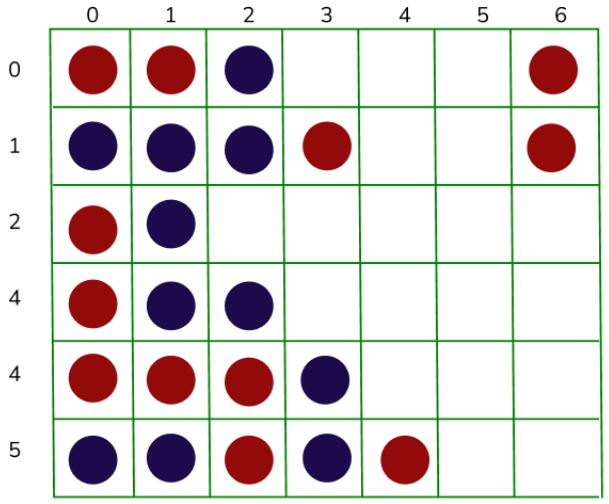
# ARTIFICIAL INTELLIGENCE(CSEN2031 )

## UNIT-III

## Adversarial Search and Logical Agents




By

**Dr. Satyabrata Dash**
Assistant Professor
Department of Computer Science and Engineering, GITAM Deemed to be University

# Vision & Mission of the Department

- **Vision statement**

To be an exceptional Centre of Excellence in Computing, cultivating highly skilled computer engineers with cutting-edge knowledge and research expertise. By advancing a culture of honesty and compassion, we aim to innovate and meet the evolving demands of society and industry, making a significant difference in the world.

- **Mission Statements**

**01** To be an application-oriented education ecosystem immersed in holistic development and to drive impactful, integrated research programs in computer science engineering

**02** Offer a flexible curriculum with high-quality teaching methods that empower students with problem-solving skills, elevate their career opportunities to prepare them for higher studies and lifelong learning to nurture valuable futures with global perspectives.

**03** Aim to address real-world challenges through applied research using emerging technologies and a strong sense of social responsibility.

**04** Committed to fostering ethical values, professional behavior, and innovative research through internships, research projects, and mentorship programs.

GITAM
DEEMED TO BE UNIVERSITY

# ARTIFICIAL INTELLIGENCE(CSEN2031 )

## Course Description:

This course enables the students to think critically about what makes humans intelligent, and how computer scientists are designing computers to act more like us. Artificial Intelligence (AI) is the study of how to make computers make things that can ‘think’ and act the right way, given the circumstances. AI plays an important role in the design and development of systems with intelligent behaviour. The primary objective of this course is to provide an introduction to the basic principles and applications of Artificial Intelligence.

## Course Educational Objectives:

- To understand the fundamentals of Artificial Intelligence
- To solve problems by using search algorithms
- To gain insight into competitive environments using adversarial search algorithms
- To learn knowledge representation and knowledge representation techniques
- To address the uncertainty and to learn the ways of learning

GITAM
DEEMED TO BE UNIVERSITY

# ARTIFICIAL INTELLIGENCE(CSEN2031 )

- **Textbooks:**

1. Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach,3 rd Edition, Pearson, 2015

2. David L. Poole and Alan K. Mackworth, Artificial Intelligence: Foundations of Computational Agents, 2nd Edition, Cambridge University Press, 2017

- **References:**

1. George F Lugar, Artificial Intelligence, Pearson, 6th edition(2017)
2. Elaine Rich, Kevin Knight and Shivashankar B. Nair, Artificial Intelligence, TMH Education Pvt. Ltd., 2008.
3. Dan W. Patterson, Introduction to Artificial Intelligence and Expert Systems, Pearson.
4. David Poole Book Reference site: https://artint.info/2e/html/ArtInt2e.html
5. Russel lecture reference site: https://inst.eecs.berkeley.edu/~cs188/sp19/
6. Microsoft AI project s Site: https://www.microsoft.com/en-us/ai/ai-lab-projects
7. https://nptel.ac.in/courses/106/105/106105079/ (Nptel course on artificial Intelligence by Prof. Dasgupta maps to the syllabus and beyond and can be used as listening material for the relevant topics)

# UNIT III: Adversarial Search and Logical Agents 9 hours,

## Adversarial Search:

- 2-Player Games
- Optimal Decisions in Games: AND-OR graph, Minimax algorithm, Alpha-Beta Pruning. Chance based games.

## Constraint satisfaction problem (CSP):

- Constraint Networks, Solving CSP by Search Logical Agents:
- Knowledge-based Agents,
- Propositional Logic,
- Propositional Theorem Proving: Inference.

GITAM
DEEMED TO BE UNIVERSITY

# 2-Player Games

1. Two-player games in artificial intelligence are a classic domain for studying algorithms and strategies.
2. These games involve two opposing players competing against each other, and they often serve as benchmarks for developing and testing AI techniques.

**Examples:**

| Classic Board Games | Strategy Games | Card-Based and Tile Games | AI-Specific and Experimental Games |
|---|---|---|---|
| • Chess | • Tic-Tac-Toe | • Magic: The Gathering (1v1 Mode) | • Mini-Max Tic-Tac-Toe Variants |
| • Go | • Dots and Boxes | • Dominoes | • Iterated Prisoner's Dilemma |
| • Backgammon | • Hex | • Gin Rummy (Head-to-Head Mode) | • Alpha-Beta Sudoku Challenges |
| • Connect Four | • Shogi (Japanese Chess) | | |

# AI Techniques for Two-Player Games

**1. Search Algorithms**:

- **Minimax**: Fundamental algorithm for perfect information games.
- **Alpha-Beta Pruning**: Reduces the number of nodes evaluated in the search tree.

**2. Reinforcement Learning**:

- Used for games with vast state spaces and less clear rules, allowing AI to learn strategies through trial and error.

**3. Monte Carlo Tree Search (MCTS)**:

- Effective for games like Go, which have a large branching factor.

**4. Neural Networks**:

- Enhance game evaluation and decision-making in complex games like Chess and Go (*AlphaZero*).

**5. Evolutionary Algorithms**:

- Optimize strategies over generations of simulated play.

GITAM
DEEMED TO BE UNIVERSITY

# Key Concepts for Game Playing

- **Perfect Information Games**: Both players have complete knowledge of the game state (**Chess, Go**).

AI employs techniques like the *Minimax Algorithm* with enhancements like *Alpha-Beta Pruning* for efficiency.

- **Imperfect Information Games**: Players have incomplete knowledge of the game state (**Poker, Stratego**).AI uses techniques such as *Monte Carlo Tree Search (MCTS)*, *Bayesian Models*, or *Reinforcement Learning*.
- **Zero-Sum Games**: A win for one player is a loss for the other. Most two-player games fall under this category (**Tic-Tac-Toe, Connect 4**).
- **Game Trees**: Represent possible game states and moves. Search algorithms explore this tree to determine optimal moves.
- **Evaluation Functions**: Estimate the desirability of a game state, helping AI decide on moves when exhaustive search is impractical.

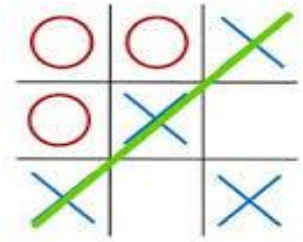
## Key Concepts for Game Playing( Terminology)

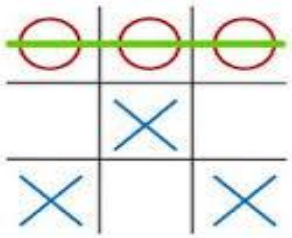
1. **Players:** We call them Max and Min.
2. **Initial State:** Includes board position and whose turn it is.
3. **Operators:** These correspond to legal moves.
4. **Terminal Test:** A test applied to a board position which determines whether the game is over. In chess, for example, this would be a checkmate or stalemate situation.
5. **Utility Function**: A function which assigns a numeric value to a terminal state. For example, in chess the outcome is win (+1), lose (-1) or draw (0). Note that by convention, we always measure utility relative to Max.

# Introduction to Tic-Tac-Toe game playing Strategies

## Rules of the Game

1. The game is to be played between two people (in this program between HUMAN and COMPUTER).
2. One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
3. The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).
4. If no one wins, then the game is said to be draw.





## Tic Tac Toe Game playing strategies

- Two players

  **1. HUMAN**

  **2. COMPUTER.**

- **The objective is to write a computer program in such a way that Computer wins most of the time. !!!!!**

# Introduction to Tic-Tac-Toe game playing Strategies

Three approaches are presented to play this game which increase in

1. Complexity
2. Use of generalization
3. Clarity of their knowledge
4. Extensibility of their approach

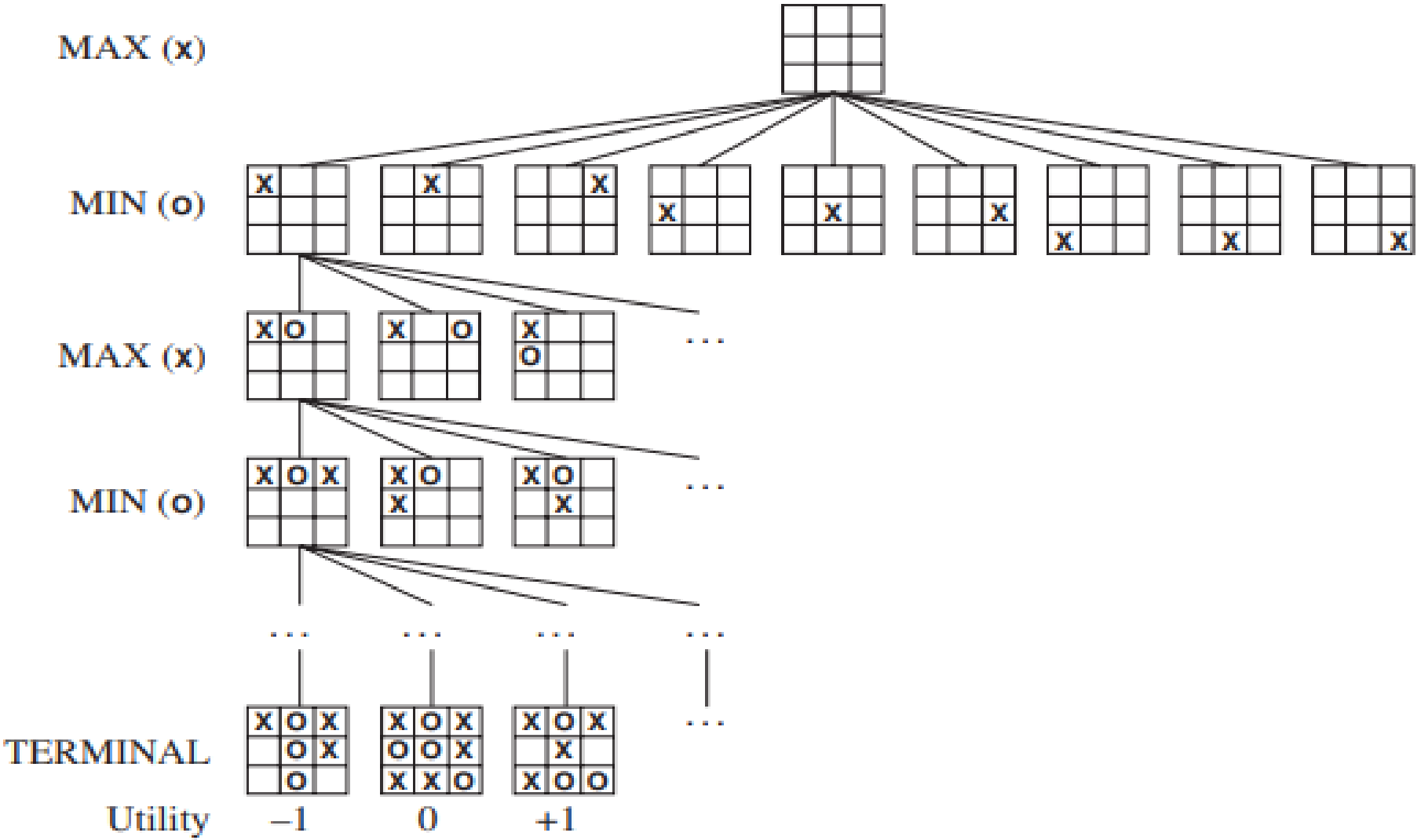
•These approaches will move towards being representations of what we will call AI techniques and can br used in any Board game playing strategy

**Tic Tac Toe Board/ Noughts and crosses/ Xs and Os**

It is two players, *X* and *O*, game who take turns marking the spaces in a 3×3 grid.

The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

# Introduction to Tic-Tac-Toe game playing Strategies



# Tic-Tac-Toe game playing :Approach 1

1. Consider a Board having nine elements vector(3×3grid) the number of grids are starting from 1 to 9.

**(Board Position)**

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

2.Each element will contain

- 0 for blank
- 1 indicating X player move
- 2 indicating O player move

3.Computer may play as X or O player.

**4.First player who so ever is always plays X.**

# Move Table MT

1. MT is a vector of $3^9$ elements, each element of which is a nine element vector representing board position.

2. Total of $3^9$ **(19683)** elements in MT

| Index | Current Board position | New Board position |
|---|---|---|
| 0 | 000000000 | 000010000 |
| 1 | 000000001 | 020000001 |
| 2 | 000000002 | 000100002 |
| 3 | 000000010 | 002000010 |
| : | | |
| : | | |

# Algorithm: Approach-1

| To make a move in the game, following steps are required |
|---|
| 1. View the vector (board) as a ternary number and convert it to itscorresponding decimal number. |
| 2. Use the computed number as an index into the MT and access thevector stored there.<br><br>3. The selected vector represents the way the board will lookafter the move. |
| 4. Set board equal to that vector. |

# Optimal Decisions in Games: Problem Reduction and Game Playing

**Need and Objective:**

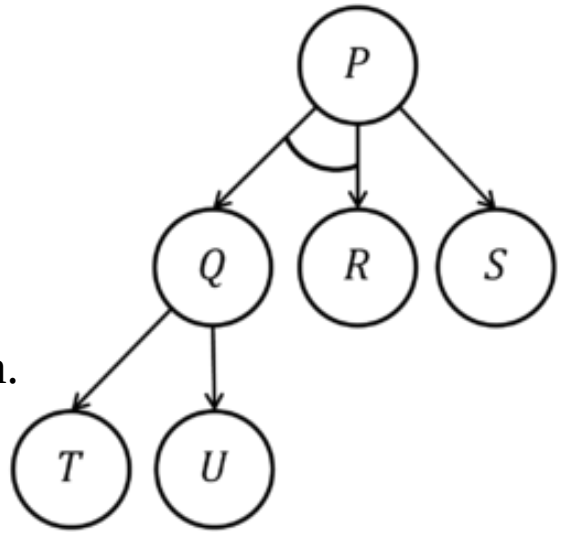
1. Sometimes problems only seem **hard to solve**.
2. A hard problem may be one that can be **reduced** to a number of simple problems...and,
3. when each of the **simple problems is solved, then the hard problem has been solved**.
4. This is the basic Concept behind the method of **problem reduction**.

- We already know about the **divide and conquer strategy**, a solution to a problem can be obtained by decomposing it into smaller sub-problems.
- Each of this sub-problem can then be solved to get its sub solution.
- These sub solutions can then recombined to get a solution as a whole.
- That is called is **Problem Reduction**.
- **This method generates arc which is called as AND arcs**.
- One AND arc may point to any number of successor nodes, all of which must be solved in order for an arc to point to a solution.

GITAM
DEEMED TO BE UNIVERSITY

# AND-OR graph

**Problem Reduction**

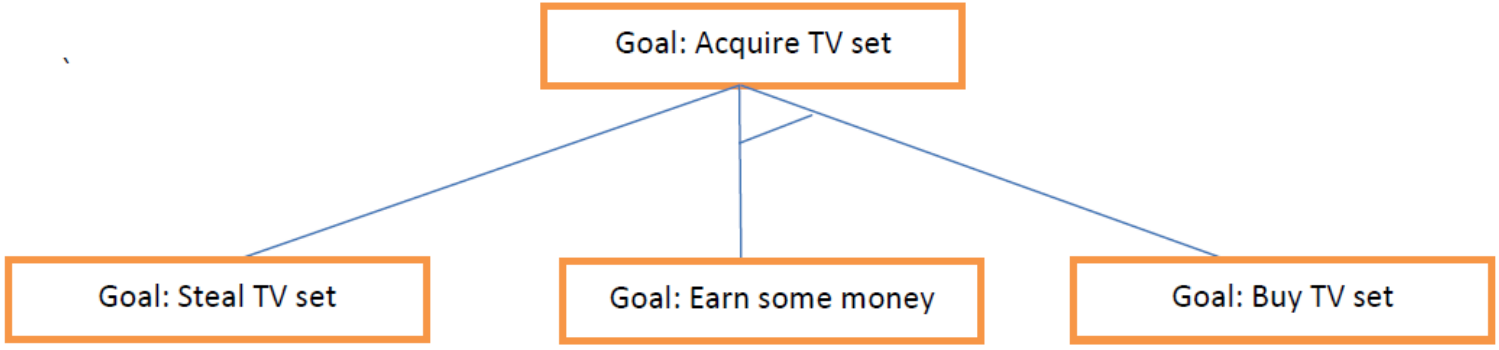



1. So far **search strategies** discussed were for **OR graphs**.
   - Here several arcs indicate a different ways of solving problem.
2. Another kind of structure is **AND-OR graph (tree).**
3. Useful for representing the solution of problem by decomposing it into smaller sub-problems.
4. Each sub-problem is solved and final solution is obtained by combining solutions of each sub-problem.
5. **Decomposition generates arcs that we will call AND arc.**
6. **One AND arc may point to any number of successors, all of which must be solved.**
7. **Such structure is called AND–OR graph rather than simply AND graph.**

GITAM
DEEMED TO BE UNIVERSITY

# AND-OR graph

An AND-OR graph is a type of directed graph used in problem-solving and decision-making scenarios. It is particularly useful in representing hierarchical relationships, dependencies, or logical structures. This graph extends the traditional graph model by introducing two types of nodes: **AND nodes** and **OR nodes**, which represent different logical operations.




A SIMPLE AND-OR GRAPH

# AND-OR graph

## Structure of an AND-OR Graph

### Nodes:

**AND nodes**: These require all child nodes to be satisfied (or true) for the parent node to be satisfied.

**OR nodes**: These require at least one child node to be satisfied for the parent node to be satisfied.

### Edges:

Directed edges connect nodes and represent dependencies or relationships.

The direction of edges indicates how information or conditions flow in the graph.

### Representation

An AND-OR graph is typically used to represent solutions to problems that can be broken into sub-problems, some of which require all conditions to be met (AND) and others that need only one condition (OR).

# AND-OR graph(Algorithm)

**Algorithm**

**Input**: An AND-OR graph where each node represents a state and edges define dependencies or choices.

**Initialization**: Start from an initial node or state.

**Search Process**:

**Expand Nodes**: Select a node to expand based on a search strategy (e.g., depth-first, breadth-first).

**Check Dependencies**:

- If it's an AND node, all child nodes are expanded.
- If it's an OR node, choose one child node to expand.

**Check Goal**: Verify if a goal state is reached.

**Path Construction**:

- If a solution is found, trace the path back to the root.
- If no solution exists for the current path, backtrack and try alternatives (for OR nodes) or mark failure (for AND nodes).

**Output**: A solution path (if one exists) or a determination that no solution exists.

# Minimax algorithm

1. The **minimax algorithm** is a decision-making algorithm widely used in two-player, turn-based games such as chess, checkers, or tic-tac-toe.
2. It is a **recursive or backtracking algorithm** which is used in decision-making and game theory.
3. It provides an optimal move for the player assuming that opponent is also playing optimally.
4. In this algorithm two players play the game, one is called **MAX** and other is called **MIN.**
5. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
6. Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
7. The minimax algorithm performs a **depth-first search algorithm** for the exploration of the complete game tree.
8. The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

GITAM
DEEMED TO BE UNIVERSITY

# Minimax algorithm

## Key Concepts

**Game Tree**:

The algorithm explores a tree structure where nodes represent game states, and edges represent possible moves.

**Root:** Current state of the game.

**Leaves:** Final states (win, lose, or draw).

Players:

**Maximizer:** The player attempting to maximize their score.

**Minimizer:** The opponent trying to minimize the Maximizer's score.

**Utility Function:**

Assigns a numeric value to each terminal state. Examples:

**Win: +1**

**Loss: -1**

**Draw: 0**

These values guide the algorithm in decision-making.

# Minimax algorithm

**Algorithm**

If the current node is a terminal state (win, lose, or draw), return the utility value.

- If it's the **Maximizer's turn**:

  Initialize the best value as -∞.

  For each possible move:

    Compute the minimax value of the child node.

    Update the best value with the maximum of the current best value and the child's value.

  Return the best value.

- If it's the **Minimizer's turn**:

  Initialize the best value as +∞.

  For each possible move:

    Compute the minimax value of the child node.

    Update the best value with the minimum of the current best value and the child's value.

  Return the best value.

# Minimax algorithm

**How It Works**

**Recursion**:

The algorithm explores all possible moves recursively down the tree.

Alternates between maximizing and minimizing at each level of the tree.

**Backtracking**:

Starting from the leaves, the algorithm propagates the optimal scores back up the tree.

**Decision**:

The root node (current state) chooses the move with the best score based on the minimax values.

# Alpha-Beta pruning

1. The word **‘pruning’** means cutting down branches and leaves
2. Alpha-beta pruning is nothing but the **pruning of useless branches in decision trees**.
3. This alpha-beta pruning algorithm was discovered independently by researchers in the 1900s.
4. Alpha-beta pruning is a modified version of the **minimax algorithm**.
5. It is an **optimization technique** for the **minimax algorithm**.
6. It used primarily in two-player games like chess, checkers, or tic-tac-toe.
7. It significantly reduces the number of nodes evaluated in the game tree, improving the efficiency of decision-making without affecting the correctness of the result.

GITAM
DEEMED TO BE UNIVERSITY

# Alpha-Beta pruning

## Terminologies

1. **Minimax Algorithm**: Determines the best move for a player by simulating all possible moves and their outcomes, assuming both players play optimally.
2. **Alpha (α)**: The best value that the maximizing player (Max) can guarantee. Initially set to -∞.
3. **Beta (β)**: The best value that the minimizing player (Min) can guarantee. Initially set to +∞.
4. **Pruning :**Alpha-Beta pruning skips the evaluation of parts of the game tree that cannot affect the final decision. If a branch cannot improve the outcome for the current player, it is pruned (discarded).

GITAM
DEEMED TO BE UNIVERSITY

# Alpha-Beta pruning( Algorithm)

**Algorithm Steps**

**Initialize α and β**: Set alpha to -∞ and beta to +∞.

**Traverse the Tree**:

If the current player's turn is Max, try to maximize the value.

If it’s Min’s turn, try to minimize the value.

**Prune Branches**:

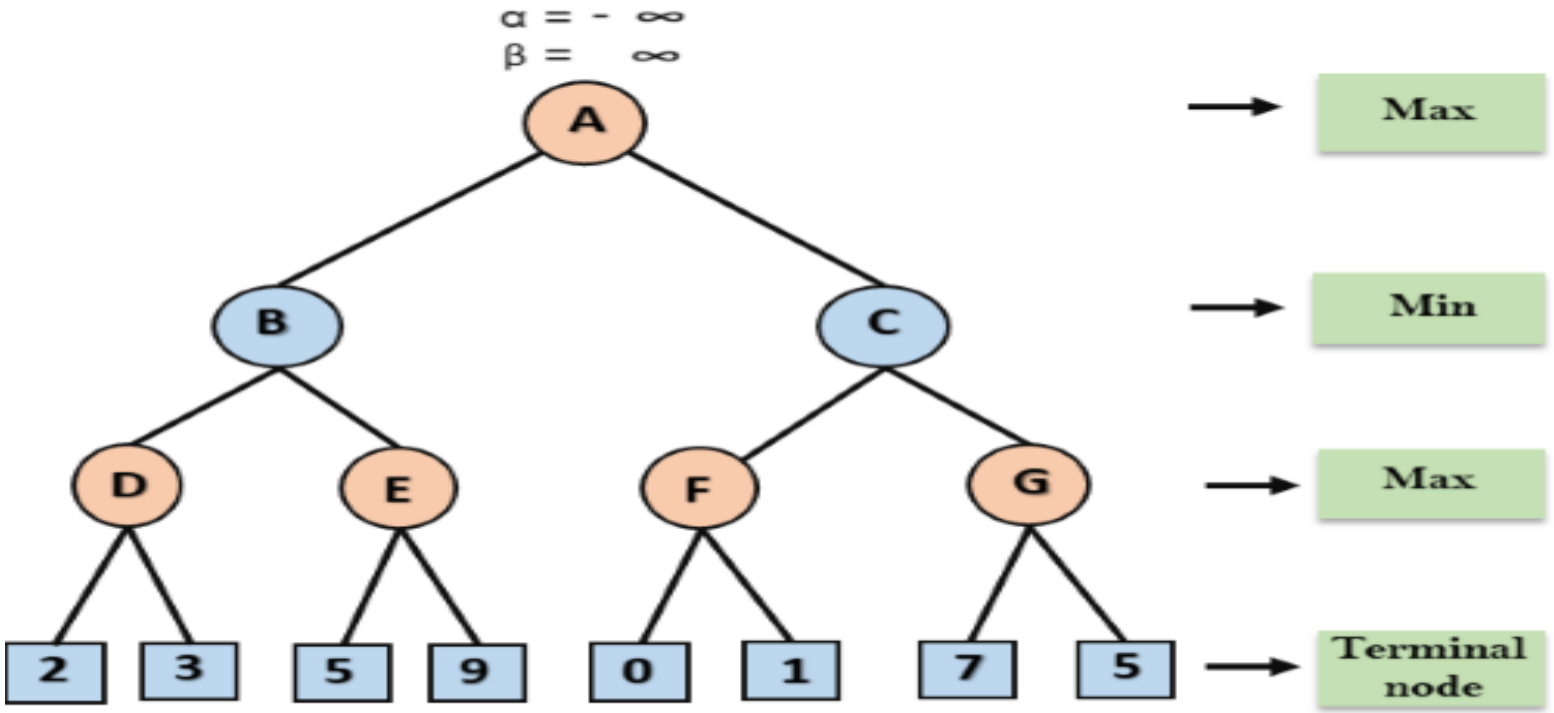
If the current node's value exceeds **β**, stop exploring further (Min will never allow this branch to be chosen).

If the current node's value is less than **α**, stop exploring further (Max will never allow this branch to be chosen).

# Alpha-Beta pruning ( Example)

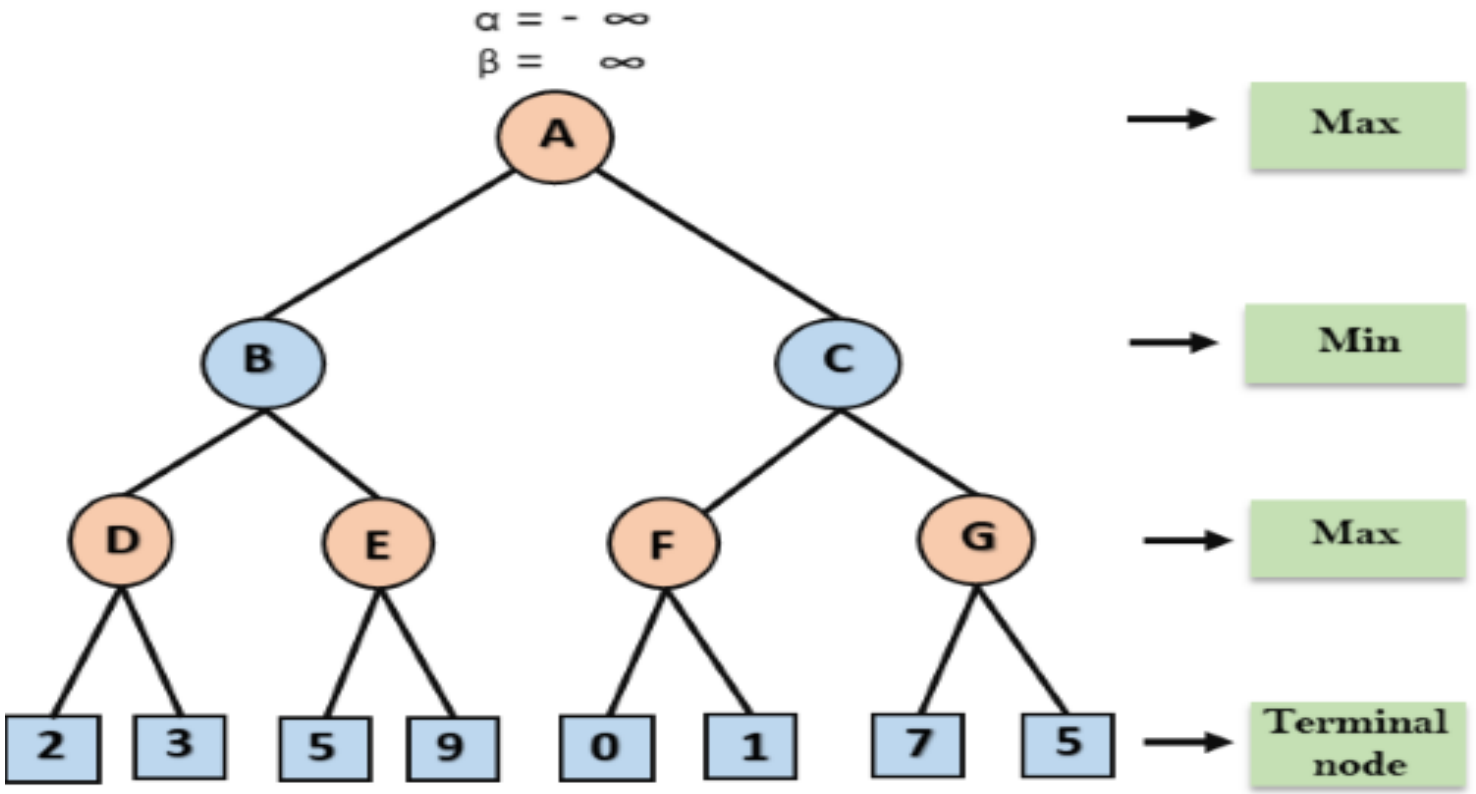
Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

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



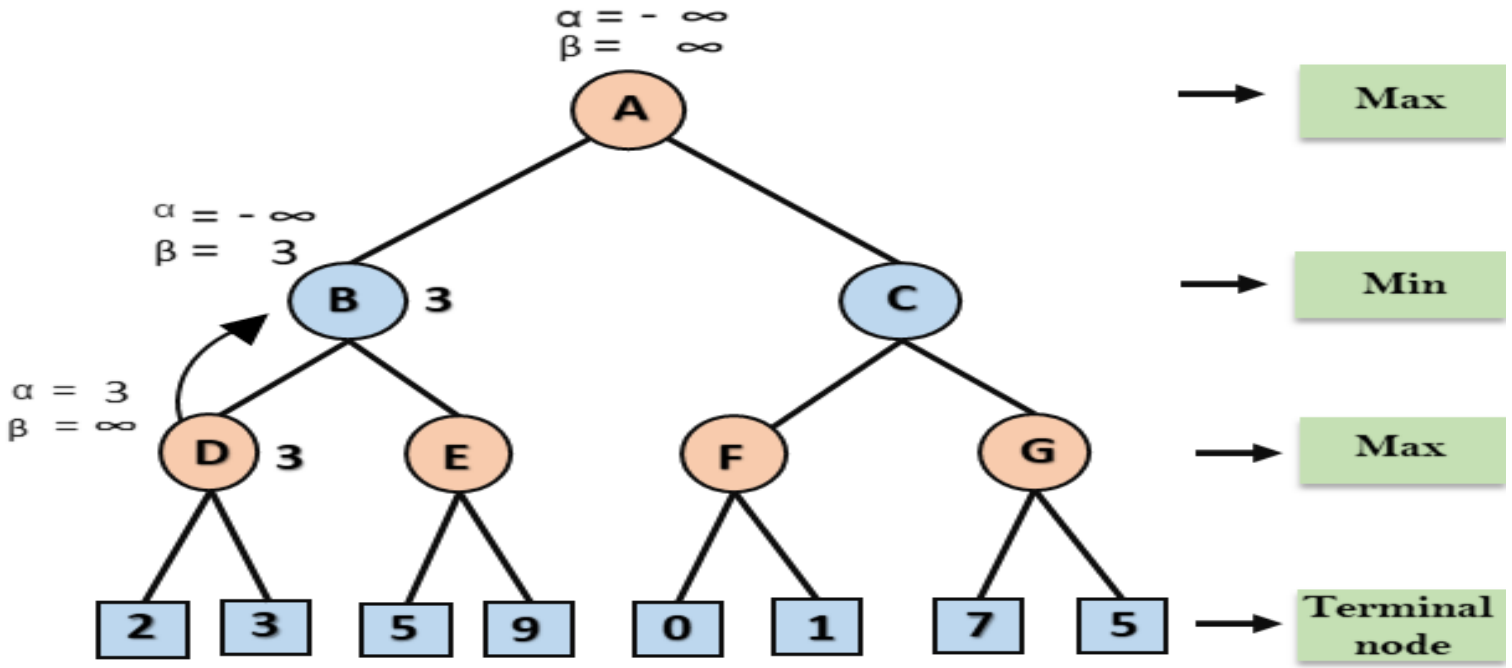
# Alpha-Beta pruning ( Example)

**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.



# Alpha-Beta pruning ( Example)

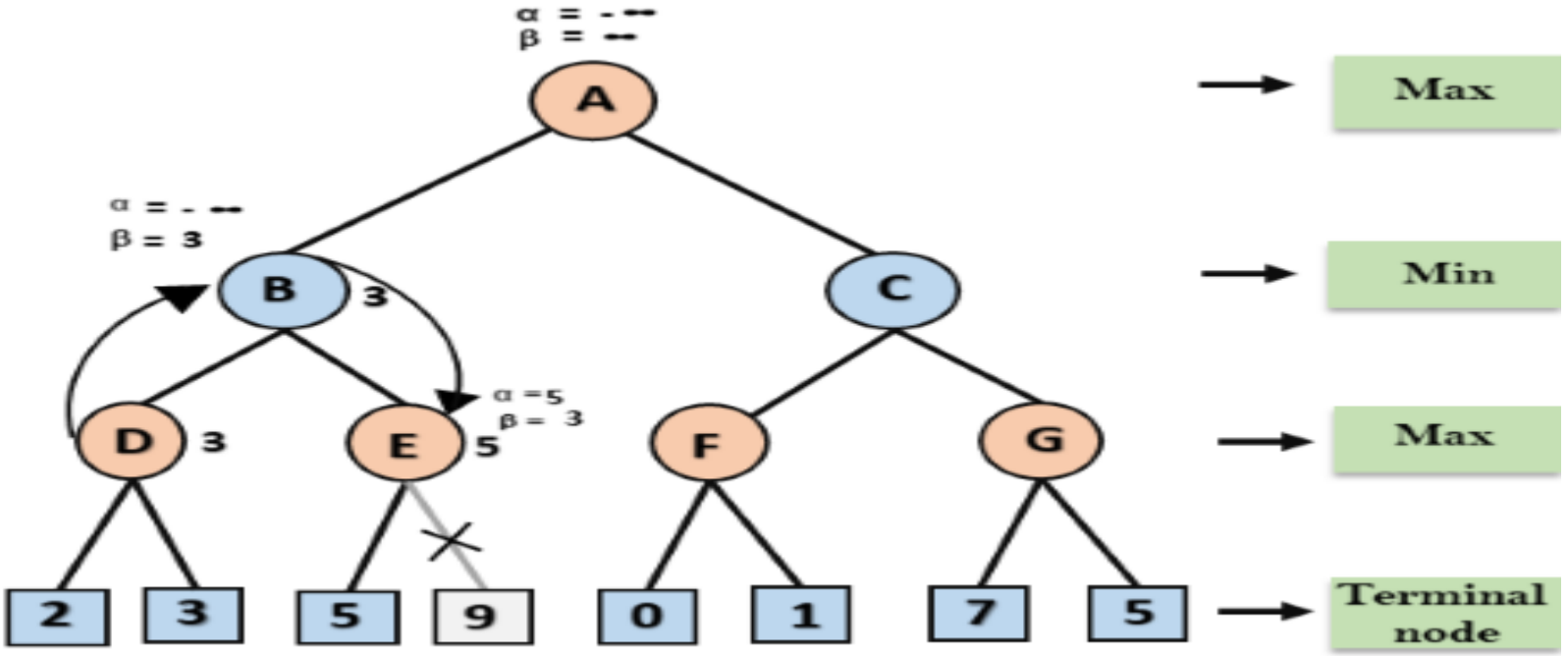
**Step 3:** Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.




In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

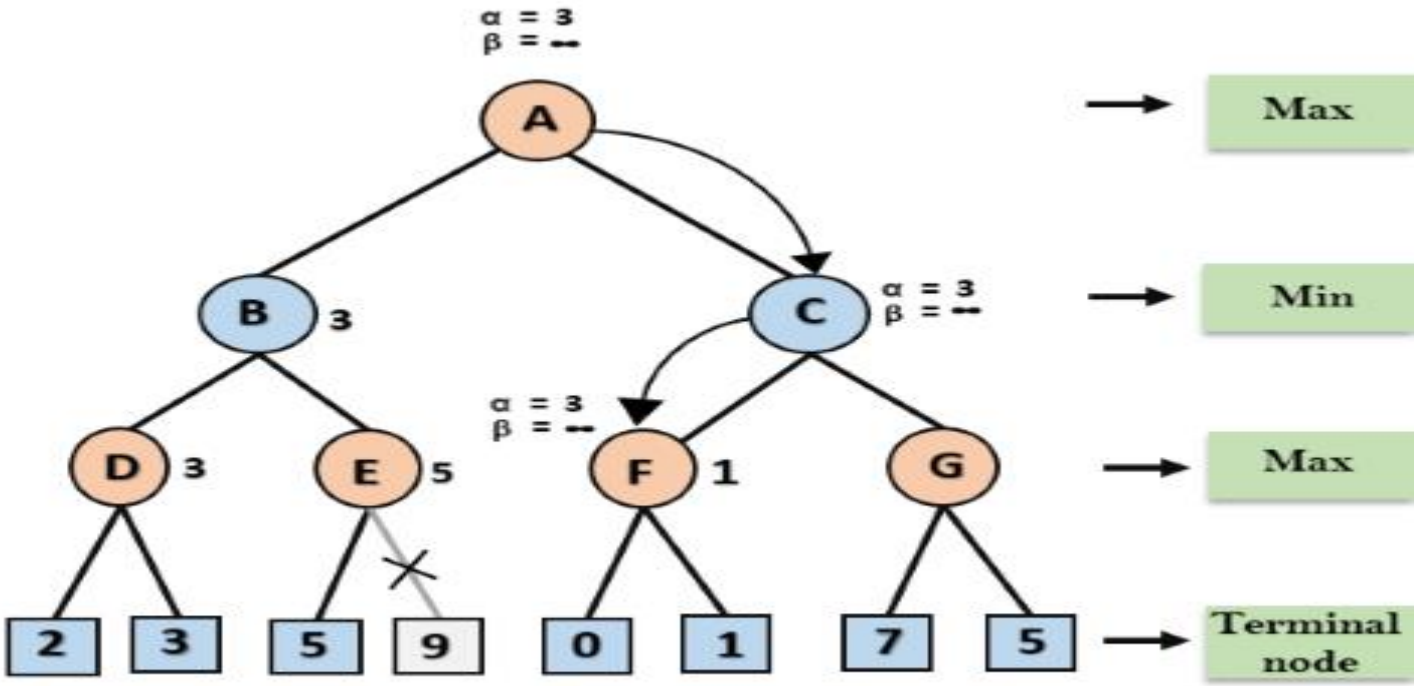
# Alpha-Beta pruning ( Example)

**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 (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



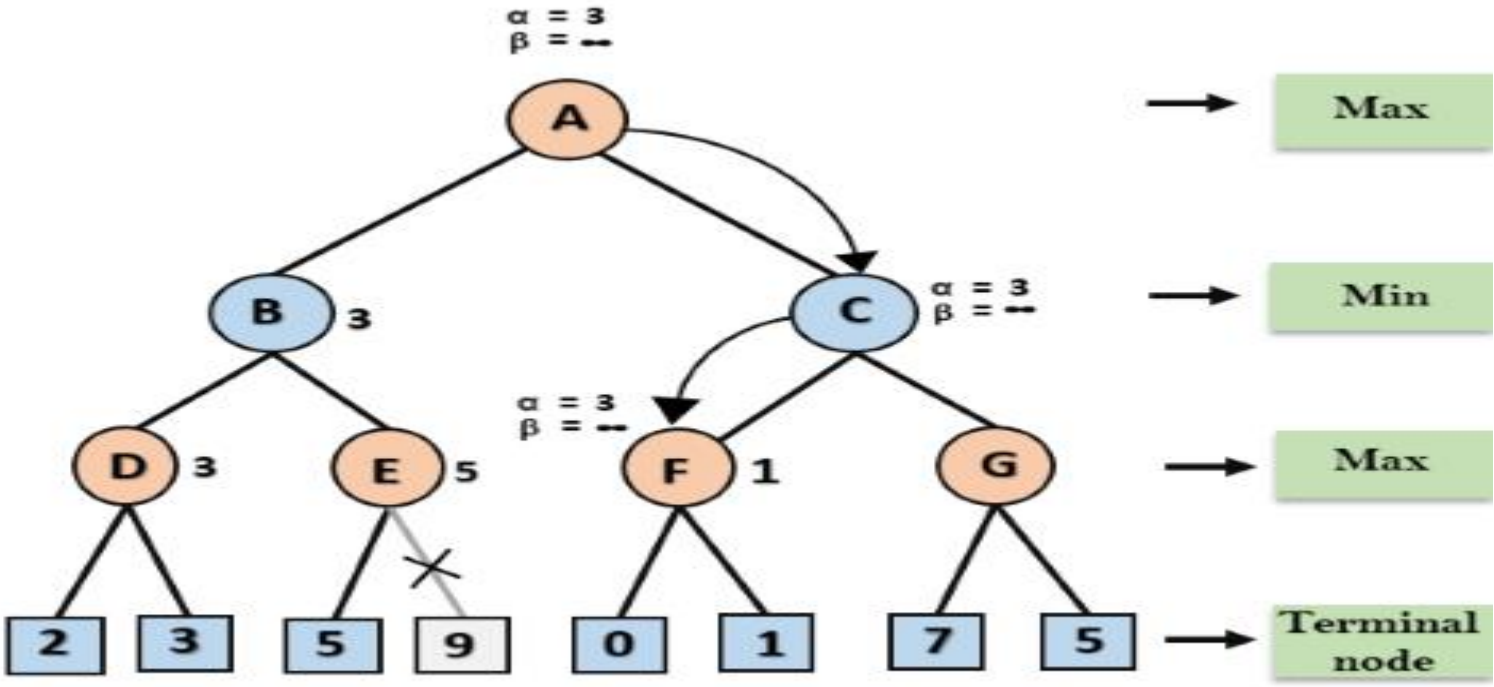
# Alpha-Beta pruning ( Example)

**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 (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
At node C, α=3 and β= +∞, and the same values will be passed on to node F.



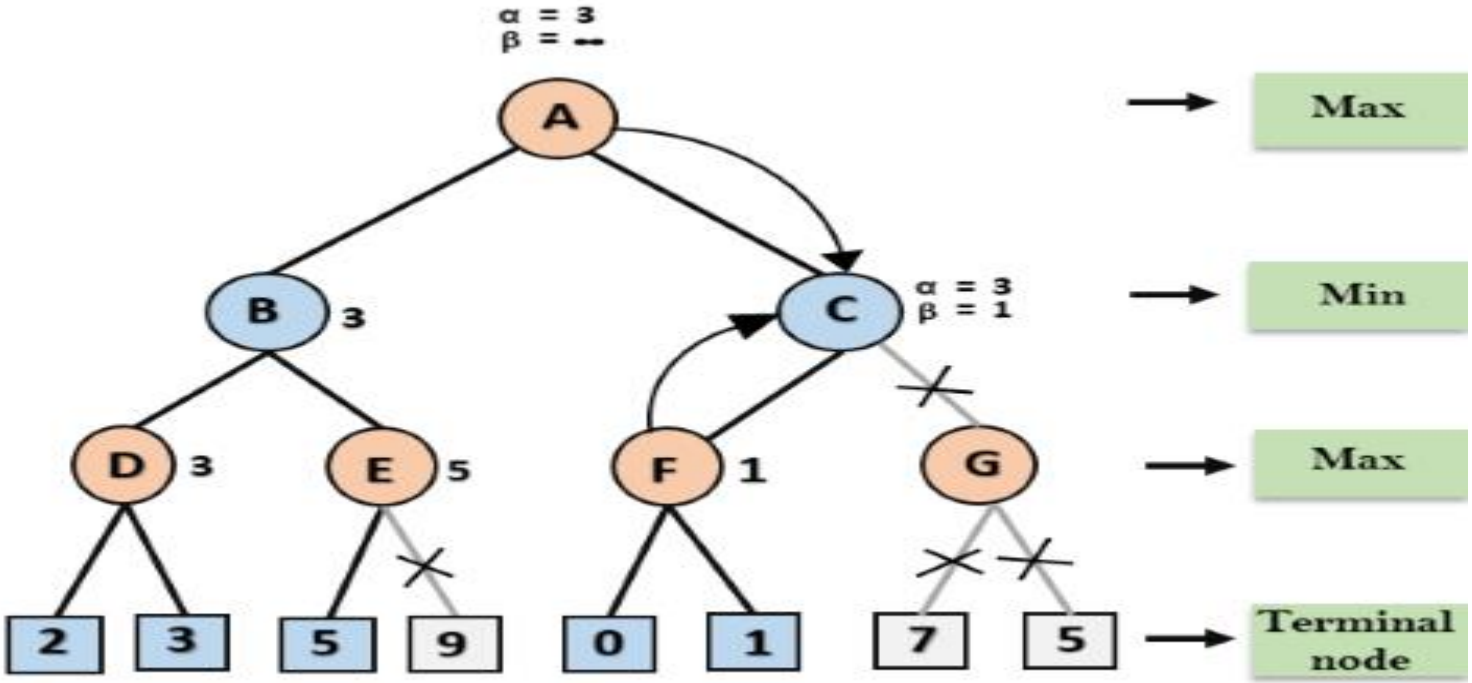
# Alpha-Beta pruning ( Example)

**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.



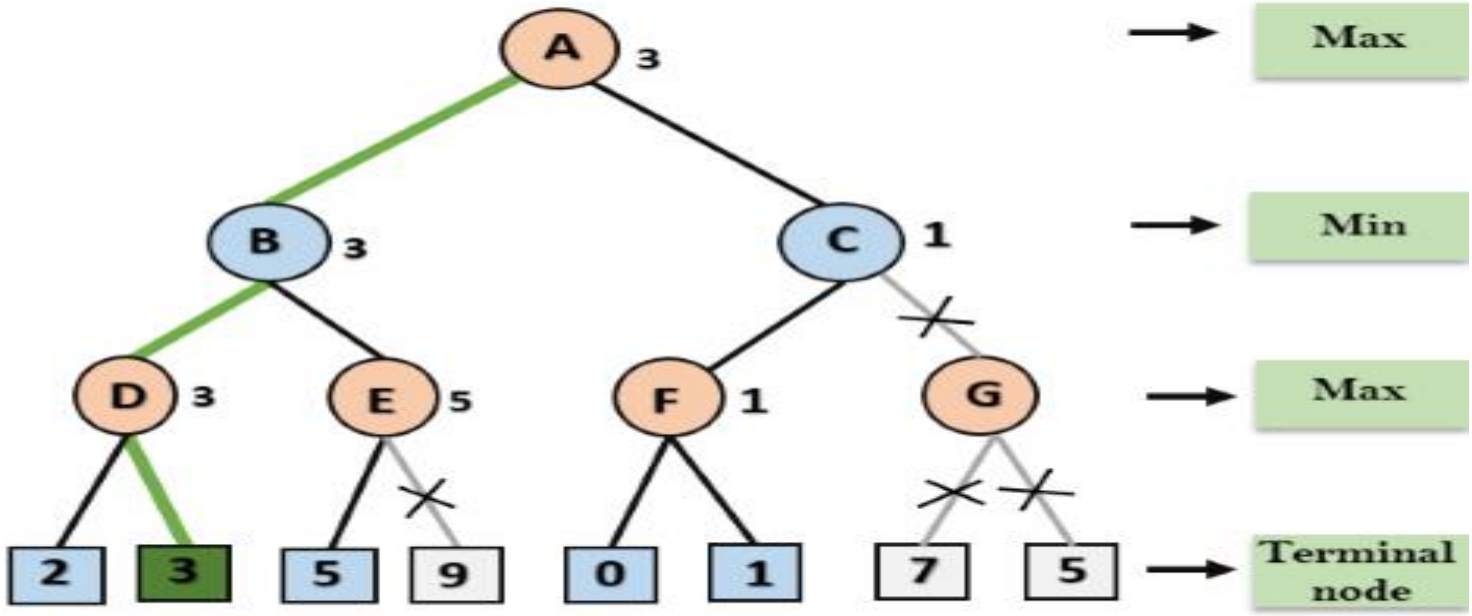
# Alpha-Beta pruning ( Example)

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



# Alpha-Beta pruning ( Example)

**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 the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



# Constraint satisfaction problem (CSP):

# Constraint satisfaction problem (CSP):

1. A **Constraint Satisfaction Problem (CSP)** is a mathematical problem defined by a set of **variables**, a set of **domains** (possible values for each variable), and a set of **constraints** that restrict the values the variables can take.
2. Many AI problems can be viewed as problems of constrained satisfaction in which the goal is to solve some problem state that satisfies a given set of constraints.
3. **Applications of CSP:**

- **AI and Robotics:** Scheduling and planning tasks.
- **Operations Research:** Timetable scheduling, vehicle routing.
- **Computer Vision:** Image recognition, object tracking.
- **Natural Language Processing:** Parsing sentences with grammar rules.

## Constraint satisfaction problem (CSP):

### Key Components of CSP:

### Variables (X)

A finite set of variables: ***X1,X2,...,Xn***

### Domains (D)

Each variable ***Xi*** has a domain ***Di,*** which is the set of possible values it can take.

### Constraints (C)

A set of constraints that define allowable combinations of values for subsets of variables.

## Constraint satisfaction problem (CSP):

**Example of a CSP:**

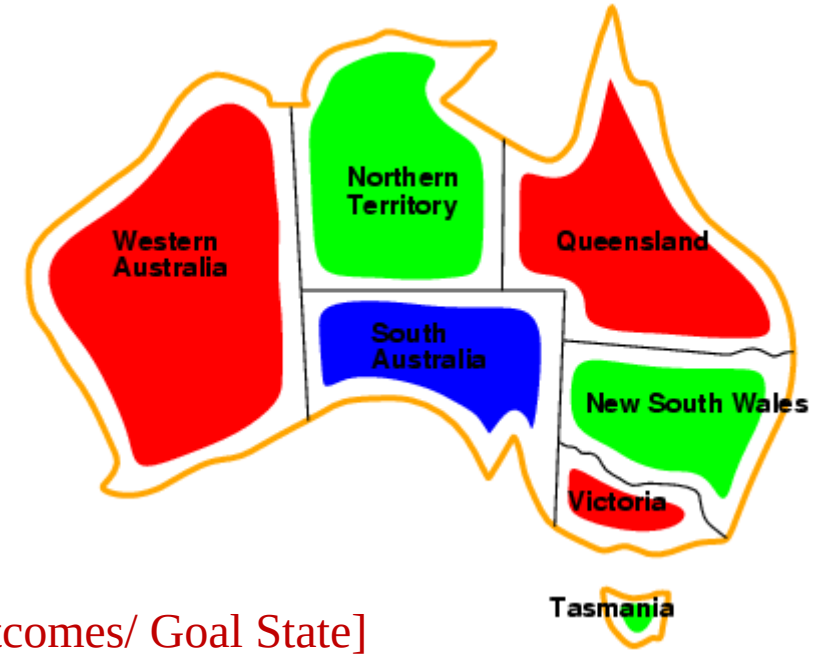
1. Consider the **Map Coloring Problem** (e.g., coloring a map where no two adjacent regions have the same color).
2. **Variables:** Different regions (e.g., {A, B, C, D}).
3. **Domains:** Available colors (e.g., {Red, Green, Blue}).
4. **Constraints:** Adjacent regions cannot have the same color.

## Constraint satisfaction problem (CSP): Example

1. **state** is defined by variables $X_i$ with values from domain $D_i$
2. **goal test** is a set of constraints specifying allowable combinations of values for subsets of variables
3. Allows useful general-purpose algorithms with more power than standard search algorithms

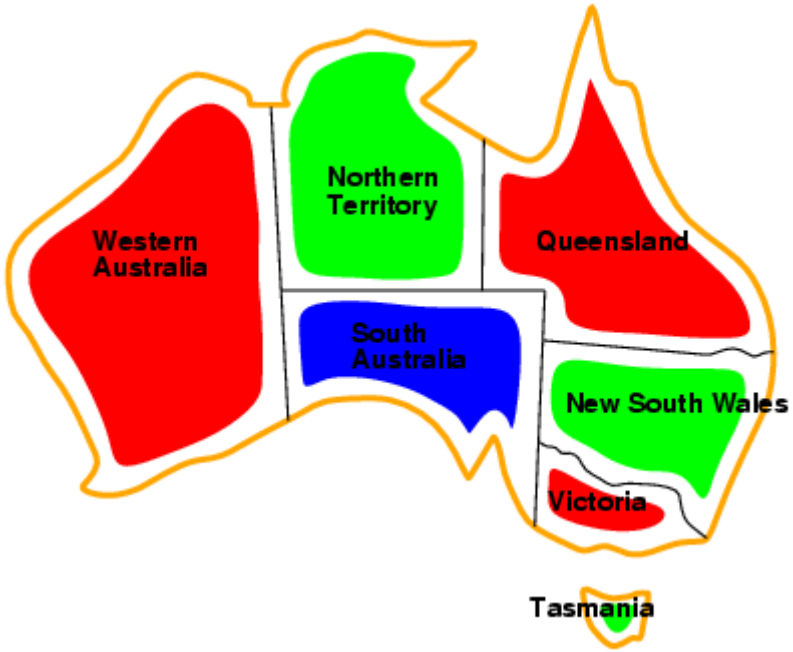
## Constraint satisfaction problem (CSP): Example

1. **Variables** ***WA, NT, Q, NSW, V, SA, T***
2. **Domains** $D_i$ = **{red, green, blue}**
3. **Constraints:** adjacent regions must have different colors e.g., WA ≠ NT




[Outcomes/ Goal State]

## Constraint satisfaction problem (CSP):




**Solutions** are complete and consistent assignments, e.g., **WA = red, NT = green, Q = red,**

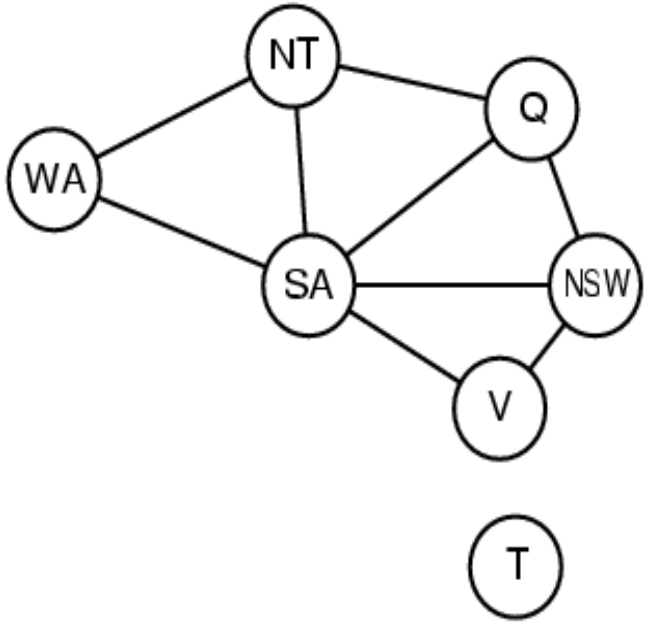
**NSW = green, V = red, SA = blue, T = green**

# Constraint satisfaction problem (CSP):

## Constraint graph

**Binary CSP:** each constraint relates two variables

**Constraint graph:** nodes are variables, arcs are constraints

## Constraint satisfaction problem (CSP):

**Varieties of constraints**

**Unary constraints involve a single variable,**

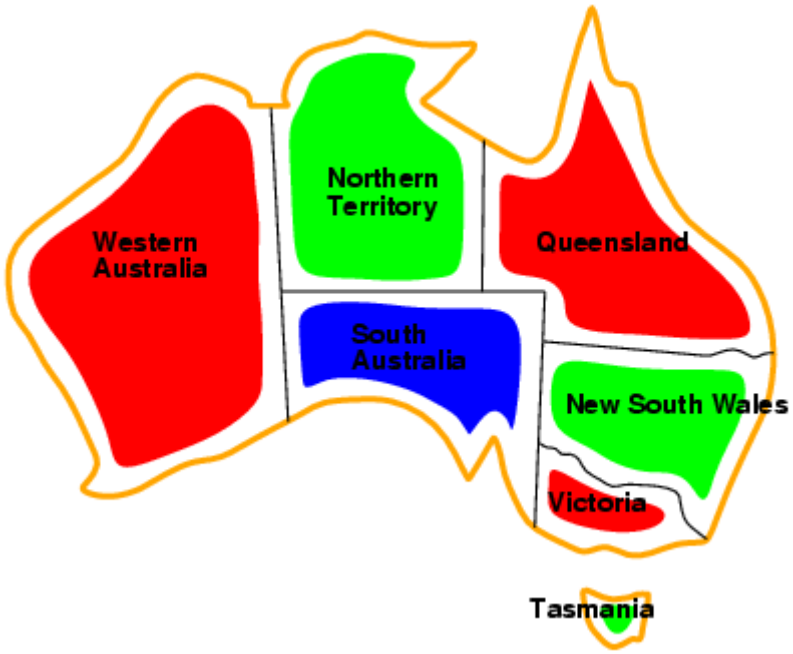
e.g., SA ≠ green

**Binary constraints involve pairs of variables,**

e.g., SA ≠ WA

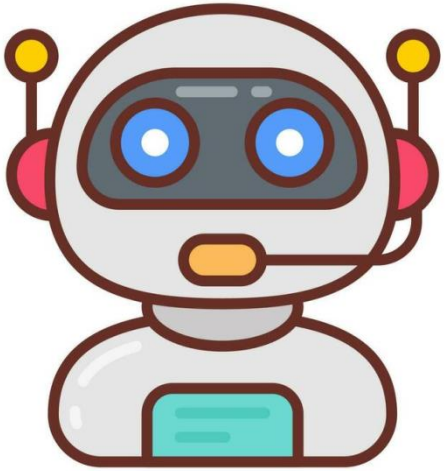
**Higher-order constraints involve 3 or more variables,**

e.g., SA ≠ WA ≠ NT



# Knowledge-based Agents

## What is an AI Agent?



An **AI Agent** is a system that:

- **Perceives** the environment through **sensors.**
- **Processes** data or information to **make decisions**.
- **Acts** on the environment using **actuators or other means**.
- **Continuously evaluates** its performance based on **feedback** or **predefined criteria**.

## Examples of AI Agents:

1. A self-driving car that navigates roads.
2. A chatbot interacting with users.
3. A robotic vacuum cleaner that avoids obstacles.

GITAM
DEEMED TO BE UNIVERSITY

# Interaction Between AI Agents and Environment

The interaction process involves:

1. **Perception:** Sensors capture the environment's state.
2. **Decision-Making:** The agent determines the best course of action.
3. **Action:** The agent executes the chosen action, which impacts the environment.
4. **Feedback Loop:** The environment responds to the action, completing the cycle.

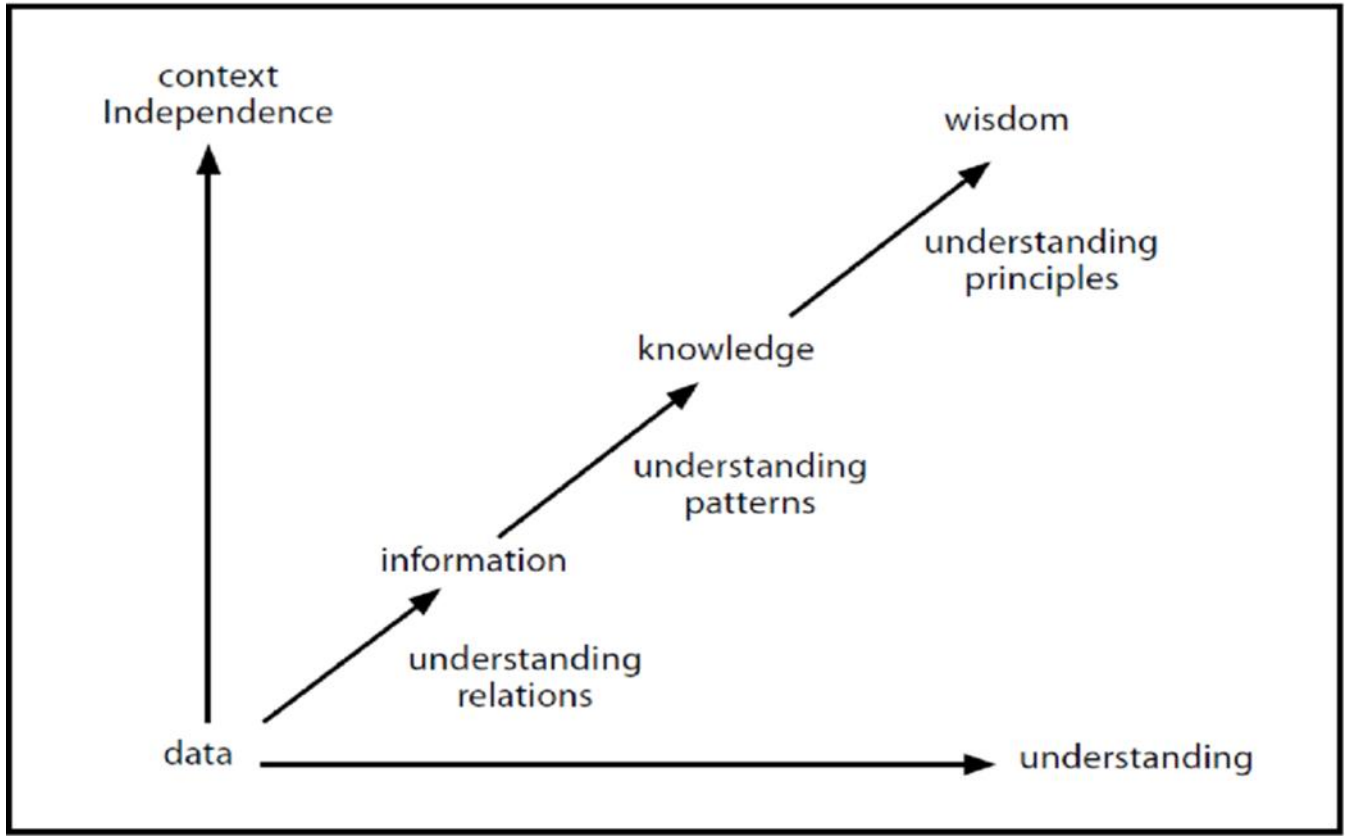
# Types of AI Agents

1. **Simple Reflex Agents:** React directly to stimuli (e.g., thermostats).
2. **Model-Based Reflex Agents:** Use an internal model of the environment for decision-making.
3. **Goal-Based Agents:** Operate to achieve specific objectives (e.g., route optimization).
4. **Utility-Based Agents:** Evaluate and prioritize multiple goals based on utility (e.g., trading algorithms).
5. **Learning Agents:** Adapt behavior over time to improve performance (e.g., AlphaGo).

# Knowledge-based Agents

## What is Knowledge?

1. Knowledge is the **information about a domain that can be used to solve problems in that domain**.
2. To solve many problems requires much knowledge, and this knowledge must be represented in the computer. As part of designing a program to solve problems, we must define how the knowledge will be represented.



# Knowledge-based Agents

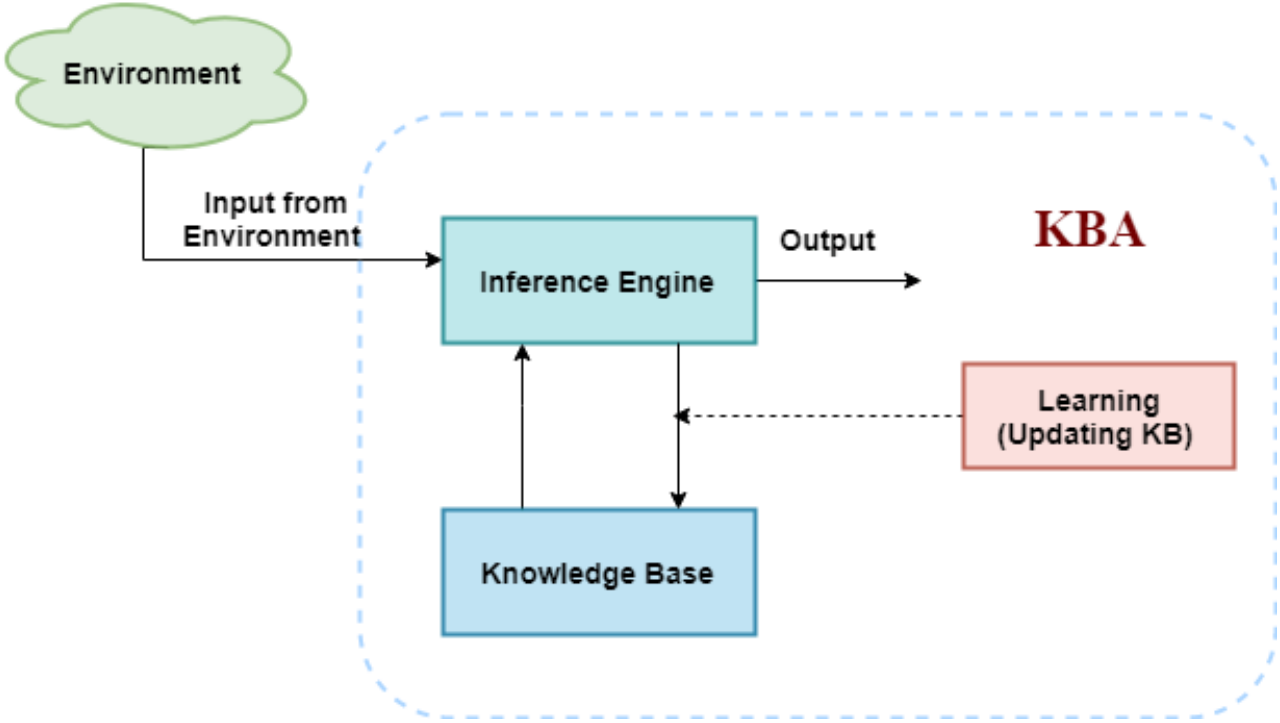
**Kind of knowledge which needs to be represented in AI systems.**

Following are the kind of knowledge which needs to be represented in AI systems:

1. **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
2. **Events:** Events are the actions which occur in our world.
3. **Performance:** It describe behavior which involves knowledge about how to do things.
4. **Meta-knowledge:** It is knowledge about what we know.
5. **Facts:** Facts are the truths about the real world and what we represent.
6. **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

# Knowledge-based Agents

1. A **Knowledge-Based Agent (KBA)** is an AI system that utilizes a structured knowledge base to make decisions, reason about its environment, and perform intelligent actions.
2. These agents operate using a **knowledge base (KB)**, which stores facts, rules, and heuristics, and an **inference engine**, which applies logical reasoning to derive conclusions.



# Knowledge-based Agents

**Key Components of Knowledge-Based Agents:**

1. **Knowledge Base (KB)** : A structured repository of facts, rules, and ontologies.
2. **Inference Engine** : Uses reasoning techniques (e.g., logical inference, rule-based processing) to derive new knowledge.
3. **Knowledge Representation** : Methods such as propositional logic, first-order logic, semantic networks, or ontologies.
4. **Reasoning Mechanisms** : Deductive, inductive, or abductive reasoning to infer new facts and make informed decisions.

GITAM
DEEMED TO BE UNIVERSITY

# Knowledge-based Agents

**Applications:**

1. **Expert Systems** (e.g., medical diagnosis, legal advisory)
2. **Autonomous Decision-Making** (e.g., robotics, intelligent assistants)
3. **Natural Language Processing** (e.g., chatbots, semantic search)
4. **Smart Manufacturing** (e.g., Industry 5.0 process optimization)

GITAM
DEEMED TO BE UNIVERSITY

# Propositional Logic
# Propositional Theorem Proving: Inference

# Propositional Logic

## What is Logic?

1. Logic is the basis of all mathematical reasoning, and of all automated reasoning. The rules of logic specify the meaning of mathematical statements.
2. In **simple words**, **logic** is "the study of correct reasoning, especially regarding making inferences." **Logic** began as a philosophical term and is now used in other disciplines like math and computer science.

## Importance of Mathematical Logic

1. The rules of logic give precise meaning to mathematical statements. These rules are used to distinguish between valid and invalid mathematical arguments.
2. Apart from its importance in understanding mathematical reasoning, logic has numerous applications in Computer Science, varying from design of digital circuits, to the construction of computer programs and verification of correctness of programs.

GITAM
DEEMED TO BE UNIVERSITY

# Propositional Logic

•**Logic** is a study of principles used to

  •**distinguish correct from incorrect reasoning**.

•Formally it deals with

  •the notion of truth in an abstract sense and is concerned with the principles of valid inferencing.

•**A proposition** is the basic building block of logic.{ **It is defined as a declarative sentence that is either True or False, but not both.}**

•The **Truth Value** of a proposition is **True(denoted as T)** if it is a true statement, and **False(denoted as F)** if it is a false statement.

•example,

**The sun rises in the East and sets in the West.**

**1 + 1 = 2**

**'b' is a vowel.**

•Given some propositions to be true in a given context,

  •logic helps in inferencing new proposition, which is also true in the same context.

# Propositional Logic

1. Propositional Logic, also called **Sentential Logic** or **Boolean Logic**, is a formal system in logic that deals with propositions and their relationships using logical connectives.
2. It is widely used in **mathematics, computer science, artificial intelligence, and philosophy** for reasoning and problem-solving.

# Propositional Logic

Suppose we are given a set of propositions such as

**“It is hot today" and**

**“If it is hot it will rain", then**

**we can infer that “It will rain today".**

**Definition of Propositional Formula**

1) Any atom P is a formula.

2) If A is a formula so is ¬A.

3) If A, B are formulas, so is (A ∧ B),

4) If A, B are formulas, so is (A ∨ B).

All (propositional) formulas are constructed from atoms using rules 2) - 4).

GITAM
DEEMED TO BE UNIVERSITY

# Propositional Logic

## Well-formed formula

Propositional calculus is the formal basis of **logic** dealing with the notion and usage of words such as **"NOT," "OR," "AND,"** and **"implies."** Many systems of propositional calculus have been devised which attempt to achieve consistency, completeness, and independence of axioms.

The term "sentential calculus" is sometimes used as a synonym for propositional calculus.

Well-formed formula is defined as:

1. An **atom(a)** is a **well-formed formula**.
2. If **a** is a well-formed formula, then **~a** is a well-formed formula.
3. If **a** and **b** are well formed formulae, then **(a and b), (a or b ), (a -> b), (a <->b )** are also well-formed formulae.
4. A **propositional expression** is a **well-formed formula** if and only if it can be obtained by using above conditions.

# Truth Table

1. Truth table gives us operational definitions of important logical operators.
2. By using truth table, the truth values of well-formed formulae are calculated.
3. Truth table elaborates all possible truth values of a formula.
4. The meanings of the logical operators are given by the following truth table.

| P | Q | ~P | P ∧ Q | P∨Q | P → Q | P ←→ Q |
|---|---|---|---|---|---|---|
| T | T | F | T | T | T | T |
| T | F | F | F | T | F | F |
| F | T | T | F | T | T | F |

# Propositional Logic

# Propositional Logic

*Reading Books Will Improve Your Knowledge*

*Thank You*

**Dr.Satyabrata Dash(SMIEEE)**
Assistant Professor
Department of Computer Science and Engineering, GITAM
Deemed to be University
sdash@gitam.edu

GITAM
DEEMED TO BE UNIVERSITY