A **fifth generation programming language** (abbreviated as **5GL**) is a programming language based on solving problems using constraints given to the program, rather than using an algorithm written by a programmer. Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.

While fourth-generation programming languages are designed to build specific programs, fifth-generation languages are designed to make the computer solve a given problem without the programmer.

Fifth-generation languages are used mainly in artificial intelligence research. Prolog, OPS5, and Mercury are examples of fifth-generation languages

Most notably, from 1982 to 1993 Japan put much research and money into their fifth generation computer systems project, hoping to design a massive computer network of machines using these tools.

However, as larger programs were built, the flaws of the approach became more apparent. It turns out that, given a set of constraints defining a particular problem, deriving an efficient algorithm to solve it is a very difficult problem in itself. This crucial step cannot yet be automated and still requires the insight of a human programmer

- Functional Programming Language such as Haskell is considered fifth Generation P.L.
- 5th Generation Languages were built on LISP.

- PROLOG is a higher ~~lan~~ level language compared to LISP.

Introduction to Propositional & Predicate Logic

Laws of Traditional Logic

Name	Law	Meaning
identity	A is A	A thing always equals itself
Contradiction	A is not both A and not-A	No statement is both True and False
Excluded Middle	A is not. not-A	Every statement is either True or False

Propositional Logic

- A proposition is a statement.
- The rules of deductive inference for proposition are given below, where letters P and Q represent any proposition.

- If $P \rightarrow Q$ and P is True
  Then Q is True (Modus Ponen)

- If $P \rightarrow Q$ and Q is False
  Then P is False (Modus Tollen)

- If P and Q are not both True and P is True, then Q is False (Exclusive OR)

- If either P or Q is True and P is not True then Q is True (Inclusive OR)

Propositional Logic recognizes four possible statement forms ~~such as~~ each expressing relationship between two classes (subject and predicate class).

Classes	Example
All S is P	All Americans are car drivers
No S is P	No Indian is two-headed
Some S is P	Some Americans are Taxi-drivers
some S is not P	Some taxi drivers are not aggressive

Look at the following statement

All Americans are car drivers

All Californians are Taxi drivers

Infer $\Rightarrow$ All Californians are car-drivers

The boolean algebra, Demorgan's law etc. can be used to derive a calculus of such Proposition.

e.g. 'Implication', $X \rightarrow Y$ can be represented as $!X + Y$ having a truth table as follows:

X	Y	$X \rightarrow Y$	$(!X + Y)$
T	T	T	
T	F	F	
F	T	T	
F	F	T	

The operators (connectives) used in Propositional calculus are following:

Name	Symbol
Not	!
AND	.
OR	+
Implies	$\rightarrow$

Writing English sentences in symbolic form

(1) If either Jerry takes Calculus or ken takes sociology, then larry will take English.

P: Jerry takes Calculus

Q: Ken takes Sociology

R: Larry will takes English

$$(P \vee Q) \rightarrow R$$

(2) The crop will be displayed if there is flood.

P: There is flood

Q: The crop will be destroyed

$P \rightarrow Q$ Means ?

(a) Q is necessary for P

(b) P is sufficient for Q

Ans. All are True

eg. If you are beheaded, Then you will die

Same as: You have been beheaded only if you die

(Example of Equivalence between

If P Then Q

&

P only if Q

)

Examples of PL sentences

- P means "It is hot." Hot
- Q means "It is humid." Humid
- R means "It is raining." Raining
- $(P \wedge Q) \rightarrow R$ (Hot ∧ Humid) → Raining.

  "If it is hot and humid, then it is raining"
- $Q \rightarrow P$ Humid → Hot.

  "If it is humid, then it is hot"
- A better way:

  Hot = "It is hot"

  Humid = "It is humid"

  Raining = "It is raining"

Proving things

- A **proof** is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.
- The last sentence is the **theorem** (also called goal or query) that we want to prove.
- Example for the "weather problem" given above.

1 Humid	Premise	"It is humid"
2 Humid→Hot	Premise	"If it is humid, it is hot"
3 Hot	Modus Ponens(1,2)	"It is hot"
4 (Hot∧Humid)→Rain	Premise	"If it's hot & humid, it's raining"
5 Hot∧Humid	And Introduction(1,~~2~~ 3)	"It is hot and humid"
6 Rain	Modus Ponens(4,5)	"It is raining"

Proving things

- A **proof** is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.
- The last sentence is the **theorem** (also called goal or query) that we want to prove.
- Example for the "weather problem" given above.

1 Humid	Premise	"It is humid"
2 Humid→Hot	Premise	"If it is humid, it is hot"
3 Hot	Modus Ponens(1,2)	"It is hot"
4 (Hot∧Humid)→Rain	Premise	"If it's hot & humid, it's raining"
5 Hot∧Humid	And Introduction(1,2)	"It is hot and humid"
6 Rain	Modus Ponens(4,5)	"It is raining"

Statement

All the declarative sentences to which it is possible to assign one and only one of two possible truth values (True or False) are called statements

Examples

1. Canada is a country
2. Moscow is the capital of Spain
3. This statement is False
4. $1 + 101 = 110$
5. Close the door

Truth values of above statements (if they are statements!)

1. True

2. False

3. Not a statement as we can not properly assign truth value to it.

4. False (In Decimal System.
True, if binary system)

5. Not a statement

Well-Formed Formula (WFF)

(Also called Sentence)

Rule 1 : A statement variable alone is a WFF

Rule 2 : If 'A' is a WFF, Then
$\neg A$ is also a WFF

Rule 3 : If A and B are WFF, Then
$(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$,
$(A \rightleftharpoons B)$ are WFFs

Rule 4 : A string of symbols containing the statement variables, connectives and parenthesis is WFF if and only if it can be obtained by finite applications of rules 1, 2 and 3.

Examples of WFF

$(P \rightarrow (P \vee Q))$,

$(P \rightarrow (Q \rightarrow R))$

Not WFF

$(P \rightarrow Q$

$(P \wedge Q) \rightarrow R)$ etc.

Special Form of WFF

WFF	Value	Name
$A + !A$	Always True	Tautology
$A_9 !A$	Always False	Contradiction

Semantic properties of WFF

Major properties of WFF are :

Valid : True under all interpretations.

Satisfiable : True under atleast one interpretation

Unsatisfiable : True under no interpretation

Inference Rules

Logical interence is used to create new sentences that follows logically from a given set of Predicate Calculus Sentences (KB).

Two properties of Inference rules :

I) Sound : An inference rule is sound if every sentence X produced by an inference rule operating on a KB logically follows from the KB (i.e. it doesn't create any contradiction

II) Complete : An inference rule is complete if it is able to produce every expression that

Sound rules of inference

- Here are some examples of sound rules of inference
  - *A rule is sound if its conclusion is true whenever the premise is true*
- Each can be shown to be sound using a truth table

RULE	PREMISE	CONCLUSION
Modus Ponens	$A, A \rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\neg\neg A$	A
Unit Resolution	$A \vee B, \neg B$	A
Resolution	$\mathbf{A \vee B, \neg B \vee C}$	$\mathbf{A \vee C}$

Soundness of modus ponens

A	B	$A \rightarrow B$	OK?
True	True	True	√
True	False	False	√
False	True	True	√
False	False	True	√

Clausal Form

- A literal is either a positive or negated atomic formula.
- Each clause is a set of literals connected by OR.

Negated literals are placed at the end of the clause.

Thus, a schematic view of the clause is:

$$P_1 + P_2 + \ldots + P_n + N_1 + N_2 + \ldots + N_m$$

If C, D, E & F are atomic formula then the clause $C + D + !E + !F$ states that either C or D is True if E and F are both true.

Horn Clause

It's a clause containing only one positive literal and may be written in slightly different form known as arrow notation.

Thus, a typical horn clause is:

C + !E + ! F + !G, which can be written as:

C ← E, F, G.

~~If~~ states

Here "←" is "Is Implied By"

Commas denote "AND" Operation

PROLOG is based on Horn Clause.

PROLOG Syntax

A Horn clause in which a conclusion is followed by zero or more conditions, is written in PROLOG as follows:

Conclusion :- Condition 1, Condition 2, ...
Condition N.

Here, ":-" is to be read as "if"
"," is AND
"." is a syntactic element.

The clause is read as :

"The conclusion is True if condition 1 and condition 2 and ... condition N are all True".

Example : We can assert a fact by writing a Horn clause without any condition

eg. smoker (john).

It's read as "John is a smoker"

- A rule is another form of Horn clause

eg has-sore-throat (john) :- smoker (john).

Horn sentence or Horn Clause

It has the form

$$P1 \wedge P2 \wedge P3 \ldots \wedge Pn \rightarrow Q$$

Or Equivalently

$$\neg P_1 \vee \neg P_2 \vee \neg P_3 \ldots \vee \neg P_n \vee Q$$

Programming in PROLOG consists of:

- Declaring some facts about objects and their relationship
- Defining some rules about objects and their relationship
- Asking questions about object and their relationship

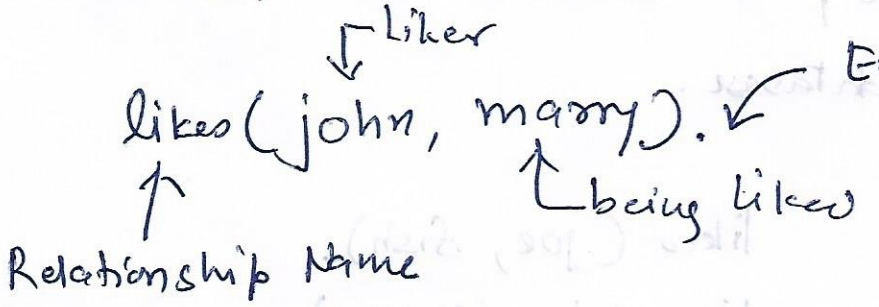
FACTS

Suppose, we want to tell PROLOG fact that "John likes Marry".

Objects - John, Marry

Relationship - likes

In PROLOG, it's written as:



Relationship Name, Object Name must begin with lower case letters.

Thus, likes (john, marry) ≠ likes (marry, john).

Example of facts:

Gold is Valuable	valuable (gold).
Jane is female	female (jane).
John owns gold	owns (john, gold).
John is the father of Marry	father (john, marry).
John gives the book to Marry	gives (john, book, marry).

- The name of the relationship that just precedes the bracket is called Predicate.
- In PROLOG, a collection of facts and rules is called database.

Example
Database I

likes (joe, fish).
likes (joe, marry).
likes (marry, book).
likes (john, book).

Let's Query PROLOG

?- likes (joe, money).

no

?- likes (joe, fish).

Yes

?- rises (sun, east).

no

This "NO" is not same as "False".

No in PROLOG means "Nothing matches in the Database".

II

human (socrates).

human (aristotle).

athenian (socrates).

goal: athenian (socrates).

Yes

goal: greek (socrates).

NO ← "Not Provable".

To ask what objects does John like, we need variables.

- A variable name in PROLOG begins with a Capital Letter.
- A variable in PROLOG is said to be bound (instantiated) when it stands for some object.

III

```
likes (john, flowers)
likes ( john, mary).
likes ( paul, mary).
```

goal: likes (john, X).

X = flowers

X = mary

2 solutions (Reply from Turbo PROLOG)

(Std. PROLOG / LPA WIN-PROLOG will give one soln at a time.

";" has to be pressed to get more soln).

Initially X is free.

- PROLOG searches database from the top.
- It matches first fact.
- X gets bound to flowers.
- Then X becomes free (uninstantiated) and it gets bound to "marry".

```
?- likes(X, marry).

X = john
X = paul
2 solutions
```

Conjunctions

"Do John and Marry like each other?"

Solution I: Ask to PROLOG – "Do John like Marry?"

If answer is YES.

Then again ask – "Do Marry like John?"

If answer is YES, Then answer to the original question is YES.

Entire question is asked in single query through Conjunction represented as "," .

goal : likes (marry, john) , likes (john, marry).

Subgoal 1 Subgoal 2

To ask PROLOG

" Is there anything which John and Marry both like ?"

goal : like (marry, X) , like (john, X).

IV

likes (marry, food).
likes (marry, wine).
likes (john, wine).
likes (john, marry).

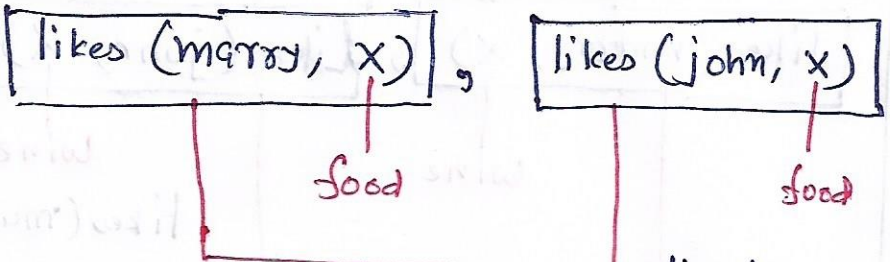
?- likes (marry, X) , likes (john, X).

food food

likes (marry, food).
likes (marry, wine).
likes (john, wine).
likes (john, marry).

1. The first goal succeeds, instantiating X to food.
2. Next, attempt to satisfy the second goal

goal: likes(marry, X), likes(john, X)

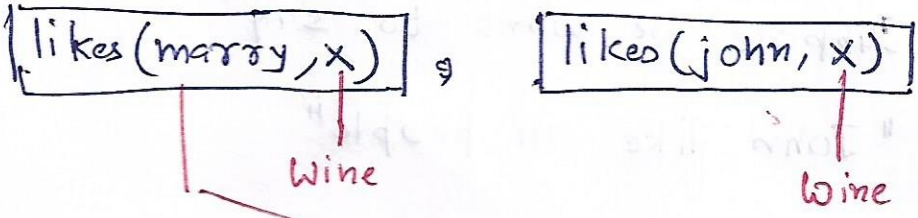


3. The second goal fails.

4. Next backtrack; forget the previous X, and attempt to re-satisfy the first goal

likes(marry, food).
likes(marry, wine).
likes(john, wine).
likes(john, marry).

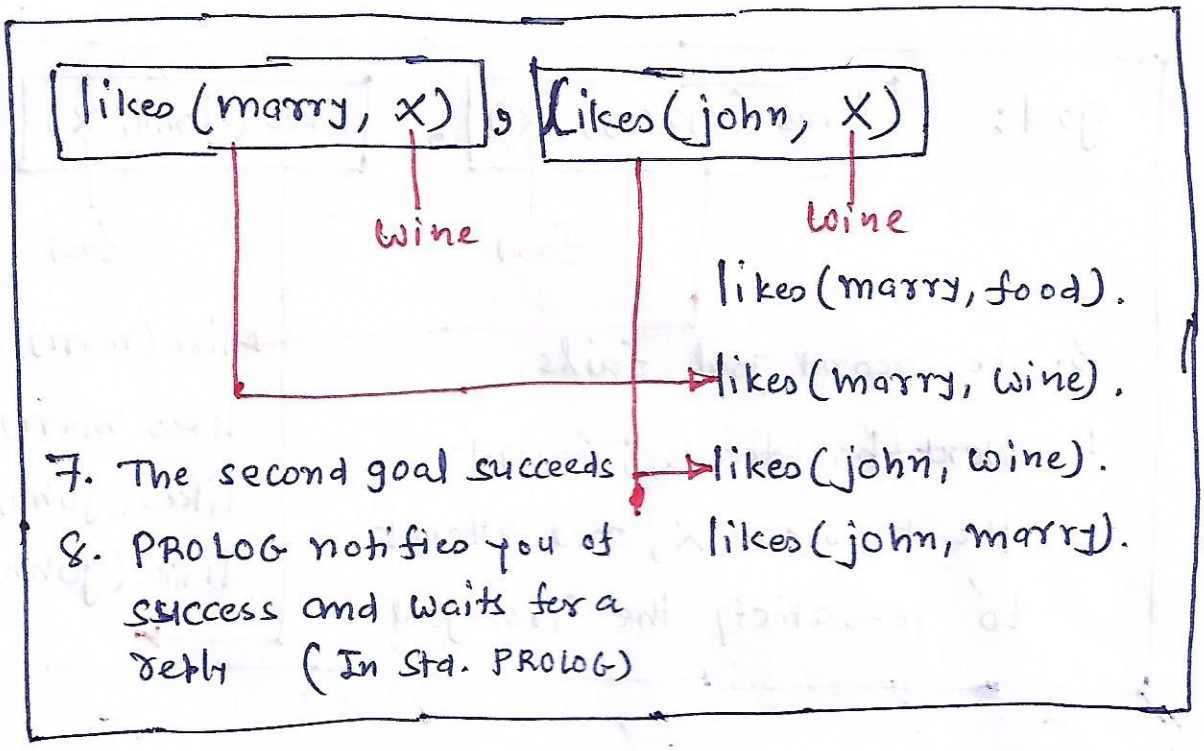
goal: likes(marry, X), likes(john, X)



5. The first goal succeeds again, instantiating X to wine.

likes(marry, food).
likes(marry, wine).
likes(john, wine).
likes(john, marry).

6. Next attempt to satisfy the second goal



Rules

Suppose we want to say

"John like all people"

It's reframed as:

"John likes an object provided it is a person."

This is use of Rules in PROLOG

likes (john, X) :- person (X).

More examples :

- John likes anyone who likes wine.

```
likes (john, X)  :-  likes (X, wine).
```

- John likes any female who likes wine.

```
likes( john, X) :- female (X), likes (X, wine).
```

- Defining sisterof (X, Y) predicate

```
sisterof (X, Y) :- female (X), parents(X, M,F),
                   parents (Y, M, F).
```

V Database

```
male (albert).
male (edward).
female (alice).
female (victoria).
parents (edward, victoria, albert).
parents (alice, victoria, albert).
sisterof (X, Y) :- female (X), parents (X, M, F),
```