Introduction to AI and Production Systems

Module 1

AI Definition

1. “Artificial intelligence is the art of creating machines that perform functions that require intelligence when performed by people. (Kurzweil, 1990)”
2. “Artificial intelligence is the study of how to make computers do things at which, at the moment, people are better.”
3. “An agent is something that acts. A rational agent is an agent that acts so as to achieve the best outcome or when there is uncertainty, the best expected outcome. Artificial intelligence is the study of the general principles of rational agents and of the components for constructing them.”

Task Domains of AI

- **1. Mundane Tasks (Common-Sense AI)**
- These are tasks that humans perform effortlessly but are challenging for AI due to their need for perception, reasoning, and natural language understanding.
- **Perception:** Vision, speech recognition
- **Natural Language Processing (NLP):** Understanding, generating, and translating human language
- **Commonsense Reasoning:** Making everyday decisions based on incomplete information
- **Robotics:** Navigating, grasping objects, and interacting with the physical world

- **2. Formal Tasks (Well-Defined Problems)**
- These tasks involve structured problems with well-defined rules and clear evaluation criteria.
- **Mathematics and Logic:** Theorem proving, solving algebraic equations
- **Game Playing:** Chess, Go, and other board games with strict rules
- **Planning and Search:** Finding optimal solutions using algorithms like A* and Minimax

- **3. Expert Tasks (Specialized AI Applications)**
- These are tasks that require deep domain knowledge and specialized reasoning.
- **Medical Diagnosis:** AI-based systems assisting in detecting diseases
- **Engineering Design:** AI-driven simulations for product development
- **Financial Analysis:** AI predicting stock market trends and risks
- **Scientific Discovery:** AI assisting in hypothesis generation and experimentation

Problems

- To build a system to solve a particular problem, we need 4 things

1. Define the problem precisely
2. Analyze the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best problem-solving technique and apply it

Defining the problem as a state space search

- In AI, problems are solved by searching through a **state space**.
- A **state space** consists of all possible configurations of the problem.
- The **search process** finds a path from an **initial state** to a **goal state** using a set of rules (operators).

- A problem is formally defined as:
- **State space -**that contains all the possible configurations of the relevant objects
- **Initial State** – The starting configuration of the problem.
- **Operators** – Set of rules that describe the actions available
- **Goal State** – The desired solution state(s).

- The problem can be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found.

- **Search Trees and Search Graphs**
- **Search Tree**: A tree structure representing possible moves from the initial state.
- **Search Graph**: A graph where nodes represent states and edges represent operators.
- **Solution Path**: A sequence of moves leading to the goal.

Water Jug Problem

- You are given two jugs with fixed capacities:
- A **4-liter jug (Jug X)**
- A **3-liter jug (Jug Y)**
- An **unlimited water supply** and the ability to **empty** or **transfer** water between jugs.
- The goal is to measure **exactly 2 liters** of water in 4 liter jug.

Water Jug Problem

- A **state** is represented as **(x, y)**, where:
- **x** is the amount of water in 4 gallon jug.
- **y** is the amount of water in 3 gallon jug.
- **Initial State:**
- (0, 0) → Both jugs are empty.
- **Goal State:**
- (2, y)

Operators

- **Fill Jug X** → (4, y)
- **Fill Jug Y** → (x, 3)
- **Empty Jug X** → (0, y)
- **Empty Jug Y** → (x, 0)
- **Pour water from X to Y** until Y is full or X is empty
  - If $x+y \le 3$ → $(0, x + y)$
  - Else $x-(3-y),3$
- **Pour water from Y to X** until X is full or Y is empty
  - If $x+y \le 4$ → $(x + y, 0)$
  - Else → $4,y-(4-x)$

State space graph for water jug problem

- **(0,0)** → Initial state
- **(4,0)** → Fill Jug X
- **(1,3)** → Pour from X to Y (3 liters transferred, 1 remains in X)
- **(1,0)** → Empty Jug Y
- **(0,1)** → Pour from X to Y
- **(4,1)** → Fill Jug X again
- **(2,3)** → Pour from X to Y (2 liters transferred, reaching the goal!)

Missionaries and Cannibals Problem

- **1. Problem Statement**
- There are **three missionaries** and **three cannibals** on one side of the river.
- A boat can carry at most **two people** at a time.
- The goal is to transport **all six people to the other side** without violating safety constraints:
  - **Missionaries must never be outnumbered by cannibals** on either side.

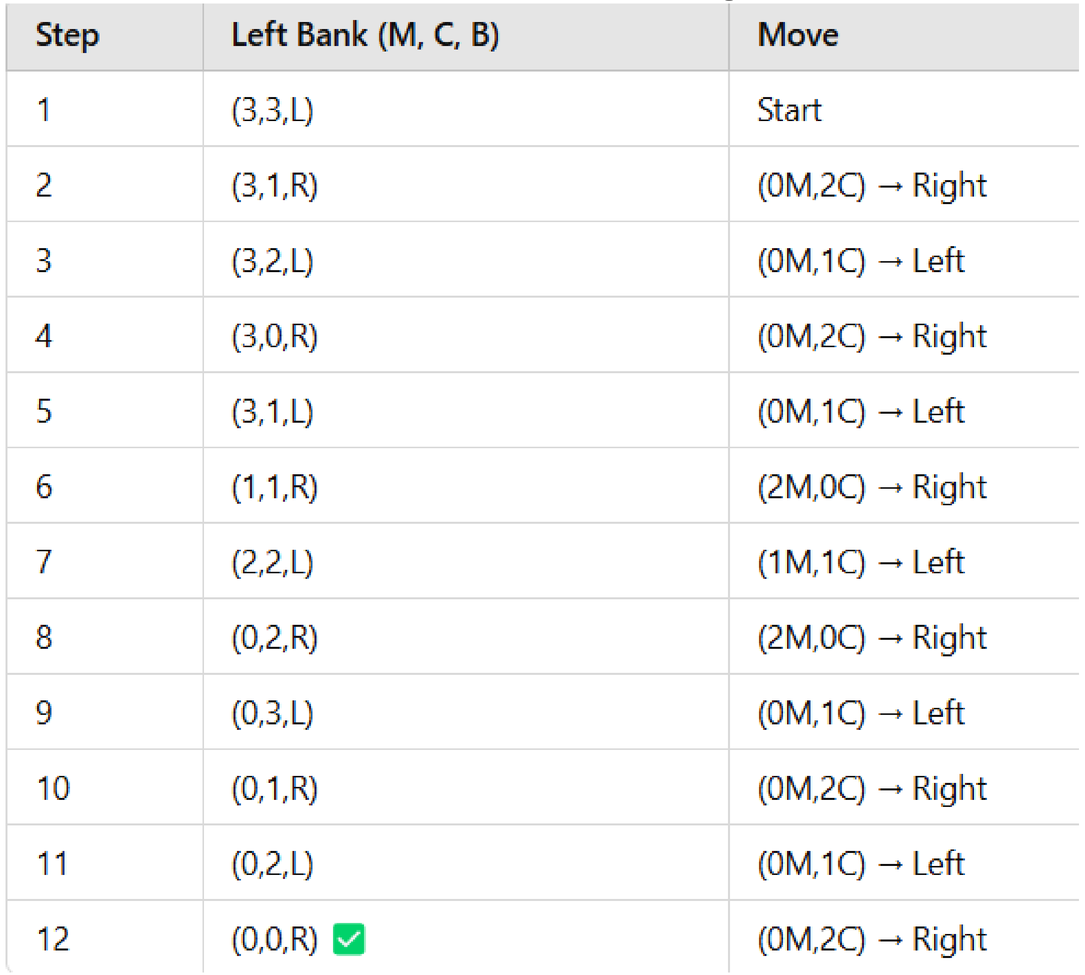
Missionaries and Cannibals Problem

- **Initial State**: Three missionaries and three cannibals on one side of the river.
- **Operators**: Boat moves 1 or 2 people across the river.
- **Constraints**: Cannibals must never outnumber missionaries on either side.
- **Goal State**: All safely transported across the river.

- Each **state** is represented as **(M, C, B)**, where:
- **M** = Number of missionaries on the left bank.
- **C** = Number of cannibals on the left bank.
- **B** = Boat position (L = left, R = right).
- **Initial State:**
- (3,3,L)(All missionaries and cannibals on the left bank)
- **Goal State:**
- (0,0,R)(All missionaries and cannibals safely on the right bank)

- **Constraints:**
- Cannibals can never outnumber missionaries on either bank unless there are **no missionaries** present.
- The boat cannot move without at least **one person** onboard.

Solution to Missionaries problem



Step	Left Bank (M, C, B)	Move
1	(3,3,L)	Start
2	(3,1,R)	(0M,2C) → Right
3	(3,2,L)	(0M,1C) → Left
4	(3,0,R)	(0M,2C) → Right
5	(3,1,L)	(0M,1C) → Left
6	(1,1,R)	(2M,0C) → Right
7	(2,2,L)	(1M,1C) → Left
8	(0,2,R)	(2M,0C) → Right
9	(0,3,L)	(0M,1C) → Left
10	(0,1,R)	(0M,2C) → Right
11	(0,2,L)	(0M,1C) → Left
12	(0,0,R) ✅	(0M,2C) → Right

Problem-Playing Chess

- Each state in the chess game is a **unique board configuration** (position of all pieces).
- The **initial state** is the standard chessboard setup at the beginning of the game.
- A **goal state** could be:
  - **Checkmate** (opponent’s king is trapped).
  - **Stalemate** (game ends in a draw).

Problem-Playing Chess

- **Operators (Legal Moves)**
- Operators are the **legal moves** available to each piece.
- Example operators:
  - **Pawn moves**: Forward one square (or two if it's the first move).
  - **Knight moves**: L-shaped moves.
  - **Castling**: A special king-rook move.
  - **Captures**: Taking opponent’s pieces.

Problem-Playing Chess

- **State Space**
- The **state space** is the **set of all possible board positions** from the initial state.
- A complete chess game can have **10^40 to 10^50** possible states!
- Due to this huge number, AI does not explore all states blindly but uses efficient search techniques.

Types of problems

- **. Well-Defined Problems** – Clear starting point, goal, and rules!
  Example: Solving a Sudoku puzzle!
- **Ill-Defined Problems** – No clear path, multiple solutions!
  Example: Writing a creative story!

Problem characteristics

Different characteristics of problems that help to choose appropriate method for a particular problem

- Problem Decomposability
- Solution steps can be ignored or undone.
- The problem’s universe is predictable or not .
- Solution is Absolute or Relative?
- Is the solution a State or Path ?
- Role of Knowledge in Problem-Solving
- Interaction with a Person (Human Involvement)

Problem decomposability

- Some problems can be **broken down into smaller subproblems**, solved independently, and then combined to form the final solution.
- This makes the problem **easier to solve** and more **efficient**

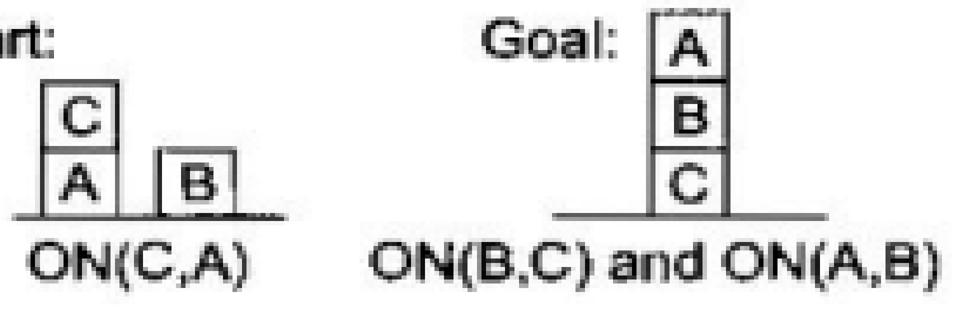
Examples illustrating problem characteristics

- Example 1
- Consider the problem: Evaluate the integral
- We show that this problem is decomposable t $\int (x^2 + \sin^2 x)\, dx.$ oblems Since,
- the prob $\int (x^2 + \sin^2 x)\, dx = \int x^2\, dx + \int \sin^2 x\, dx,$ pler subproblems. • Problem (i): Evaluate. • Problem (ii): Evaluate

$\int x^2\, dx$ $\int \sin^2 x\, dx,$

$$\begin{tikzcd}
& \boxed{\int (x^2 + \sin^2 x)\, dx} \arrow[dl] \arrow[dr] & & \\
\boxed{\int x^2\, dx} \arrow[d] & & \boxed{\int \sin^2 x\, dx} \arrow[d] & \\
\boxed{\frac{1}{3}x^3} & & \boxed{\int \frac{1}{2}(1 - \cos 2x)\, dx} \arrow[dl] \arrow[dr] & \\
& \boxed{\int \frac{1}{2}\, dx} \arrow[d] & & \boxed{\int -\frac{1}{2}\cos 2x\, dx} \arrow[d] \\
& \boxed{\frac{1}{2}x} & & \boxed{-\frac{1}{4}\sin 2x}
\end{tikzcd}$$

- Example 2 Consider a simple blocks world problem



- The following are the solution steps : UNSTACK(C,A), PUTDOWN(C) , PICKUP(B), STACK(B,A), STACK(C,B) Since, the steps are interdependent the problem cannot be decomposed into two independent problems.

2 Solution steps can be ignored or undone.

- **Example 1** Suppose our goal is to prove a theorem in mathematics. On the way, we prove some preliminary results, say Result 1, Result 2, Result 3, and then finally we prove the theorem. The steps in the proof look like this:
- Result 1
- Result 2
- Result 3
- Theorem
- Suppose, we later realize that Result 2 is not actually needed in proving the theorem. Then, we may ignore Result 2 and present the steps of the proof as follows:
- Result 1
- Result 3
- Theorem
- In this example, the solution steps can be ignored.

- **Example 2** Consider the 8-puzzle. The solution involves a sequence of moves. In the process of finding a solution, after some moves, we realize that a certain previous move has be reversed. The previous move can be undone by backtracking the moves. In this example, the solution steps can be undone and backtracked.

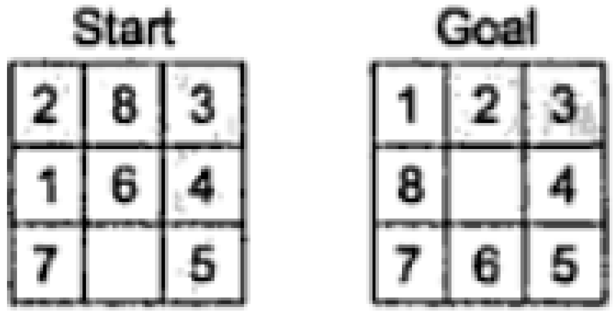


Fig. 2.12 *An Example of the 8-Puzzle*

- **Example 3** Consider the problem of playing chess. If a chess playing programme makes a wrong move, the rules of the game do not permit undoing the move. In this example, the solution steps cannot be undone or ignored.

3.The problem universe is predictable

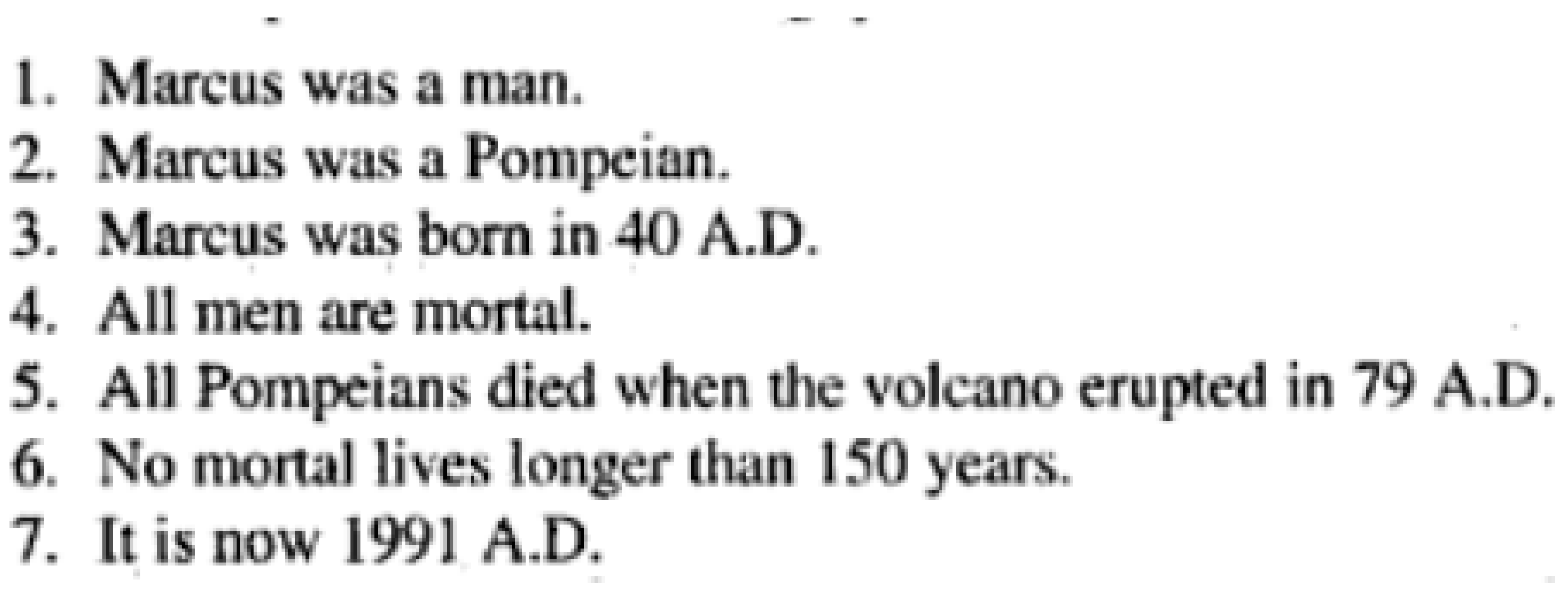
- **Example 1** In the 8-puzzle, every time we make a move we know exactly what will happen. This means that it is possible to plan an entire sequence of moves. Thus in this problem, the universe is predictable.

- **Example 2** In a game like bridge, this is not possible because a player does not know where the various cards are and what the other players will do on their turns. In this problem, the universe is unpredictable.

4. There are obvious good solutions without comparison to all other possible solutions.

- **Example 1** Consider a mathematical problem. In general, there may be many methods for solving the problem. Any method is a good method without comparison to other methods provided it solves the problem. In general, any “any-path problem” is an example of a problem having obvious good solutions without comparison to other possible solutions

- Is Marcus alive?



1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 1991 A.D.

		Justification
1.	Marcus was a man.	axiom 1
4.	All men are mortal.	axiom 4
8.	Marcus is mortal.	1, 4
3.	Marcus was born in 40 A.D.	axiom 3
7.	It is now 1991 A.D.	axiom 7
9.	Marcus' age is 1951 years.	3, 7
6.	No mortal lives longer than 150 years.	axiom 6
10.	Marcus is dead.	8, 6, 9
	OR	
7.	It is now 1991 A.D.	axiom 7
5.	All Pompeians died in 79 A.D.	axiom 5
11.	All Pompeians are dead now.	7, 5
2.	Marcus was a Pompeian.	axiom 2
12.	Marcus is dead.	11, 2

Fig. 2.13 *Two Ways of Deciding That Marcus Is Dead*

- **Example 2** A “best-path problem” is a problem having no obvious good solutions. The travelling salesman problem is an example for a best-path problem. The travelling salesman problem can be formulated as follows: “Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Desired solution is a state of the universe or a path to a state.

- **Example 1**
- In the missionaries and cannibals problem, if we organise the various states in the form of a tree, it can be seen that the solution to the problem is a path connecting the various states

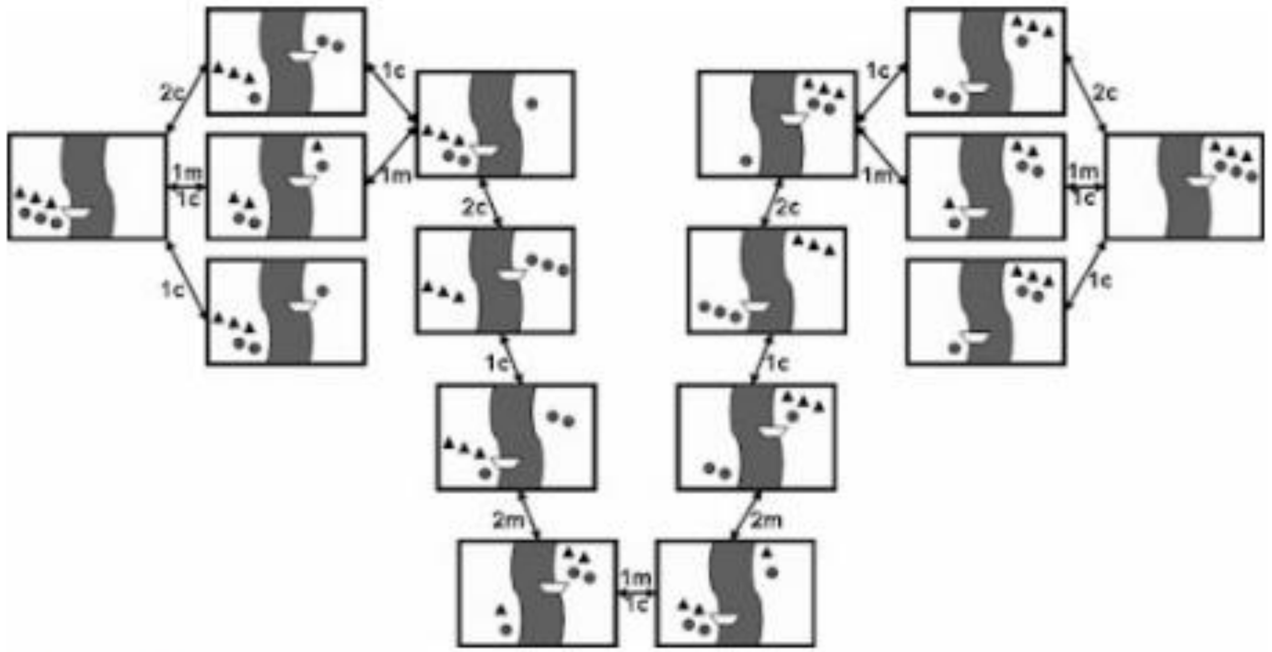


Figure 2.13: The state space of the missionaries and cannibals problem (with triangles representing missionaries and circles representing cannibals

- **Example 2** In the cryptarithmetic puzzle, the solution to the problem is a state of the problem, namely, the state representing the assignment of digits to the letters in the puzzle.

Requires lots of knowledge; or, uses knowledge to constrain solutions

- **Example 1** Consider the problem of playing chess. The amount of knowledge required to play chess is very little: just the rules of the game! Additional knowledge about strategies may be used to make intelligent moves!!
- **Example 2** Consider the problem of scanning daily newspapers to decide which are supporting and which are opposing a political party in an upcoming election. It is obvious that a great deal of knowledge is required to solve this problem.

Problem requires periodic interaction between human and computer.

- Example Even in a so called “fully automated system” situations constantly arise that call for human intervention. When the machines get thrown off track, or become faulty, experts have to be summoned to step in and troubleshoot the problems.

Production System in AI

- A **Production System** in AI is a framework used for problem-solving and knowledge representation.
- It consists of a set of rules (productions) and a mechanism for applying these rules to a given state of a system

Components of a Production System

1. Set of rules

- Each rule is in the form of **IF-THEN** statements
- Each rule in the set of rules consists of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
- Example:
  - **IF** the traffic light is red, **THEN** stop the car.
  - **IF** the goal is reached, **THEN** terminate the search.

2. Knowledge databases

- The databases contain whatever information is appropriate for the particular task.
- Some parts of the database may be permanent.
- The database includes a working memory whose elements may pertain only to the solution of the current problem.

Working Memory

- Stores the current state of the system.
- Changes dynamically as rules are applied.
- Example: In a chess game, the current board position is stored here.

- **3. Control strategy**
  - Determines which rule(s) to apply based on the **current state**.
  - Uses **conflict resolution strategies** to choose between multiple applicable rules.
  - **Types of Control Strategies:**
  - **Data-Driven (Forward Chaining)**
    - Starts with **known facts** and applies rules to reach a conclusion.
    - Example: **Expert Systems like MYCIN** (Medical Diagnosis).
  - **Goal-Driven (Backward Chaining)**
    - Starts with a **goal** and works **backward** to find supporting facts.
    - Example: **Prolog Programming** (AI-based logic programming).

- **Conflict Resolution Strategies**
  - When multiple rules match, the system decides **which rule to apply first**.
  - Strategies include:
    - **Specificity** – More specific rules are preferred.
    - **Recency** – The most recently applied rule takes priority.
    - **Priority Ordering** – Rules are assigned fixed priority values.

4. Interpreter/Rule Applier

- Executes the selected rule and **updates the working memory**.
- Continues until a **goal state** is reached.

Advantages of Production System

- **Modularity** – Rules are independent, making the system easy to modify.
- **Flexibility** – Can handle different types of problems.
- **Scalability** – Works well for large knowledge bases.
- **Transparency** – The rule-based approach makes reasoning clear
- **helpful in a real-time environment** and applications

Disadvantages of production system

- **Computational Cost** – Large rule sets slow down processing.
- **Difficult Conflict Resolution** – Deciding the best rule can be complex.
- **No Learning Ability** – Classical production systems do not learn from past cases.
- It is very **difficult to analyze the flow of control** within a production system.

Types of production systems

- **1. Monotonic Production System**
- **Rules, once applied, never need to be retracted.**
- The knowledge base **only expands**; no information is removed.
- Example: **Mathematical Theorem Proving**
  - Once a theorem is proven, it remains valid.

- **2. Non-Monotonic Production System**
- **Applied rules can be retracted or revised based on new knowledge.**
- New information may contradict previous conclusions.
- Example: **Medical Diagnosis Systems**
  - A diagnosis may change when new symptoms appear.

3. Partially Commutative Production System

- **The order of rule application does not affect the final result.**
- No matter which rules are selected first, the final solution remains the same.
- Example: **Pathfinding Algorithms** like A* Search.

4.Non-Partially Commutative Production System

- **The order of rule application affects the final outcome.**
- If rules are applied in different sequences, different results may occur.
- Example: **Chemical synthesis**
    - applying production rules in different sequences leads to different chemical products, side reactions, or yields.

5. Commutative Production System

- Production system that is **both monotonic and partially commutative**
- Example: **Mathematical Theorem Proving**

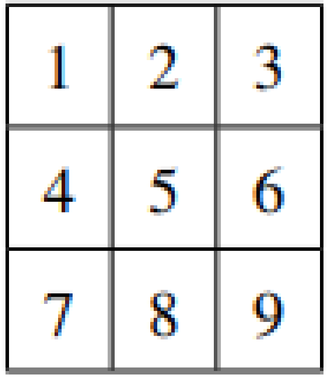
	Monotonic	Nonmonotonic
Partially commutative	Theorem proving	Robot navigation
Not partially commutative	Chemical synthesis	Bridge

Remarks

1. It can be shown that, depending on how the operators are chosen, the 8-puzzle and the blocks world problem are partially commutative systems.
2. Production systems that are not partially commutative are useful in which irreversible changes occur. For example, the process to produce a desired chemical compound may involve may irreversible steps.
3. Commutative production systems are useful for solving ignorable problems like the problem of proving a theorem in mathematics.
4. Nonmonotonic partially commutative production systems are useful for problems in which the order of operations is not important. For example, robot navigation is such a problem.

Production system example: 3 × 3 knight’s tour

- A knight’s tour is a sequence of moves of a knight on a chessboard such that the knight visits every square exactly once. In the 3 × 3 knight’s tour problem, instead of the usual 8 × 8 board in chess, we consider a 3 × 3 board
- **Problem statement** Using the production system model, determine whether a path exists from square 1 to square 2 in the 3 × 3 knight’s tour problem



1	2	3
4	5	6
7	8	9

Solution

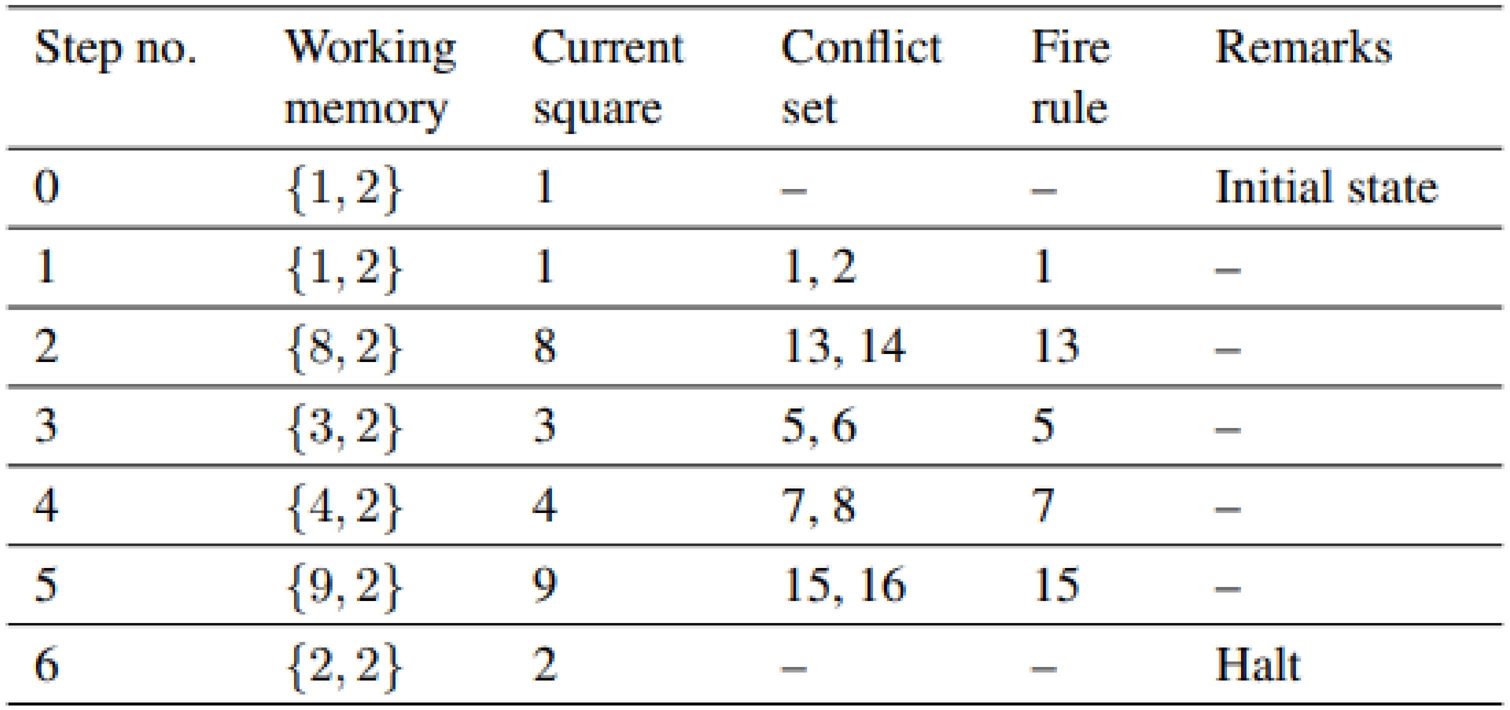
- **Step 1. Problem formulation as a production system**
- **(i) Knowledge database**
- Since there is no information other than that contained in the statement of the problem, the database contains only the working memory. The working memory contains the following two facts: {knight on square 1, knight on square 2} We denote this initial working memory by the set {1, 2}.

- **(ii) Production rules**

Rule No.	Condition		Action
1	knight on square 1	$\rightarrow$	knight on square 8
2	knight on square 1	$\rightarrow$	knight on square 6
3	knight on square 2	$\rightarrow$	knight on square 9
4	knight on square 2	$\rightarrow$	knight on square 7
5	knight on square 3	$\rightarrow$	knight on square 4
6	knight on square 3	$\rightarrow$	knight on square 8
7	knight on square 4	$\rightarrow$	knight on square 9
8	knight on square 4	$\rightarrow$	knight on square 3
9	knight on square 6	$\rightarrow$	knight on square 1
10	knight on square 6	$\rightarrow$	knight on square 7
11	knight on square 7	$\rightarrow$	knight on square 2
12	knight on square 7	$\rightarrow$	knight on square 6
13	knight on square 8	$\rightarrow$	knight on square 3
14	knight on square 8	$\rightarrow$	knight on square 1
15	knight on square 9	$\rightarrow$	knight on square 2
16	knight on square 9	$\rightarrow$	knight on square 4

- **(iii) Conflict resolution strategy**
- Select and fire the first rule in the conflict set that does not lead to a repeated state.
- **(iv) Interpreter** Interpreter is the select-execute loop

- **Step 2. Application of the rules**
- Table shows the sequence of the applications of the rules starting from the state represented by “knight on square 1” in the working memory.



Step no.	Working memory	Current square	Conflict set	Fire rule	Remarks
0	$\{1, 2\}$	1	–	–	Initial state
1	$\{1, 2\}$	1	1, 2	1	–
2	$\{8, 2\}$	8	13, 14	13	–
3	$\{3, 2\}$	3	5, 6	5	–
4	$\{4, 2\}$	4	7, 8	7	–
5	$\{9, 2\}$	9	15, 16	15	–
6	$\{2, 2\}$	2	–	–	Halt

- **Step 3. Conclusion**
- There exists a path from the initial state “knight on square 1” to the goal state “knight on square 2”.

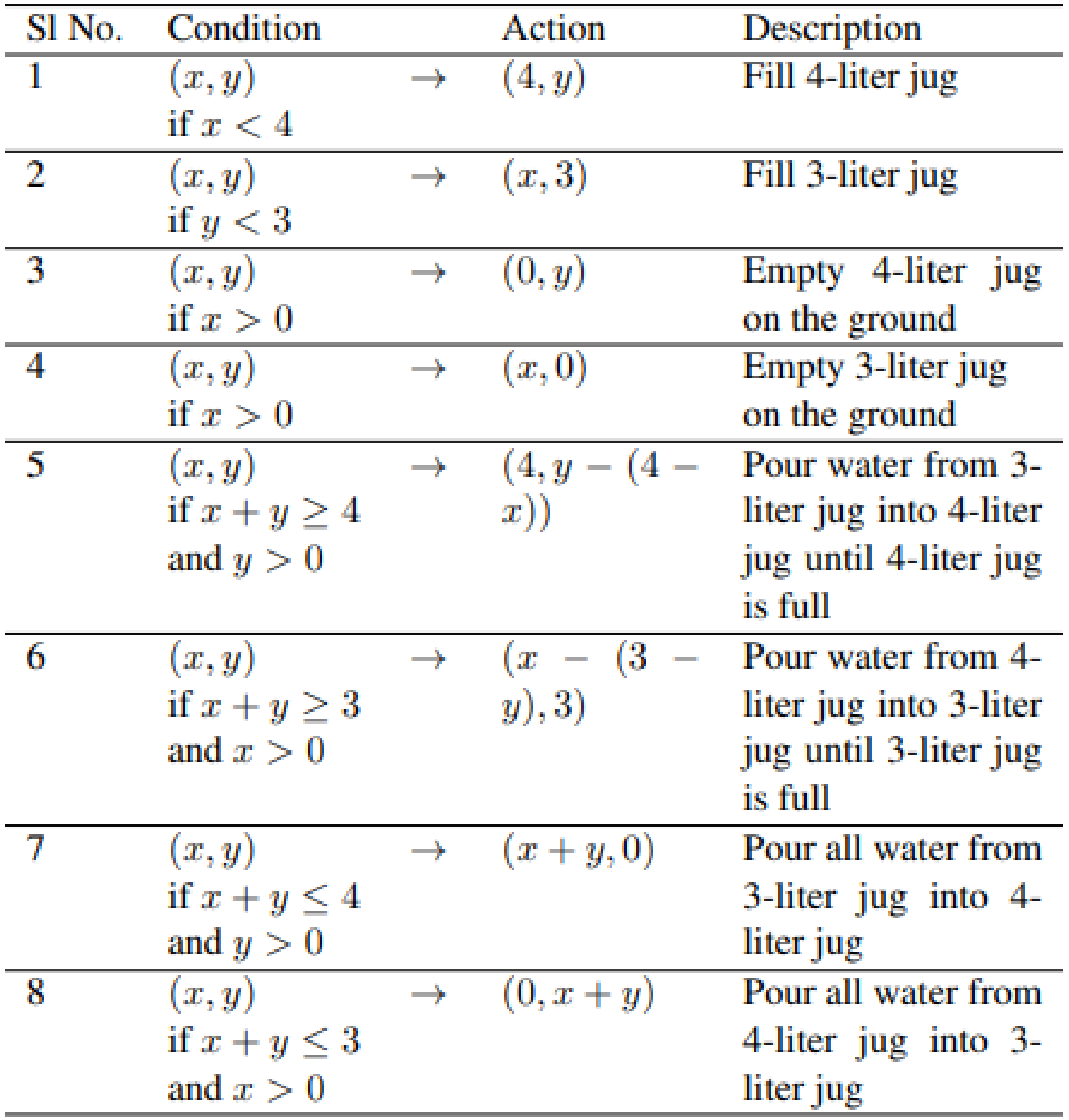
Production system example: The water jug problem

- **Problem statement**
- We have two jugs of capacity 4 liters and 3 liters, and a tap with an endless supply of water. The objective is to obtain 2 liters of water exactly in the 4-liter jug with the minimum steps possible.

Solution

- **Step 1. Problem formulation as a production system**
- we represent a state of the problem by an ordered pair (x, y) of integers where x denotes the number of liters of water in the 4-liter jug and y the number of liters of water in the 3-liter jug.
- **1. Knowledge database** Since there is no information other than that contained in the statement of the problem, the database contains only the working memory. The working memory contain the following two facts: {(0, 0),(2, 0)} where (0, 0) is the initial state and (2, 0) is the goal state.

- 2. Production rules



Sl No.	Condition		Action	Description
1	(x, y) if $x < 4$	$\rightarrow$	$(4, y)$	Fill 4-liter jug
2	(x, y) if $y < 3$	$\rightarrow$	$(x, 3)$	Fill 3-liter jug
3	(x, y) if $x > 0$	$\rightarrow$	$(0, y)$	Empty 4-liter jug on the ground
4	(x, y) if $x > 0$	$\rightarrow$	$(x, 0)$	Empty 3-liter jug on the ground
5	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow$	$(4, y - (4 - x))$	Pour water from 3-liter jug into 4-liter jug until 4-liter jug is full
6	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow$	$(x - (3 - y), 3)$	Pour water from 4-liter jug into 3-liter jug until 3-liter jug is full
7	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow$	$(x + y, 0)$	Pour all water from 3-liter jug into 4-liter jug
8	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow$	$(0, x + y)$	Pour all water from 4-liter jug into 3-liter jug

- **3. Conflict resolution strategy**
- Select and fire the first rule in the conflict set that does not lead to a repeated state.
- **4. Interpreter**
- Interpreter is the select-execute loop.

- **Step 2. Application of the rules**
- The various steps in the solution procedure obtained by using the forward chaining method

Step No.	Working memory	Current state	Conflict set	Fire rule	Remarks
0	$\{(0,0),(2,0)\}$	$(0,0)$	–	–	Initial state
1	$\{(0,0),(2,0)\}$	$(0,0)$	1, 2	1	–
2	$\{(4,0),(2,0)\}$	$(4,0)$	2, 3, 4, 6	2	–
3	$\{(4,3),(2,0)\}$	$(4,3)$	3, 4, 5, 6	3	–
4	$\{(0,3),(2,0)\}$	$(0,3)$	1,7	7	–
5	$\{(3,0),(2,0)\}$	$(3,0)$	1, 2, 3, 4, 6, 8	2	–
6	$\{(3,3),(2,0)\}$	$(3,3)$	1, 3, 4, 5, 6	5	–
7	$\{(4,2),(2,0)\}$	$(4,2)$	2, 3, 4, 5, 6	3	–
8	$\{(0,2),(2,0)\}$	$(0,2)$	1, 2, 7	7	–
9	$\{(2,0),(2,0)\}$	$(2,0)$	–	–	Goal state

- Let us examine the working of the conflict resolution strategy in this example. Consider Step No. 6 in Table 2.5. The current state is (3, 3). If we examine the rules, it can be seen that this state satisfies the conditions of the Rules 1, 3, 4, 5, and 6 and hence the conflict set is {Rule 1, Rule 3, Rule 4, Rule 5, Rule 6}. The conflict resolution strategy we have adopted is "select and fire the first rule that does not lead to a repeated state". If we select Rule 1 and fire it, the resulting state is (4, 3) and we have obtained this state in Step No. 3. Similarly if we select Rule 3 and fire it the resulting state is (0, 3) which we have already obtained in Step No. 4. Proceeding like this, we see that that the first rule in the conflict set that does not lead to a repeated state is Rule 5 and we select it and fire it to obtain the non-repeated state (4, 2).

- **3. Conclusion**
- The operations specified in Table specify a method of getting 2 liters of water in the 4-liter jug. However, this need not necessarily be the optimum solution in the sense that there may be a solution with less number of operations.

Blocks World Problem

- It involves a set of blocks placed on a table and a robotic arm that manipulates them to achieve a goal configuration.

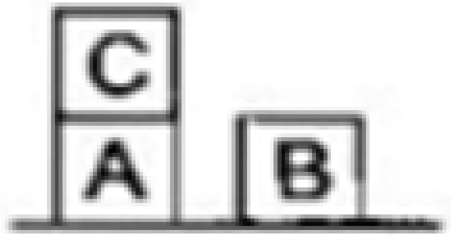
Blocks World Problem

- The world consists of:
- A **set of blocks** (e.g., A, B, C, ...).
- A **table** where blocks can be placed.
- A **robotic arm** that can pick up and place blocks.
- The goal is to transform an **initial state** into a **goal state** by moving the blocks according to certain rules.

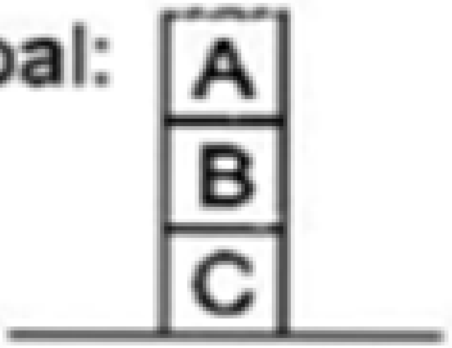
Operators in blocks world

- **Pick Up (X)** – Pick up block X from the table (only if it’s clear).
- **Put Down (X)** – Place block X on the table
- **Stack (X, Y)** – Place block X on top of block Y (only if Y is clear).
- **Unstack (X, Y)** – Remove block X from on top of block Y.

Start:



Goal:



- UNSTACK(C,A)
- PUTDOWN(C)
- PICKUP(B)
- STACK(B,A)
- STACK(C,B)

8 PUZZLE Problem

- It involves a **3×3 grid** with **8 numbered tiles (1-8) and one blank space**, where the goal is to rearrange the tiles into a specified order by sliding tiles into the blank space.
- **State Space:** A **3×3** grid with 8 tiles and 1 blank space.
- **Initial State:** Any random configuration of tiles.
- **Goal State:** A predefined arrangement (usually in ascending order).

Operators:

The **blank space ("_") can move**:

- **Up**
- **Down**
- **Left**
- **Right**

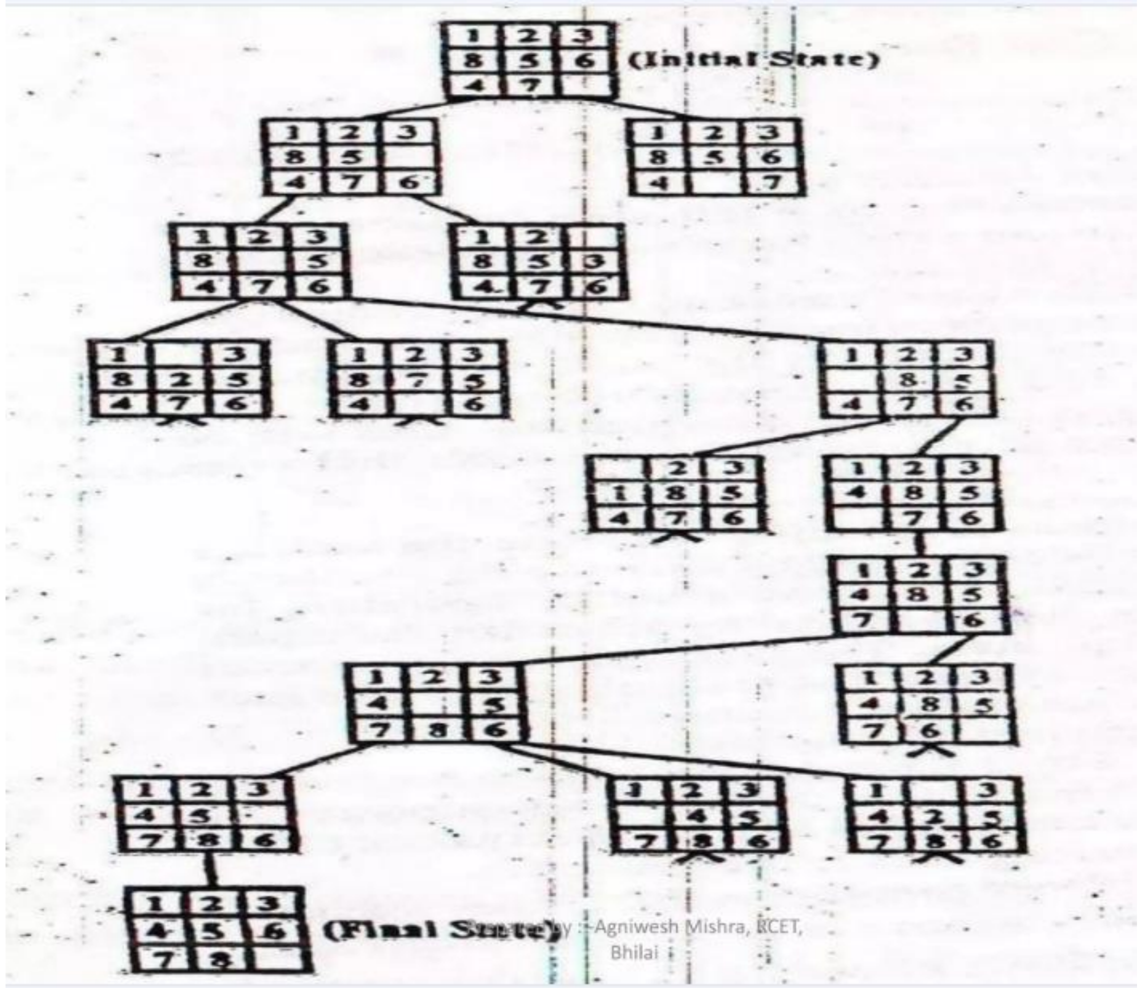
Not all moves are always possible (e.g., if the blank is in the top row, it can't move up).

1	2	3
8	5	6
4	7	

A Start Configuration

1	2	3
4	5	6
7	8	

A Goal Configuration


(Initial State)
(Final State)