
ARTIFICIAL INTELLIGENCE

Dr. Rajeshwari.J

2.4 ADVERSARIAL SEARCH

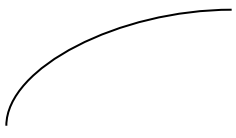
Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search. Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**. **Tic-tac-toe, chess, checkers**, etc., are such type of games where no luck factor works, only mind works.

Games:

Mathematically, this search is based on the concept of **'Game Theory.'** ***According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.'***

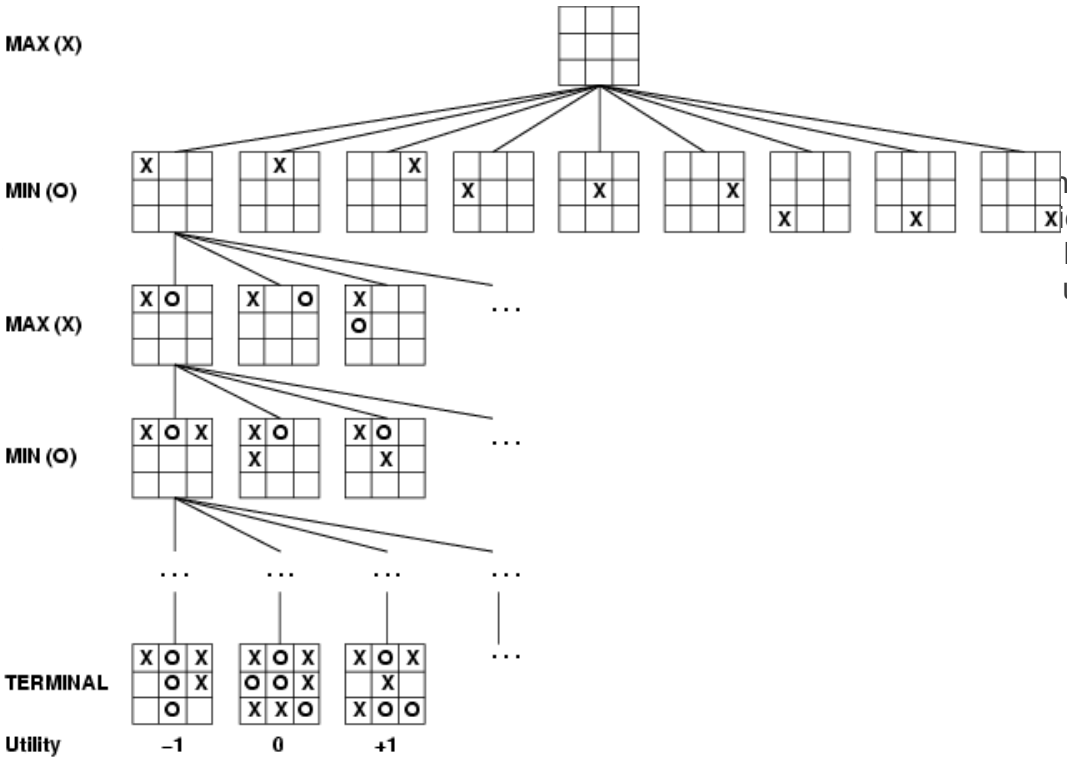
Elements of Game Playing search
To play a game, we use a game tree to know all the possible choices and to pick the best one out. There are following elements of a game-playing:

- **S_0:** It is the initial state from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves to be used in a state.
- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended and returns true.
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**. The price which the winner will get i.e.
- **(-1):** If the PLAYER loses.
- **(+1):** If the PLAYER wins.
- **(0):** If there is a draw between the PLAYERS.



We are opponents- I win, you loose.

Let's understand the working of the elements with the help of a game tree designed for tic-tac-toe. Here, the *node represents the game state and edges represent the moves taken by the players.*



ish the game
ich will decide
be an optimal
under limited

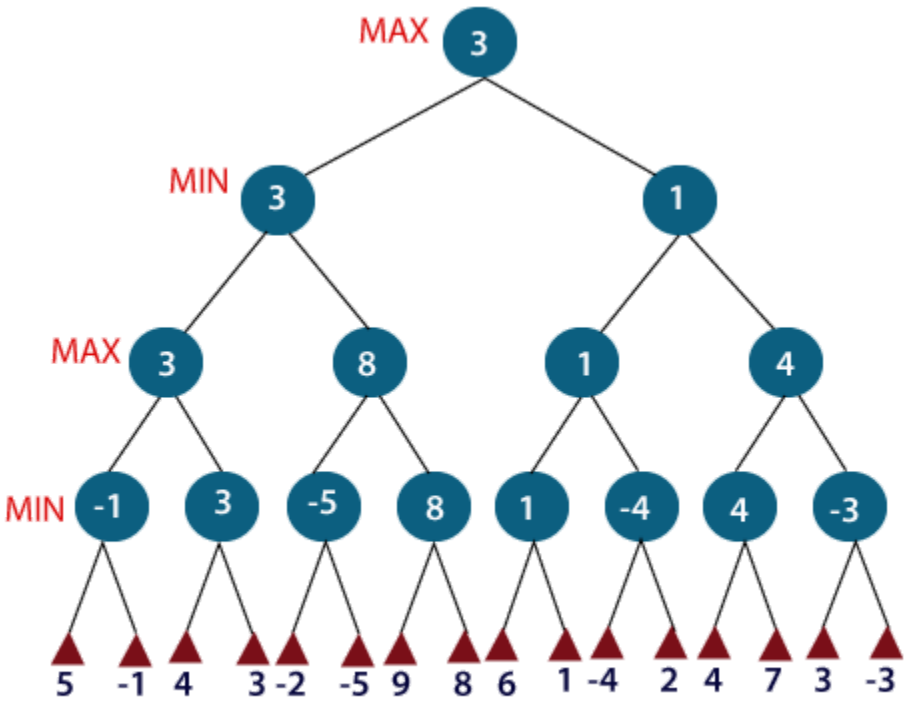
Players will be two namely:

- **MIN:** Decrease the chances of **MAX** to win the game.
- **MAX:** Increases his chances of winning the game.

Algorithm:

MINIMAX algorithm is a backtracking algorithm where it backtracks to pick the best move out of several choices. MINIMAX strategy follows the **DFS (Depth-first search)** concept. Here, we have two players **MIN and MAX,** and the game is played alternatively between them, i.e., when **MAX** made a move, then the next turn is of **MIN.** It means the move made by MAX is fixed and, he cannot change it. The same concept is followed in DFS strategy, i.e., we follow the same path and cannot change in the middle. That's why in MINIMAX algorithm, instead of BFS, we follow DFS.

- Keep on generating the game tree/ search tree till a limit **d.**
- Compute the move using a heuristic function.
- Propagate the values from the leaf node till the current position following the minimax strategy.
- Make the best move from the choices.

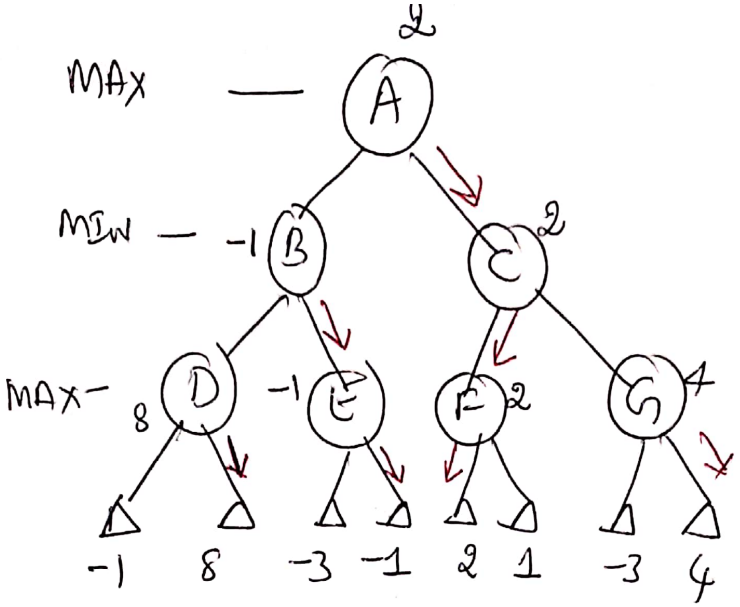


For example, in the above figure, the two players **MAX** and **MIN** are there. **MAX** starts the game by choosing one path and propagating all the nodes of that path. Now, **MAX** will backtrack to the initial node and choose the best path where his utility value will be the maximum. After this, its **MIN** chance. **MIN** will also propagate through a path and again will backtrack, but **MIN** will choose the path which could minimize **MAX** winning chances or the utility value.

So, if the level is minimizing, the node will accept the minimum value from the successor nodes. If the level is maximizing, the node will accept the maximum value from the successor.

```
Function minmax(N)
Begin
If Nis a leaf then
        Return the payoff(rewards) of the leaf
```

```
Else
  Let N1,N2… b ethe successor of the node
  If N is the minnode then
    Return min(Minimax(N1), Minimax(N2)….)
Else
  Return max(Minimax(N1), Minimax(N2)….)
```



Basic Strategy

→ Assume one can assign a payoff to each final position → called utility (Rewards)

→ Propopagate the values backwards from final position.

→ Assume the opponent always makes moves worst for us.

→ Pick best move on own turn.

→ At D it chooses MAX 8 because. It our turn we need to increase the utility (Rewards).

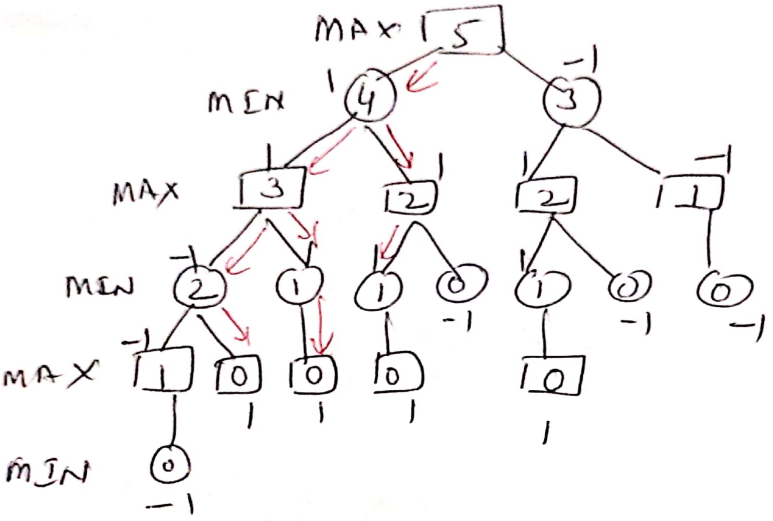
→ At B it chooses MIN -1 because the opponent wants to decrease the utility (Rewards) of the MAX player

→ If Root node gets final value as +ve, then MAX has played well

→ If Root node gets final value as -ve, then Min has played well, MAX has lost

→ If root node gets final value as zero, then both players played equally well and hence it draw.

Game tree for 5-stone



MAX node Removes first either 2 or 1 node At (3) (4)

and continues.

MAX gets reward as 1

MIN gets reward as -1

If we see the root MAX will win the game since it has the value 1 and choose the possible paths to get the reward as 1

NOTE: Extra Information: The utility value is just some arbitrary value that the player receives when arriving at a certain state in the game. For instance, in Tic-tac-toe, your utility function could simply be 1 for a win, 0 for a tie, or -1 for a loss. Running minmax on this would at best find a set of actions that result in 1 (a win).

Another example would be chess (not that you can feasibly run minimax on a game of chess). Say your utility function comes from a certain number that is based on the value of the piece you captured or lost

Alpha-beta Pruning

Drawback of MINIMAX algorithm:

Alpha-beta pruning is an advance version of MINIMAX algorithm. The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity. But as we know, the performance measure is the first consideration for any optimal algorithm. Therefore, alpha-beta pruning reduces this drawback of minimax strategy by less exploring the nodes of the search tree.

The method used in alpha-beta pruning is that it **cutoff the search** by exploring less number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique

Alpha-beta pruning works on two threshold values, i.e., **α (alpha)** and **β (beta).**

- **α:** It is the best highest value, a **MAX** player can have. It is the lower bound, which represents negative infinity value.
- **β:** It is the best lowest value, a **MIN** player can have. It is the upper bound which represents positive infinity.

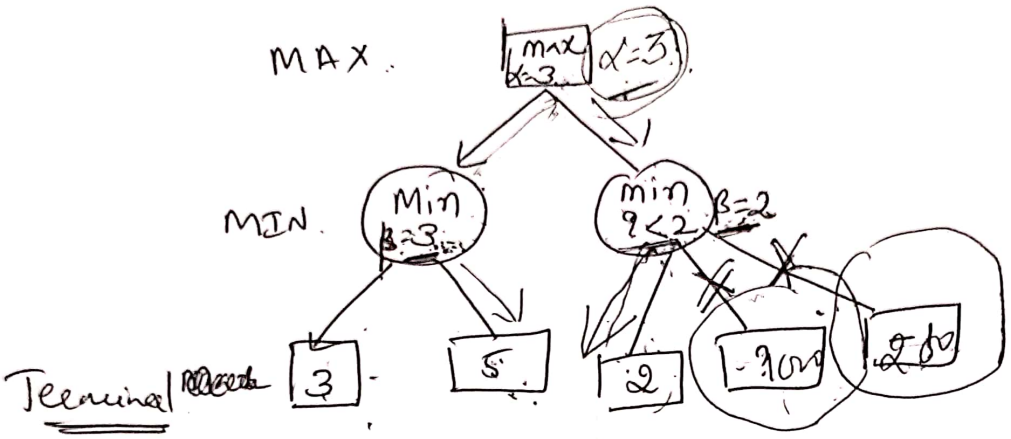
So, each MAX node has α-value, which never decreases, and each MIN node has β-value, which never increases.

Working of Alpha-beta Pruning:(Algorithm)
Consider the below example of a game tree where **P** and **Q** are two players. The game will be played alternatively, i.e., chance by chance. Let, **P** be the player who will try to win the game by maximizing its winning chances. **Q** is the player who will try to minimize **P'**s winning chances. Here, **α** will represent the maximum value of the nodes, which will be the value for **P** as well. **β** will represent the minimum value of the nodes, which will be the value of **Q**.

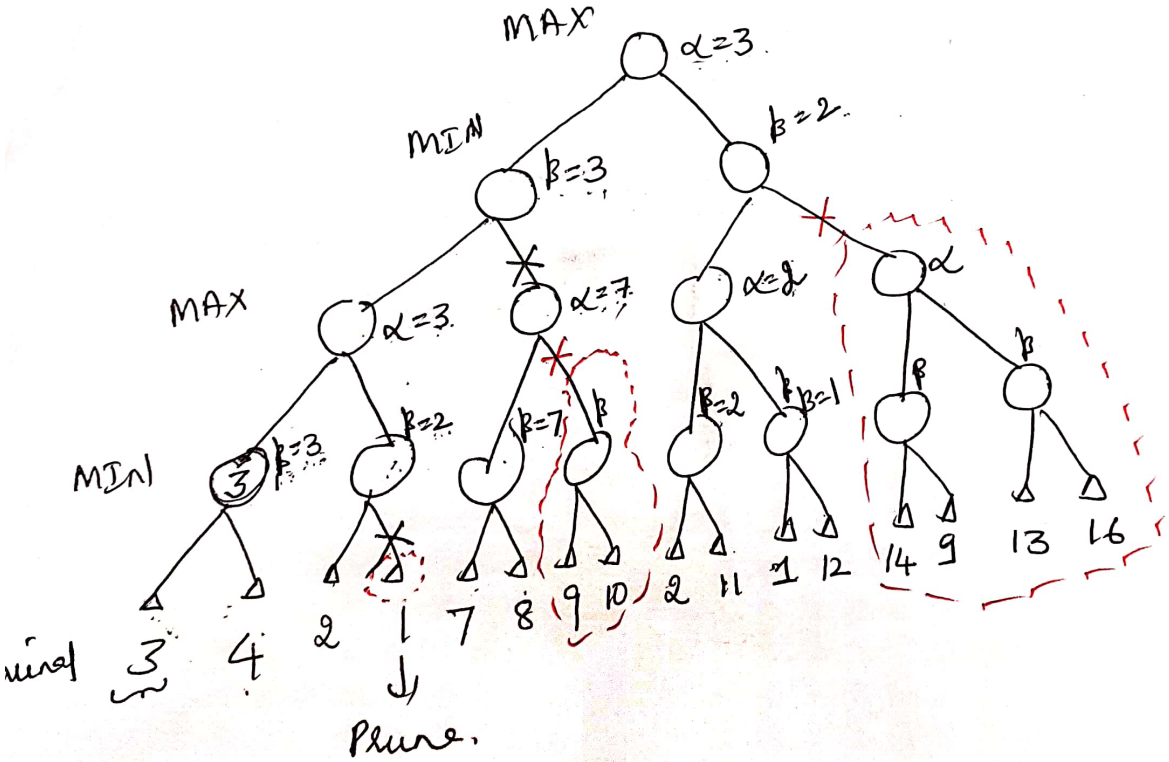
- Both will play the game alternatively.
- The game will be started from the last level of the game tree, and the value will be chosen accordingly.
- Like in the below figure, the game is started by player Q. He will pick the leftmost value of the TERMINAL and fix it for beta (β). Now, the next TERMINAL value will be compared with the β-value. If the value will be smaller than or equal to the β-value, replace it with the current β-value otherwise no need to replace the value.

- After completing one part, move the achieved β-value to its upper node and fix it for the other threshold value, i.e., α.
- Now, its P turn, he will pick the best maximum value. P will move to explore the next part only after comparing the values with the current α-value. If the value is equal or greater than the current α-value, then only it will be replaced otherwise we will prune the values.
- The steps will be repeated unless the result is not obtained.

Alpha Beta Pruning



Alphabeta pruning
→ cut off search by exploring less no of nodes



Note:
Utility can be thought of as a way to "score" each possible move based on its potential to result in a win. How utility is calculated is entirely up to the programmer. It can incorporate a large variety of factors and weigh them as the programmer sees fit. For instance, number of blank spaces on the board, the location of the opponent's current pieces, the location of our current pieces, how close we are to a winning formation, etc. all might be factors to consider in calculating the utility of a particular move. Let's take tic-tac-toe, for example, which can have relatively simple utility measures. The figure below displays a tic-tac-toe board midway through the game with a very simple (probably not optimal) utility rule. We can see that it's *X*'s turn, and there are only 3 possible moves, and hence, 3 child nodes. For each possible move, utility is calculated using the below utility rule.

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

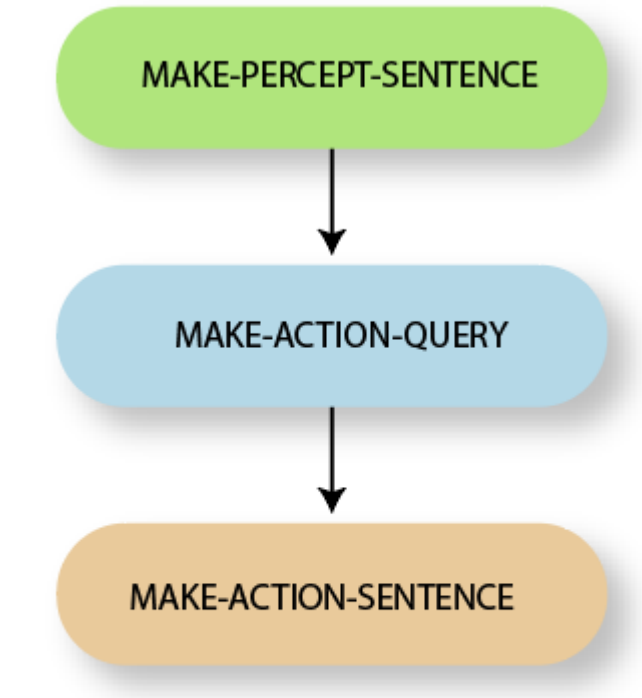
Knowledge base: A Knowledge base represents the actual facts which exist in the real world. It is the central component of a knowledge-based agent. It is a set of sentences which describes the information related to the world.

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.

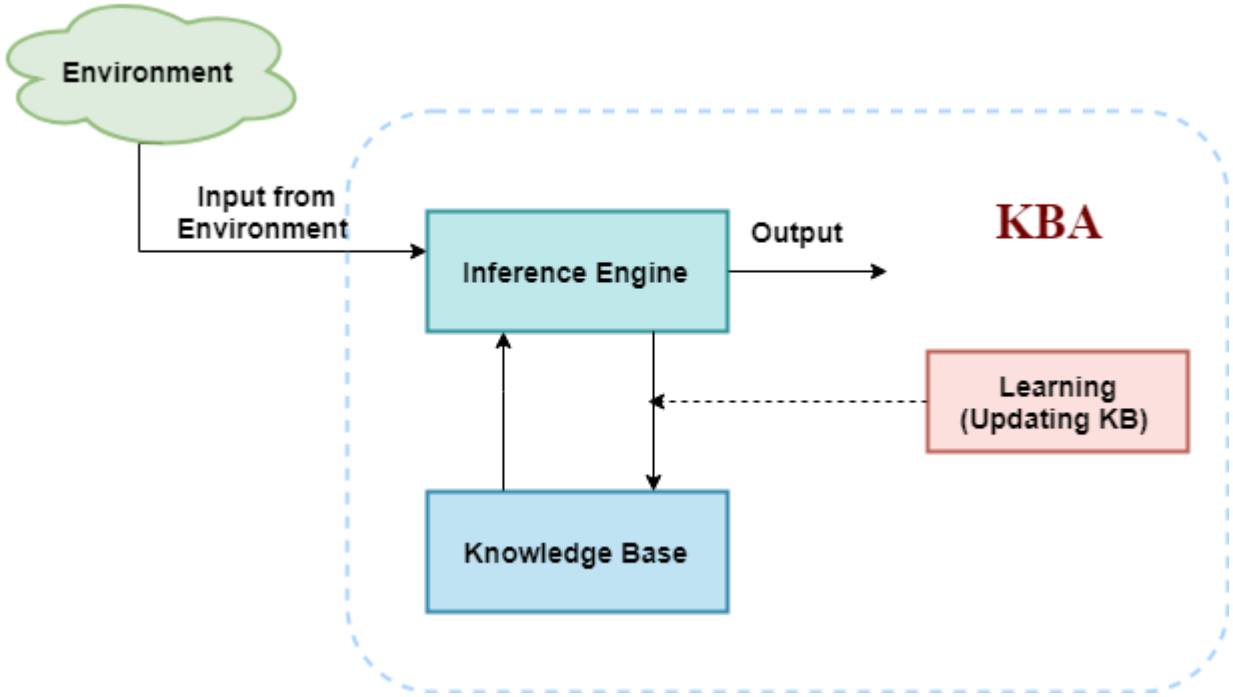
- **MAKE-ACTION-SENTENCE()**

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.

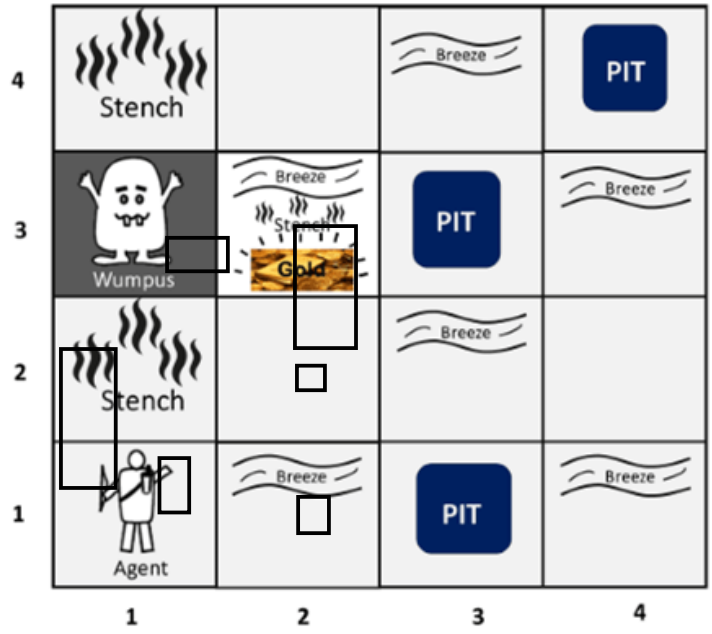
Inference system generates new facts so that an agent can update the KB.

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:

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

a. The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
b. There will be glitter in the room if and only if the room has gold.
c. 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:

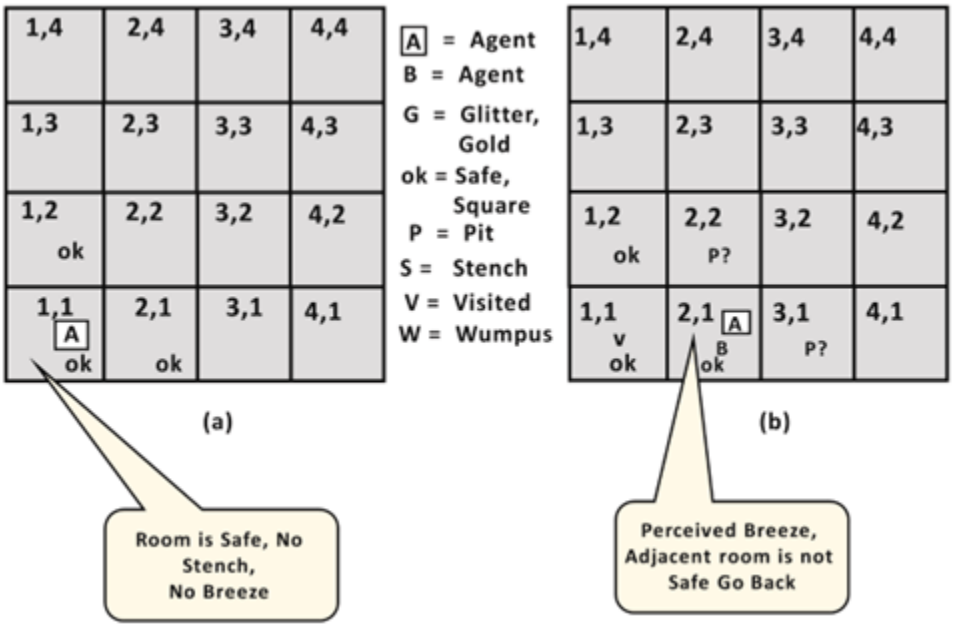
- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **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 $\mathbf{a}_2 \Leftrightarrow \mathbf{a}_3$. 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 Λ 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** ∨ **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 ∧ A) = A (A∨ A) = A
Commutative Law	(A ∧ B) = (B ∧ A) (A∨ B) = (B ∨ A)
De morgan's Law	~(A ∧ B) =(~A ∨ ~B) ~(A ∨ B) = (~A ∧ ~B)
Associative Law	A∨(B ∨ C) = (A ∨ B) ∨ C A ∧(B ∧ C) = (A ∧B) ∧ C
Distributive Law	A ∧(B ∨ C) = (A ∧ B) ∨ (A ∧ C)A ∨ (B ∧ C) = (A ∨ B) ∧ (A ∨ C)
Contrapositive Law	A → B = ~A → ~B ~A → ~B (Converse of Inverse)
Implication Removal	A → B = ~A ∨ B
Biconditional Removal	A ⇔ B = (A → B) ∧ (B → A)
Absorption Law	A ∧ (A ∨ B) ≡ A A∨ (A ∧ B) ≡A
Double-negation elimination	~(~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 V Q) → (P Λ 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 → B) Λ A) → 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) ∧ (B →A)

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

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

- **¬ (¬A) ≡ A** (double-negation elimination
- **¬ (A ∧ B) ≡ (¬A ∨ ¬B)** (De Morgan)
- **¬(A ∨ B) ≡ (¬A ∧ ¬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 Biconditional using rule.

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

2) Remove implication 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 :- 1) 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 united.

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

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) Theefoee 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.

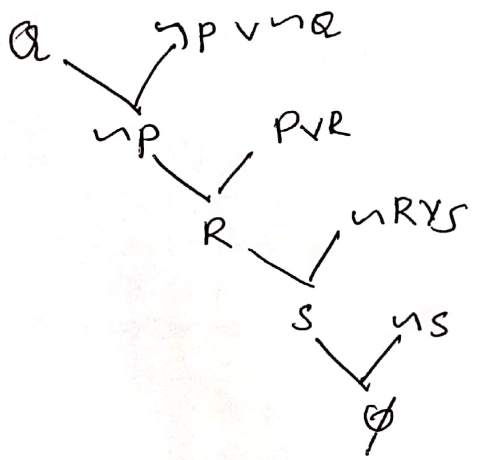
4) $\neg Q \vee S$

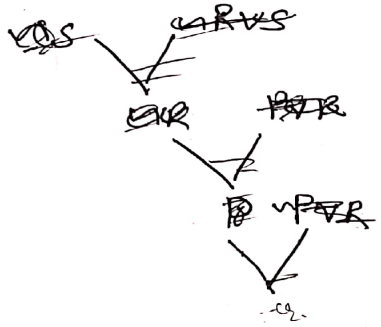
Negate $\neg(\neg Q \vee S)$

$= Q \wedge \neg S$

$= Q$

$\neg S$





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)

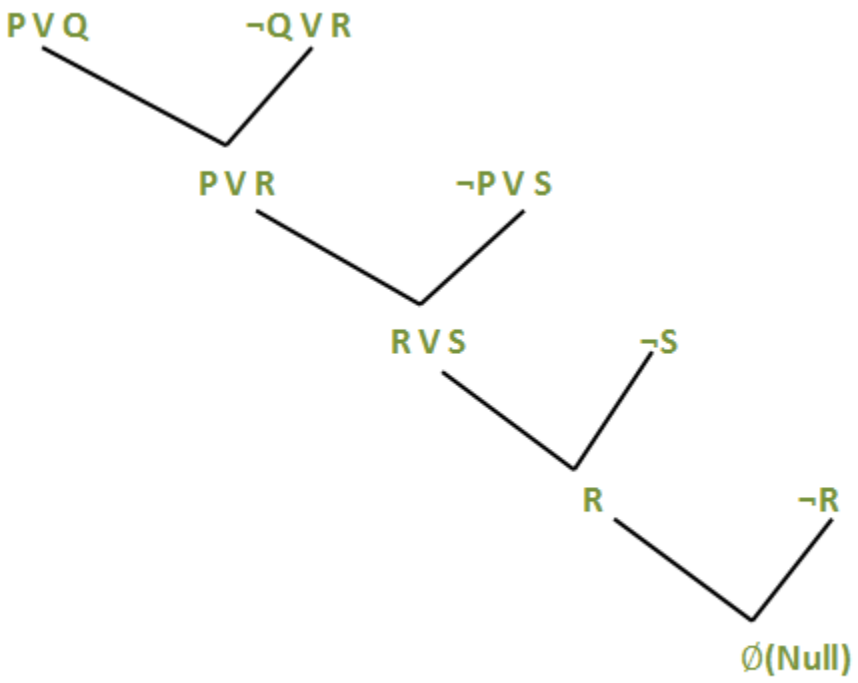
Negation of Goal (¬R): It will not rain.

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 (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.

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.