**Prolog** is a [logic programming](https://en.wikipedia.org/wiki/Logic_programming)language associated with [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) and [computational linguistics](https://en.wikipedia.org/wiki/Computational_linguistics).[[1]](https://en.wikipedia.org/wiki/Prolog#cite_note-Clocksin2003-1)[[2]](https://en.wikipedia.org/wiki/Prolog#cite_note-Bratko2012-2)[[3]](https://en.wikipedia.org/wiki/Prolog#cite_note-Covington1994-3)

Prolog has its roots in [first-order logic](https://en.wikipedia.org/wiki/First-order_logic), a [formal logic](https://en.wikipedia.org/wiki/Formal_logic), and unlike many other [programming languages](https://en.wikipedia.org/wiki/Programming_language), Prolog is intended primarily as a [declarative](https://en.wikipedia.org/wiki/Declarative_programming" \o "Declarative programming)programming language: the program logic is expressed in terms of relations, represented as facts and [rules](https://en.wikipedia.org/wiki/Rule_of_inference). A computation is initiated by running a *query* over these relations.[[4]](https://en.wikipedia.org/wiki/Prolog#cite_note-lloyd84-4)

The language was first conceived by [Alain Colmerauer](https://en.wikipedia.org/wiki/Alain_Colmerauer) and his group in Marseille, France, in the early 1970s and the first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel.[[5]](https://en.wikipedia.org/wiki/Prolog#cite_note-Kowalski-5)[[6]](https://en.wikipedia.org/wiki/Prolog#cite_note-6)

Prolog programs specify relationships among objects and properties of object

When we say “Amit has a Bike”, we are declaring the ownership relationship between two objects; Amit and the bike

When we ask, “Does Amit own the bike?” Then we are trying to find out about a relationship.

Relationships can be also rules such as

Two people are brother if

They are both male and

They both have the same parents(F and M)(A and B are not same)

A rule allows us to find out about a relationship even if the relationship isn’t explicitly stated as a fact.

**Facts Rules and Queries**

Prolog programs describe relations, defined by means of clauses Pure Prolog is restricted to [**Horn clauses**](https://en.wikipedia.org/wiki/Horn_clauses). There are two types of clauses: facts and rules.

A rule is of the form

Head :- Body.

and is read as "Head is true if Body is true". A rule's body consists of calls to predicates, which are called the rule's goals.

**Facts:**

Clauses with **empty bodies** are called facts.

Facts describe **explicit** relationships between objects and properties objects might have

An example of a fact is:

cat(tom).

which is equivalent to the rule:

cat(tom) :- true.

Ram has phone number 123123123 is written in prolog as

phoneno(ram,123123123).

**Note**:

* Names of properties /relationships begin with lowercase letter
* The **Relationship name** appears as the first term
* Objects appear as comma separated arguments within parenthesis
* A period “.” must end a fact
* Objects also begin with lower case letters. The also can begin with digits like 1234 and can be strings of characters enclosed in quotes e.g color(penink,’red’).
* phoneno(ram,123123123). is also called a **predicate** or a **clause**

**Facts about a Department of BCS-**

%teaches(X,Y): person X teaches course Y

teaches(ram, CS301).

%student(X,Y): person X student course Y

student(hari, CS301).

Together, these facts will form **Prolog’s database**/ **knowledge base**

**Rules**

Clauses with bodies are called **rules**. An example of a rule is:

animal(X) :- cat(X).

Consider the following case which produces a general rule-

One teacher will guide a student if the student studies that very course id on which the teacher teaches

In prolog this will be written as:

Guide(Teacher,Student):-

Teaches(Teacher,Coursed).

Studies(Student, Courseid).

**Facts are unit clauses and rules are non-unit clauses.**

**Queries**

Queries will be based on facts and rules. We can ask questions based on the stored information.

Suppose we want to know if ram lectures in CSC301 or not, then we can ask:

?-teaches(ram,CSC301).

Yes

In GNU prolog, queried are terminated by a full stop.

To answer this query, prolog consults its database to see if this is a known fact or not.

We can also ask-

?-teaches(ram,X).

X=CSC301

If answer is true or yes then the query is succeeded

If answer is false or no then the query failed.

Syntax of a clause

:- means “if” or “is implied by” also called the neck symbol.

The built-in [predicate](https://en.wikipedia.org/wiki/Predicate_(mathematics)) ,/2(meaning a 2-arity [operator](https://en.wikipedia.org/wiki/Operator_(programming)) with name ,) denotes [conjunction](https://en.wikipedia.org/wiki/Logical_conjunction) of goals, and ;/2 denotes [disjunction](https://en.wikipedia.org/wiki/Logical_disjunction). Conjunctions and disjunctions can only appear in the body, not in the head of a rule.

Example on clause writing:

P:- Q;R.

Can also be written as

P:-Q.

P:-R.

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

Is understood as

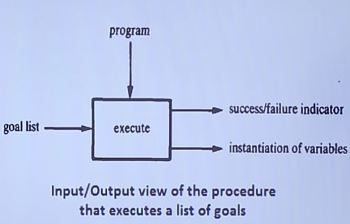
P;-(Q,R);(S,T,U).

Can also be written as

P:-Q,R.

P:-S,T,U.

How does a prolog program execute

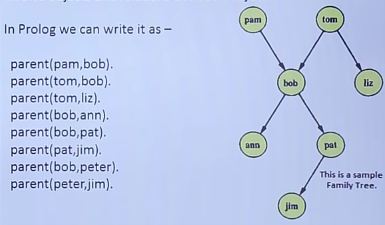


**Declarative and procedural semantics**

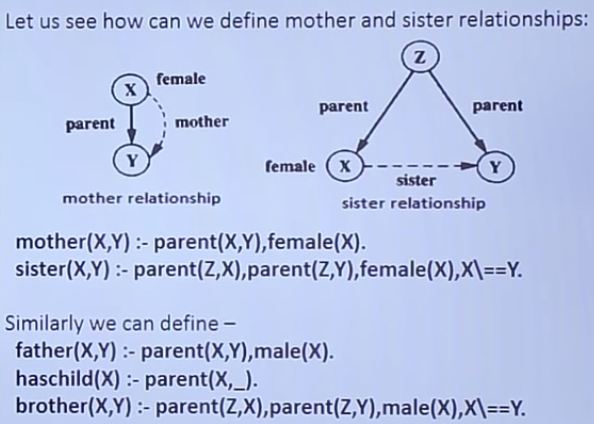
**The Declarative Semantics** of prolog defines whether a goal is true with respect to a given program, if it is true, for what instantiation of variables it is true.

**The Procedural Semantics** of prolog is a procedure for satisfying a list of goals in the context of a given program. The procedure outputs the truth or falsity of the goal list and the corresponding instantiations of variables. The procedure automatically backtracks to examine alternatives.

**Family Relationship in prolog**

****

The arguments of relations can {among other things} be: **concrete objects, or constants (**such as pat and jim**),** general objects such as X and Y. Objects of the first kind in our programs are called **atoms.** Objects of the second kind are called variables.



X\==Y means X is not equal to Y

\_ means some variable X in prolog

Example of execution using trace and no trace

{trace}

| ?- mother(pam,X).

1 1 Call: mother(pam,\_42) ?

2 2 Call: parent(pam,\_42) ?

2 2 Exit: parent(pam,bob) ?

3 2 Call: female(pam) ?

3 2 Exit: female(pam) ?

1 1 Exit: mother(pam,bob) ?

X = bob

yes

{trace}

| ?- mother(tom,X).

1 1 Call: mother(tom,\_42) ?

2 2 Call: parent(tom,\_42) ?

2 2 Exit: parent(tom,bob) ?

3 2 Call: female(tom) ?

3 2 Fail: female(tom) ?

2 2 Redo: parent(tom,bob) ?

2 2 Exit: parent(tom,liz) ?

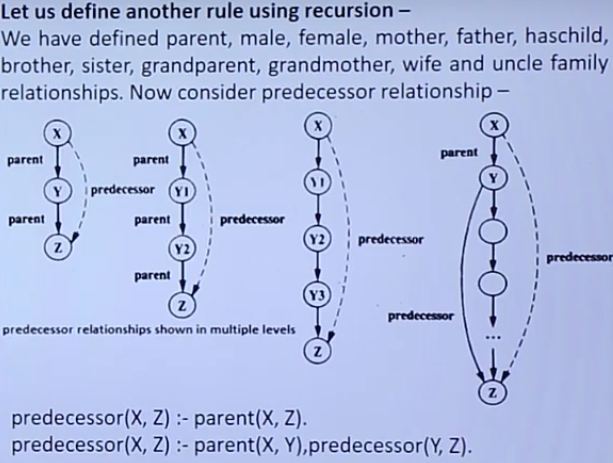
3 2 Call: female(tom) ?

3 2 Fail: female(tom) ?

1 1 Fail: mother(tom,\_42) ?

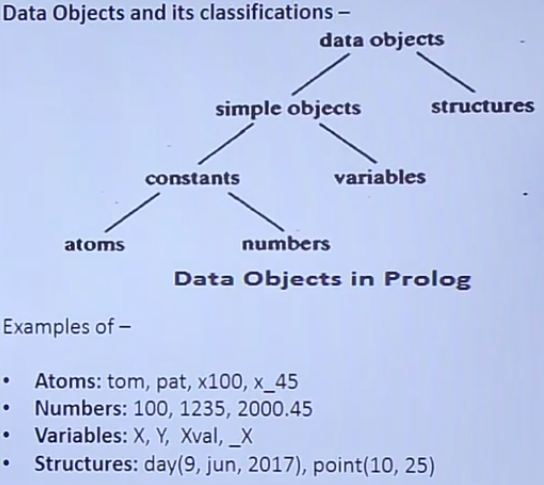
(31 ms) no

**Recursion in Family relationship:Prolog**



Check with trace predecessor(peter,X).

**Data Objects in prolog**

****

An **Atom** is a general-purpose name with no inherent meaning.

1. It can be strings of letters,digits and the underscore character,’\_’, starting with a lower-case letter. As example: azahar,b59,b\_59,b\_59AB,b\_x25,antara\_sarkar etc
2. Strings of special characters:

< ------ >

=======>

……..

.:.

::=

When using atoms of this form, some care is necessary because some strings of special characters already have a predefined meaning; an example is ‘:-‘.

1. Strings of characters enclosed in single quotes. This is useful if we want,

For example to have an atom that starts with a capital letter. By inclosing it in quotes we make it distinguishable from variables:

‘Rubai’

‘Ram\_Bahadur’

‘Hari Parsad’

**Numbers**

Numbers can be **floats** or **integers** (-16383 to 16383)

e.g. 100,4,-89,1020

e.g. 3.14159,-0.00062,450.18

**Variables**

Variables are strings of letters, digits and underscore characters. They start with an upper-case letter or an underscore character. They resemble placeholders for arbitrary terms E.g.

X

Sum

Memer\_name

Student\_list

Shoppinglist

\_a50

\_15

In a clause when a variable is used once only then the variable name can be replaced by so-called ‘anonymous’ variable, which is written as a single underscore’\_’ character. For example-

hasachild(X):-parent(X,Y). can also be written as,

hasachild(X):-parent(X,\_).

**Structures(compound terms)-functor followed by comma separated arguments**

Structures are objects that have multiple components. The components themselves can, in turn, be structures

For example the date can be viewed as a structure with three components: day,month,year.

Then the date 9th june,2017 can be written as :

date(9,june,2017)

Tree representation

Prolog representation functor:date, arguments 9,june,2017

The number of arguments is called the terms **arity**.

 An atom can be regarded as a compound term with **arity** zero

To represent a point, aline segment and a triangle using structure in prolog, consider the following statements.

P1:point(1,1)

P2:point(2,3)

S:seg(P1,P2): seg(point(1,1),point(2,3))

T: triangle(point(4,Z),point(6,4),point(7,1))

Special cases of compound terms:

* A *List* is an ordered collection of terms. It is denoted by square brackets with the terms separated by commas or in the case of the empty list, []. For example, [1,2,3] or [red,green,blue].
* *Strings*: A sequence of characters surrounded by quotes is equivalent to either a list of (numeric) character codes, a list of characters (atoms of length 1), or an atom depending on the value of the Prolog flag double\_quotes. For example, "to be, or not to be".

**Representation of Lists**

The list is a simple data Structure widely used in non-numeric programming. List can be either empty or non-empty in the first case, the list is simply written as a prolog atom, []. IN the second case, the list can be viewed as consisting of two things:

1. The first item, called the **head** of the list;
2. The remaining part of the list called the **tail**.

For example

[red,green,blue,whte,dark]

The head is red and the tail is the list

[green,blue,white,dark]

Let us consider L=[a,b,c]

If we write tail=[b,c] then we can also write L=[a | Tail]

Here the vertical bar separates the head and tail.

So, following list representations are also valid

[a,b,c]= [ x | [ b , c ] ] = [a, b | [c] ] = [a, b, c | [ ] ]

**Operation on list :**

1. **Membership**

Problem statement:

To check whether an object X is member of List L or not.

list\_member(X, L).

Where X is an object and L is a list

The goal list\_members(X,L) is true if X occurs in L.

For example,

list\_member(b,[a, b, c]) is true

list\_member(b, [a, [b,c] ] is not true but

list\_member([b,c],[a,[b,c]]) is true.

The program for membership relation can be based on the following observation:

X is a member of L if either

* 1. X is the head of L, or
  2. X is a member of the tail of L.

1. **Length Calculation**

Problem Statement:

Calculate the number of items of a given list.

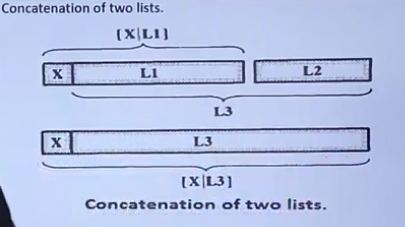
list\_length( List, N)

Which will count the elements in a list ‘List’ and instantiate ‘N’ to their number.

If the list is empty it has a length of 0

If the list is not empty then List =[Head | Tail]; then its length is equal to 1 plus the length of the tail Tail.

1. **Concatination of two lists**

****

1. **Operations in words**

**Problem statement**

We shall have to define a relation

inwords(List1,List2)

To translate a list of numbers between 0 to 9 to a list of the corresponding words. For Example:

inwords([3,5,1,3]), [three, five, one, three]).

Use the following as an auxillary relation:

interm(0,zero). interm(1,one). interm(2,two). … .

1. **Delete an Item**

We can have two cases

If X is the head of the list then the result after the deletion is the tail of the list

If X is in the tail then it is deleted from there

1. **\*Permutation**

Find all possible permutation of all the items of a given list

list\_permutation([a,b,c],P);

P: [a,b,c];

P: [a,c,b];

P: [b,a,c];

1. **\*Append**

list\_append(L1,L2,L3).

2.