ARTIFICIAL INTELLIGENCE

19CS507

UNIT III - Symbolic Logic

Propositional logic in Artificial intelligence

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions.
- A proposition is a declarative statement which is either true or false.
- It is a technique of knowledge representation in logical and mathematical form.

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) 3+3= 7(False proposition)
4. d) 5 is a prime number.

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and logical connectives.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called tautology, and it is also called a valid sentence.
- A proposition formula which is always false is called Contradiction.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "Where is Rohini", "How are you", "What is your name", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

a. Atomic Propositions
b. Compound propositions

- Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

1. a) 2+2 is 4, it is an atomic proposition as it is a true fact.
2. b) "The Sun is cold" is also a proposition as it is a false fact.

Compound proposition:

Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

1. a) "It is raining today, and street is wet."
2. b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

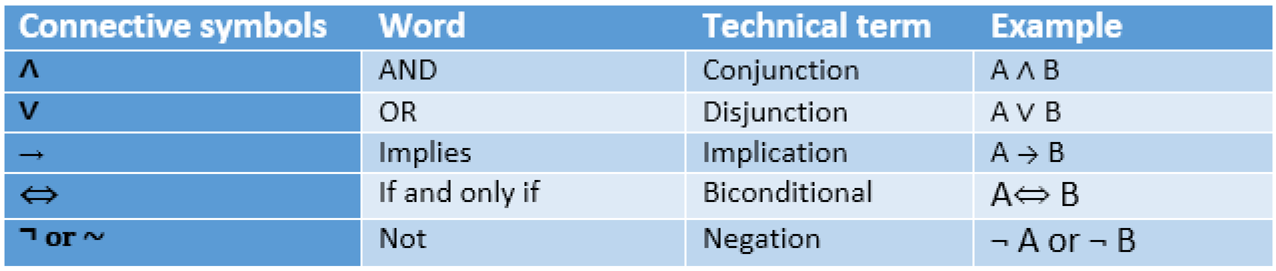
- Logical connectives are used to connect two simpler propositions or representing a sentence logically.
- We can create compound propositions with the help of logical connectives.

There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.

2. **Conjunction:** A sentence which has ∧ connective such as, P ∧ Q is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as,
   P= Rohan is intelligent,
   Q= Rohan is hardworking. → P∧ Q.
3. **Disjunction:** A sentence which has ∨ connective, such as P ∨ Q. is called disjunction, where P and Q are the propositions.
   Example: "Ritika is a doctor or Engineer",
   Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as P ∨ Q.
4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
   If it is raining, then the street is wet.
   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. **Biconditional:** A sentence such as P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive
   P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:



Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Truth Table:

- In propositional logic, we need to know the truth values of propositions in all possible scenarios.
- We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called Truth table.

Following are the truth table for all logical connectives:

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

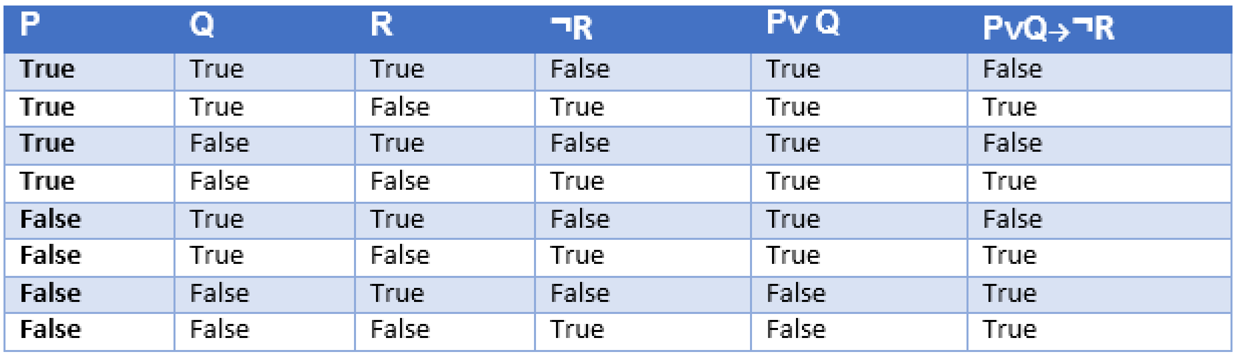
P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.



P	Q	R	¬R	Pv Q	PvQ→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

First-Order Logic in Artificial intelligence

- In the topic of Propositional logic, we have seen that how to represent statements using propositional logic.
- But unfortunately, in propositional logic, we can only represent the facts, which are either true or false.
- PL is not sufficient to represent the complex sentences or natural language statements.
- The propositional logic has very limited expressive power.
- Consider the following sentence, which we cannot represent using PL logic.

  - "Some humans are intelligent", or
  - "Sachin likes cricket."

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
  - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus,
  - **Relations:** It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
  - **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
  a. Syntax
  b. Semantics

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....

Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as Predicate (term1, term2, ..., term n).

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

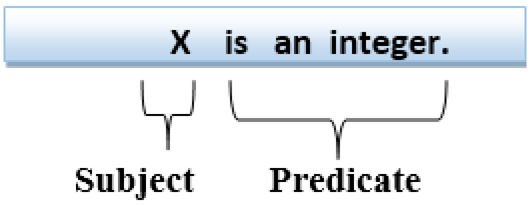
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject**: Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
  a. Universal Quantifier, (for all, everyone, everything)
  b. Existential quantifier, (for some, at least one).

Universal Quantifier:

- Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.
- The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

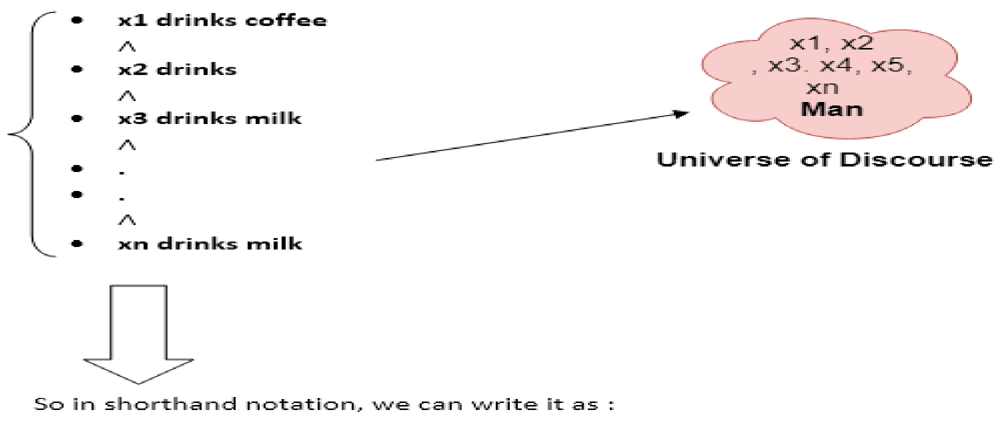
If x is a variable, then ∀x is read as:

- For all x
- For each x
- For every x.

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

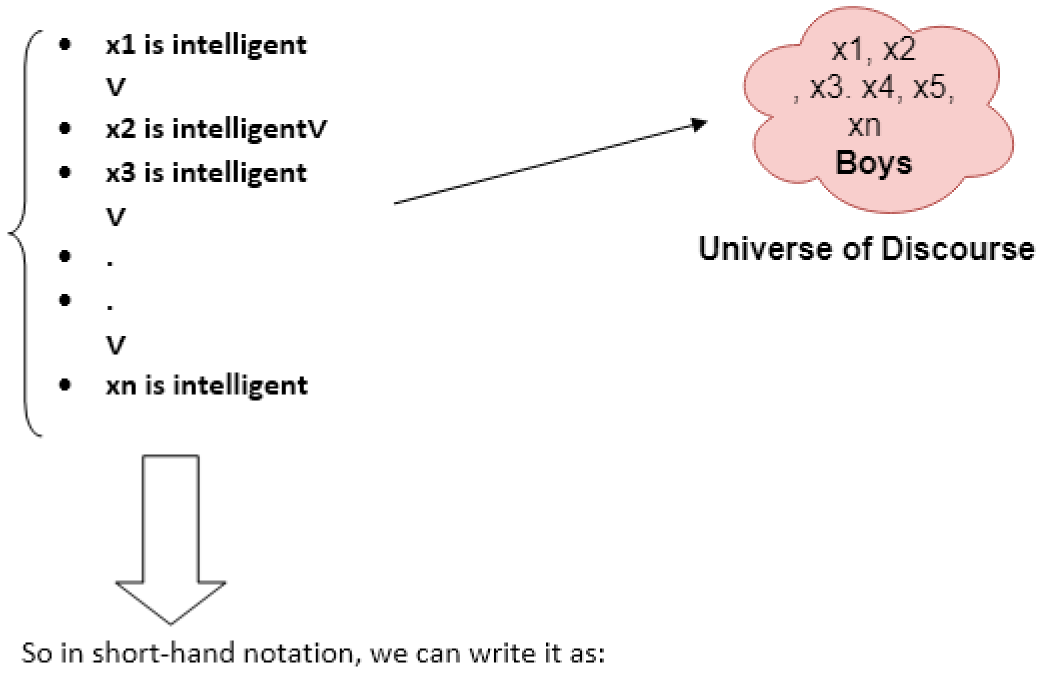
- Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.
- It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- There exists a 'x.'
- For some 'x.'
- For at least one 'x.'

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "fly(bird)."
And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "respect(x, y)," where x=man, and y= parent.
Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "play(x, y)," where x= boys, and y= game. Since there are some boys so we will use ∃, and it will be represented as:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "like(x, y)," where x= student, and y= subject.
Since there are not all students, so we will use ∀ with negation, so following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. **Only one student failed in Mathematics.**
In this question, the predicate is "failed(x, y)," where x= student, and y= subject.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

$\exists$(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

- The quantifiers interact with variables which appear in a suitable way.

There are two types of variables in First-order logic which are given below:

Free Variable:

- A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable:

- A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Representing INSTANCE and ISA Relationships

- Specific attributes instance and isa play important role particularly in a useful form of reasoning called property inheritance.

- The predicates instance and isa explicitly captured the relationships they are used to express, namely class membership and class inclusion.

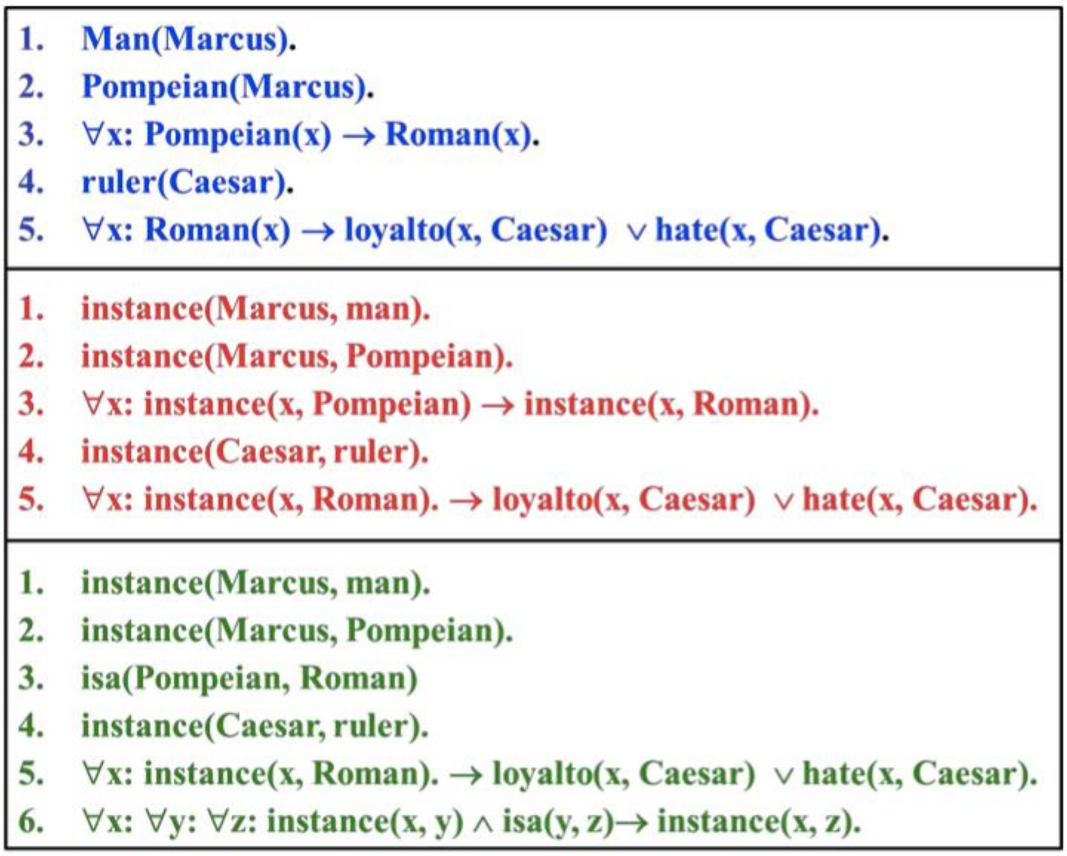
- Below figure shows the first five sentences of the last section represented in logic in three different ways.

- The first part of the figure contains the representations we have already discussed.

- In these representations, class membership is represented with unary predicates (such as Roman), each of which corresponds to a class.

- Asserting that P(x) is true is equivalent to asserting that x is an instance (or element) of P.

- The second part of the figure contains representations that use the instance predicate explicitly.



1. Man(Marcus).
2. Pompeian(Marcus).
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x).
4. ruler(Caesar).
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

---

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. $\forall x$: instance(x, Pompeian) $\rightarrow$ instance(x, Roman).
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

---

1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. isa(Pompeian, Roman)
4. instance(Caesar, ruler).
5. $\forall x$: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).
6. $\forall x$: $\forall y$: $\forall z$: instance(x, y) $\wedge$ isa(y, z)$\rightarrow$ instance(x, z).

- The predicate instance is a binary one, whose first argument is an object and whose second argument is a class to which the object belongs.

- But these representations do not use an explicit isa predicate.

- Instead, subclass relationships, such as that between Pompeians and Romans, are described as shown in sentence 3.

- The implication rule states that if an object is an instance of the subclass Pompeian then it is an instance of the superclass Roman.

- Note that this rule is equivalent to the standard set-theoretic definition of the subclass-superclass relationship.

- The third part contains representations that use both the instance and isa predicates explicitly.

- The use of the isa predicate simplifies the representation of sentence 3, but it requires that one additional axiom (shown here as number 6) be provided.

Computable Functions and Predicates

A computable predicate may include computable functions such as +, -, *, etc. For example, gt(x-y,10) →bigger(x) contains the computable predicate gt which performs the greater than function. Note that this computable predicate uses the computable function subtraction.

Predicate Logic:

- Predicate Logic deals with predicates, which are propositions, consist of variables.

Predicate Logic - Definition

- A predicate is an expression of one or more variables determined on some specific domain.

- A predicate with variables can be made a proposition by either authorizing a value to the variable or by quantifying the variable.

The following are some examples of predicates.

- Consider E(x, y) denote "x = y"
- Consider X(a, b, c) denote "a + b + c = 0"
- Consider M(x, y) denote "x is married to y."

Quantifier:

- The variable of predicates is quantified by quantifiers.
- There are two types of quantifier in predicate logic - Existential Quantifier and Universal Quantifier.

Existential Quantifier:

- If p(x) is a proposition over the universe U.
- Then it is denoted as ∃x p(x) and read as "There exists at least one value in the universe of variable x such that p(x) is true.
- The quantifier ∃ is called the existential quantifier.
- There are several ways to write a proposition, with an existential quantifier, i.e.,
- (∃x∈A)p(x) or ∃x∈A such that p (x) or (∃x)p(x) or p(x) is true for some x ∈A.

Universal Quantifier:

- If p(x) is a proposition over the universe U.
- Then it is denoted as ∀x,p(x) and read as "For every x∈U,p(x) is true."
- The quantifier ∀ is called the Universal Quantifier.
- There are several ways to write a proposition, with a universal quantifier.

∀x∈A,p(x) or p(x), ∀x ∈A Or ∀x,p(x) or p(x) is true for all x ∈A.

Negation of Quantified Propositions:

- When we negate a quantified proposition, i.e., when a universally quantified proposition is negated, we obtain an existentially quantified proposition,and when an existentially quantified proposition is negated, we obtain a universally quantified proposition.
- The two rules for negation of quantified proposition are as follows. These are also called DeMorgan's Law.

Propositions with Multiple Quantifiers:

- The proposition having more than one variable can be quantified with multiple quantifiers.
- The multiple universal quantifiers can be arranged in any order without altering the meaning of the resulting proposition.
- Also, the multiple existential quantifiers can be arranged in any order without altering the meaning of the proposition.
- The proposition which contains both universal and existential quantifiers, the order of those quantifiers can't be exchanged without altering the meaning of the proposition, e.g., the proposition $\exists x \forall y\ p(x,y)$ means "There exists some x such that p (x, y) is true for every y."

Syntax & Semantics of FOPL

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
    - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
    - **Relations:** It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
    - **Function:** Father of, best friend, third inning of, end of, ...
- **As a natural language, first-order logic also has two main parts:**
    a. Syntax
    b. Semantics

Syntax of First-Order logic:

- The syntax of FOL determines which collection of symbols is a logical expression in first-order logic.
- The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as Predicate (term1, term2, ..., term n).

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

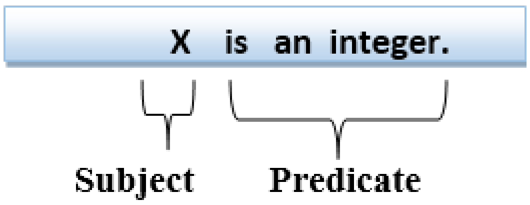
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
  a. Universal Quantifier, (for all, everyone, everything)
  b. Existential quantifier, (for some, at least one).

Universal Quantifier:

- Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.
- The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

Normal Forms

Normal Forms

- The problem of finding whether a given statement is tautology or contradiction or satisfiable in a finite number of steps is called the Decision Problem.
- For Decision Problem, construction of truth table may not be practical always.
- We consider an alternate procedure known as the reduction to normal forms.

There are two such forms:

1. Disjunctive Normal Form (DNF)
2. Conjunctive Normal Form

Disjunctive Normal Form (DNF):

If p, q are two statements, then "p or q" is a compound statement, denoted by p ∨ q and referred as the disjunction of p and q. The disjunction of p and q is true whenever at least one of the two statements is true, and it is false only when both p and q are false

p	**q**	**p ∨ q**
T	T	T
T	F	T
F	T	T
F	F	F

Example: - if p is "4 is a positive integer" and q is "$\sqrt{\boxed{5}}$ is a rational number", then p ∨ q is true as statement p is true, although statement q is false.

Conjunctive Normal Form:

If p, q are two statements, then "p and q" is a compound statement, denoted by p ∧ q and referred as the conjunction of p and q. The conjunction of p and q is true only when both p and q are true, otherwise, it is false

p	**q**	**p ∧ q**
T	T	T
T	F	F
F	T	F
F	F	F

Example: if statement p is "6<7" and statement q is "-3>-4" then the conjunction of p and q is true as both p and q are true statements.

Unification &Resolution

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\Psi_1\sigma = \Psi_2\sigma$, then it can be expressed as UNIFY(Ψ_1, Ψ_2).
- Example: Find the MGU for Unify{King(x), King(John)}

Let Ψ_1 = King(x), Ψ_2 = King(John), Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, P(x, y), and P(a, f(z)).

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x, y)......... (i)

P(a, f(z))......... (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as a/x and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: [a/x, f(z)/y].

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Resolution in FOL

Resolution

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions.
- It was invented by a Mathematician John Alan Robinson in the year 1965.
- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements.
- Unification is a key concept in proofs by resolutions.
- Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

Clause:

- Disjunction of literals (an atomic sentence) is called a clause. It is also known as a unit clause.

Conjunctive Normal Form:

- A sentence represented as a conjunction of clauses is said to be conjunctive normal form or CNF.

The resolution inference rule:

- The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \ldots\ldots \vee l_k, \qquad m_1 \vee \ldots\ldots \vee m_n}{\mathrm{SUBST}(\theta, l_1 \vee \ldots\ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots\ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots\ldots \vee m_n)}$$

- Where l_i and m_j are complementary literals.
- This rule is also called the binary resolution rule because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)]   and   [¬ Loves(a, b) V ¬Kills(a, b)]

Where two complimentary literals are: Loves (f(x), x) and ¬ Loves (a, b)

steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

a. John likes all kind of food.

b. Apple and vegetable are food

c. Anything anyone eats and not killed is food.

d. Anil eats peanuts and still alive

e. Harry eats everything that Anil eats.

Prove by resolution that:

f. John likes peanuts.

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

a. ∀x: food(x) → likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) Λ alive(Anil).

e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**

g. ∀x: alive(x) →¬ killed(x)

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

- Eliminate all implication (→) and rewrite
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).
- Move negation (¬)inwards and rewrite

. ∀x ¬ food(x) V likes(John, x)
  a. food(Apple) Λ food(vegetables)
  b. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  c. eats (Anil, Peanuts) Λ alive(Anil)
  d. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  e. ∀x ¬killed(x)] V alive(x)
  f. ∀x ¬ alive(x) V ¬ killed(x)
  g. likes(John, Peanuts).
- Rename variables or standardize variables

. ∀x ¬ food(x) V likes(John, x)

a. food(Apple) Λ food(vegetables)
b. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
c. eats (Anil, Peanuts) Λ alive(Anil)
d. ∀w¬ eats(Anil, w) V eats(Harry, w)
e. ∀g ¬killed(g)] V alive(g)
f. ∀k ¬ alive(k) V ¬ killed(k)
g. likes(John, Peanuts).

- Eliminate existential instantiation quantifier by elimination.
  In this step, we will eliminate existential quantifier ∃, and this process is known as Skolemization. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.
- Drop Universal quantifiers.
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

. ¬ food(x) V likes(John, x)

a. food(Apple)
b. food(vegetables)
c. ¬ eats(y, z) V killed(y) V food(z)
d. eats (Anil, Peanuts)
e. alive(Anil)
f. ¬ eats(Anil, w) V eats(Harry, w)
g. killed(g) V alive(g)
h. ¬ alive(k) V ¬ killed(k)
i. likes(John, Peanuts).

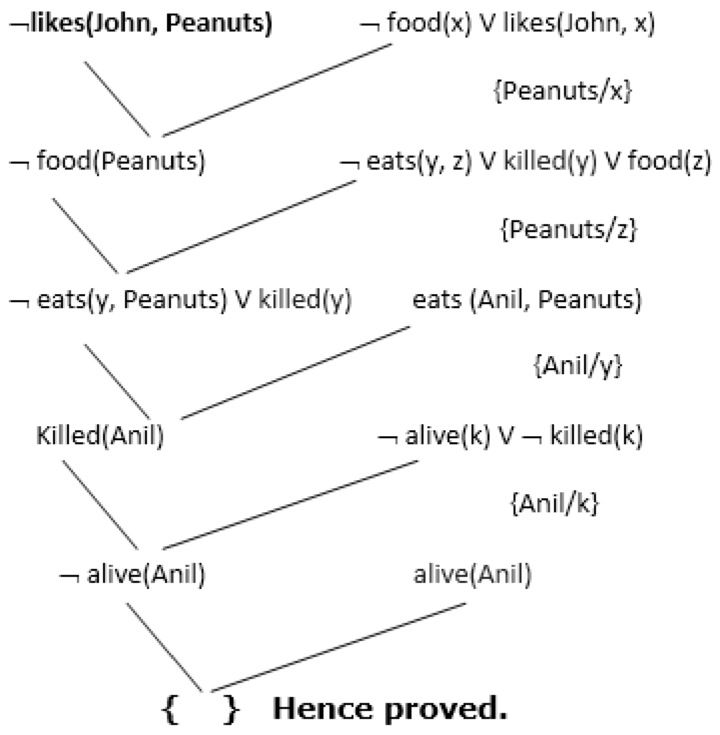
- Distribute conjunction ∧ over disjunction ¬.
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, ¬likes(John, Peanuts) , and likes(John, x) get resolved(canceled) by substitution of {Peanuts/x}, and we are left with ¬ food(Peanuts)
- In the second step of the resolution graph, ¬ food(Peanuts) , and food(z) get resolved (canceled) by substitution of { Peanuts/z}, and we are left with ¬ eats(y, Peanuts) V killed(y) .
- In the third step of the resolution graph, ¬ eats(y, Peanuts) and eats (Anil, Peanuts) get resolved by substitution {Anil/y}, and we are left with Killed(Anil) .

- In the fourth step of the resolution graph, Killed(Anil) and ¬ killed(k) get resolve by substitution {Anil/k}, and we are left with ¬ alive(Anil) .
- In the last step of the resolution graph ¬ alive(Anil) and alive(Anil) get resolved.

Representation Using Rules

Knowledge-Based System

- A knowledge-based system is a system that uses artificial intelligence techniques to store and reason with knowledge. The knowledge is typically represented in the form of rules or facts, which can be used to draw conclusions or make decisions.
- One of the key benefits of a knowledge-based system is that it can help to automate decision-making processes. For example, a knowledge-based system could be used to diagnose a medical condition, by reasoning over a set of rules that describe the symptoms and possible causes of the condition.
- Another benefit of knowledge-based systems is that they can be used to explain their decisions to humans. This can be useful, for example, in a customer service setting, where a knowledge-based system can help a human agent understand why a particular decision was made.
- Knowledge-based systems are a type of artificial intelligence and have been used in a variety of applications including medical diagnosis, expert systems, and decision support systems.

Knowledge-Based System in Artificial Intelligence:

- An intelligent agent needs knowledge about the real world to make decisions and reasoning to act efficiently.
- Knowledge-based agents are those agents who have the capability of maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take action. These agents can represent the world with some formal representation and act intelligently.

Why use a knowledge base?

- A knowledge base inference is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

- Inference means deriving new sentences from old. The inference-based system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. The inference system applies logical rules to the KB to deduce new information.
- The inference system generates new facts so that an agent can update the KB.

An inference system works mainly in two rules which are given:

- Forward chaining
- Backward chaining

Various levels of knowledge-based agents:

A knowledge-based agent can be viewed at different levels which are given below:

1. Knowledge level:

- Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are.
- With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Logical level:

- At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics.
- At the logical level, an encoding of knowledge into logical sentences occurs.
- At the logical level we can expect to the automated taxi agent to reach to the destination B.

3. Implementation level:

- This is the physical representation of logic and knowledge.
- At the implementation level agent perform actions as per logical and knowledge level.
- At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.
- Knowledge-based agents have explicit representation of knowledge that can be reasoned.
- They maintain internal state of knowledge, reason over it, update it and perform actions accordingly.
- These agents act intelligently according to requirements.

- Knowledge based agents give the current situation in the form of sentences.
- They have complete knowledge of current situation of mini-world and its surroundings. These agents manipulate knowledge to infer new things at "Knowledge level".

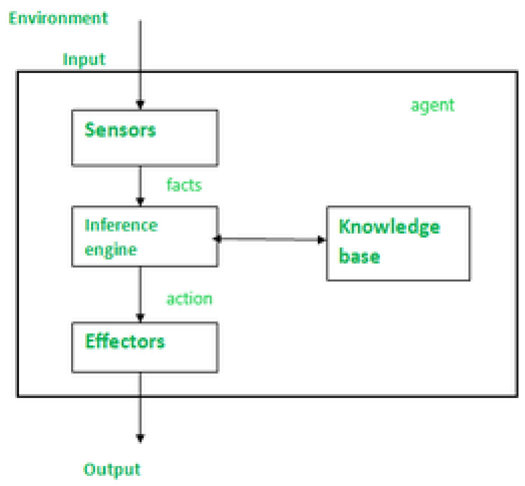
knowledge-based system has following features:

- Knowledge base (KB): It is the key component of a knowledge-based agent.
- These deal with real facts of world.
- It is a mixture of sentences which are explained in knowledge representation language.

Inference Engine(IE): It is knowledge-based system engine used to infer new knowledge in the system.

Actions performed by an agent:

- Inference System is used when we want to update some information (sentences) in Knowledge-Based System and to know the already present information.
- This mechanism is done by TELL and ASK operations.
- They include inference i.e. producing new sentences from old.
- Inference must accept needs when one asks a question to KB and answer should follow from what has been Told to KB.
- Agent also has a KB, which initially has some background Knowledge.
- Whenever, agent program is called, it performs some actions.



A Knowledge based system behavior can be designed in following approaches:-

Declarative Approach:

- In this beginning from an empty knowledge base, the agent can TELL sentences one after another till the agent has knowledge of how to work with its environment.
- This is known as the declarative approach.
- It stores required information in empty knowledge-based system.

This converts required behaviors directly into program code in empty knowledge-based system. It is a contrast approach when compared to Declarative approach. In this by coding behavior of system is designed .

Natural Deduction

- The natural deduction system is essentially a Frege system with an additional rule which allows to prove an implication $\varphi \to \psi$ by taking φ as an assumption and deriving ψ.
- The fact that this rule can be simulated in a Frege system is called the deduction theorem and the rule is called the deduction rule.

Natural Deduction:

- Natural Deduction (ND) is a common name for the class of proof systems composed of simple and self-evident inference rules based upon methods of proof and traditional ways of reasoning that have been applied since antiquity in deductive practice.

- The first formal ND systems were independently constructed in the 1930s by G. Gentzen and S. Jaśkowski and proposed as an alternative to Hilbert-style axiomatic systems.

- Gentzen introduced a format of ND particularly useful for theoretical investigations of the structure of proofs. Jaśkowski instead provided a format of ND more suitable for practical purposes of proof search.

- Since then many other ND systems were developed of apparently different character.

- What is it that makes them all ND systems despite the differences in the selection of rules, construction of proof, and other features? First of all, in contrast to proofs in axiomatic systems, proofs in ND systems are based on the use of assumptions which are freely introduced but discharged under some conditions.

- Moreover, ND systems use many inference rules of simple character which show how to compose and decompose formulas in proofs.

- Finally, ND systems allow for the application of different proof-search strategies.

- Thanks to these features proofs in ND systems tend to be much shorter and easier to construct than in axiomatic or tableau systems.

- These properties of ND make them one of the most popular ways of teaching logic in elementary courses.

- In addition to its educational value, ND is also an important tool in proof-theoretical investigations and in the philosophy of meaning (specifically, of the meaning of logical constants).

- This article focuses on the description of the main types of ND systems and briefly mentions more advanced issues concerning normal proofs and proof-theoretical semantics.

Natural Deduction:

- Testing whether a proposition is a tautology by testing every possible truth assignment is expensive—there are exponentially many.
- We need a deductive system, which will allow us to construct proofs of tautologies in a step-by-step fashion.
- The system we will use is known as natural deduction.

- The system consists of a set of rules of inference for deriving consequences from premises.
- One builds a proof tree whose root is the proposition to be proved and whose leaves are the initial assumptions or axioms (for proof trees, we usually draw the root at the bottom and the leaves at the top).
- For example, one rule of our system is known as modus ponens.
- Intuitively, this says that if we know P is true, and we know that P implies Q, then we can conclude Q.

$$\frac{P \qquad P \Rightarrow Q}{Q} \text{ (modus ponens)}$$

- The propositions above the line are called premises; the proposition below the line is the conclusion.
- Both the premises and the conclusion may contain metavariables (in this case, P and Q) representing arbitrary propositions.
- When an inference rule is used as part of a proof, the metavariables are replaced in a consistent way with the appropriate kind of object (in this case, propositions).

Most rules come in one of two flavors:

- introduction or elimination rules. Introduction rules introduce the use of a logical operator, and elimination rules eliminate it.
- Modus ponens is an elimination rule for ⇒. On the right-hand side of a rule, we often write the name of the rule.
- This is helpful when reading proofs.
- In this case, we have written (modus ponens). We could also have written (⇒-elim) to indicate that this is the elimination rule for ⇒.

Rules for Conjunction:

Conjunction (∧) has an introduction rule and two elimination rules:

$$\frac{P \qquad Q}{P \wedge Q} \text{ (∧-intro)} \qquad \frac{P \wedge Q}{P} \text{ (∧-elim-left)} \qquad \frac{P \wedge Q}{Q} \text{ (∧-elim-right)}$$

Rule for T:

The simplest introduction rule is the one for T. It is called "unit". Because it has no premises, this rule is an axiom: something that can start a proof.

$$\frac{}{T}\ (\text{unit})$$

Rules for Implication:

- In natural deduction, to prove an implication of the form P ⇒ Q, we assume P, then reason under that assumption to try to derive Q. If we are successful, then we can conclude that P ⇒ Q.
- In a proof, we are always allowed to introduce a new assumption P, then reason under that assumption.
- We must give the assumption a name; we have used the name x in the example below. Each distinct assumption must have a different name.

$$\frac{}{[x : P]}\ (\text{assum})$$

- Because it has no premises, this rule can also start a proof. It can be used as if the proposition P were proved.
- The name of the assumption is also indicated here.
- However, you do not get to make assumptions for free! To get a complete proof, all assumptions must be eventually discharged.
- This is done in the implication introduction rule. This rule introduces an implication P ⇒ Q by discharging a prior assumption [x : P].
- Intuitively, if Q can be proved under the assumption P, then the implication P ⇒ Q holds without any assumptions.
- We write x in the rule name to show which assumption is discharged.
- This rule and modus ponens are the introduction and elimination rules for implications.
- Reductio ad absurdum (RAA) is an interesting rule.
- It embodies proofs by contradiction.

- It says that if by assuming that P is false we can derive a contradiction, then P must be true. The assumption x is discharged in the application of this rule.
- This rule is present in classical logic but not in intuitionistic (constructive) logic.
- In intuitionistic logic, a proposition is not considered true simply because its negation is false.

Excluded Middle:

Another classical tautology that is not intuitionistically valid is the the law of the excluded middle, $P \vee \neg P$. We will take it as an axiom in our system. The Latin name for this rule is tertium non datur, but we will call it magic.

Structured Representations of Knowledge

- Knowledge representation is a fundamental concept in artificial intelligence (AI) that involves creating models and structures to represent information and knowledge in a way that intelligent systems can use.
- Objects, events, performance, meta-knowledge, facts, and knowledge-base are the different kinds of knowledge.

STRUCTURED REPRESNTATION OF KNOWLEDGE:

- Representing knowledge using logical formalism, like predicate logic, has several advantages.
- They can be combined with powerful inference mechanisms like resolution, which makes reasoning with facts easy.
- But using logical formalism complex structures of the world, objects and their relationships, events, sequences of events etc. can not be described easily.

A good system for the representation of structured knowledge in a particular domain should posses the following four properties:

(i) **Representational Adequacy:-** The ability to represent all kinds of knowledge that are needed in that domain.

(ii) **Inferential Adequacy :-** The ability to manipulate the represented structure and infer new structures.

(iii) **Inferential Efficiency:-** The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.

(iv) **Acquisitional Efficiency :-** The ability to acquire new information easily, either by direct insertion or by program control.

The techniques that have been developed in AI systems to accomplish these objectives fall under two categories:

1. **Declarative Methods:-** In these knowledge is represented as static collection of facts which are manipulated by general procedures. Here the facts need to be stored only one and they can be used in any number of ways. Facts can be easily added to declarative systems without changing the general procedures.

3. **Procedural Method:-**
    - In these knowledge is represented as procedures. Default reasoning and probabilistic reasoning are examples of procedural methods. In these, heuristic knowledge of "How to do things efficiently "can be easily represented.

    - In practice most of the knowledge representation employ a combination of both.
    - Most of the knowledge representation structures have been developed to handle programs that handle natural language input.
    - One of the reasons that knowledge structures are so important is that they provide a way to represent information about commonly occurring patterns of things .

- such descriptions are some times called schema.
- One definition of schema is
- “Schema refers to an active organization of the past reactions, or of past experience, which must always be supposed to be operating in any well adapted organic response”.

- By using schemas, people as well as programs can exploit the fact that the real world is not random. There are several types of schemas that have proved useful in AI programs. They include

(i**) Frames:-** Used to describe a collection of attributes that a given object possesses (**eg: description of a chair).**

(**ii) Scripts:-** Used to describe common sequence of events **(eg:- a restaurant scene).**

(iii) **Stereotypes :-** Used to described characteristics of people.

(iv**) Rule models:-** Used to describe common features shared among a set of rules in a production system.

- Frames and scripts are used very extensively in a variety of AI programs.
- Before selecting any specific knowledge representation structure, the following issues have to be considered.

(i) The basis properties of objects , if any, which are common to every problem domain must be identified and handled appropriately.

(ii) The entire knowledge should be represented as a good set of primitives.

(iii) Mechanisms must be devised to access relevant parts in a large knowledge base.

Semantic Nets

- A semantic network is a knowledge structure that depicts how concepts are related to one another and illustrates how they interconnect.
- Semantic networks use artificial intelligence (AI) programming to mine data, connect concepts and call attention to relationships.

SEMANTIC NETWORK EXAMPLES:

Some of the examples of Semantic Networks are:

- **WordNet:** A lexical database of English that groups English words into sets of synonyms (synsets), provides definitions, and records semantic relations between them.
- **Gellish Model:** It is a formal language that is defined as a network of relations between concepts and names of concepts.
- **Logical Descriptions:** Semantic Networks can also be used to represent logical descriptions like Charles Sanders Peirce’s existential graphs or John F. Sowa's conceptual graphs.

COMPONENTS OF SEMANTIC NETWORKS:

Semantic Networks can further be defined by specifying its fundamental components, which are:

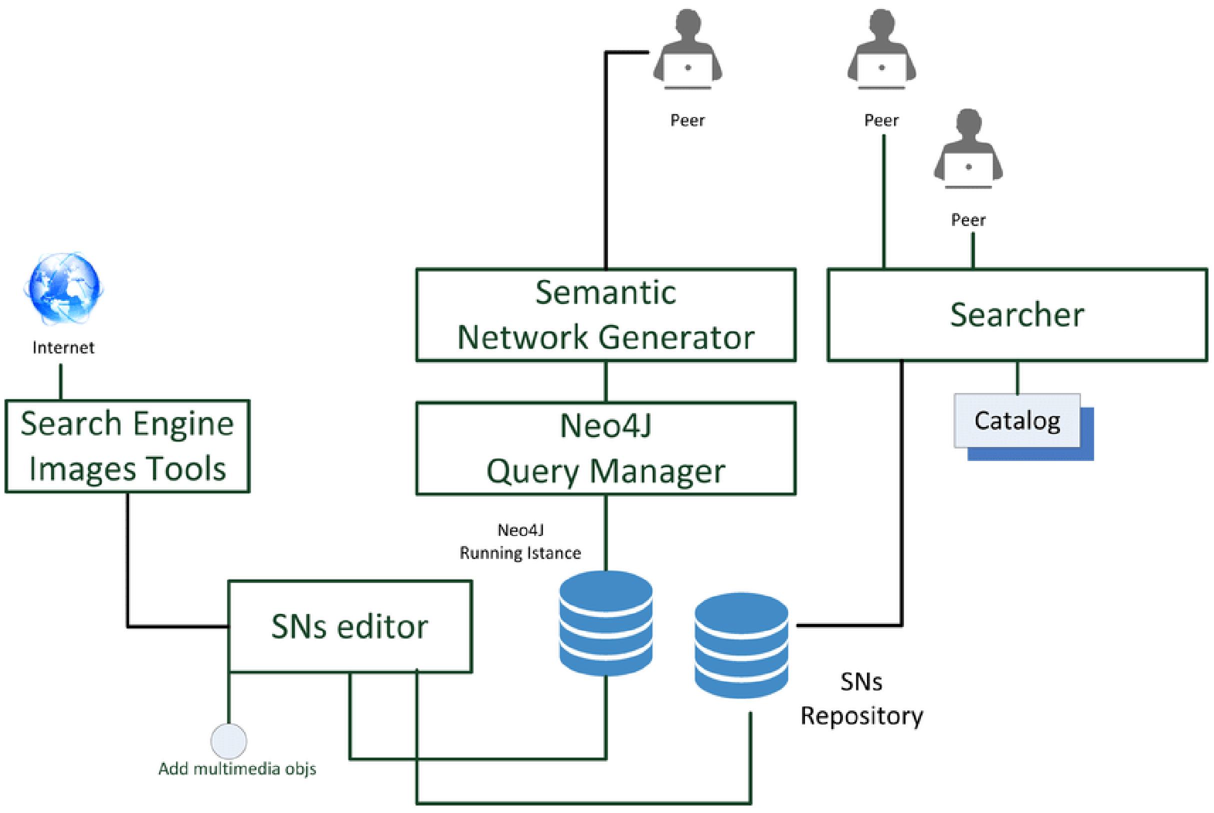
- **Lexical Components - Consists of:**
    - Nodes represent the object or concept.
    - Links: Denoting relation between nodes.
    - Labels: Denoting particular objects & relations.

- **Structural Component -** Here the links and nodes form a directed graph wherein the labels are placed on the link and nodes.

- **Semantic Component** - The meanings here are related to the links and labels of nodes, whereas the facts are dependent on the approved areas.
- **Procedural Part -** The creation of new links and nodes is permitted by constructors, whereas the destructors are responsible for the removal of links and nodes.

SEMANTIC NETWORK ARCHITECTURE:

- As stated earlier, the semantic network is a simple knowledge representation technique.
- It uses graphic notation to represent knowledge or data, wherein a graph of labeled nodes and labels are used, with directed arcs to encode knowledge.
- It follows a simple and comprehensible architecture, which helps add and change information efficiently.



TYPES OF SEMANTIC NETWORKS:

- Semantic networks were developed initially for computers in 1956 by Richard H. R. of the Cambridge Language Research Unit (CLRU), for machine translation of natural languages.
- However, now it is used for a variety of functions, like knowledge representation.
- There are currently six types of semantic networks that enable declarative graphic representation, which is further used to represent knowledge and support automated systems for reasoning about the knowledge.

These six types of semantic networks are:

- **Definitional Networks:** Emphasize the subtype or is-a relationship between a concept type and a newly defined subtype.
- **Assertional Networks**: Designed to assert propositions.
- **Implicational Networks:** Uses implications as the primary relationship for connecting nodes.
- **Executable Networks:** Contain mechanisms that can cause some change to the network itself.
- **Learning Networks:** It builds or extends the representation by acquiring knowledge from examples.
- **Hybrid Networks:** These combine two or more of the previous techniques, either in a single network or in separate, but closely interacting networks.

APPLICATION OF SEMANTIC NETWORKS:

With the growing need for intelligent machines, the application of semantic networks is also increasing.
Therefore, listed here are some of the areas where Semantic Networks are applied or used:

- In natural language processing applications like semantic parsing, word sense disambiguation, etc.
- Specialized retrieval tasks, like plagiarism detection.

- Knowledge Graph proposed by Google in 2012 uses semantic networks in the search engines.

ADVANTAGES & DISADVANTAGES OF SEMANTIC NETWORKS:

As one of the oldest and the most effective techniques or Knowledge Representation, semantic networks offers various advantages, a few of which are:

1. ADVANTAGES:

- It is simple and comprehensible.
- Efficient in space requirement.
- Easily clusters related knowledge.
- It is flexible and easy to visualize.
- It is a natural representation of knowledge.
- Conveys meaning in a transparent manner.

2. DISADVANTAGES:

Though the importance of Semantic Networks is immense in Knowledge Representation, we must consider the drawbacks it offers, such as:

- Inheritance cause problems.
- Links on objects represent only binary options.
- Interactable for large domains.
- Don't represent performances or meta-knowledge effectively.
- It's difficult to express some properties using Semantic Networks, like negation, disjunction, etc.

Partioned Semantic Nets

Partitioned Semantic Networks:

- Semantic Networks are simple representation technique that provides graphical representation.
- The knowledge we use can be more complex.
- It may contain complex facts and statements here, and semantic networks can only provide partially accurate results.
  Partitioned semantic networks overcome the limitations of semantic networks.
- It is a type of semantic network.

DIFFERENCE BETWEEN SEMANTIC NETS AND FRAMES:

- Semantic networks and frames are both knowledge representation techniques.
- Though they are categorized into the same group, their functions and working are vastly different from one another.
- Therefore, to help you understand the differences between the two, we are here with a detailed comparison of semantic nets and frames.

Semantic Nets

- A knowledge-base that represents systematic relations between concepts in a network.
- It has nodes that represent objects and arcs and describes their relations.
- With Semantic Networks, adding information and making inferences is fast and easy.
- It categorizes objects in different forms and also links them.
- Semantic networks vary in type and can represent very diverse systems.

Frames

- Represent related knowledge about a narrow subject that has default knowledge.
- It has slots that further have values, which are known as facets.
- Frames add a procedural attachment & inherit properties from generic frames.
- It makes the programming languages easier by grouping the related data.

- It is vastly used in natural language and information processing.

- A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world.
- Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations.
- It consists of a collection of slots and slot values.

- Frame Representation A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world.
- Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations.
- It consists of a collection of slots and slot values.
- These slots may be of any type and sizes.
- Slots have names and values which are called facets.

Facets:

- The various aspects of a slot is known as Facets.
- Facets are features of frames which enable us to put constraints on the frames.
- **Example:** IF-NEEDED facts are called when data of any particular slot is needed.
- A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values.
- A frame is also known as slot-filter knowledge representation in artificial intelligence. Frames are derived from semantic networks and later evolved into our modern-day classes and objects.
- A single frame is not much useful.
- Frames system consist of a collection of frames which are connected.
- In the frame, knowledge about an object or event can be stored together in the knowledge base.

- The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Conceptual Dependency

- Conceptual Dependency theory is based on the use of limited number of primitive concepts and rules of formation to represent any natural language statement.
- Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.
- The agent and the objects are represented.

Conceptual Dependency:

- In 1977, Roger C. Schank has developed a Conceptual Dependency structure. The Conceptual Dependency is used to represent knowledge of Artificial Intelligence.
- It should be powerful enough to represent these concepts in the sentence of natural language.
- It states that different sentence which has the same meaning should have some unique representation.

There are 5 types of states in Conceptual Dependency:

1. Entities
2. Actions
3. Conceptual cases
4. Conceptual dependencies
5. Conceptual tense

Main Goals of Conceptual Dependency:

1. It captures the implicit concept of a sentence and makes it explicit.
2. It helps in drawing inferences from sentences.
3. For any two or more sentences that are identical in meaning. It should be only one representation of meaning.

4. It provides a means of representation which are language independent.
5. It develops language conversion packages.

Rules of Conceptual Dependency:

Rule-1: It describes the relationship between an actor and the event he or she causes.

Rule-2: It describes the relationship between PP and PA that are asserted to describe it.

Rule-3: It describes the relationship between two PPs, one of which belongs to the set defined by the other.

Rule-4: It describes the relationship between a PP and an attribute that has already been predicated on it.

Rule-5: It describes the relationship between two PPs one of which provides a particular kind of information about the other.

Rule-6: It describes the relationship between an ACT and the PP that is the object of that ACT.

Rule-7: It describes the relationship between an ACT and the source and the recipient of the ACT.

Rule-8: It describes the relationship between an ACT and the instrument with which it is performed. This instrument must always be a full conceptualization, not just a single physical object.

Rule-9: It describes the relationship between an ACT and its physical source and destination.

Rule-10: It represents the relationship between a PP and a state in which it started and another in which it ended.

Rule-11: It represents the relationship between one conceptualization and another that causes it.

Rule-12: It represents the relationship between conceptualization and the time at which the event occurred described.

Rule-13: It describes the relationship between one conceptualization and another, that is the time of the first.

Rule-14: It describes the relationship between conceptualization and the place at which it occurred.

Conceptual Graphs

- Conceptual graphs (CGs) are a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence.
- Their purpose is to express meaning in a form that is logically precise, humanly readable, and computationally tractable.
- Conceptual graphs (CGs) are a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence.
- They express meaning in a form that is logically precise, humanly readable, and computationally tractable.
- With a direct mapping to language, conceptual graphs serve as an intermediate language for translating computer-oriented formalisms to and from natural languages.
- With their graphic representation, they serve as a readable, but formal design and specification language (J. Sowa).

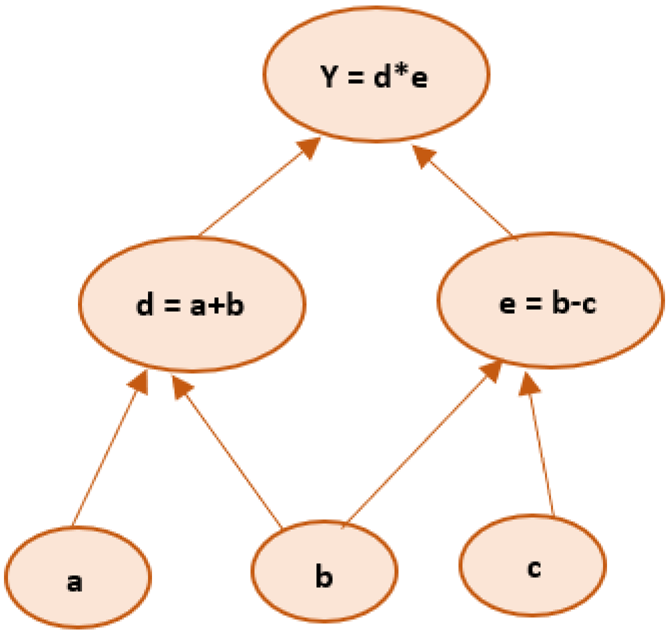
These can be used for two different types of calculations:

1. Forward computation
2. Backward computation

The following sections define a few key terminologies in computational graphs.

- A variable is represented by a node in a graph. It could be a scalar, vector, matrix, tensor, or even another type of variable.
- A function argument and data dependency are both represented by an edge. These are similar to node pointers.
- A simple function of one or more variables is called an operation. There is a set of operations that are permitted. Functions that are more complex than these operations in this set can be represented by combining multiple operations.

Here, we have three operations, addition, subtraction, and multiplication. To create a computational graph, we create nodes, each of them has different operations along with input variables. The direction of the array shows the direction of input being applied to other nodes.

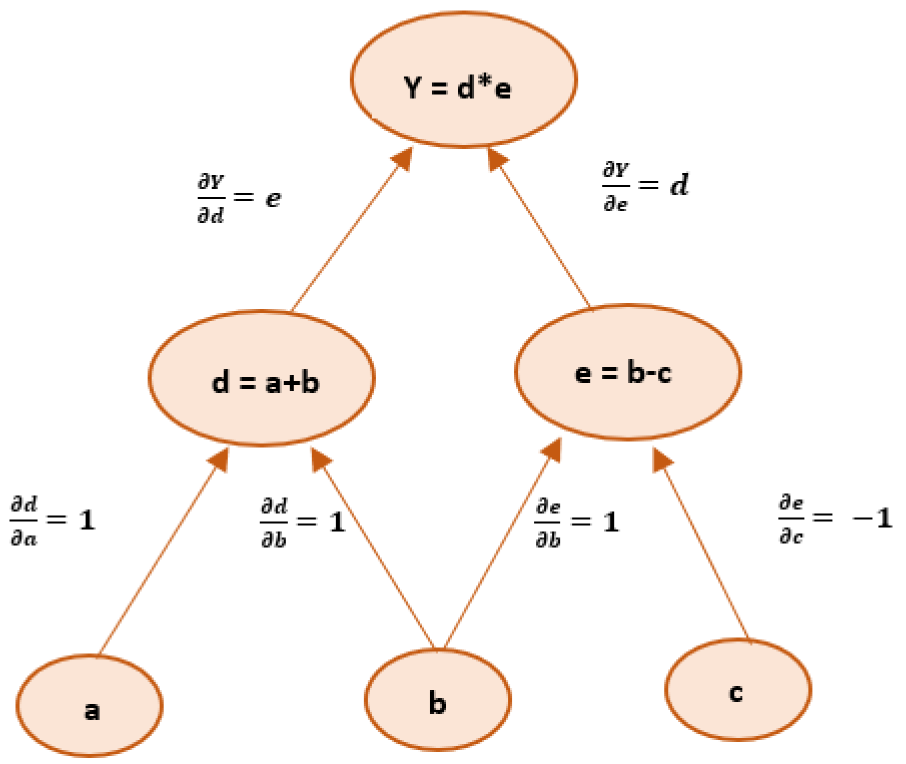


We can find the final output value by initializing input variables and accordingly computing nodes of the graph.

Computational Graphs in Deep Learning

- Computations of the neural network are organized in terms of a forward pass or forward propagation step in which we compute the output of the neural network, followed by a backward pass or backward propagation step, which we use to compute gradients/derivatives.
- Computation graphs explain why it is organized this way.
- If one wants to understand derivatives in a computational graph, the key is to understand how a change in one variable brings change on the variable that depends on it.
- If a directly affects c, then we want to know how it affects c.
- If we make a slight change in the value of a how does c change? We can term this as the partial derivative of c with respect to a.

Graph for backpropagation to get derivatives will look something like this:


Y = d*e
$\frac{\partial Y}{\partial d} = e$
$\frac{\partial Y}{\partial e} = d$
d = a+b
e = b-c
$\frac{\partial d}{\partial a} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial e}{\partial b} = 1$
$\frac{\partial e}{\partial c} = -1$
a
b
c

We have to follow chain rule to evaluate partial derivatives of final output variable with respect to input variables: a, b, and c. Therefore the derivatives can be given as :

$$\frac{\partial Y}{\partial a} = \frac{\partial Y}{\partial d} \times \frac{\partial d}{\partial a} = e \times 1 = e$$

$$\frac{\partial Y}{\partial b} = \frac{\partial Y}{\partial d} \times \frac{\partial d}{\partial b} = e \times 1 = e$$

$$\frac{\partial Y}{\partial c} = \frac{\partial Y}{\partial e} \times \frac{\partial e}{\partial c} = d \times -1 = -d$$

This gives us an idea of how computational graphs make it easier to get the derivatives using backpropagation.

Types of computational graphs:

Type 1: Static Computational Graphs

- **Involves two phases:-**
    - **Phase 1:-** Make a plan for your architecture.
    - **Phase 2:-** To train the model and generate predictions, feed it a lot of data.
- The benefit of utilizing this graph is that it enables powerful offline graph optimization and scheduling.
- As a result, they should be faster than dynamic graphs in general.
- The drawback is that dealing with structured and even variable-sized data is unsightly. As the forward computation is performed, the graph is implicitly defined.

Type 2:

- This graph has the advantage of being more adaptable.
- The library is less intrusive and enables interleaved graph generation and evaluation. The forward computation is implemented in your preferred programming language, complete with all of its features and algorithms.
- Debugging dynamic graphs is simple.
- Because it permits line-by-line execution of the code and access to all variables, finding bugs in your code is considerably easier.
- If you want to employ Deep Learning for any genuine purpose in the industry, this is a must-have feature.
- The disadvantage of employing this graph is that there is limited time for graph optimization, and the effort may be wasted if the graph does not change.

Scripts

Intro to Scripts:

- Postman is not only used for manual testing, but we can also do automation testing of API. Postman requests and collections can be applied to dynamic behavior.
- In Postman Scripts are the lines of code that allow you to automate an API test. It offers you to write pre-request and test scripts.

    1. Before sending a request, a pre-request script will run and,
    2. After receiving a response, test scripts will run.

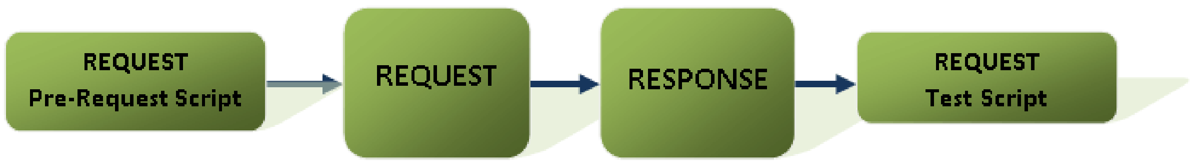
- In Postman, we can write the tests, pass the data between the requests, and change the parameters.
- It allows adding test script and pre-request script to a collection, a folder, a request, or a request not attached to a group.

Postman Sandbox:

- To write the script in Postman, we use Postman Sandbox.
- Postman sandbox has been written in Javascript. It is an execution environment. Since the sandbox has written in Javascript, therefore, it will only receive a script written in Javascript.
- Hence, we have to write the script in javascript to make the code executable in Postman.

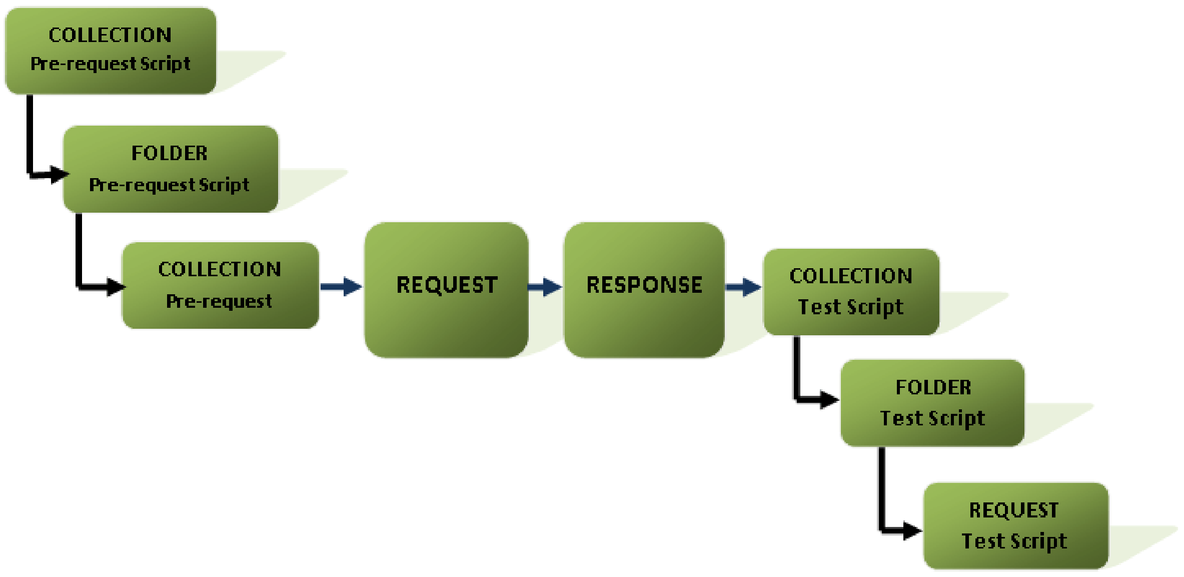
Execution Order of Scripts in Postman:

In Postman, the order of script execution for a single request is as follows:



- A pre-request script associated with a request will execute before sending the request.
- Test script associated with a request will execute after sending the request.

The order of script execution for every request in a collection is as follows:

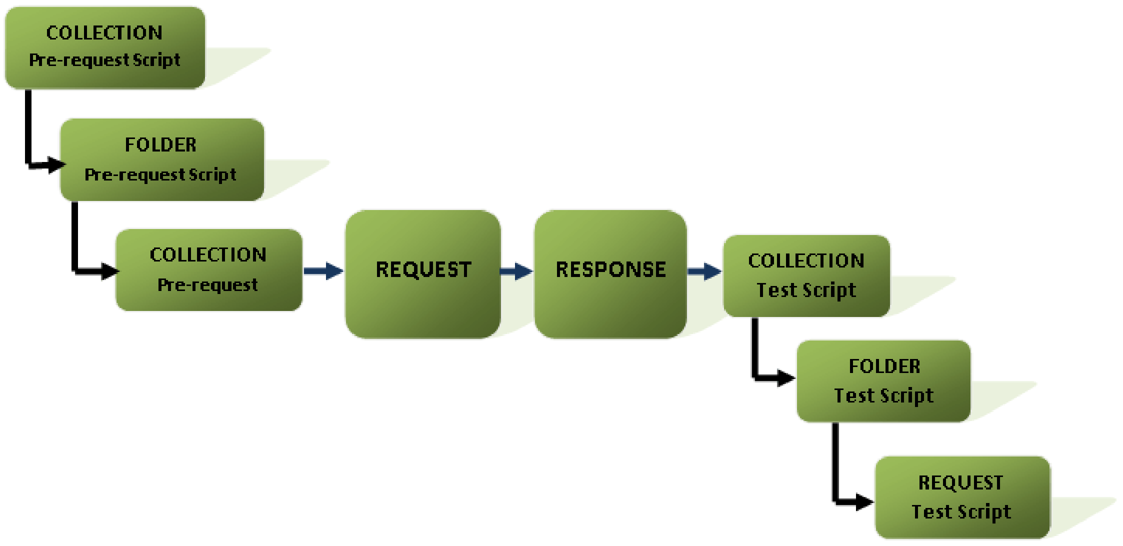


- Before every request in the collection, a pre-request script associated with a collection will run.
- Before every request in the folder, a pre-request script associated with a folder will run.
- After every request in the collection, a test script associated with a collection will run.
- After a request in the folder, a test script associated with a folder will run.

The scripts should always run according to the following hierarchy for every request in a collection:

- Collection level script
- Folder level script
- Request level script
- The same order will be applied for both pre-request and test scripts.
- A pre-request script associated with a request will execute before sending the request.
- Test script associated with a request will execute after sending the request.

The order of script execution for every request in a collection is as follows:



- Before every request in the collection, a pre-request script associated with a collection will run.
- Before every request in the folder, a pre-request script associated with a folder will run.
- After every request in the collection, a test script associated with a collection will run.
- After a request in the folder, a test script associated with a folder will run.

The scripts should always run according to the following hierarchy for every request in a collection:

- Collection level script
- Folder level script
- Request level script
    - The same order will be applied for both pre-request and test scripts.

CYC

- CYC, which stands for CycL Knowledge Base, is a large-scale artificial intelligence project that aims to create a knowledge base containing a broad range of common-sense knowledge that would enable computers to reason more like humans.

- The project began in 1984 and is led by computer scientist Doug Lenat.

CYC

- CYC is an example of a frame-based representational system of knowledge, which is, in a way, the opposite of an expert system.
- Whereas an expert system has detailed knowledge of a very narrow domain, the developers of CYC have fed it information on over 100,000 different concepts from all fields of human knowledge.
- CYC also has information of over 1,000,000 different pieces of "common sense" knowledge about those concepts.
- The system has over 4000 different types of links that can exist between concepts, such as inheritance, and the "is–a" relationship that we have already looked at.
- The idea behind CYC was that humans function in the world mainly on the basis of a large base of knowledge built up over our lifetimes and our ancestors' lifetimes.
- By giving CYC access to this knowledge, and the ability to reason about it, they felt they would be able to come up with a system with common sense. Ultimately, they predict, the system will be built into word processors.