

UNIT-4: Adversarial Search

Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the solution which is often expressed in the form of a sequence of actions.
- But, there might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.**
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.

Types of Games in AI:

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

- **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. Such games are also called as stochastic games.
Example: Backgammon, Monopoly, Poker, etc.

Zero-Sum Game

- Zero-sum games are adversarial search which involves pure competition.
- In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.

- One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as ply.
- Chess and tic-tac-toe are examples of a Zero-sum game.

Zero-sum game: Embedded thinking

The Zero-sum game involved embedded thinking in which one agent or player is trying to figure out:

- What to do.
- How to decide the move
- Needs to think about his opponent as well
- The opponent also thinks what to do

Each of the players is trying to find out the response of his opponent to their actions. This requires embedded thinking or backward reasoning to solve the game problems in AI.

Formalization /Elements of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p . It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, $\frac{1}{2}$. And for tic-tac-toe, utility values are +1, -1, and 0.

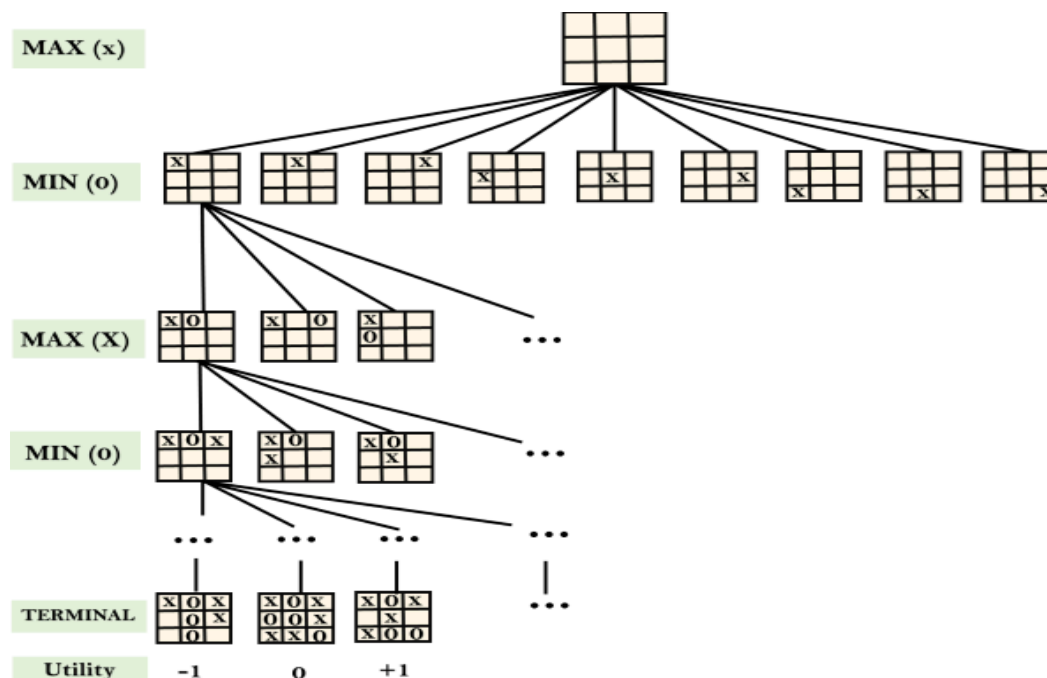
Game tree:

A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, actions function, and result Function.

Example: Tic-Tac-Toe game tree:

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree
- MIN minimizes the result.



Example Explanation:

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

Hence adversarial Search for the minimax procedure works as follows:

- It aims to find the optimal strategy for MAX to win the game.
- It follows the approach of Depth-first search.
- In the game tree, optimal leaf node could appear at any depth of the tree.
- Propagate the minimax values up to the tree until the terminal node discovered.

In a given game tree, the optimal strategy can be determined from the minimax value of each node, which can be written as MINIMAX(n). MAX prefer to move to a state of maximum value and MIN prefer to move to a state of minimum value then:

$$\text{For a state } S \text{ MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{If } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{If } \text{PLAYER}(s) = \text{MIN}. \end{cases}$$

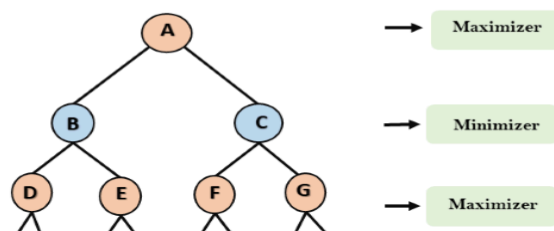
Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

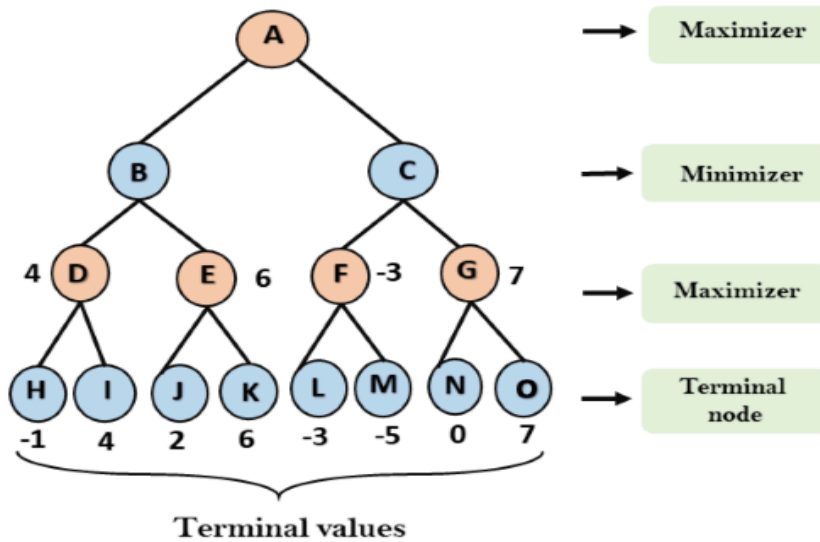
- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value = - infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



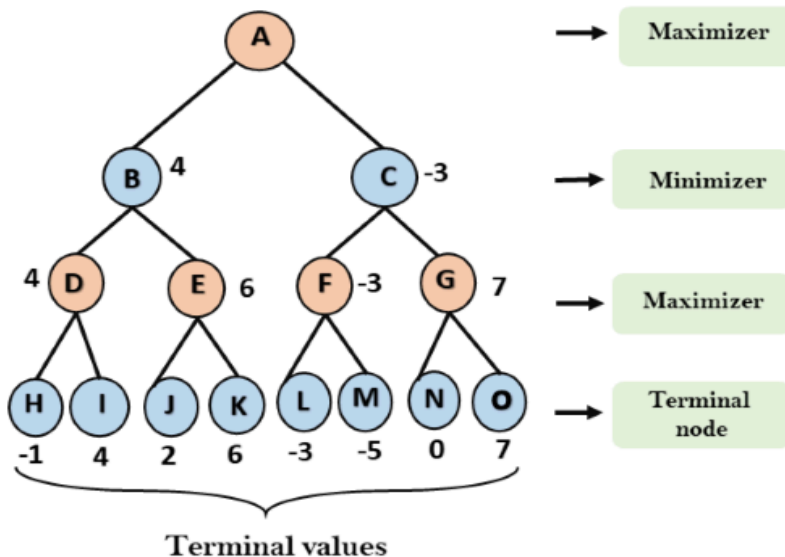
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

-
- For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
 - For Node E $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
 - For Node F $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
 - For node G $\max(0, -\infty) = \max(0, 7) = 7$
-



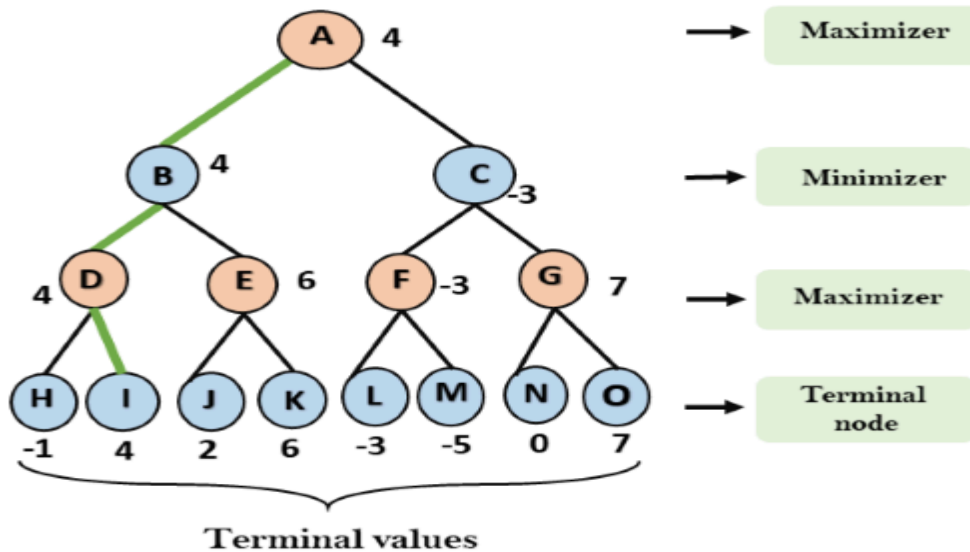
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.

- For node B = $\min(4, 6) = 4$
- For node C = $\min(-3, 7) = -3$



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A $\max(4, -3) = 4$



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is $O(bm)$.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning**.

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
 - a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$.
 - a. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.

The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is: $\alpha \geq \beta$

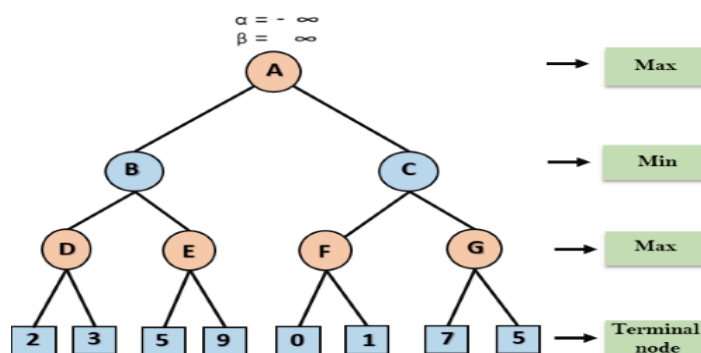
Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

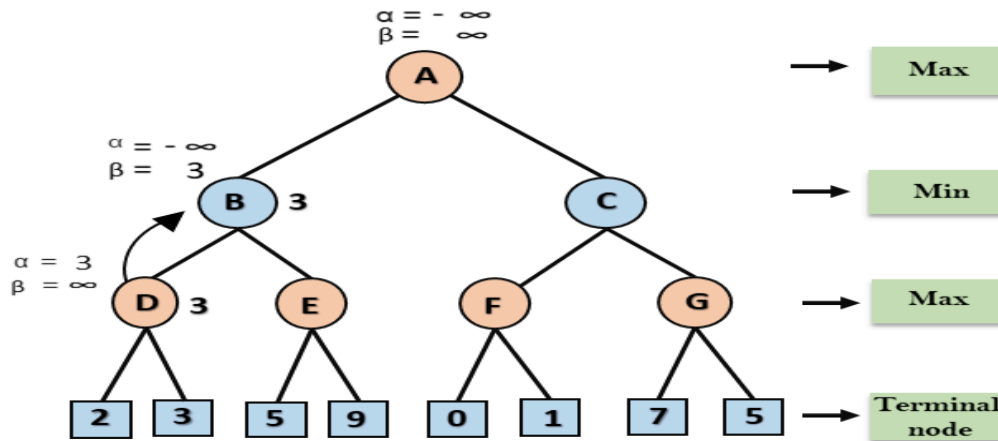
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where $\alpha = -\infty$ and $\beta = +\infty$, these value of alpha and beta passed down to node B where again $\alpha = -\infty$ and $\beta = +\infty$, and Node B passes the same value to its child D.



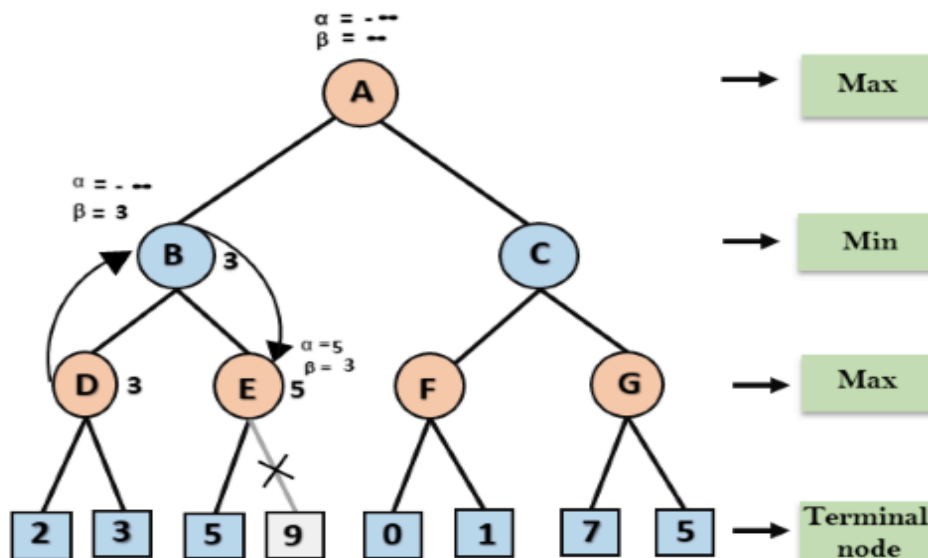
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max ($2, 3$) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now $\beta = +\infty$, will compare with the available subsequent nodes value, i.e. min ($\infty, 3$) = 3, hence at node B now $\alpha = -\infty$, and $\beta = 3$.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of $\alpha = -\infty$, and $\beta = 3$ will also be passed.

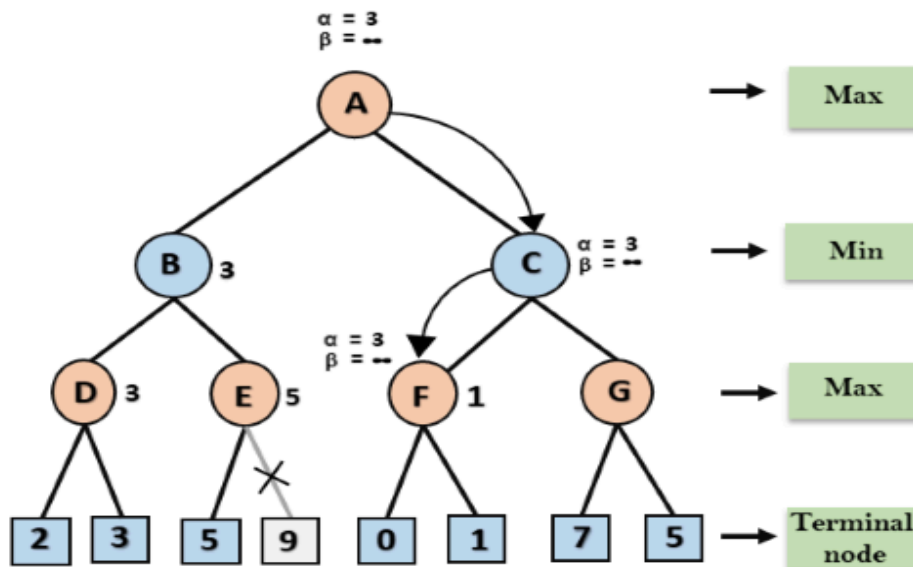
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max ($-\infty, 5$) = 5, hence at node E $\alpha = 5$ and $\beta = 3$, where $\alpha \geq \beta$, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



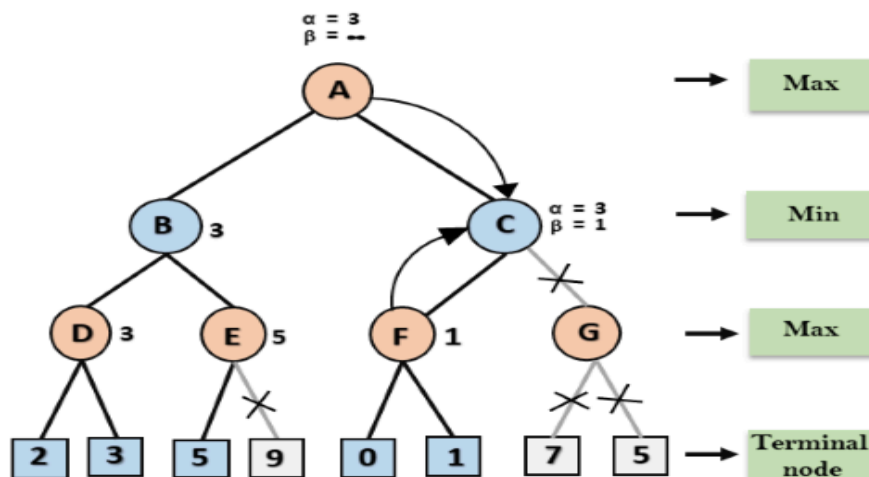
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as $\max(-\infty, 3) = 3$, and $\beta = +\infty$, these two values now passes to right successor of A which is Node C.

At node C, $\alpha=3$ and $\beta = +\infty$, and the same values will be passed on to node F.

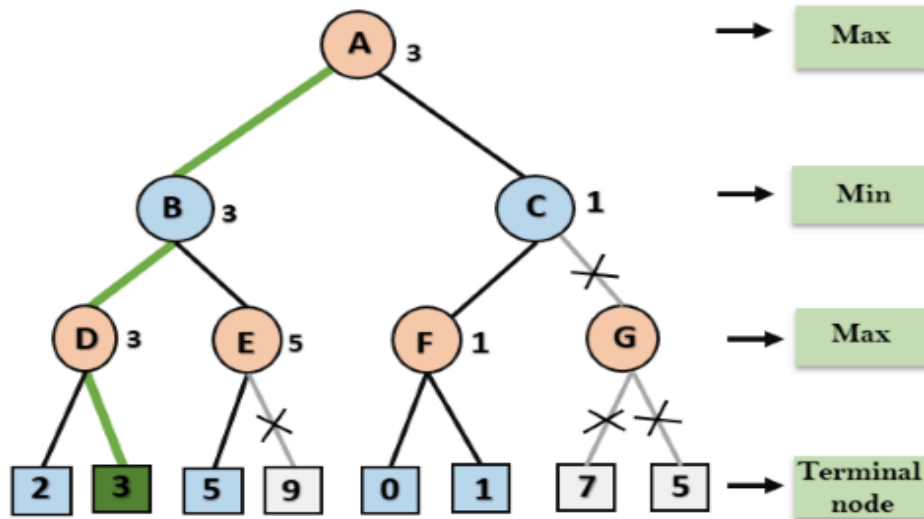
Step 6: At node F, again the value of α will be compared with left child which is 0, and $\max(3,0) = 3$, and then compared with right child which is 1, and $\max(3,1) = 3$ still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C $\alpha=3$ and $\beta = +\infty$, here the value of beta will be changed, it will compare with 1 so $\min(\infty, 1) = 1$. Now at C, $\alpha=3$ and $\beta=1$, and again it satisfies the condition $\alpha \geq \beta$, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is $\max(3, 1) = 3$. Following is the final game tree which is showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

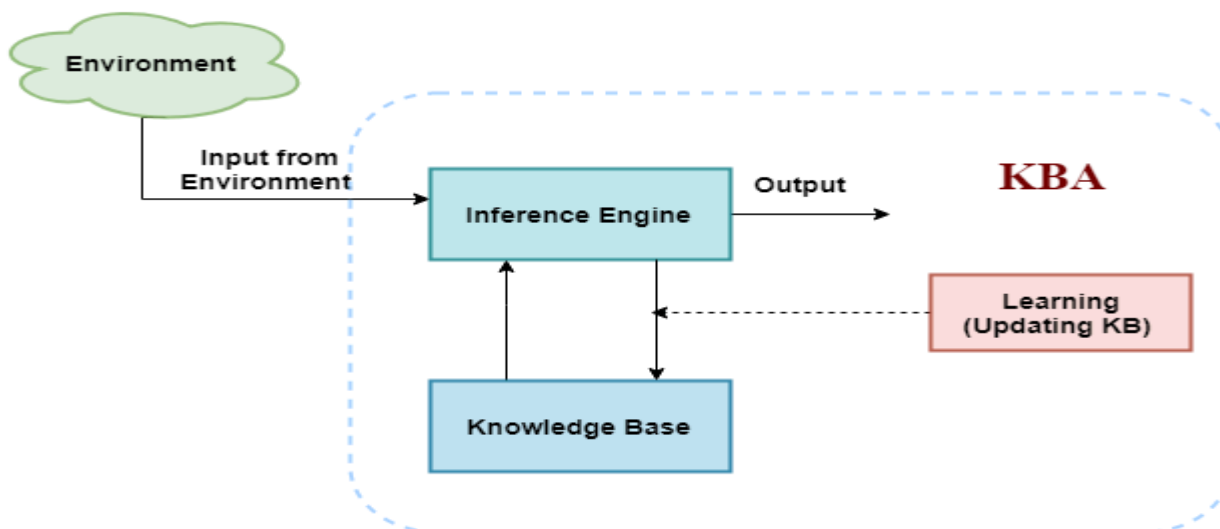
Knowledge-Based Agent in Artificial intelligence

- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.**
- Knowledge-based agents are composed of two main parts:
 - **Knowledge-base and**
 - **Inference system.**

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

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. The Knowledge-base of KBA stores fact about the world.

Why use a knowledge base?

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

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

- **Forward chaining**
- **Backward chaining**

Operations Performed by KBA

Following are three operations which are performed by KBA in order to show the intelligent behavior:

1. **TELL:** This operation tells the knowledge base what it perceives from the environment.
2. **ASK:** This operation asks the knowledge base what action it should perform.
3. **Perform:** It performs the selected action.

Each time when the function is called, it performs its three operations:

- Firstly it TELLS the KB what it perceives.
- Secondly, it asks KB what action it should take
- Third agent program TELLS the KB that which action was chosen.

Various levels of knowledge-based agent:

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

1. Knowledge level

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

2. Logical level:

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

3. Implementation level:

This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Types of knowledge:

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Following are the various types of knowledge:



1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a field or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

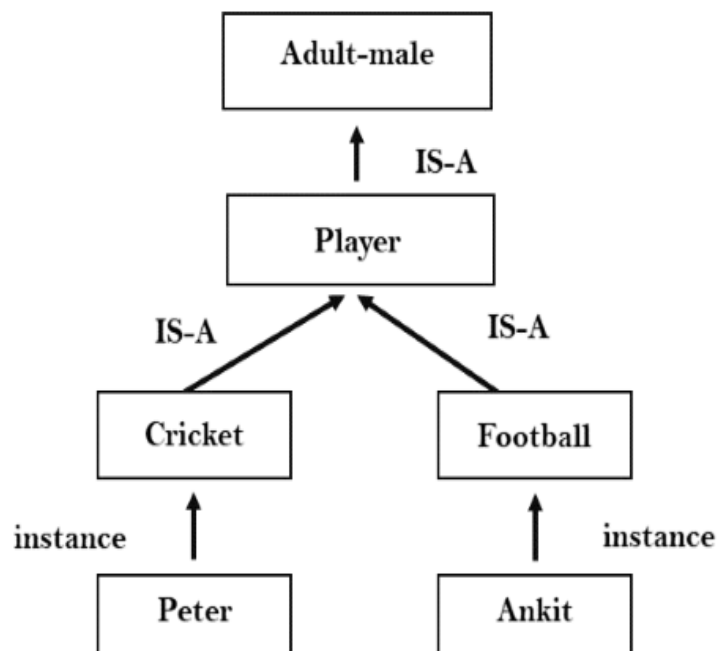
	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

Example: The following is the simple relational knowledge representation.

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.

Example:



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:

a. Marcus is a man

b. All men are mortal

Then it can represent as;

man(Marcus)

$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Issues in knowledge representation:

The main objective of knowledge representation is to draw the conclusions from the knowledge, but there are many issues associated with the use of knowledge representation techniques.

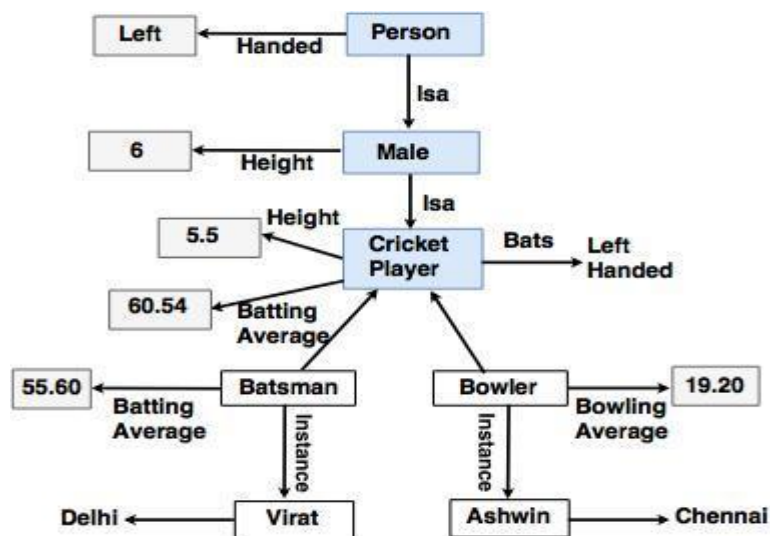


Fig: Inheratable Knowledge Representation

Some of them are listed below:

Refer the above diagram to the following issues.

1.Important Attributes:

Any attribute of objects so basic that they occur in almost every problem domain?

There are two attributed “instance” and “is-a”, that are general significance. These attributes are important because they support property inheritance.

2.Relationship among attributes:

The attributes we use to describe objects are themselves entities that we represent.

The relationship between the attributes of an object, independent of specific knowledge they encode, may hold properties like:

1. **Inverse** — This is about consistency check, while a value is added to one attribute. The entities are related to each other in many different ways.
2. **Existence in an isa hierarchy** — This is about generalization-specification, like, classes of objects and specialized subsets of those classes, there are attributes and specialization of attributes. For example, the attribute height is a specialization of general attribute physical-size which is, in turn, a specialization of physical-attribute. These generalization-specialization relationships are important for attributes because they support inheritance.
3. **Technique for reasoning about values** — This is about reasoning values of attributes not given explicitly. Several kinds of information are used in reasoning, like, height: must be in a unit of length, Age: of a person cannot be greater than the age of person’s parents. The values are often specified when a knowledge base is created.
4. **Single valued attributes** — This is about a specific attribute that is guaranteed to take a unique value. For example, a baseball player can at time have only a single height and be a member of only one team. KR systems take different approaches to provide support for single valued attributes.

3.Choosing Granularity:

At what level of detail should the knowledge be represented?

It is necessary to know:

- At what level should the knowledge be represented and what are the primitives?
- Should there be a small number or should there be a large number of low-level primitives or High-level facts.
- High-level facts may not be adequate for inference while Low-level primitives may require a lot of storage.

Example of Granularity:

- Suppose we are interested in following facts:

John spotted Sue.

This could be represented as

Spotted (agent(John),object (Sue))

Such a representation would make it easy to answer questions such are:

- Who spotted Sue?

Suppose we want to know:

- Did John see Sue?

Given only one fact, we cannot discover that answer.

We can add other facts, such as

Spotted(x, y) -> saw(x, y)

We can now infer the answer to the question.

4.Set of objects:

How should sets of objects be represented?

There are certain properties of objects that are true as member of a set but not as individual;

Example: Consider the assertion made in the sentences:

“there are more sheep than people in Australia”, and

“English speakers can be found all over the world.”

To describe these facts, the only way is to attach assertion to the sets representing people, sheep, and English.

This is done,

- in logical representation through the use of universal quantifier, and
- in hierarchical structure where node represent sets and inheritance propagate set level assertion down to individual.

5. Finding Right structure:

Given a large amount of knowledge stored in a database, how can relevant parts are accessed when they are needed?

This is about access to right structure for describing a particular situation.

This requires, selecting an initial structure and then revising the choice.

While doing so, it is necessary to solve following problems:

- How to perform an initial selection of the most appropriate structure.
- How to fill in appropriate details from the current situations.
- How to find a better structure if the one chosen initially turns out not to be appropriate.
- What to do if none of the available structures is appropriate.
- When to create and remember a new structure.

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

a) Forward chaining

b) Backward chaining

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: $(\neg p \vee \neg q \vee k)$. It has only one positive literal k.

It is equivalent to $p \wedge q \rightarrow k$.

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.
- Consider the following famous example which we will use in both approaches:

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
American(p) $\wedge$ weapon(q) $\wedge$ sells(p, q, r) $\wedge$ hostile(r) $\rightarrow$ Criminal(p) ... (1)
- Country A has some missiles. **?p Owns(A, p) $\wedge$ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
Owns(A, T1) (2)
Missile(T1) (3)
- All of the missiles were sold to country A by Robert.
?p Missiles(p) $\wedge$ Owns(A, p) $\rightarrow$ Sells(Robert, p, A) (4)
- Missiles are weapons.
Missile(p) $\rightarrow$ Weapons(p) (5)
- Enemy of America is known as hostile.
Enemy(p, America) $\rightarrow$ Hostile(p) (6)
- Country A is an enemy of America.
Enemy(A, America) (7)
- Robert is American
American(Robert). (8)

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1)**. All these facts will be represented as below.

American (Robert)	Missile (T1)	Owns (A,T1)	Enemy (A, America)
-------------------	--------------	-------------	--------------------

Step-2:

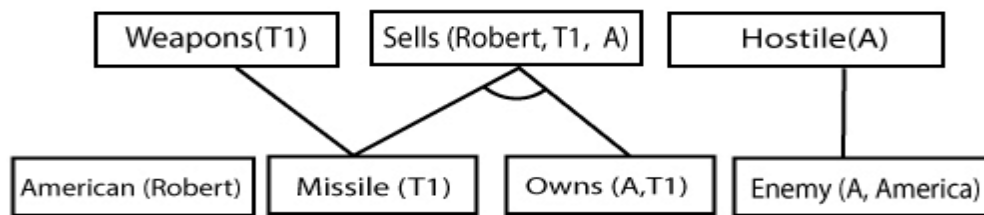
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

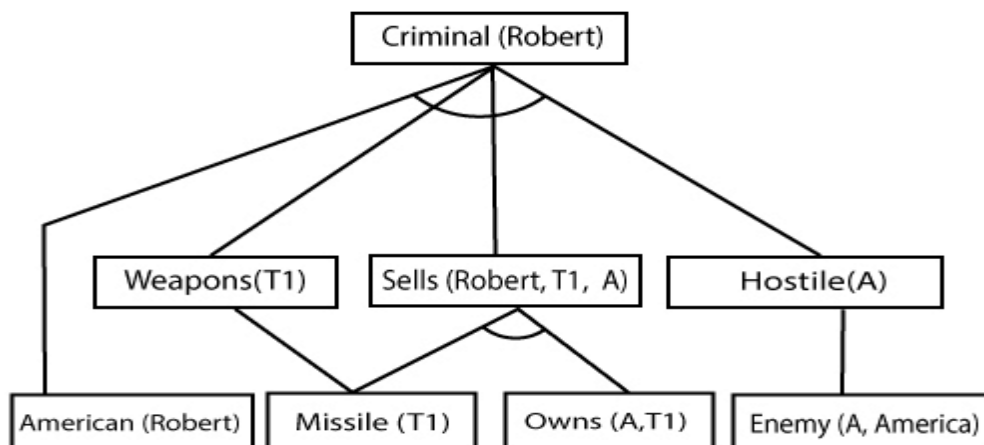
Rule-(4) satisfy with the substitution $\{p/T1\}$, so **Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution (p/A) , so Hostile(A) is added and which infers from Rule-(7).



Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution $\{p/Robert, q/T1, r/A\}$, so we can add **Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.

- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

- **American (p) $\wedge$ weapon(q) $\wedge$ sells (p, q, r) $\wedge$ hostile(r) $\rightarrow$ Criminal(p) ... (1)**
- **Owns(A, T1) ... (2)**
- **Missile(T1) ... (3)**
- **?p Missiles(p) $\wedge$ Owns (A, p) $\rightarrow$ Sells (Robert, p, A) ... (4)**
- **Missile(p) $\rightarrow$ Weapons (p) ... (5)**
- **Enemy(p, America) $\rightarrow$ Hostile(p) ... (6)**
- **Enemy (A, America) ... (7)**
- **American(Robert). ... (8)**

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

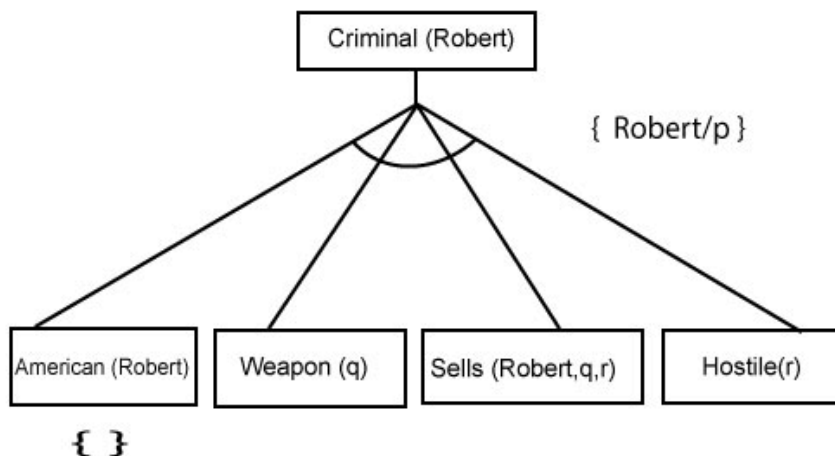
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



Step-2:

At the second step, we will infer other facts from goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/p}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

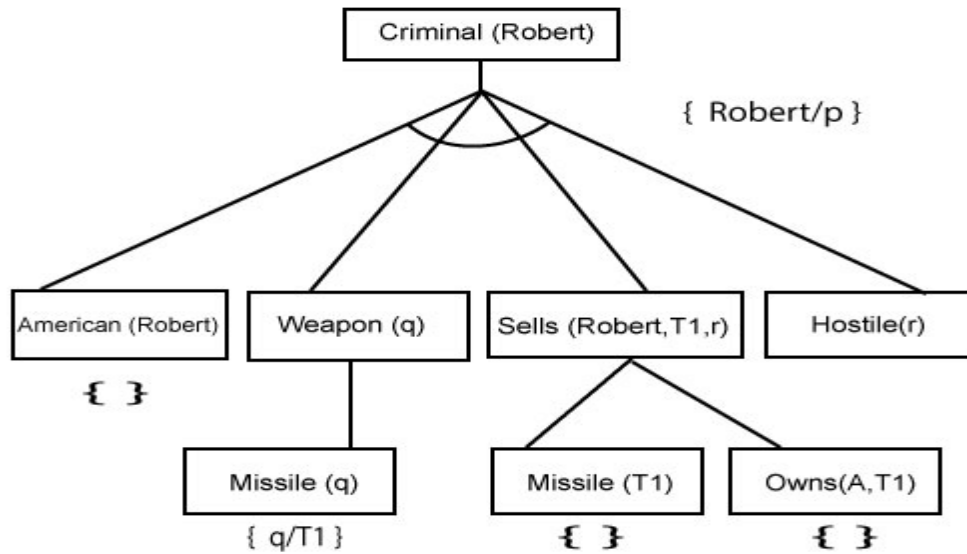
Here we can see **American (Robert)** is a fact, so it is proved here.



Step-3: At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.

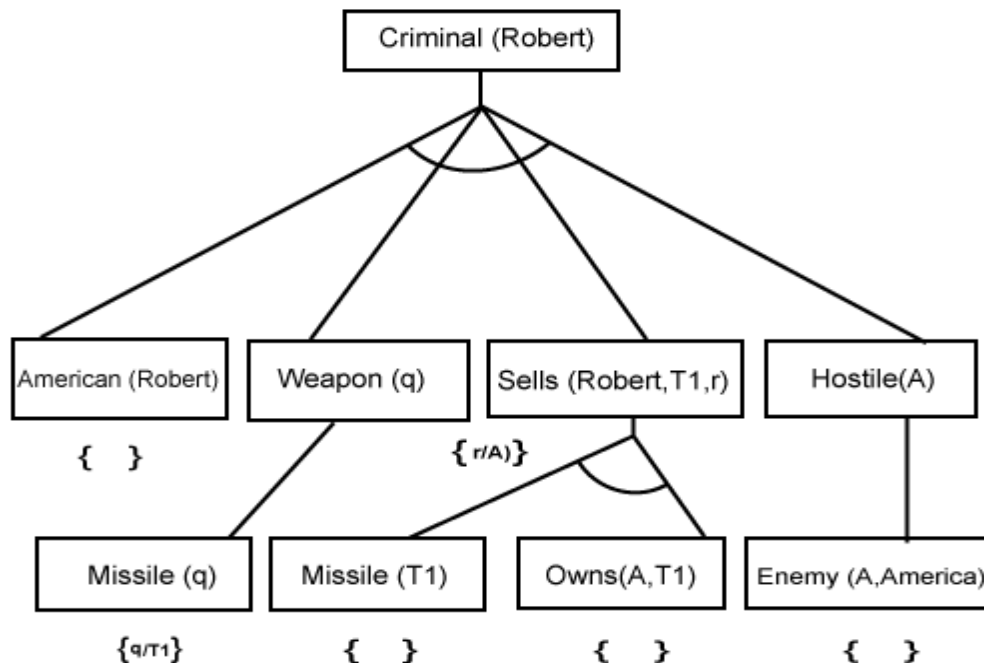
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) from Sells(Robert, T1, r) which satisfies the **Rule-4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Difference between backward chaining and forward chaining

Following is the difference between the forward chaining and backward chaining:

S. No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data until it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

