- pattern-matching mechanism
- backtracking strategy that searches for possible solutions
- uniform data structures from which programs are built

PROLOG

- Prolog program is simply based on predicate logic known as Horn clause. In prolog, we compose the program using facts and rules and we pose a query on query prompt about the facts and rules we inserted.
- **Horn Clause :** Horn clause consists of head (left hand side) and body (right hand side). Head can have 0 or 1 predicate and body can have list of predicates. That means LHS has only single literal and RHS can have more than one literals.

Clause

- Written as : h :- b (where b is p1, p2, p3, .., pn and ' **:-** ' operator is read as '**if**')
- Read as : h is true *if* b is true. In other words h is true if all the predicates on right side are true.
- Syntax : Start the statement (relationship or Object) with lowercase letters and end with '**.**' period (full stop) - for all facts, rules and queries.
- Examples :

Headless Horn Clauses :

son(john). // Read as : john is son

son(john,mary). // Read as : john is son of mary

Headed Horn Clauses :

boy(john) :- son(john,mary). // Read as : john is a boy if he is son of mary

Facts

- **Facts :** Facts are those statements that state the objects or describe the relationship between objects. For an instance when we say *john likes piano*, we are showing the 'like' relationship between two objects 'john and piano' and in prolog this fact can be written as *likes(john,piano)*. Facts are also known as "unconditional horn clauses" and they are always true.

- Examples: logic_programming. // Read as : logic programming
  music_student(john). // Read as : john is a music student
  likes('John', car(bmw)) // Read as : john likes bmw car
  gives(john, chocolate, jane). // Read as : john gives chocolate to jane

Rules

- **Rules :** Rules are the conditional statements about objects and their relationships.
- Examples: If we want to say that *john and jane are friends if john likes jane and jane likes john*. Then in prolog this *friends* rule can be written as,
- friends(john,jane) :- likes(john,jane), likes(jane,john).
- Examples: Rules with variables.

likes(john, X) :- car(X). // Read as : john likes X if X is a car.
friends(X, Y) :- likes(X, Y), likes(Y, X). // Read as : X and Y are friends if X likes Y and Y likes X. OR Two people are friends if they like each other.

- Example : For below Prolog Program, we will ask two queries about the facts we have provided.
  Program :likes(alice,john). // Read as : alice likes john
  likes(john,mary). // Read as : john likes mary
- Queries : ?- likes(alice,john). // Read as :- Does alice like john?
  true.
  ?- likes(alice,mary). // Read as :- Does alice like mary?

Variables in Prolog

- In prolog variables always start with an uppercase letter or an underscore and it can be written in head or body or the goals.
- Examples : Who, What, X, Y, ABC, Abc, A12
  _who, _x, _variable, _abc,
  _Abc, _12

Variables and Unification

- How do we say something like "What does Fred eat"? Suppose we had the following fact in our database: eats(fred,mangoes).
- How do we ask what fred eats? We could type in something like?- eats(fred,what). However Prolog will say no. The reason for this is that what does not match with mangoes.
- In order to match arguments in this way we must use a Variable. The process of matching items with variables is known as **unification.**
- Thus returning to our first question we can find out what fred eats by typing

  ?- eats(fred,X).

  X=mangoes
- As a result of this query, the variable X has matched (or unified) with mangoes.

Cont..

- **Comments in Prolog :** Single-line or Multi-line comments begin with /* and end with */ , but cannot be nested. % is also used only for single line comment, but it should be at the starting of the line.

Searching in Prolog

- Suppose that we have the following database
  - eats(fred,pears).
  - eats(fred,t_bone_steak).
  - eats(fred,apples).

  So far we have only been able to ask if fred eats specific things. Suppose that I wish to instead answer the question, "What are all the things that fred eats". To answer this I can use variables again. Thus I can type in the query: ?- eats(fred,X)

- As we have seen earlier, Prolog will answer with
  - *X = pears*
- This is because it has found the first clause in the database. At this point Prolog allows us to ask if there are other possible solutions. When we do so we get the following.
  - *X = t_bone_steak*
- if I ask for another solution, Prolog will then give us.
  - *X = apples*
- If we ask for further solutions, Prolog will answer no, since there are only three ways to prove fred eats something. The mechanism for finding multiple solution is called **backtracking**. This is an essential mechanism in Prolog and we shall see more of it later.

- If we have this fact and rule
  - rainy(london).
  - rainy(bangkok).
  - dull(X):- rainy(X).
  - We can ask (or query) prolog on its command prompt
    - ?- dull(C). (is there a C that makes this predicate true?)
    - It will automatically try to substitute atoms in its fact into its rule such that our question gives the answer true
    - in this example, we begin with dull(X), so the program first chooses an atom for X, that is london (our first atom in this example)
    - The program looks to see if there is rainy(london). There is!
    - So the substitution gives the result “true”

- The Prolog will answer
  - C= london
- To find an alternative answer, type “;” and “Enter”
- It’ll give C= bangkok
- If it cannot find any more answer, it will answer “no”

Cont…

```
/* Clause 1 */  located_in(atlanta,georgia).
/* Clause 2 */  located_in(houston,texas).
/* Clause 3 */  located_in(austin,texas).
/* Clause 4 */  located_in(toronto,ontario).
/* Clause 5 */  located_in(X,usa) :- located_in(X,georgia).
/* Clause 6 */  located_in(X,usa) :- located_in(X,texas).
/* Clause 7 */  located_in(X,canada) :- located_in(X,ontario).
/* Clause 8 */  located_in(X,north_america) :- located_in(X,usa).
/* Clause 9 */  located_in(X,north_america) :- located_in(X,canada).
```

- To ask whether atlanta is in georgia:

?- located_in(atlanta,georgia).

- This query matches clause 1. So prolog replies “yes”.

?- located_in(atlanta,usa).

This query can be solve by calling clause 5, and then clause 1. So prolog replies “yes”.

?-located_in(atlanta,texas).

this query gets “no” as its answer because this fact cannot be deduced from the knowledge base.

The query **succeeds** if it gets a “yes” and **fails** if it gets a “no”.

Arithmetic functions

- If we ask ?- (2+3) = 5
- Prolog will answer “no” because it sees (2+3) as a +(2,3) structure
- Prolog thus have a special function is(X,Y) This function will compare X and the arithmetic value of Y (there are prefix and infix versions of this function)
- So, asking ?-is(X,1+2). will return X=3
- But asking ?- is(1+2,4-1). will return “no”
  - Because it only evaluates the second argument :P
  - so we should ask ?- is(Y,1+2), is(Y,4-1) instead

Prolog Syntax

- The central data structure in Prolog is that of a term.
- There are terms of four kinds: **atoms, numbers, variables, and compound terms**. Atoms and numbers are sometimes grouped together and called atomic terms.

Atoms

- Atoms are usually strings made up of lower- and uppercase letters, digits, and the underscore, starting with a lowercase letter. The following are all valid Prolog atoms: elephant, b, abcXYZ, x_123, another_pint_for_me_please
- On top of that also any series of arbitrary characters enclosed in single quotes denotes an atom. ’This is also a Prolog atom.’

Numbers

- All Prolog implementations have an integer type: a sequence of digits, optionally preceded by a - (minus). Some also support floats. Check the manual for details.

Variables.

- Variables are strings of letters, digits, and the underscore, starting with a capital letter or an underscore. Examples: X, Elephant, _4711, X_1_2, MyVariable, _
- The last one of the above examples (the single underscore) constitutes a special case. It is called the anonymous variable and is used when the value of a variable is of no particular interest.

Compound terms.

- Compound terms are made up of a functor (a Prolog atom) and a number of arguments (Prolog terms, i.e., atoms, numbers, variables, or other compound terms) enclosed in parentheses and separated by commas. The following are some examples for compound terms:

  is_bigger(horse, X), f(g(X, _), 7), ’My Functor’(dog)

The sets of compound terms and atoms together form the set of Prolog predicates. A term that doesn’t contain any variables is called a *ground term.*

Prolog Lists

- Its one of the most important data structures in Prolog.
- Lists are contained in square brackets with the elements being separated by commas. Example: **[elephant, horse, donkey, dog].**
- This is the list of the four atoms elephant, horse, donkey, and dog. Elements of lists could be any valid Prolog terms, i.e., atoms, numbers, variables, or compound terms. The empty list is written as [ ]

Cont..

- The following is another example for a (slightly more complex) list: [elephant, [ ], X, parent(X, tom), [a, b, c], f(22)]
- The first element of a list is called its **head** and the remaining list is called the **tail**. An empty list doesn’t have a head. A list just containing a single element has a head (namely that particular single element) and its tail is the empty list.

- We can place a special symbol | (pronounced 'bar') in the list to distinguish between the first item in the list and the remaining list. For example, consider the following.
- [first,second,third] = [A|B]where *A = first* and *B=[second,third]*

Some basic list manipulation Predicates

- **length/2:** The second argument is matched with the length of the list in the first argument.

  Example: ?- length([elephant, [], [1, 2, 3, 4]], Length).

  Length = 3

  Yes

- **member/2:** The goal member(Elem, List) will succeed, if the term Elem can be matched with one of the members of the list List.
  - Example: ?- member(dog, [elephant, horse, donkey, dog, monkey]).

    Yes

- **append/3:** Concatenate two lists.
- **last/2:** This predicate succeeds, if its second argument matches the last element of the list given as the first argument of last/2.
- **reverse/2:** This predicate can be used to reverse the order of elements in a list.
  - Example: ?- reverse([1, 2, 3, 4, 5], X).

    X = [5, 4, 3, 2, 1]

    Yes

- **select/3:** Given a list in the second argument and an element of that list in the first, this predicate will match the third argument with the remainder of that list.
  - Example: ?- select(bird, [mouse, bird, jellyfish, zebra], X).

    X = [mouse, jellyfish, zebra]

    Yes