DAYANANDA SAGAR COLLEGE OF ENGINEERING
*(An Autonomous Institute Affiliated to VTU, Belagavi)*
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

# Knowledge Based Agents in AI

Intelligent agent should have the knowledge about the world.
Knowledge is the basic element for a human brain to know and understand the things logically. When a person becomes knowledgeable about something, he is able to do that thing in a better way. In AI, the agents which copy such an element of human beings are known as knowledge-based agents.

**Knowledge-based agent uses some task-specific knowledge to solve a problem efficiently.**

**A knowledge-based system comprises of two distinguishable features which are:**

- A Knowledge base
- An Inference Engine

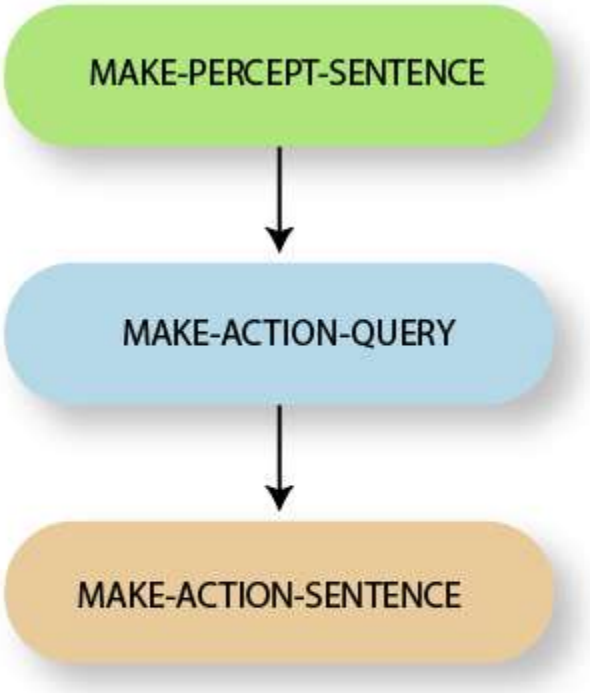
Knowledge base: **A Knowledge base represents the actual facts which exist in the real world. It is the central component of a knowledge-based agent. It is a set of sentences which describes the information related to the world.**

Inference Engine: **It is the engine of a knowledge-based system which allows to infer new knowledge in the system.**

When there is a need to **add/update** some new information or sentences in the knowledge-based system, we require an inference system. Also, to know what information is already known to the agent, we require the inference system. The technical words used for describing the mechanism of the inference system are: **TELL** and **ASK**. When the agent solves a problem, it calls the agent program each time. **The agent program performs three things:**

1. It **TELLS** the knowledge base what it has perceived from the environment.
2. It **ASKS** the knowledge base about the actions it should take?
3. It **TELLS** the action which is chosen, and finally, the agent executes that action.

## Generic Knowledge based agent




Functions hiding details of Knowledge Representation Language

**The functions are discussed below:**

- **MAKE-PERCEPT-SENTENCE()**

This function returns a sentence which tells the percieved information by the agent at a given time.

- **MAKE-ACTION-QUERY()**

This function returns a sentence which tells what action the agent must take at the current time.

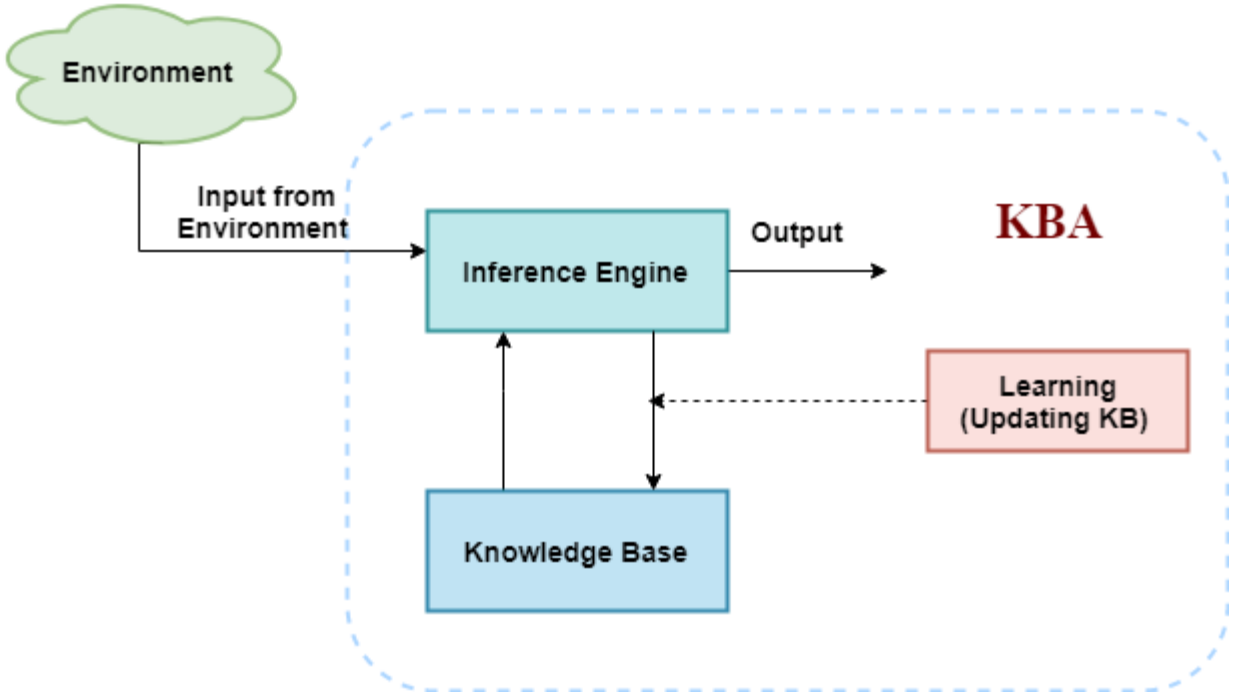
- **MAKE-ACTION-SENTENCE()**

This function returns a sentence which tells an action is selected as well as executed.

Let's understand the working of these functions under the Inference engine with the help of the below function:

functionKB-AGENT(percept ) returnsan action
persistent: KB, a knowledge base
t, a counter, initially 0, indicating time
TELL(KB,MAKE-PERCEPT-SENTENCE(percept , t ))
action ←ASK(KB,MAKE-ACTION-QUERY(t ))
TELL(KB,MAKE-ACTION-SENTENCE(action, t ))
t ←t + 1

## The architecture of knowledge-based agent:




The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) take input from the environment by perceiving the environment. The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

**Knowledge base:**

Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language.

**Inference system:**

Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.

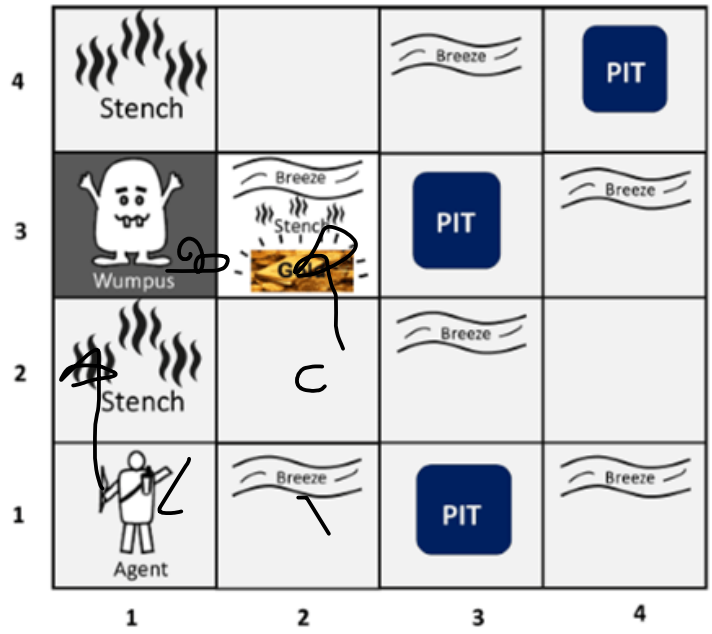
Inference system generates new facts so that an agent can update the KB.

# The Wumpus World in Artificial intelligence

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game **Hunt the Wumpus** by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

Following is a sample diagram for representing the Wumpus world. It is showing some rooms with Pits, one room with Wumpus and one agent at (1, 1) square location of the world.




**There are also some components which can help the agent to navigate the cave. These components are given as follows:**

a. The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.

b. The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.

c. There will be glitter in the room if and only if the room has gold.

d. The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

# PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

## Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- The game ends if either agent dies or came out of the cave.

## Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

## Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

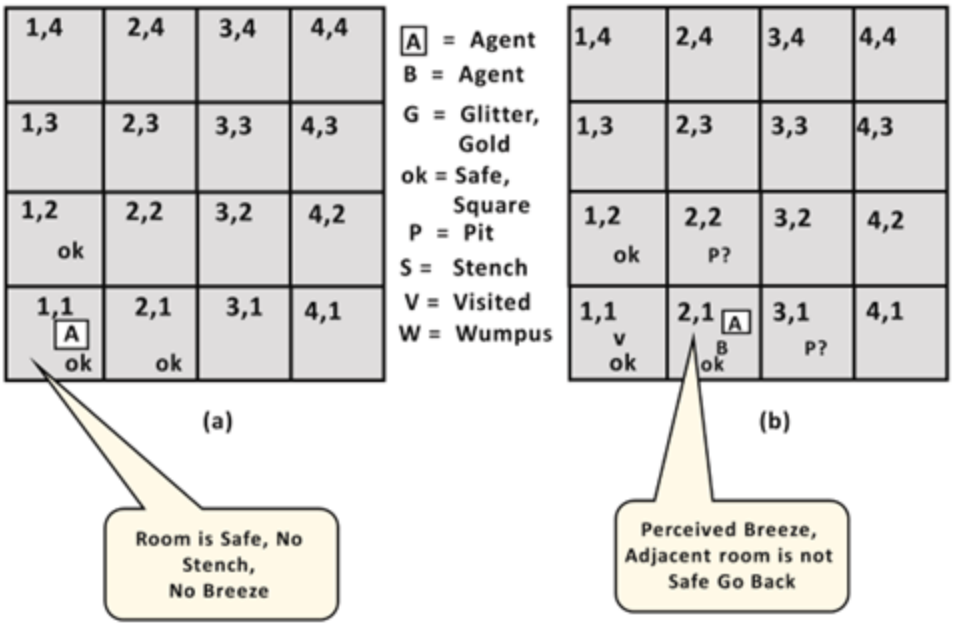
## Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.
- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
  **[Stench, Breeze, None, None, None]**.

# The Wumpus world Properties:

- **Partially observable:** The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- **Deterministic:** It is deterministic, as the result and outcome of the world are already known.
- **Sequential:** The order is important, so it is sequential.
- **Static:** It is static as Wumpus and Pits are not moving.
- **Discrete:** The environment is discrete.
- **One agent:** The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

## Exploring the Wumpus world:




**Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.**

**Agent's First step:**

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.

**Agent's second Step:**

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is

around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares

**Agent's third step:**

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2]

**Agent's fourth step:**

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

# Propositional Logic

There should be proper knowledge representation and reasoning. The knowledge should be represented properly. Otherwise the AI system will do wrong analysis. If we don’t represent properly then there will be syntax and semantics error. Output of the propositional logic is either true or false but not both.

The knowledge can be represented in different ways:
-Propositional logic

-Predicate logic
-Rules-If then
-Semantic net-Google Graphs
-frames
-Scripts.

**Syntax and Semantics of Propositional Logic**

*Syntax and semantics define a way to determine the truth value of the sentence.*

**Syntax:** The statements given in a problem are represented via propositional symbols. Each sentence consists of a single propositional symbol. The propositional symbol begins with an uppercase letter and may be followed by some other subscripts or letters. We have two fixed propositional symbols, i.e., True and False.

- **not(¬):** It is known as the **negation** of a sentence. A literal can be a positive literal or a negative literal.

- **and(Λ):** When a sentence is having **(Λ)** as the main connective. It is known as **Conjunction,** and its parts are known as **Conjuncts**.

- **or(V):** When a sentence is having **(V)** as the main connective. It is known as **Disjunction,** and its parts are known as **Disjuncts**.
-
- **implies(=>):** When **($Y_1$ V $Y_2$) => $Y_3$** is given, it is known as the **Implication of a sentence**. It is like **if->then** clause, where if this implies then it will happen. Implication is sometimes referred to as **Rules or if-then** statement. It can also be d**eno**ted as ( ) or ( ).
- **if and only if (**ó**):** It represents implication at both sides where the expression is $a_2 \Leftrightarrow a_3$. Such type of connective is called **biconditional implication**. It returns true if both sides satisfy one another, else returns false. This can also be denoted as (≡).

**Precedence Order of the Connectives**
Below table shows the precedence order of the connectives in their decreasing order:

| **Name** | **Symbol** |
|---|---|
| Parenthesis/ Brackets | () |
| Negation/not | ¬ or ~ |
| Conjuction/and | Λ |
| Disjunction/or | V |

| | |
|---|---|
| Implication | → |
| Biconditional/ if and only if | ó |

**Semantics:** It defines the rules to determine the truth of a sentence with respect to a specific model. A semantic should be able to compute the truth value of any given sentence.
**There are following five rules regarding the semantics of the complex sentences P and Q in a given model m :**
**¬P:** Its value will be false, iff it is true in the model m.
**(P Λ Q):** Its value is true, iff both P and Q are true in m.
**(P v Q):** Its value is true, iff either P is true, or Q is true in m.
**(P=> Q):** Its value is true, iff the value of P is false, and that of Q is true in m.
**(P ⇔ Q):** The value will be true, iff P and Q value is either true or false in the given model m.
**Note:** Here, iff means if and only if.
**These five connectives can also be understood with the help of the below described truth table:**

| P | Q | ¬ P | P ∧ Q | P v Q | P → Q | P ↔ 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 five logical Connectives

**Examples of Propositional Logic**
**Example 1: Consider the given statement:**
**If it is humid, then it is raining.**
**Solution:** Let, P and Q be two propositions.
P=It is humid.

Q=It is raining.

It is represented as (**P→Q).**

**Example 2: It is noon and Ram is sleeping.**
**Solution:** A= It is noon.
B= Ram is sleeping.

It is represented as **(A ∧ B).**

**Example 3: If it is raining, then it is not sunny.**

**Solution:** P= It is raining.
Q= It is sunny.

It is represented as **P → ( ~Q)**

**Example 4: Ram is a man or a boy.**
**Solution:** X= Ram is a man.
Y= Ram is a boy.

It is represented as **(X** V **Y).**

**Example 5: I will go to Delhi if and only if it is not humid.**
**Solution:** A= I will go to Delhi.
B= It is humid.

It is represented as **(A ⇔ B).**
**There can be many examples of Propositional logic.**

# Propositional Theorem Proving

*Theorem proving means to apply rules of inference directly to the sentences.*
**There are following concepts which are used for theorem proving:**

- **Logical Equivalence:** If the value of P and Q is true in the same set of models, then they are said to be logically equivalence.

| Rule Name | Rule |
|---|---|
| Idempotency Law | (A Λ A) = A (AV A) = A |
| Commutative Law | (A Λ B) = (B Λ A) (AV B) = (B V A) |
| De morgan's Law | ~(A Λ B) =( ~A V ~B) ~(A V B) = (~A Λ ~B) |
| Associative Law | AV(B V C) = (A V B) V C A Λ(B Λ C) = (A ΛB) Λ C |
| Distributive Law | A Λ(B V C) = (A Λ B) V (A Λ C)A V (B Λ C) = (A V B) Λ (A V C) |
| Contrapositive Law | A → B = ~A → ~B<br>~A → ~B (Converse of Inverse) |
| Implication Removal | A → B = ~A V B |

| | |
|---|---|
| Biconditional Removal | $A \Leftrightarrow B = (A \rightarrow B) \wedge (B \rightarrow A)$ |
| Absorption Law | $A \wedge (A \vee B) \equiv A$ $A \vee (A \wedge B) \equiv A$ |
| Double-negation elimination | $\sim(\sim A)=A$ |

**Table defining the rules used in Propositional logic where A, B, and C represents some arbitrary sentences.**

- **Validity:** If a sentence is valid in all set of models, then it is a valid sentence. Validity is also known as **tautology,** where it is necessary to have true value for each set of model.
- **Satisfiability:** If a sentence is true atleast for some set of values, it is a satisfiable sentence.

**Let's understand validity and satisfiability with the help of examples:**
**Example 1:**
**$(P \vee Q) \rightarrow (P \wedge Q)$**

| P | Q | P V Q | P Λ Q | (P V Q) → (P Λ Q) |
|---|---|---|---|---|
| False | False | False | False | True |
| False | True | True | False | False |
| True | False | True | False | False |
| True | True | True | True | True |

**Example 2:**

| A | B | A → B | (A → B) Λ A | ((A → B) Λ A) → B |
|---|---|---|---|---|
| False | False | True | False | True |
| False | True | True | False | True |
| True | False | False | False | True |
| True | True | True | True | True |

**$((A \rightarrow B) \wedge A) \rightarrow B$**
So, it is clear from the truth table that the given expression is valid as well as satisfiable.

# Inference Rules in Proposition Logic

Inference rules are those rules which are used to describe certain conclusions. The inferred conclusions lead to the desired goal state.

**There are following laws/rules used in propositional logic:**

- Modus Ponens: "If A is true, then B is true. A is true. Therefore, B is true."
- Modus Tollens: "If A is true, then B is true. B is not true. Therefore, A is not true."

- **Modus Tollen:** Let, P and Q be two propositional symbols:

**Rule:** Given, the negation of Q as (~Q).
If P→Q, then it will be (~P), i.e., the negation of P.
**Example:** If Aakash goes to the temple, then Aakash is a religious person. Aakash is not a religious person. Prove that Aakash doesn't go to temple.
**Solution:** Let, P= Aakash goes to temple.
Q= Aakash is religious. Therefore, (~Q)= Aakash is not a religious person.
**To prove:** ~P→~Q
By using **Modus Tollen** rule, P→Q, i.e., ~P→~Q (because the value of Q is (~Q)).
**Therefore, Aakash doesn't go to the temple.**

- **Modus Ponen:** Let, P and Q be two propositional symbols:

**Rule:** If P→Q is given, where P is positive, then Q value will also be positive.
**Example:** If Sheero is intelligent, then Sheero is smart. Sheero is intelligent. Prove that Sheero is smart.
**Solution:** Let, A= Sheero is intelligent.
B= Sheero is smart.
**To prove:** A→B.
By using **Modus Ponen** rule, A→B where A is positive. Hence, the value of B will be true. **Therefore, Sheero is smart.**

- **Syllogism:** It is a type of logical inference rule which concludes a result by using deducting reasoning approach. It is a valid deductive argument having two premises and a conclusion.
-

**Rule:** If there are three variables say P, Q, and R where
P→Q and Q→R then P→R.
**Example:** Given a problem statement:
If Ram is the friend of Shyam and Shyam is the friend of Rahul, then Ram is the friend of Rahul.

**Solution:** Let, P= Ram is the friend of Shyam.
Q= Shyam is the friend of Rahul.
R= Ram is the friend of Rahul.
It can be represented as: **If (P→Q) Λ (Q→R)= (P→R).**

- **Disjunctive Syllogism:** IN propositional logic, **disjunctive syllogism** (also known as **disjunction elimination** and **or elimination**, is a valid rule of inference. If we are told that at least one of two statements is true; and also told that it is not the former that is true; we can infer that it has to be the latter that is true. If *P* is true or *Q* is true and *P* is false, then *Q* is true. The reason this is called "disjunctive syllogism" is that, first, it is a syllogism, a three-step argument, and second, it contains a logical disjunction, which simply means an "or" statement. "P or Q" is a disjunction.

**Rule:** If (~P) is given and (P V Q), then the output is Q.
**Example:** Sita is not beautiful or she is obedient.
**Solution:** Let, (~P)= Sita is beautiful.
Q= She is obedient.
P= Sita is not beautiful.
It can be represented as (P V Q) which results Sita is obedient.

# Resolution Method in AI

Resolution method is an inference rule which is used in both Propositional as well as First-order Predicate Logic in different ways. This method is basically used for proving the satisfiability of a sentence. In resolution method, we use **Proof by Refutation** technique to prove the given statement.
The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause(which indicates contradiction). Resolution method is also called **Proof by Refutation**. Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true.

## Resolution Method in Propositional Logic

In propositional logic, resolution method is the only inference rule which gives a new clause when two or more clauses are coupled together.

**The process followed to convert the propositional logic into resolution method contains the below steps:**

- Convert the given axiom into clausal form,
- Apply and proof the given goal using negation rule.

- Use those literals which are needed to prove.
- Iteratively apply resolution to the set and add the resolvent to the set
- Continue until no further resolvents can be obtained or a null clause can be obtained.

## Conjunctive Normal Form(CNF)

In propositional logic, the resolution method is applied only to those clauses which are disjunction of literals. **There are following steps used to convert into CNF:**

1) Eliminate bi-conditional implication by replacing A ⇔ B with (A → B) Λ (B →A)

2) 2) Eliminate implication by replacing A → B with ¬A V B.

3) In CNF, negation(¬) appears only in literals, therefore we move it inwards as:

- **¬ ( ¬A) ≡ A** (double-negation elimination
- **¬ (A Λ B) ≡ ( ¬A V ¬B)** (De Morgan)
- **¬(A V B) ≡ ( ¬A Λ ¬B)** (De Morgan)

4) Finally, using distributive law on the sentences, and form the CNF as:
**$(A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \ldots \wedge (A_n \vee B_n)$.**

We illustrate the procedure by converting the sentence B1,1 ⇔ (P1,2 ∨ P2,1) into CNF. The steps are as follows:

1. Eliminate ⇔, replacing α ⇔ β with (α ⇒ β) ∧ (β ⇒ α).
   (B1,1 ⇒ (P1,2 ∨ P2,1)) ∧ ((P1,2 ∨ P2,1) ⇒ B1,1) .
2. Eliminate ⇒, replacing α ⇒ β with ¬α ∨ β:
   (¬B1,1 ∨ P1,2 ∨ P2,1) ∧ (¬(P1,2 ∨ P2,1) ∨ B1,1) .

3. CNF requires ¬ to appear only in literals, so we "move ¬ inwards" by repeated application of the following equivalences
   ¬(¬α) ≡ α (double-negation elimination)
   ¬(α ∧ β) ≡ (¬α ∨ ¬β) (De Morgan)
   ¬(α ∨ β) ≡ (¬α ∧ ¬β) (De Morgan)
   In the example, we require just one application of the last rule:
   (¬B1,1 ∨ P1,2 ∨ P2,1) ∧ ((¬P1,2 ∧ ¬P2,1) ∨ B1,1).

   4. Now we have a sentence containing nested ∧ and ∨ operators applied to literals. We apply the distributivity law from Figure 7.11, distributing ∨ over ∧ wherever possible.
   (¬B1,1 ∨ P1,2 ∨ P2,1) ∧ (¬P1,2 ∨ B1,1) ∧ (¬P2,1 ∨ B1,1).

   The original sentence is now in CNF, as a conjunction of three clauses.

Steps involved in converting propositional logic (PL) Sentences into CNF (Convolutional Neural Network)

→ Knowledge is represented in PL sentences.

→ PL sentences are complex. So we need to convert into CNF, so that machine can understand.

Rules to convert PL to CNF

1) Remove Biconditions using rule.

$$\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$

2) Remove implications using rule.

$$\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$$

3) Move negation inwards.

$$\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$
$$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$$
$$\neg(\neg\alpha) = \alpha$$

4) Apply Distributive and/or commutative law.

$$\left.\begin{aligned} \alpha \wedge (\beta \vee \gamma) &\equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \\ \alpha \vee (\beta \wedge \gamma) &\equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma) \end{aligned}\right\} \text{Distributive.}$$

$$\left.\begin{aligned} \alpha \wedge \beta &\equiv \beta \wedge \alpha \\ \alpha \vee \beta &\equiv \beta \vee \alpha \end{aligned}\right\} \text{commutative law.}$$

Ex:- $A \leftrightarrow (B \vee C)$

Step1 :-) Remove $\leftrightarrow$

$(A \rightarrow (B \vee C)) \wedge ((B \vee C) \rightarrow A)$

Step 2) Remove $\rightarrow$

$(\neg A \vee (B \vee C)) \wedge (\neg (B \vee C) \vee A)$

Step3) Move negation inward.

$(\neg A \vee (B \vee C)) \wedge ((\neg B \wedge \neg C) \vee A)$

Step 4) Distributive and Commutative

$(\neg A \vee B \vee C) \wedge ((\neg B \vee A) \wedge (\neg C \vee A))$

$(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$

# Conjuctive Normal Form

1) If maid stole Jewelleey then
butlee is not guilty

2) Eithee maid stole the jewelleey or
she milk the cow.

3) If maid milk the cow then
butlee got the ceeam

4) Theefore if butlee is guilty then
he got the ceeam.

Propositional Logic: PL

P: Maid stole the jewelleey.
Q: Butlee is guilty.
R: Maid milk the cow.
S: Butlee got the ceeam.

1) $P \rightarrow \neg Q$
2) $P \vee R$
3) $R \rightarrow S$
4) $Q \rightarrow S$

Applying the resolution method:

1) $P \rightarrow \neg Q$ will be $\neg P \vee \neg Q$
2) not needed (required)
3) $R \rightarrow S$ will be $\neg R \vee S$
4) $Q \rightarrow S$ will be $\neg Q \vee S$

We have to prove that the last statement If bultee is guilty then he got the cream. So we have to negate the last statement.

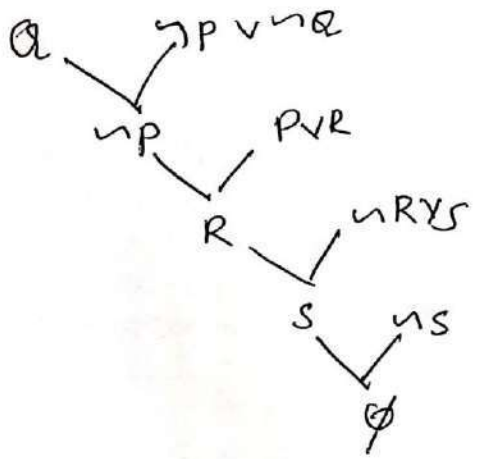
4) $\neg Q \vee S$

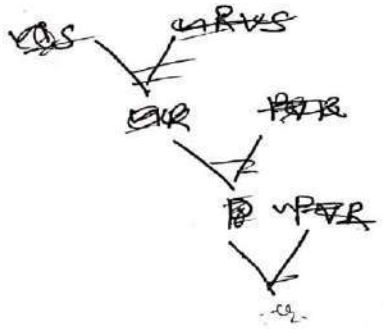
Negate $\neg(\neg Q \vee S)$

$= Q \wedge \neg S$

$= Q$

$\neg S$



## Example OF Propositional Resolution

Consider the following Knowledge Base:

1. The humidity is high or the sky is cloudy.
2. If the sky is cloudy, then it will rain.
3. If the humidity is high, then it is hot.
4. It is not hot.

**Goal:** It will rain.

Use propositional logic and apply resolution method to prove that the goal is derivable from the given knowledge base.

**Solution:** Let's construct propositions of the given sentences one by one:

1. Let, P: Humidity is high.
   Q: Sky is cloudy.

It will be represented as **P V Q.**

2) Q: Sky is cloudy. …**from(1)**

Let, R: It will rain.

It will be represented as b**Q →  R.**

3) P: Humidity is high. …**from(1)**

Let, S: It is hot.

It will be represented as **P  →  S.**

4) ¬S: It is not hot.

# Applying resolution method:

In (2), Q → R will be converted as (¬Q V R)

In (3), P →  S will be converted as (¬P V S)

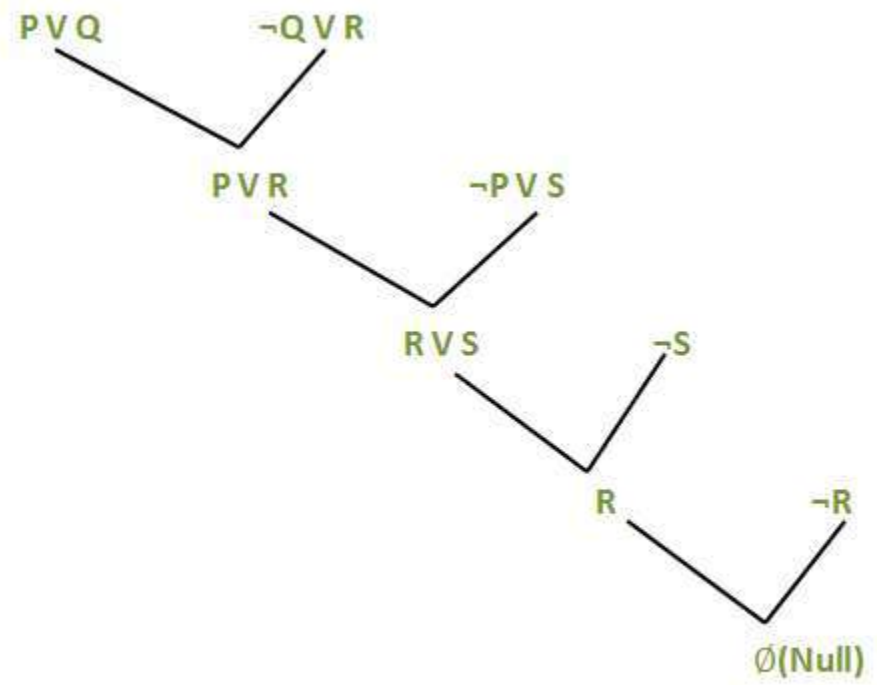
**Negation of Goal (¬R):** It will not rain.

**Finally, apply the rule as shown below:**

**After** applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a **Null clause ( Ø ).** Hence, the goal is achieved. Thus, It is not raining.

**Note:** We can have many examples of Proposition logic which can be proved with the help of Propositional resolution method.

We should prove that **Negation of Goal (¬R):** It will not rain is false then **R which is a goal it is raining is true.**




**After** applying Proof by Refutation (Contradiction) on the goal, the problem is solved, and it has terminated with a **Null clause ( ∅ ).** Hence, the goal is achieved. Thus, It is not raining.
**Note:** We can have many examples of Proposition logic which can be proved with the help of Propositional resolution method.

**Horn clauses:**

Horn clause, which is a disjunction of literals of which at most one is positive
Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. For example, the definite clause (¬L1,1 ∨ ¬Breeze ∨ B1,1) can be written as the implication (L1,1 ∧ Breeze) ⇒ B1,1. In the implication form, the sentence is easier to understand: it says that if the agent is in [1,1] and there is a breeze, then [1,1] is breezy.
Inference with Horn clauses can be done through the forward-chaining and backwardchaining algorithms.
Example: For example, if I am indoors and hear rain starting to fall, it might occur to me that the picnic will be canceled.
Inference with Horn clauses can be done through the forward-chaining and backwardchaining algorithms.

# Forward Chaining

Forward Chaining is the process which works on the basis of available data to make certain decisions. Forward chaining is the process of chaining data in the forward direction. In forward chaining, we start with the available data and use inference rules to extract data until the goal

is reached. Forward chaining is the concept of data and decision. From the available data, expand until a decision is made.

## Forward Chaining in Propositional Logic

--It is a form of reasoning which starts with atomic sentences in the knowledge base and applies the inference rules in the forward direction to extract more data until the goal is reached.

**Properties**
--It is a process of making the conclusion based on known facts or data by starting from the initial state and reach the goal.
--It is also called as data driven because we reach the goal using the available data.

**Let's see an example:**

1. If D barks and D eats bone, then D is a dog.
2. If V is cold and V is sweet, then V is ice-cream.
3. If D is a dog, then D is black.
4. If V is ice-cream, then it is Vanilla.

**Derive forward chaining using the given known facts to prove Tomy is black.**

- Tomy barks.
- Tomy eats bone.

**Solution:** Given Tomy barks.
From (1), it is clear:
If Tomy barks and Tomy eats bone, then Tomy is a dog.
From (3), it is clear:
If Tomy is a dog, then Tomy is black.
Hence, it is proved that **Tomy is black**.

# Backward Chaining

## Backward Chaining in Propositional Logic

A Backward chaining algorithm is a form of reasoning which starts with the goal and works backwards chaining through rules to find the known facts that supports the goal.

Properties:
1.Backward chaining is based on modus ponen inference rule.
2. In Backward chaining goal is broken into subgoals to prove the facts are true.
3. It is a goal driven approach as goal decides which goal is selected or used.

**Example of Backward Chaining in Propositional Logic**
**Let's consider the previous section example:**
**Given that:**

1. If D barks and D eats bone, then D is a dog.

2. If V is cold and V is sweet, then V is ice-cream.
3. If D is a dog, then D is black.
4. If V is ice-cream, then it is Vanilla.

**Derive backward chaining using the given known facts to prove Tomy is black.**

- Tomy barks.
- Tomy eats bone.

**Solution:**

1. **On replacing D with Tomy in (3), it becomes:**

If Tomy is a dog, then Tomy is black.

Thus, the goal is matched with the above axiom.

- **Now, we have to prove Tomy is a dog. …(new goal)**

Replace D with Tomy in (1), it will become:

If Tomy barks and Tomy eats bone, then Tomy is a dog.

## Effective Prepositional Model Checking

One approach based on backtracking search, and one local hill-climbing search. These algorithms are part of the "technology" of propositional logic. The algorithms we describe are for checking satisfiability:

A complete backtracking algorithm: DPLL algorithm

**a**) **Early termination**: The algorithm detects whether the sentence must be true or false, even with a partially completed model. A clause is true if any literal is true, even if the other literals do not yet have truth values; hence, the sentence as a whole could be judged true even before the model is complete. For example, the sentence $(A \vee B) \wedge (A \vee C)$ is true if A is true, regardless of the values of B and C. Early termination avoids examination of entire subtrees in the search space.

**b**)**A pure symbol** is a symbol that always appears with the same "sign" in all clauses. For example, in the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, and $(C \vee A)$

**c) Unit clause heuristic**: A unit clause was defined earlier as a clause with just one literal. For example, if the model contains B = true, then $(\neg B \vee \neg C)$ simplifies to $\neg C$, which is a unit clause.

DAYANANDA SAGAR COLLEGE OF ENGINEERING Dept. of Information Science & Engg
*(An Autonomous Institute Affiliated to VTU, Belagavi)*
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078
Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

# First Order Logic

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:** It refers to an entity that exists in the real world. **For example,** Ram, John, etc. are referred to as Objects.

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

- **Relations:** The relation of an object with the other object defines its relation**. For example,** brother, mother, king, etc. are some types of relations which exist in the real world.
- **Functions:** Any function performed by the object/on the object. **For example** writes, eats, etc. are some of the functions.

- # 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** | ∀, ∃ |

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

# 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)**.
- A term is a logical expression that refers to an object.

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

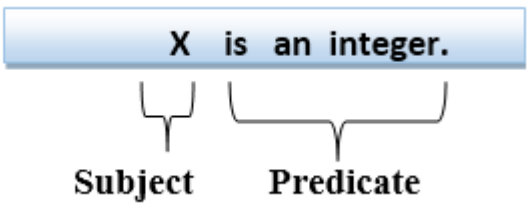
# Complex Sentences:

- 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:

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

- 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:

1. **Universal Quantifier, (for all, everyone, everything)**

# 1.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 ∀.

***Note: In universal quantifier we use implication "→".***

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

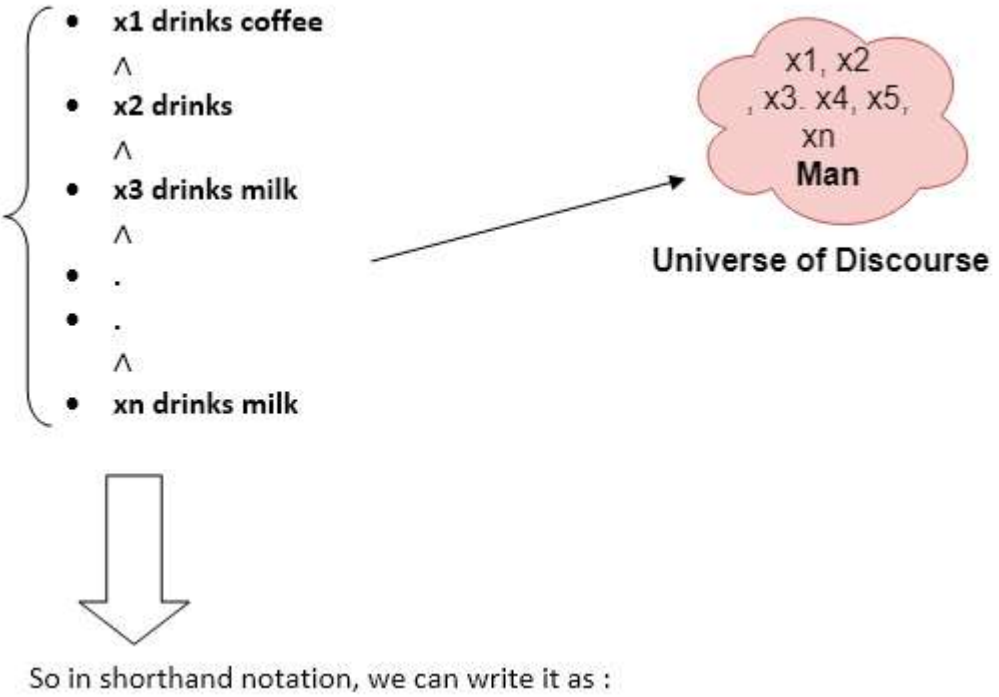
- **For all x**
- **For each x**
- **For every x.**

# Example:

**All man drink coffee.**

Let a variable x

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering




**∀x man(x) → drink (x, coffee).**

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

## 2. 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 ∃,

***Note: In Existential quantifier we always use AND or Conjunction symbol (∧).***

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

- **There exists a 'x.'**

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

- **For some 'x.'**
- **For at least one 'x.'**

# Example:

**Some students are intelligent.**

**∃x: student(x) ∧ intelligent(x)**

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

### 3. Nested Quantifiers:

It is the nesting of the same type of quantifier. One predicate is nested under the other predicate. Two quantifiers are nested if one is within the scope of the other.

Ex1:Everybody loves somebody
**Example** ∃ y ∀ x Loves(x, y)
For every person, there is someone that person loves
Ex2: Brothers are siblings
**∀x ∀y brother(x,y) →Sibling(x,y)**

4. **Equality:** We use the equality symbol to express that two terms refer to the same object**. For example,** Eleveenth_President(India)= Dr. APJ Abdul Kalam. Here, both LHS is equal to RHS. It means that both terms refer to the same entity/ person.

   Equality can also be used with the negation to insist that two terns are not the same object.
   Richard has at least two brothers
   ∃ x,y Brother(x, Richard) ∧ Brother (x, Richard) ∧ ¬(x==y)

Some Examples of FOL using quantifier:

DAYANANDA SAGAR COLLEGE OF ENGINEERING Dept. of Information Science & Engg
*(An Autonomous Institute Affiliated to VTU, Belagavi)*
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078
Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

**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)].**

Some more examples:

**Example 1:** Lipton is a tea.
**Solution:** Here, the object is Lipton.
It will be represented as **Tea(Lipton).**
**Note:** In this example, there is no requirement of quantifiers because the quantity is not specified in the given predicate.

**Example 2:** Every man is mortal.
**Solution:** Here, the quantifier is the universal identifier, and the object is man.
Let x be the man.
Thus, it will be represented as **∀ x: man(x) → mortal(x).**

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

**Example 3:** All girls are beautiful.
**Solution:** Here, we are talking about all girls. It means universal quantifier will be used. The object is girls. Let, y be the girls.
Therefore, it will be represented as **∀y:girls(y) → beautiful(y).**

**Example 4:** All that glitters is not gold.
**Solution:** Here, we will represent gold as x.
Therefore, it will be represented as **∀x: glitters(x) → ¬gold(x).**

**Example 5:** Some boys are obedient.
**Solution:** Here, boys are objects. The quantifier used will be existential quantifier. Let x be the boys. Thus, it will be represented as
**∃x: boys(x) → obedient(x).**

**Example 6:** Some cows are black and some cows are white.
**Solution:** Let, x be the cows. Therefore, it will be represented as:
**∃x: cows(x) → black(x) Λ white(x).**

**Example 6:** All kings are person:i

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

For all x, if x is a king, then x is a person

## Some more examples:

1. "Every house is a physical object" is translated as

∀x house(x) → physical object(x), where house and physical object are unary predicate symbols.

2. "Some physical objects are houses"'

is translated as ∃x.(physical object(x) ∧ house(x))

3. "every house is owned by somebody" is translated as
   ∀x ∃y. (house(x) → owns(y, x)),

4. Some Dogs bark
   ∃x.(dog(x) ∧ bark(x))

DAYANANDA SAGAR COLLEGE OF ENGINEERING Dept. of Information Science & Engg
*(An Autonomous Institute Affiliated to VTU, Belagavi)*
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078
Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

# Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

**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**".

## FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

**1. Universal Generalization:**

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

**Example:** Let's represent, P(c): "**A byte contains 8 bits**", so for **∀ x P(x)** "**All bytes contain 8 bits**.", it will also be true.

**2. Universal Instantiation:**

- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$ .

**Example:1.**

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c)

**Example: 2.**

Every man is mortal.

It is represented as ∀ x: man(x) → mortal(x).
In UI, we can infer different sentences as:
man(John) → mortal(John)
man(Aakash) → mortal(Aakash), etc.

**3. Existential Instantiation:**

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- Notation: Let, the variable be v which is replaced by a constant symbol k for any sentence α. The value of k is unique as it does not appear for any other sentence in the knowledge base. Such type of constant symbols are known as Skolem constant. As a result, EI is a special case of Skolemization process.

It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$

**Example:**

**For example: ∃x:steal(x, Money).**
We can infer from this: **steal(Thief, Money)**

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

**4. Existential introduction**

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$
- **Example: Let's say that,**
  "Priyanka got good marks in English."
  "Therefore, someone got good marks in English."

# Generalized Modus Ponens Rule:

For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.

Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."

According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)}{SUBST(\theta, q)}$$

t is a lifted version of Modus Ponen as it uplifts the Modus Ponens from ground propositions to FOPL. Generalized Modus Ponen is more generalized than Modus Ponen. It is because, in generailzed, the known facts and the premise of the implication are matched only upto a substitution, instead of its exact match.

**Example:**

**We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.**

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

1. Here let say, p1' is king(John) p1 is king(x)
2. p2' is Greedy(y) p2 is Greedy(x)
3. θ is {x/John, y/John} q is evil(x)
4. SUBST(θ,q).

## Unification

# 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 $\Psi_1$ and $\Psi_2$ be two atomic sentences and $\sigma$ be a unifier such that, $\boldsymbol{\Psi_1\sigma = \Psi_2\sigma}$, then it can be expressed as **UNIFY($\boldsymbol{\Psi_1}$, $\boldsymbol{\Psi_2}$)**.

**Example1: Find the MGU for Unify{King(x), King(John)}**

Let $\Psi_1$ = King(x), $\Psi_2$ = 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.

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

**E.g.2.** 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.

# Unification Algorithm:

**Algorithm: Unify($\Psi_1$, $\Psi_2$)**

```
Step. 1: If Ψ1 or Ψ2 is a variable or constant, then:
      a) If Ψ1 or Ψ2 are identical, then return NIL.
      b) Else if Ψ1is a variable,
            a. then if Ψ1 occurs in Ψ2, then return FAILURE
            b. Else return { (Ψ2/ Ψ1)}.
      c) Else if Ψ2 is a variable,
            a. If Ψ2 occurs in Ψ1 then return FAILURE,
            b. Else return {( Ψ1/ Ψ2)}.
      d) Else return FAILURE.
Step.2: If the initial Predicate symbol in Ψ1 and Ψ2 are not same, then
return FAILURE.
Step. 3: IF Ψ1 and Ψ2 have a different number of arguments, then return
FAILURE.
Step. 4: Set Substitution set(SUBST) to NIL.
```

Dr. Rajeshwari.J
Professor
Dayananda Sagar College of Engineering

Step. 5: For i=1 to the number of elements in $\Psi_1$.

a) Call Unify function with the ith element of $\Psi_1$ and ith element of $\Psi_2$, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

# Examples

**1. Find the MGU of UNIFY(prime (11), prime(y))**

Here, $\Psi_1$ = {prime(11) , and $\Psi_2$ = prime(y)}
$S_0$ => {prime(11) , prime(y)}
SUBST θ= {11/y}

$S_1$ => {prime(11) , prime(11)} , **Successfully unified.**
**Unifier: {11/y}.**

**2.UNIFY(knows(Richard, x), knows(Richard, John))**

Here, $\Psi_1$ = knows(Richard, x), and $\Psi_2$ = knows(Richard, John)
$S_0$ => { knows(Richard, x); knows(Richard, John)}
SUBST θ= {John/x}
$S_1$ => { knows(Richard, John); knows(Richard, John)}, **Successfully Unified.**
**Unifier: {John/x}.**

- ✓ the ―highest‖ common descendant of any two nodes is the result of applying their most general unifier.
- ✓ predicate with n arguments contains O(2n ) nodes (in our example, we have two arguments, so our lattice has four nodes)
- ✓ Repeated constants = slightly different lattice.

***3. Forward Chaining***

***First-Order Definite Clauses:***

A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. The following are first-order definite clauses: Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified.

$$King(x) \wedge Greedy(x) \Rightarrow Evil(x) .$$
$$King(John) .$$
$$Greedy(y) .$$

Consider the following problem;

***"The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American."***

We will represent the facts as first-order definite clauses

". . . It is a crime for an American to sell weapons to hostile nations":

**Example Knowledge Base:**

- ….it is a crime for an American to sell weapons to hostile nations:
  **Rule 1.** $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- Nono . . . has some missiles,
  i.e., $\exists\, x\ Owns(Nono, x) \wedge Missile(x)$:
  **Rule 2.** $Owns(Nono, M_1)$ **and**
  **Rule 3.** $Missile(M_1)$
- . . . all of its missiles were sold to it by Colonel West
  **Rule 4.** $\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- Missiles are weapons:
  **Rule 5.** $Missile(x) \Rightarrow Weapon(x)$
- An enemy of America counts as "hostile":
  **Rule 6.** $Enemy(x, America) \Rightarrow Hostile(x)$
- West, who is American . . .
  **Rule 7.** $American(West)$
- The country Nono, an enemy of America . . .
  **Rule 8.** $Enemy(Nono, America)$

***A simple forward-chaining algorithm:***

- Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions lo the known facts
- The process repeats until the query is answered or no new facts are added. Notice that a fact is not "new" if it is just *renaming*of a known fact.

We will use our crime problem to illustrate how FOL-FC-ASK works. The implication sentences are (1), (4), (5), and (6). Two iterations are required:

- ✓ On the first iteration, rule (1) has unsatisfied premises.
  Rule (4) is satisfied with *{x/Ml),* and *Sells (West,* M1, *Nono)* is added. Rule (5) is satisfied with *{x/M1) a*nd *Weapon (M1)* is added.
  Rule (6) is satisfied with *{x/Nono},* and *Hostile (Nono)* is added.
- ✓ On the second iteration, rule (1) is satisfied with *{x/West, Y/MI, z /Nono),* and *Criminal* (*West)* is added.
  It is ***sound,*** because every inference is just an application of Generalized Modus Ponens, it is ***complete***for definite clause knowledge bases; that is, it answers every query whose answers are entailed by any knowledge base of definite clauses

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  inputs: KB, the knowledge base, a set of first-order definite clauses
          α, the query, an atomic sentence
  local variables: new, the new sentences inferred on each iteration

  repeat until new is empty
    new ← { }
    for each rule in KB do
      (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
      for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n)
                  for some p'1, ..., p'n in KB
        q' ← SUBST(θ, q)
        if q' does not unify with some sentence already in KB or new then
          add q' to new
          φ ← UNIFY(q', α)
          if φ is not fail then return φ
    add new to KB
  return false
```

**Figure 9.3** A conceptually straightforward, but very inefficient, forward-chaining algorithm. On each iteration, it adds to $KB$ all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in $KB$. The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.

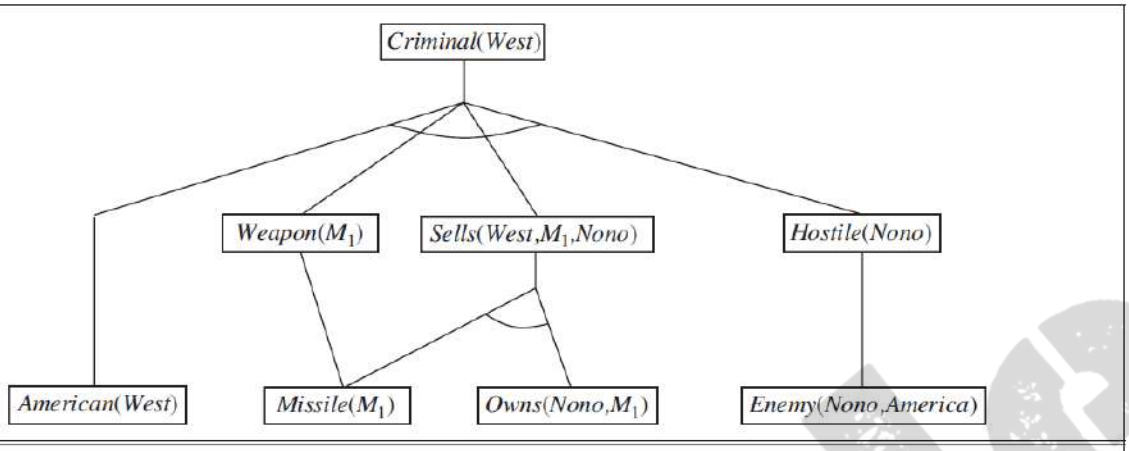



**Figure 9.4** The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.

**Efficient forward chaining:**

The above given forward chaining algorithm was lack with efficiency due to the the three sources of complexities:

- ✓ Pattern Matching
- ✓ Rechecking of every rule on every iteration even a few additions are made to rules
- ✓ Irrelevant facts

***1.*** **Matching rules against known facts:**

For example, consider this rule,

Missile(x) A Owns (Nono, x) =>Sells (West, x, Nono).

The algorithm will check all the objects owned by Nono in and then for each object, it could check whether it is a missile. This is the ***conjunct ordering problem:***

―Find an ordering to solve the conjuncts of the rule premise so that the total cost is minimized‖. The **most constrained variable** heuristic used for CSPs would suggest ordering the conjuncts to look for missiles first if there are fewer missiles than objects that are owned by Nono.

The connection between pattern matching and constraint satisfaction is actually very close. We can view each conjunct as a constraint on the variables that it contains-for example, Missile(x) is a unary constraint on x. Extending this idea, we can express everyfinite-domain CSP as a single definite clause together with some associated ground facts. Matching a definite clause against a set of facts is NP-hard

***2.*** **Incremental forward chaining:**

On the second iteration, the rule ***Missile (x) =>Weapon (x)***

Matches against Missile (M1) (again), and of course the conclusion Weapon(x/M1) is already known so nothing happens. Such redundant rule matching can be avoided if we make the following observation:

―Every new fact inferred on iteration t must be derived from at leastone new fact inferred on iteration t – 1‖.

This observation leads naturally to an incremental forward chaining algorithm where, at iteration t, we check a rule only if its premise includes a conjunct p, that unifies with a fact p: newly inferred at iteration t - 1. The rule matching step then fixes p, to match with p', but allows the other conjuncts of the rule to match with facts from any previous iteration.

***3.*** **Irrelevant facts:**

- One way to avoid drawing irrelevant conclusions is to use backward chaining.
- Another solution is to restrict forward chaining to a selected subset of rules
- A third approach, is to rewrite the rule set, using information from the goal.so that only relevant variable bindings-those belonging to a so-called **magic** set-are considered during forward inference.

For example, if the goal is Criminal (West), the rule that concludes Criminal (x) will be rewritten to include an extra conjunct that constrains the value of x:

Magic(x) A American(z) A Weapon(y)A Sells(x, y, z) A Hostile(z) =>Criminal(x )

The fact *Magic (West)* is also added to the KB. In this way, even if the knowledge base contains facts about millions of Americans, only Colonel West will be considered during the forward inference process.

***4. Backward Chaining***

This algorithm work backward from the goal, chaining through rules to find known facts that support the proof. It is called with a list of goals containing the original query, and returns the set of all substitutions satisfying the query. The algorithm takes the first goal in the list and finds every clause in the knowledge base whose **head,** unifies with the goal. Each such clause creates a new recursive call in which **body,** of the clause is added to the goal stack .Remember that facts are clauses with a head but no body, so when a goal unifies with a known fact, no new sub goals are added to the stack and the goal is solved. The algorithm for backward chaining and proof tree for finding criminal (West) using backward chaining are given below.

```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

generator FOL-BC-OR(KB, goal, θ) yields a substitution
  for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs, rhs) ← STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ' in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
      yield θ'

generator FOL-BC-AND(KB, goals, θ) yields a substitution
  if θ = failure then return
  else if LENGTH(goals) = 0 then yield θ
  else do
    first, rest ← FIRST(goals), REST(goals)
    for each θ' in FOL-BC-OR(KB, SUBST(θ, first), θ) do
      for each θ'' in FOL-BC-AND(KB, rest, θ') do
        yield θ''
```

**Figure 9.6** A simple backward-chaining algorithm for first-order knowledge bases.

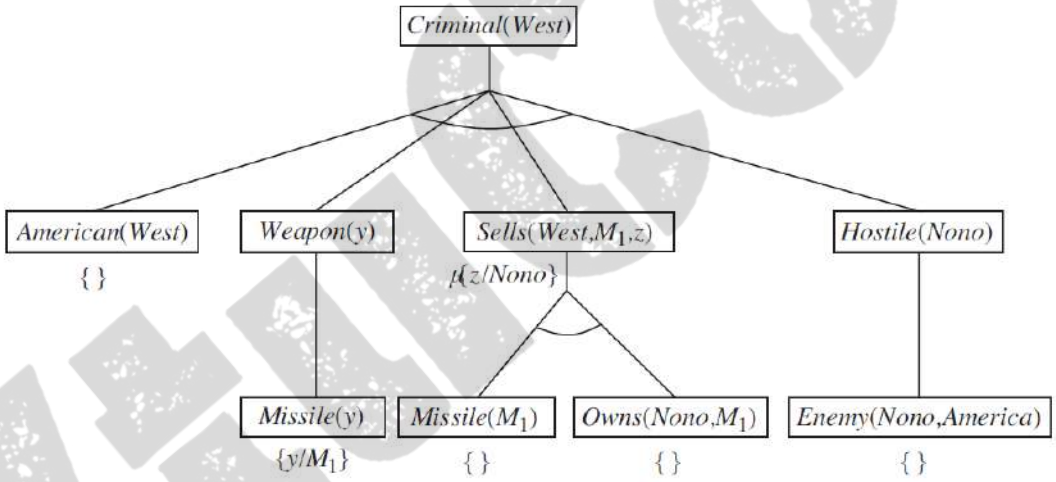



**Figure 9.7** Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove $Criminal(West)$, we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally $Hostile(z)$, $z$ is already bound to $Nono$.