SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution
Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AI&ML

FOUNDATIONS OF ARTIFICIAL INTELLIGENCE

II YEAR - III SEM

UNIT II – Logical Reasoning

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

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

Following are some basic facts about propositional logic:

- 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:

- **Atomic 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.

a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.

b) "The Sun is cold" is also a proposition as it is a **false** fact.

- **Compound propositions**

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

Example:

- a) "It is raining today, and street is wet."
- 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.

- **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**.
- **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
- **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 V 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**

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
- **All the girls are intelligent.**
- **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Inference:

- In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference.**

Inference rules:

- Inference rules are **the templates** for generating **valid arguments.** Inference rules are applied to **derive proofs in artificial intelligence**, and the proof is a sequence of the conclusion that leads to the desired goal.

- In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

- **Implication:** It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.
- **Inverse:** The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Hence from the above truth table, we can prove that P → Q is equivalent to ¬ Q → ¬ P, and Q→ P is equivalent to ¬ P → ¬ Q.

Types of Inference rules:

1. Modus Ponens:

- The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:
- **Example:**
- Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
  Statement-2: "I am sleepy" ==> P
  Conclusion: "I go to bed." ==> Q.
  Hence, we can say that, if P→ Q is true and P is true then Q will be true

Notation for Modus ponens: $\dfrac{P \rightarrow Q,\ P}{\therefore Q}$

Proof by Truth table:

P	Q	$P \rightarrow Q$
0	**0**	**0**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

2. Modus Tollens:

- The Modus Tollens rule state that if P→ Q is true and **¬ Q is true, then ¬ P** will also true. It can be represented as:

- **Statement-1:** "If I am sleepy then I go to bed" ==> P→ Q
  Statement-2: "I do not go to the bed."==> ~Q
  Statement-3: Which infers that "**I am not sleepy**" => ~P

Notation for Modus Tollens: $\dfrac{P \rightarrow Q,\ \sim Q}{\sim P}$

P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

3. Hypothetical Syllogism:

- The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

- **Statement-1:** If you have my home key then you can unlock my home. **P→Q**
  Statement-2: If you can unlock my home then you can take my money. **Q→R**
  Conclusion: If you have my home key then you can take my money. **P→R**

Proof by truth table:

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$P \rightarrow R$	
0	0	0	1	1	1	←
0	0	1	1	1	1	←
0	1	0	1	0	1	
0	1	1	1	1	1	←
1	0	0	0	1	1	
1	0	1	0	1	1	
1	1	0	1	0	0	
1	1	1	1	1	1	←

4. Disjunctive Syllogism:

- The Disjunctive syllogism rule state that if PvQ is true, and ¬P is true, then Q will be true. It can be represented as:

- **Example:**
- **Statement-1:** Today is Sunday or Monday. ==>PvQ
  Statement-2: Today is not Sunday. ==> ¬P
  Conclusion: Today is Monday. ==> Q

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

P	Q	$\neg P$	$P \vee Q$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

5.Addition:

- The Addition rule is one the common inference rule, and it states that If P is true, then PvQ will be true.

- **Example:**
- **Statement:** I have a vanilla ice-cream. ==> P
  Statement-2: I have Chocolate ice-cream.
  Conclusion: I have vanilla or chocolate ice-cream. ==> (PvQ)

Notation of Addition: $\dfrac{P}{P \vee Q}$

Proof by Truth-Table:

P	Q	$P \vee Q$	
0	0	0	
1	0	1	←
0	1	1	
1	1	1	←

6. Simplification:

- The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

P	Q	$P \wedge Q$
0	0	0
1	0	0
0	1	0
1	1	1

7. Resolution:

- The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. **It can be represented as**

Notation of Resolution $\dfrac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

Proof by Truth-Table:

P	¬P	Q	R	$P \vee Q$	¬P∧R	$Q \vee R$	
0	1	0	0	0	0	0	
0	1	0	1	0	0	1	
0	1	1	0	1	1	1	←
0	1	1	1	1	1	1	←
1	0	0	0	1	0	0	
1	0	0	1	1	0	1	
1	0	1	0	1	0	1	
1	0	1	1	1	0	1	←

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, ……
- **Relations and function: 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 , Father of, best friend, third inning of, end of, ……

- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **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.

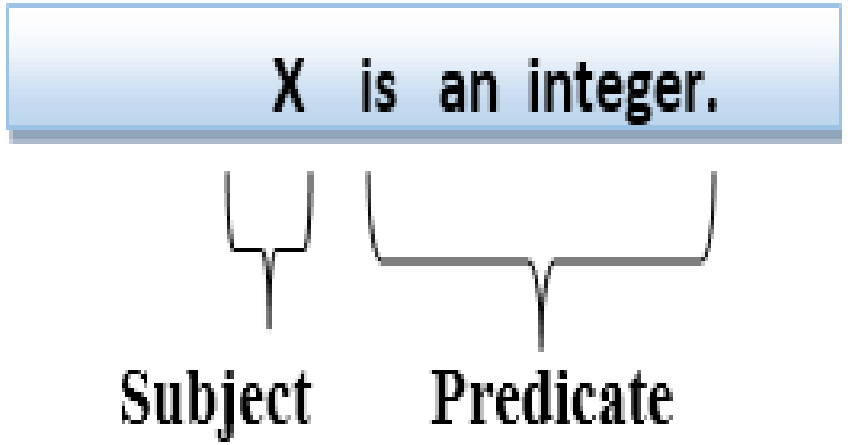
Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

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.


X is an integer.
Subject
Predicate

Inference in First-Order Logic

- Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences.

Substitution:

- Substitution is a fundamental operation performed on terms and formulas.
- It occurs in all inference systems in first-order logic.
- The substitution is complex in the presence of quantifiers in FOL.
- If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality:

- First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL.
- For this, we can use **equality symbols** which specify that the two terms refer to the same object.

  Example: Brother (John) = Smith.

- As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.
- **Example: ¬(x=y) which is equivalent to x ≠y.**

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)**.

- 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)

- 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 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**.

Steps for Resolution:

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

Example:

- John likes all kind of food.
- Apple and vegetable are food
- Anything anyone eats and not killed is food.
- Anil eats peanuts and still alive
- Harry eats everything that Anil eats.
  Prove by resolution that:
- John likes peanuts.

Step-1: Conversion of Facts into FOL

a. $\forall x$: food(x) $\rightarrow$ likes(John, x)

b. food(Apple) $\wedge$ food(vegetables)

c. $\forall x \forall y$: eats(x, y) $\wedge \neg$ killed(x) $\rightarrow$ food(y)

d. eats (Anil, Peanuts) $\wedge$ alive(Anil).

e. $\forall x$: eats(Anil, x) $\rightarrow$ eats(Harry, x)

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x)

g. $\forall x$: alive(x) $\rightarrow \neg$ killed(x)

} **added predicates.**

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

a. $\forall x \neg food(x) \vee likes(John, x)$

b. $food(Apple) \wedge food(vegetables)$

c. $\forall x \forall y \neg [eats(x, y) \wedge \neg killed(x)] \vee food(y)$

d. $eats(Anil, Peanuts) \wedge alive(Anil)$

e. $\forall x \neg eats(Anil, x) \vee eats(Harry, x)$

f. $\forall x \neg [\neg killed(x)] \vee alive(x)$

g. $\forall x \neg alive(x) \vee \neg killed(x)$

h. $likes(John, Peanuts)$.

Move negation (¬)inwards and rewrite

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ eats(x, y) V 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).

Rename variables or standardize variables

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g ¬killed(g)] V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).