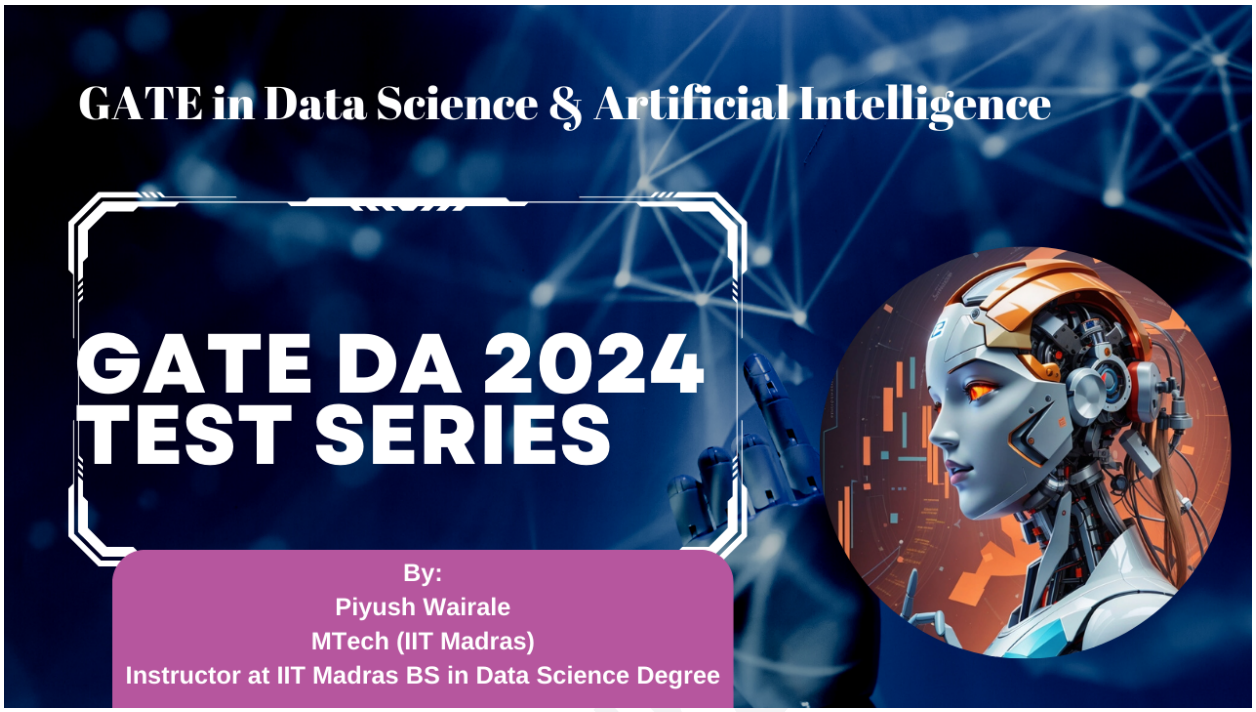


Connect with me:
Youtube: Piyush Wairale IITM
Linkedin: Piyush Wairale
Facebook: Piyush Wairale
Instagram: Piyush Wairale

Logic: Propositional and Predicate Logic

In which we design agents that can form representations of a complex world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do.

1. Humans' intelligence relies on reasoning processes that operate on internal representations of knowledge, as exemplified by knowledge-based agents in AI.
2. Problem-solving agents, while knowledgeable in limited, domain-specific ways, can benefit from more flexible and domain-independent knowledge representations, such as logic.
3. Logic serves as a general class of representations for **knowledge-based agents**, enabling them to combine and adapt information for various purposes.
4. Propositional logic, though less expressive than first-order logic, offers fundamental concepts and powerful inference technologies for building knowledge-based agents.

These sentences are expressed according to the **syntax** of the representation language, which specifies all the sentences that are well formed. The notion of syntax is clear enough in ordinary arithmetic: "x + y = 4" is a well-formed sentence, whereas "x4y+ =" is not.
A logic must also define the **semantics** or meaning of sentences. The semantics define the truth of each sentence with respect to each **possible world**. For example, the semantics for arithmetic specify that the sentence "x + y = 4" is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1. In standard logics, every sentence must be either true or false in each possible world—there is no "in between."1

1. **Syntax and Semantics:** Logic involves two key aspects - syntax, which defines the structure and grammar of sentences, and semantics, which determines the truth or falsehood of sentences in different possible worlds.
2. **Entailment and Inference:** Entailment is the idea that one sentence logically follows from another. Inference processes, like model checking, help derive conclusions from a knowledge base (KB) while ensuring the soundness of the derived statements.
3. **Soundness and Completeness:** Sound inference procedures ensure that derived conclusions are always true if the premises are true. Completeness is the ability to derive any entailed sentence, which is crucial when dealing with infinite knowledge bases.
4. **Grounding Knowledge**: The connection between logical reasoning processes and the real world is established through sensors and learning. Sensors create sentences based on

perceptual input, while learning helps derive general rules from experiences, connecting the KB to the real environment.

5. **Real-World Correspondence:** Logical reasoning processes should ensure that derived conclusions represent aspects of the real world that logically follow from the aspects represented by the premises, bridging the gap between representations and reality.

1.PROPOSITIONAL LOGIC

We now present a simple but powerful logic called propositional logic. We cover the syntax of propositional logic and its semantics—the way in which the truth of sentences is determined. Then we look at entailment—the relation between a sentence and another sentence that follows from it—and see how this leads to a simple algorithm for logical inference.

Syntax

The syntax of propositional logic defines the allowable sentences. The **atomic sentences** consist of a single **proposition symbol**. Each such symbol stands for a proposition that can be true or false.

Definition

A proposition is a declarative statement.

- It must be either TRUE or FALSE.
- It cannot be both TRUE and FALSE.
- We use T to denote TRUE and F to denote FALSE.

Example of propositions:

Example of propositions:	Example of non-propositions:
John loves CSE 191.	Does John love CSE 191?
2+3=5.	$2+3$.
2+3=8.	Solve the equation $2+x=3$.
Sun rises from West.	$2+x>8$.

Important Points:

- 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.
- 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**.
- 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.
- Connectives can be said as a logical operator which connects two sentences.

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

Atomic Propositions:

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

Compound propositions:

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

Sentence	→	*AtomicSentence* | *ComplexSentence*
AtomicSentence	→	*True* | *False* | *P* | *Q* | *R* | ...
ComplexSentence	→	(*Sentence*) | [*Sentence*]
	|	¬ *Sentence*
	|	*Sentence* ∧ *Sentence*
	|	*Sentence* ∨ *Sentence*
	|	*Sentence* ⇒ *Sentence*
	|	*Sentence* ⇔ *Sentence*
OPERATOR PRECEDENCE	:	¬, ∧, ∨, ⇒, ⇔

A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest

Semantics:

The semantics defines the rules for determining the truth of a sentence with respect to a particular model. In propositional logic, a model simply fixes the truth value—true or false—for every proposition symbol. For example, if the sentences in the knowledge base make use of the proposition symbols P1,2, P2,2, and P3,1, then one possible model is

m1 = {P1,2 = false, P2,2 = false, P3,1 = true}

With three proposition symbols, there are 2^3 = 8 possible models.

- For complex sentences, we have five rules, which hold for any subsentences P and Q in any model m (here "iff" means "if and only if"):
  - ¬P is true iff P is false in m.
  - P ∧ Q is true iff both P and Q are true in m.
  - P ∨ Q is true iff either P or Q is true in m.
  - P ⇒ Q is true unless P is true and Q is false in m.
  - P ⇔ Q is true iff P and Q are both true or both false in m.

The rules can also be expressed with **truth tables** that specify the truth value of a complex sentence for each possible assignment of truth values to its components. Truth tables for the five connectives are given-

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Truth tables for the five logical connectives. To use the table to compute, for example, the value of P $\vee$ Q when P is true and Q is false, first look on the left for the row where P is true and Q is false (the third row). Then look in that row under the P $\vee$ Q column to see the result: true

Operator	Precedence
$\neg$	1
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\leftrightarrow$	5

Implication (→) represents a structure of "if P then Q." For example, if P: "It is raining" and Q: "I'm indoors", then P → Q means "If it is raining, then I'm indoors." In the case of P implies Q (P → Q), P is called the antecedent and Q is called the *consequent*.

- When the antecedent is true, the whole implication is true in the case that the consequent is true (that makes sense: if it is raining and I'm indoors, then the sentence "if it is raining, then I'm indoors" is true).

- When the antecedent is true, the implication is false if the consequent is false (if I'm outside while it is raining, then the sentence "If it is raining, then I'm indoors" is false).
- However, when the antecedent is false, the implication is always true, regardless of the consequent. This can sometimes be a confusing concept.
- Logically, we can't learn anything from an implication (P $\rightarrow$ Q) if the antecedent (P) is false. Looking at our example, if it is not raining, the implication doesn't say anything about whether I'm indoors or not. I could be an indoors type and never walk outside, even when it is not raining, or I could be an outdoors type and be outside all the time when it is not raining. When the antecedent is false, we say that the implication is *trivially* true.

Because implication statements play such an essential role in mathematics, a variety of terminology is used to express $p \rightarrow q$:

- "if p, then q".
- "q, if p".
- "p, only if q".
- "p implies q".
- "p is sufficient for q".
- "q is necessary for p".
- "q follows from p".

Example

Proposition p: Alice is smart.
Proposition q: Alice is honest.

- $p \rightarrow q$.
  - That Alice is smart is sufficient for Alice to be honest.
  - "Alice is honest" if "Alice is smart".
- $q \rightarrow p$:
  - That Alice is smart is necessary for Alice to be honest.
  - "Alice is honest" only if "Alice is smart".

Bidirectional (⇔)is an implication that goes both directions. You can read it as "if and only if." P ⇔ Q is the same as P → Q and Q → P taken together.

For example, if P: "It is raining." and Q: "I'm indoors," then P ⇔ Q means that "If it is raining, then I'm indoors," and "if I'm indoors, then it is raining." This means that we can infer more than we could with a simple implication. If P is false, then Q is also false; if it is not raining, we know that I'm also not indoors.

Example

Proposition p: Alice is smart.
Proposition q: Alice is honest.

$\neg p \wedge q$:	Alice is not smart but honest.
$p \vee (\neg p \wedge q)$:	Either Alice is smart, or she is not smart but honest.
$p \rightarrow \neg q$:	If Alice is smart, then she is not honest.

We can also go in the other direction, translating English sentences to logical formulas:

- Alice is either smart or honest, but Alice is not honest if she is smart:
  $(p \vee q) \wedge (p \rightarrow \neg\, q)$.
- That Alice is smart is necessary and sufficient for Alice to be honest:
  $(p \rightarrow q) \wedge (q \rightarrow p)$.
  (This is often written as $p \leftrightarrow q$).

Tautology and Logically Equivalent

Definitions:

- A compound proposition that is always True is called a tautology.
- Two propositions p and q are logically equivalent if their truth tables are the same.
- Namely, p and q are logically equivalent if $p \leftrightarrow q$ is a tautology.
- If p and q are logically equivalent, we write $p \equiv q$.

Example:

Look at the following two compound propositions: $p \rightarrow q$ and $q \vee \neg p$.

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

p	q	$\neg p$	$q \vee \neg p$
T	T	F	T
T	F	F	F
F	T	T	T
F	F	T	T

- The last column of the two truth tables are identical. Therefore $(p \rightarrow q)$ and $(q \vee \neg p)$ are logically equivalent.
- So $(p \rightarrow q) \leftrightarrow (q \vee \neg p)$ is a tautology.
- Thus: $(p \rightarrow q) \equiv (q \vee \neg p)$.

De Morgan Law:

$$\neg(p \wedge q) \equiv \neg p \vee \neg q \quad (1)$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q \quad (2)$$

The following is the truth table proof for (1). The proof for (2) is similar.

p	q	$p \wedge q$	$\neg(p \wedge q)$
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

p	q	$\neg p \vee \neg q$
T	T	F
T	F	T
F	T	T
F	F	T

Distributivity

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r) \quad (1)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r) \quad (2)$$

The following is the truth table proof of (1). The proof of (2) is similar.

p	q	r	$q \wedge r$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r$	$(p \vee q) \wedge (p \vee r)$
T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	T	T	T	T
F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	F
F	F	T	F	F	F	T	F
F	F	F	F	F	F	F	F

Contrapositives

The proposition $\neg q \rightarrow \neg p$ is called the Contrapositive of the proposition $p \rightarrow q$. They are logically equivalent.

$$p \rightarrow q \quad \equiv \quad \neg q \rightarrow \neg p$$

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

p	q	$\neg q \rightarrow \neg p$
T	T	T
T	F	F
F	T	T
F	F	T

Equivalence	Name
$p \wedge T \equiv p, \quad p \vee F \equiv p$	Identity laws
$p \vee T \equiv T, \quad p \wedge F \equiv F$	Domination laws
$p \vee p \equiv p, \quad p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \vee q) \equiv \neg p \wedge \neg q$ $\neg(p \wedge q) \equiv \neg p \vee \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	Absorption laws
$p \vee \neg p \equiv T, \quad p \wedge \neg p \equiv F$	Negation laws

Logical Equivalences Involving Conditional Statements

$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv p \wedge \neg q$
$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

Logical Equivalences Involving Biconditional Statements

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

Model

The model is an assignment of a truth value to every proposition. To reiterate, propositions are statements about the world that can be either true or false. However, knowledge about the world is represented in the truth values of these propositions. The model is the truth-value assignment that provides information about the world.

For example, if P: "It is raining." and Q: "It is Tuesday.", a model could be the following truth-value assignment: {P = True, Q = False}. This model means that it is raining, but it is not Tuesday. However, there are more possible models in this situation (for example, {P = True, Q = True}, where it is both raining an a Tuesday). In fact, the number of possible models is 2 to the power of the number of propositions. In this case, we had 2 propositions, so 2^2=4 possible models.

Knowledge Base (KB)

The knowledge base is a set of sentences known by a knowledge-based agent. This is knowledge that the AI is provided about the world in the form of propositional logic sentences that can be used to make additional inferences about the world.

Entailment (⊨)

If α ⊨ β (α entails β), then in any world where α is true, β is true, too.

For example, if α: "It is a Tuesday in January" and β: "It is January," then we know that α ⊨ β. If it is true that it is a Tuesday in January, we also know that it is January. Entailment is different from implication. Implication is a logical connective between two propositions. Entailment, on the other hand, is a relation that means that if all the information in α is true, then all the information in β is true.

Inference.

Model Checking is not an efficient algorithm because it has to consider every possible model before giving the answer (a reminder: a query R is true if under all the models (truth assignments) where the KB is true, R is true as well). Inference rules allow us to generate new information based on existing knowledge without considering every possible model.

Inference rules are usually represented using a horizontal bar that separates the top part, the premise, from the bottom part, the conclusion.
The premise is whatever knowledge we have, and the conclusion is what knowledge can be generated based on the premise.

Modus Ponens

The type of inference rule we use in this example is Modus Ponens, which is a fancy way of saying that if we know an implication and its antecedent to be true, then the consequent is true as well.

$$\frac{\alpha \rightarrow \beta \qquad \alpha}{\beta}$$

And Elimination

$$\frac{\alpha \wedge \beta}{\alpha}$$

If an And proposition is true, then any one atomic proposition within it is true as well.
For example, if we know that Harry is friends with Ron and Hermione, we can conclude that

Harry is friends with Hermione.

Double Negation Elimination

$$\frac{\neg(\neg\alpha)}{\alpha}$$

A proposition that is negated twice is true.
For example, consider the proposition “It is not true that Harry did not pass the test”. We can parse it the following way: “It is not true that (Harry did not pass the test)”, or “¬(Harry did not pass the test)”, and, finally “¬(¬(Harry passed the test)).”
The two negations cancel each other, marking the proposition “Harry passed the test” as true.

Implication Elimination

$$\frac{\alpha \rightarrow \beta}{\neg\alpha \vee \beta}$$

An implication is equivalent to an Or relation between the negated antecedent and the consequent.
As an example, the proposition “If it is raining, Harry is inside” is equivalent to the proposition “(it is not raining) or (Harry is inside).”

Biconditional Elimination

$$\frac{\alpha \leftrightarrow \beta}{(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)}$$

A biconditional proposition is equivalent to an implication and its inverse with an And connective.
For example, “It is raining if and only if Harry is inside” is equivalent to (“If it is raining, Harry is inside” And “If Harry is inside, it is raining”).

De Morgan’s Law

$$\frac{\neg(\alpha \wedge \beta)}{\neg\alpha \vee \neg\beta}$$

$$\frac{\neg(\alpha \vee \beta)}{\neg\alpha \wedge \neg\beta}$$

It is possible to turn an And connective into an Or connective.
Consider the following proposition: “It is not true that both Harry and Ron passed the test.”
From this, it is possible to conclude that “It is not true that Harry passed the test” Or “It is not true that Ron passed the test.”
That is, for the And proposition earlier to be true, at least one of the propositions in the Or propositions must be true.

Distributive Property

$$\frac{(\alpha \wedge (\beta \vee \gamma))}{(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)}$$

$$\frac{(\alpha \vee (\beta \wedge \gamma))}{(\alpha \vee \beta) \wedge (\alpha \vee \gamma)}$$

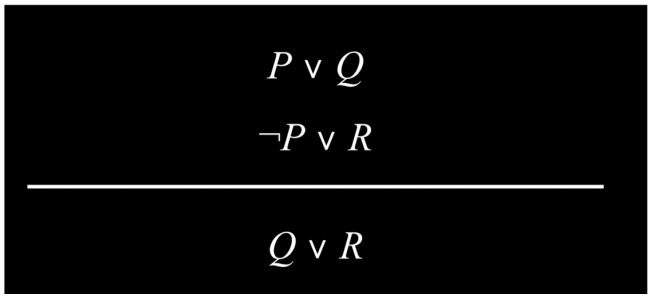
A proposition with two elements that are grouped with And or Or connectives can be distributed, or broken down into, smaller units consisting of And and Or.

Resolution

Resolution is a powerful inference rule that states that if one of two atomic propositions in an Or proposition is false, the other has to be true. For example, given the proposition "Ron is in the Great Hall" Or "Hermione is in the library", in addition to the proposition "Ron is not in the Great Hall," we can conclude that "Hermione is in the library."
More formally, we can define resolution the following way:

$$\frac{P \vee Q \quad \neg P}{Q}$$

Resolution can be further generalized. Suppose that in addition to the proposition "Ron is in the Great Hall" Or "Hermione is in the library", we also know that "Ron is not in the Great Hall" Or "Harry is sleeping." We can infer from this, using resolution, that "Hermione is in the library" Or "Harry is sleeping."
To put it in formal terms:



Resolution relies on Complementary Literals, two of the same atomic propositions where one is negated and the other is not, such as P and ¬P.
Complementary literals allow us to generate new sentences through inferences by resolution. Thus, inference algorithms locate complementary literals to generate new knowledge.

A **Clause** is a disjunction of literals (a propositional symbol or a negation of a propositional symbol, such as P, ¬P). A disjunction consists of propositions that are connected with an Or logical connective (P ∨ Q ∨ R). A conjunction, on the other hand, consists of propositions that are connected with an And logical connective (P ∧ Q ∧ R). Clauses allow us to convert any logical statement into a **Conjunctive Normal Form (CNF)**, which is a conjunction of clauses, for example: (A ∨ B ∨ C) ∧ (D ∨ ¬E) ∧ (F ∨ G).

Steps in Conversion of Propositions to Conjunctive Normal Form

- Eliminate biconditionals
  - Turn (α ↔ β) into (α → β) ∧ (β → α).
- Eliminate implications
  - Turn (α → β) into ¬α ∨ β.
- Move negation inwards until only literals are being negated (and not clauses), using De Morgan's Laws.
  - Turn ¬(α ∧ β) into ¬α ∨ ¬β

Here's an example of converting (P ∨ Q) → R to Conjunctive Normal Form:

- (P ∨ Q) → R
- ¬(P ∨ Q) ∨ R /Eliminate implication
- (¬P ∧ ¬Q) ∨ R /De Morgan's Law
- (¬P ∨ R) ∧ (¬Q ∨ R) /Distributive Law

At this point, we can run an inference algorithm on the conjunctive normal form. Occasionally, through the process of inference by resolution, we might end up in cases where a clause contains the same literal twice. In these cases, a process called factoring is used, where the duplicate literal is removed. For example, $(P \vee Q \vee S) \wedge (\neg P \vee R \vee S)$ allow us to infer by resolution that $(Q \vee S \vee R \vee S)$. The duplicate S can be removed to give us $(Q \vee R \vee S)$.

Resolving a literal and its negation, i.e. $\neg P$ and P, gives the empty clause (). The empty clause is always false, and this makes sense because it is impossible that both P and $\neg P$ are true. This fact is used by the resolution algorithm.

- To determine if $KB \models \alpha$:
  - Check: is $(KB \wedge \neg\alpha)$ a contradiction?
    - If so, then $KB \models \alpha$.
    - Otherwise, no entailment.

Proof by contradiction is a tool used often in computer science. If our knowledge base is true, and it contradicts $\neg\alpha$, it means that $\neg\alpha$ is false, and, therefore, α must be true. More technically, the algorithm would perform the following actions:

To determine if $KB \models \alpha$:

- ➢ Convert $(KB \wedge \neg\alpha)$ to Conjunctive Normal Form.
- ➢ Keep checking to see if we can use resolution to produce a new clause.
- ➢ If we ever produce the empty clause (equivalent to False), congratulations! We have arrived at a contradiction, thus proving that $KB \models \alpha$.
- ➢ However, if contradiction is not achieved and no more clauses can be inferred, there is no entailment.

Here is an example that illustrates how this algorithm might work:

- ❖ Does $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C)$ entail A?
- ❖ First, to prove by contradiction, we assume that A is false. Thus, we arrive at $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C) \wedge (\neg A)$.

- Now, we can start generating new information. Since we know that C is false (¬C), the only way (¬B ∨ C) can be true is if B is false, too. Thus, we can add (¬B) to our KB.
- Next, since we know (¬B), the only way (A ∨ B) can be true is if A is true. Thus, we can add (A) to our KB.
- Now our KB has two complementary literals, (A) and (¬A). We resolve them, arriving at the empty set, (). The empty set is false by definition, so we have arrived at a contradiction.

2.FIRST-ORDER LOGIC (Predicate Logic)

Definition

A predicate is a function. It takes some variable(s) as arguments; it returns either True or False (but not both) for each combination of the argument values.

$3 + 2 = 5$ is a proposition. But is $X + 2 = 5$ a proposition?

- Because it has a variable X in it, we cannot say it is T or F.
- So, it is not a proposition. It is called a predicate.

- In contrast, a proposition is not a function. It does not have any variable as argument. It is either True or False (but not both).
- The variables are always associated with a universe (or domain) of discourse, which tells us what combinations of the argument values are allowed.

- Suppose $P(x)$ is a predicate, where the universe of discourse for x is $\{1, 2, 3\}$. Then $P(x)$ is not a proposition, but $P(1)$ is a proposition.
- In general, a predicate is not a proposition. But when you assign values to all its argument variables, you get a proposition.

Example:

$P(x, y)$: "$x + 2 = y$" is a predicate.

- It has two variables x and y;
- Universe of Discourse: x is in $\{1, 2, 3\}$; y is in $\{4, 5, 6\}$.

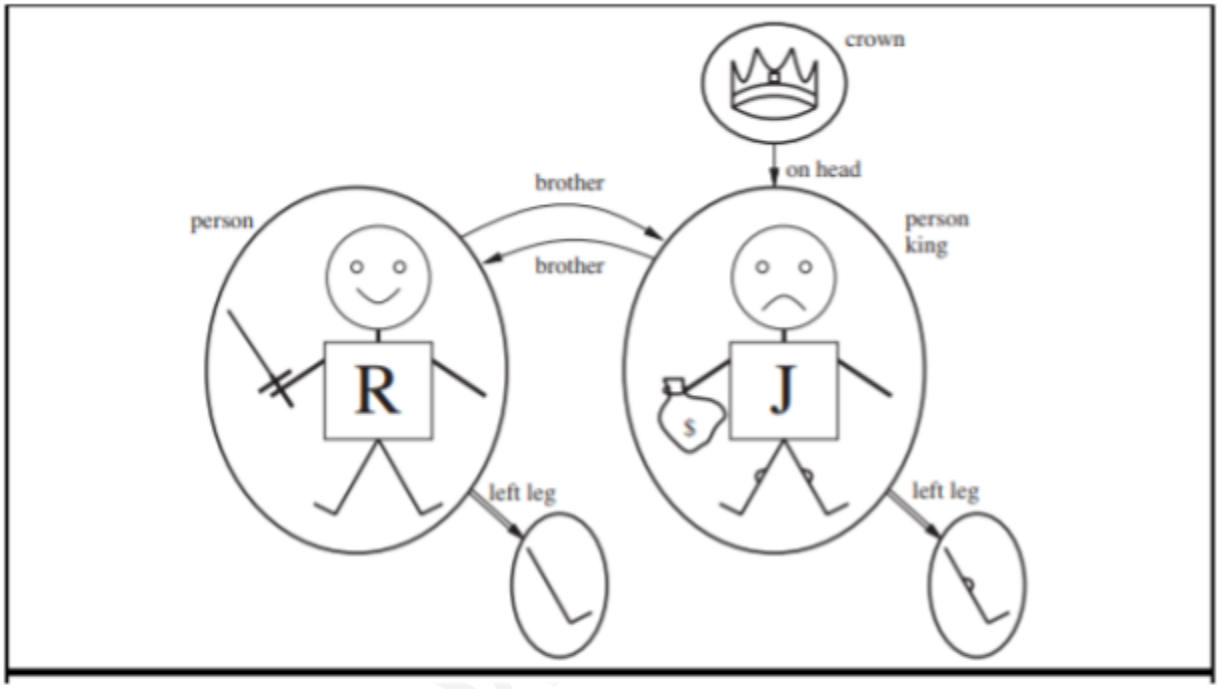
- $P(1, 4)$: $1 + 2 = 4$ is a proposition (it is F);
- $P(2, 4)$: $2 + 2 = 4$ is a proposition (it is T);
- $P(2, 3)$: meaningless (in this example), because 3 is not in the specified universe of discourse for y.

- First-order logic (FOL) is foundational in mathematics, philosophy, and artificial intelligence due to its ability to represent objects, relations, and general laws or rules.
- FOL differs from propositional logic in its ontological commitment, as it assumes that the world consists of objects with specific relations among them.
- The formal models in FOL are more complex than those in propositional logic, reflecting the richer ontology it assumes.
- Special-purpose logics, like temporal logic, make additional ontological commitments, such as considering time and ordering of events.
- Higher-order logic extends FOL by treating relations and functions as objects themselves, allowing assertions about all relations and functions.
- FOL is strictly less expressive than higher-order logic, as some sentences in the latter cannot be expressed using any finite number of FOL sentences.
- A logic's epistemological commitment refers to the possible states of knowledge regarding a fact. In both propositional and FOL, knowledge can be in three states: believing a sentence to be true, believing it to be false, or having no opinion.

Predicate logic, also known as first-order logic or predicate calculus, is a formal system in mathematical logic and philosophy. It extends propositional logic by introducing variables, quantifiers, and predicates to enable more complex and expressive representations of statements and relationships.

Syntax: First-order logic (FOL), also known as first-order predicate logic, has a well-defined syntax that consists of various elements and rules for constructing valid statements. Here are the key components of the syntax of FOL:

Object: Richard the Lionheart, King of England from 1189 to 1199; his younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; and a crown.



A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

1. **Variables:** Predicate logic introduces variables (e.g., x, y) that can represent objects or values in a domain. These variables can be quantified to specify general or specific conditions.
2. **Predicates**: Predicates are expressions that describe properties or relationships between objects or variables. They are often represented by symbols or words (e.g., P(x), Q(x, y)). Predicates can be true or false depending on the values assigned to their variables.

   We can go through each function and predicate, writing down what we know in terms of the other symbols. For example, one's mother is one's female parent:

   $\forall\ m, c\ Mother\ (c) = m \Leftrightarrow Female(m) \wedge Parent(m, c)$.

   One's husband is one's male spouse:

   $\forall\ w, h\ Husband(h, w) \Leftrightarrow Male(h) \wedge Spouse(h, w)$.

   Male and female are disjoint categories:

$\forall$ *x Male(x)* $\Leftrightarrow \neg$*Female(x)* .

Parent and child are inverse relations:

$\forall$ *p, c Parent(p, c)* $\Leftrightarrow$ *Child(c, p)* .

A grandparent is a parent of one's parent:

$\forall$ *g, c Grandparent(g, c)* $\Leftrightarrow \exists$ *p Parent(g, p)* $\wedge$ *Parent(p, c)* .

A sibling is another child of one's parents:

$\forall$ *x, y Sibling(x, y)* $\Leftrightarrow$ *x = y* $\wedge \exists$ *p Parent(p, x)* $\wedge$ *Parent(p, y)* .

3. **Quantifiers:**

Quantifiers in first-order logic allow us to express properties of entire collections of objects, making it possible to state general rules and assertions. Predicate logic uses quantifiers to express statements about the scope of variables. First-order logic contains two standard quantifiers, called universal and existential.

a. **Universal Quantifier ($\forall$)**: Represents "for all" or "for every." For example, $\forall$x P(x) means "P(x) is true for all values of x."

For Example:"All kings are persons," is written in first-order logic as

$\forall$ *x King(x)* $\Rightarrow$ *Person(x)*

$\forall$ is usually pronounced "For all ...". (Remember that the upside-down A stands for "all.") Thus, the sentence says, "For all *x*, if *x* is a king, then *x* is a person." The symbol *x* is called a **variable**. By convention, variables are lowercase letters. A variable is a term all by itself, and as such can also serve as the argument of a function—for example, *LeftLeg(x)*. A term GROUND TERM with no variables is called a **ground term.**

More precisely, $\forall$ x P is true in a given model if P is true in all possible **extended interpretations** constructed from the interpretation given in the model, where each extended interpretation specifies a domain element to which *x* refers.

For example:

x → Richard the Lionheart,
x → King John,
x → Richard's left leg,
x → John's left leg,
x → the crown.

The universally quantified sentence $\forall$ *x King(x)* $\Rightarrow$ *Person(x)* is true in the original model if the sentence *King(x)* $\Rightarrow$ *Person(x)* is true under each of the five extended interpretations. That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart is a king $\Rightarrow$ Richard the Lionheart is a person.

King John is a king ⇒ King John is a person.
Richard's left leg is a king ⇒ Richard's left leg is a person.
John's left leg is a king ⇒ John's left leg is a person.
The crown is a king ⇒ the crown is a person.

b. Existential Quantifier (∃): Represents "there exists" or "for some." For example, ∃x Q(x) means "There exists a value of x for which Q(x) is true." Universal quantification makes statements about every object. Similarly, we can make a statement about *some* object in the universe without naming it, by using an existential quantifier. To say, for example, that King John has a crown on his head, we write

$$\exists\, x\ Crown(x) \wedge OnHead(x, John)\ .$$

∃x is pronounced "There exists an x such that ..." or "For some x...".

Intuitively, the sentence $\exists\ x$ P says that P is true for at least one object x. More precisely, $\exists\ x$ P is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element. That is, *at least one* of the following is true:

Richard the Lionheart is a crown ∧ Richard the Lionheart is on John's head;
King John is a crown ∧ King John is on John's head;
Richard's left leg is a crown ∧ Richard's left leg is on John's head;
John's left leg is a crown ∧ John's left leg is on John's head;
The crown is a crown ∧ the crown is on John's head.

The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model. Notice that, by our definition, the sentence would also be true in a model in which King John was wearing two crowns. This is entirely consistent with the original sentence "King John has a crown on his head."

Just as ⇒ appears to be the natural connective to use with ∀, ∧ is the natural connective to use with ∃. Using ∧ as the main connective with ∀ led to an overly strong statement in the example in the previous section; using ⇒ with ∃ usually leads to a very weak statement, indeed. Consider the following sentence:

$$\exists\, x\ Crown(x) \Rightarrow OnHead(x, John)\ .$$

4. **Functions:** Predicate logic allows the use of functions to denote operations that produce values based on input arguments. Functions can be used in expressions and predicates (e.g., f(x) = y).

5. **Connectives:** Similar to propositional logic, predicate logic uses logical connectives (e.g., $\wedge$ for AND, $\vee$ for OR, $\neg$ for NOT) to form complex statements from simpler ones.
6. **Constants:** Constants are specific objects or values in the domain. They are treated as fixed and do not vary like variables.

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \mid [\textit{Sentence}] \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
&\mid \textit{Quantifier}\ \textit{Variable}, \ldots\ \textit{Sentence} \\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term}, \ldots) \\
&\mid \textit{Constant} \\
&\mid \textit{Variable} \\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\text{OPERATOR PRECEDENCE} &: \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

7. **Terms:**
   - Terms are expressions that represent objects in the domain of discourse.
   - They can be variables, constants, or functions applied to arguments (e.g., f(x), John).
   - Function symbols, like "LeftLeg," are used to represent relationships or properties of objects without the need for separate constant symbols for each object.

- The semantics of terms is straightforward: a term like f(t1, ..., tn) refers to the result of applying the function represented by f to objects represented by t1, ..., tn.

8. **Complex Formulas:**

- Complex formulas are built using connectives, quantifiers, and atomic formulas.
- They can be constructed by combining atomic formulas with connectives (e.g., P(x) $\wedge$ Q(y)) or by quantifying variables over formulas (e.g., $\forall$ x P(x)).

For example: Richard the Lionheart, King of England from 1189 to 1199; his younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; and a crown.

¬Brother (LeftLeg(Richard), John)
Brother (Richard, John) $\wedge$ Brother (John, Richard)
King(Richard) $\vee$ King(John)
¬King(Richard) $\Rightarrow$ King(John) .

9. **Atomic sentences:**

An atomic sentence (or atom for short) is formed from a predicate symbol optionally followed by a ATOM parenthesized list of terms, such as

Brother (Richard, John).

This states, under the intended interpretation given earlier, that Richard the Lionheart is the brother of King John. Atomic sentences can have complex terms as arguments. Thus,

Married(Father (Richard), Mother (John))

states that Richard the Lionheart's father is married to King John's mother (again, under a suitable interpretation).

An atomic sentence is true in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.

Quantifier Negation:

Example:

Consider the following two propositions:

- Not every UB student majors in computer science: $\neg\, \forall x\, D(x)$.
- There is UB student who does not major in computer science: $\exists\, x \neg\, D(x)$.

Do these two statements have the same meaning? YES.

Example:

Consider the following two propositions:

- There is no UB student living in Amherst: $\neg\, \exists\, x\, A(x)$.
- Every UB student lives in a town other than Amherst: $\forall\, x \neg\, A(x)$.

Do these two statements have the same meaning? YES.

Quantifier negation: In general we have:

$$\begin{aligned} \neg\, \forall\, x\, P(x) &\equiv \exists\, x \neg\, P(x) \\ \neg\, \exists\, x\, P(x) &\equiv \forall\, x \neg\, P(x) \end{aligned}$$

Where $\equiv$ means logical equivalence as we have seen.

Rule: to negate a quantifier:

- move the negation to the inside;
- and switch $\exists$ to $\forall$, $\forall$ to $\exists$.

Nested Quantifiers

- Sometimes we have more complicated logical formulas that require nested quantifiers.
- "Nested" means one quantifier's scope contains another quantifier.

- $\forall x \forall y P(x, y) = T$ if $P(x, y) = T$ for all x and y.
- $\forall x \exists y P(x, y) = T$ if for any x there exists y such that $P(x, y) = T$.
- $\exists x \forall y P(x, y) = T$ if there exists x such that $P(x, y) = T$ for all y.
- $\exists x \exists y P(x, y) = T$ if these exists x and there exists y such that $P(x, y) = T$.

Order Matters:

Example:

Let the universe of discourse be pairs of real numbers.

- $\forall\, x\, \exists\, y\ (y > x) \equiv$ True
  But
- $\exists\, y\, \forall\, x\ (y > x) \equiv$ False

In general:

- $\forall x \forall y P(x, y) = T$ iff $\forall y \forall x P(x, y) = T$.
- $\exists x \exists y P(x, y) = T$ iff $\exists y \exists x P(x, y) = T$.
- If $\exists x \forall y P(x, y) = T$, then $\forall y \exists x P(x, y) = T$.
- If $\forall y \exists x P(x, y) = T$, $\exists x \forall y P(x, y)$ might be F.

Scope Matters:

In general, difference in scopes of quantifiers can lead to difference in truth values.

Example:

- $\forall\, x\ (P(x) \vee \neg\, P(x)) \equiv$ True
  But the truth value of
- $\forall\, x\ P(x) \vee \neg\, P(x)$
  depends on the truth value of free variable x. (The second x is not bounded by any quantifier.) So this formula is not even a proposition!

References:

1. **CS 50: Introduction to AI course (Harvard University)**
2. **First course in Artificial Intelligence by Prof Deepak Khemani**
3. **Artificial Intelligence: A Modern Approach Textbook by Peter Norvig and Stuart J. Russell**
4. **https:// cse.buffalo. edu/~xinhe/cse191**
5. **Javatpoint. com**