

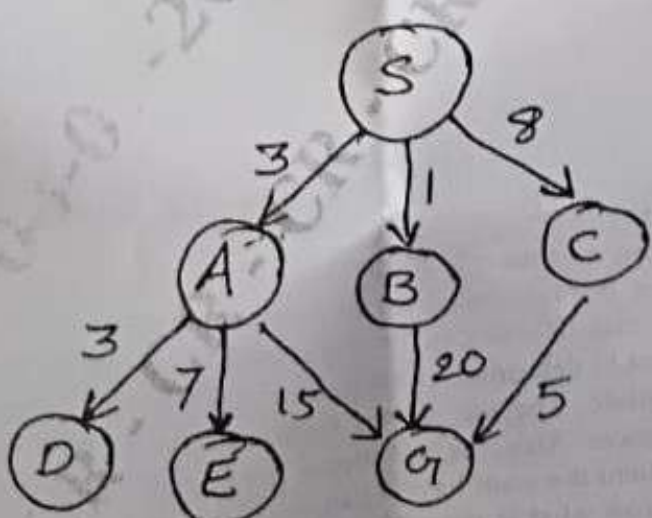
USN 1 C R 2 3 C T O 2 0

Fourth Semester B.E./B.Tech. Degree Examination, June/July 2025  
**Artificial Intelligence**

Time: 3 hrs.

Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.  
 2. M : Marks, L: Bloom's level, C: Course outcomes.

Module - 1				M	L	C
1	a.	What are the four components to define a problem? Define them.		4	L1	CO1
	b.	Compare and contrast human intelligence to artificial intelligence with numerous examples and applications.		7	L4	CO1
	c.	Explain the following: i) PEAS ii) Simple reflex agent iii) Model based agent.		9	L2	CO1
OR				8	L1	CO1
2	a.	What is AI? List out the applications of AI, state the characteristics of AI problem.		6	L4	CO1
	b.	Analyse and generalize what is a rational agent.		6	L2	CO1
	c.	Explain the structure of agents and analyse the characteristics of intelligent agents.				
Module - 2						
3	a.	You are given two jugs, a 5 liters one and a 4 liters one. A pump which has unlimited water which you can use to fill the jug, and the ground on which you get exactly 2 (two) liters of water in the 5 (five) liters of jug? Unit : Apply water Jug problem algorithm.		10	L3	CO2
	b.	Describe Depth First Search (DFS) search algorithm with an example.		10	L2	CO2
OR						
4	a.	Explain Breadth First Search (BFS) algorithm and apply BFS to find the solution for the above graph. Also find the optimum path and cost for the above graph.		10	L3	CO2
 <pre> graph TD     S((S)) -- 3 --&gt; A((A))     S -- 1 --&gt; B((B))     S -- 8 --&gt; C((C))     A -- 3 --&gt; D((D))     A -- 7 --&gt; E((E))     A -- 15 --&gt; G((G))     B -- 20 --&gt; G     C -- 5 --&gt; G           </pre>						
Fig.Q4(a)				10	L2	CO2
b.	Describe the iterative deepening depth first search with an example.					

## Module - 3

5	a.	Compare blind search and heuristic search algorithm in detail.	6	L4	CO3
	b.	Write a note on Wumpus world problem.	6	L2	CO3
	c.	Write the connectives used to form complex sentence of propositional logic. Given example for each.	8	L2	CO3

## OR

6	a.	Describe A* search algorithm with an example.	10	L3	CO3
	b.	Compare proposition logic and predicate logic in detail with example.	4	L4	CO3
	c.	Explain the following concepts with example : i) Heuristic function ii) Atomic sentence iii) Complex sentence.	6	L2	CO3

## Module - 4

7	a.	What are predicates? Explain its syntax and semantics.	5	L2	CO4
	b.	Define universal and existential instantiation and give example for both.	5	L1	CO4
	c.	Consider the following knowledge base : i) Gita likes all kinds of food ii) Mango and chapatti and food iii) Gita eats almond and is still alive iv) Anything eaten by anyone and is still alive is food Goal : Gita likes almond.	10	L3	CO4

## OR

8	a.	Write appropriate quantifiers for the following : i) Some students read well ii) Some students like some books iii) Some students like all books iv) All students like some books v) All students like no books Explain the concept of resolution in first order logic with appropriate procedure.	8	L3	CO4
	b.	Write and explain simple backward - chaining algorithm and forward - chaining algorithm for first - order knowledge bases with example. Also, explain the process of unification.	12	L3	CO4

## Module - 5

9	a.	Explain the impact of uncertainty in probabilistic reasoning.	5	L2	CO5
	b.	Explain Bayes' rule and its utilization in probabilistic reasoning.	5	L2	CO5
	c.	Write the representation of Bayes Theorem. In a class, 70% children were fall sick due to viral fever and 30% due to bacterial fever. The probability of observing temperature for viral is 0.78 and bacterial is 0.31. If a child develops high.	10	L3	CO5

## OR

10	a.	Write short notes on : i) Expert systems ii) Knowledge acquisition.	8	L2	CO5
	b.	Suppose a doctor is trying to find out if a patient is suffering from some type of cancer. If the cancer is only found on average in 2 out of every, 1000 people, the doctor's initial beliefs can be expressed as $P(\text{cancer}) = 0.002$ . There is a laboratory test to determine if the patient has cancer. Unfortunately this test is 100 % accurate. The test comes back positive in 98% of cases where the patient has cancer. Also, the test comes out negative only in 97% of the cases, where the patient does not have a cancer. If the doctor orders a test, and it comes back positive what is the probability that the patient indeed has cancer?	12	L3	CO5

# Fourth Semester B.E/B.Tech. Degree Examination ,June/July 2025

## ARTIFICIAL INTELLIGENCE-BAD402

### Module – 1

**1. a. What are the four components to define a problem? Define them. (4 Marks | L1 | CO1)**

A problem can be defined formally by five components:- (Explained with example Romania)

The initial state that the agent starts in\_ For example, the initial state for our agent in Romania might be described as *In(Arad)*

- A description of the possible actions available to the agent Given a particular state *s*, *ACTIONS(s)* returns the set of actions that can be executed in *s*. We say that each of these actions is applicable in

*s*. For example, from the state *Ir.(Arad)*, the applicable actions are { *Go(Sibiu)*, *Go(Timisoara)*, *Go(Zerind)* }.

- A description of what each action does; the formal name for this is the transition model, specified by a function *RESULT(s, a)* that returns the state that results from doing action *a* in state *s*. We also use the term successor to refer to any state reachable from a given state by a single action.<sup>2</sup> For example, we have

*RESULT(In(Arad), Go(Zerind)) = In(Zerind)* .

- Together, the initial state, actions, and transition model implicitly define the state space of the problem—the set of all states reachable from the initial state by any sequence of actions.
- The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.
- {The map of Romania shown in Figure 3.2 can be interpreted as a state-space graph if we view each road as standing for two driving actions, one in each direction.}
- A path in the state space is a sequence of states connected by a sequence of actions.



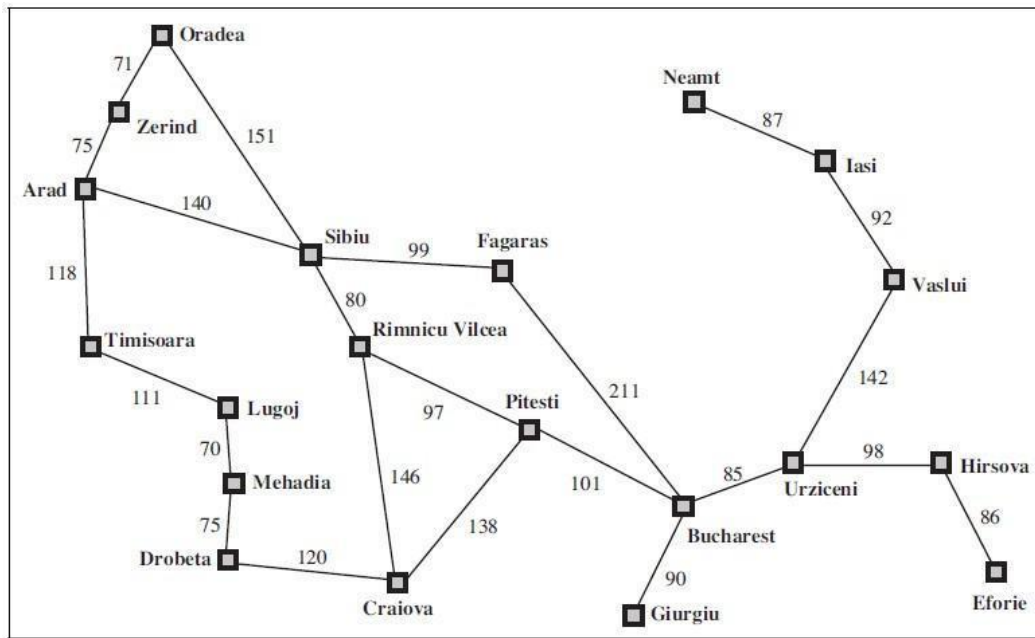


Figure 3.2 A simplified road map of part of Romania.

- The goal test, which determines whether a given state is a goal state. Sometimes there is an explicit set of possible goal states, and the test simply checks whether the given state is one of them. The agent's goal in Romania is the singleton set  $\{In(Bucharest)\}$ .
- path cost function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure. For the agent trying to get to Bucharest, time is of the essence, so the cost of a path might be its length in kilometers.

Assume that the cost of a path can be described as the *sum* of the costs of the individual actions along the path. The step cost of taking action  $a$  in state  $s$  to reach state  $s'$  is denoted by  $e(s, a, s')$ . The step costs for Romania are shown in Figure 3.2 as route distances.

### 1b. Compare and contrast human intelligence to artificial intelligence with numerous examples and applications. (7 Marks | L4 | CO1)

Human Intelligence (HI) and Artificial Intelligence (AI) are both forms of problem-solving capabilities, but they differ in origin, processing, flexibility, and learning.

Aspect	Human Intelligence	Artificial Intelligence
Origin	Natural; evolved over millions of years.	Man-made; developed through programming and machine learning.
Learning ability	Learns from experience, emotions, and abstract thinking.	Learns from data and algorithms; lacks emotional understanding.
Creativity	Highly creative and innovative (e.g., art, music, storytelling).	Limited creativity; can generate content based on training data (e.g., ChatGPT).
Decision-making	Can make intuitive and emotional decisions.	Makes logical, data-driven decisions.
Generalization	Can apply knowledge across domains (e.g., language, emotions, ethics).	Works best within specific domains (e.g., image recognition, translation).
Adaptability	Easily adapts to new, unseen situations.	Needs retraining or reprogramming for new tasks.
Memory	Limited and fallible memory.	High storage and precise retrieval.

#### Examples and Applications:

##### 1. Human Intelligence Examples:

- A teacher understanding and adapting to each student's needs.
- A doctor diagnosing rare diseases based on experience and intuition.

##### 2. Artificial Intelligence Applications:

- Medical Diagnosis: AI systems like IBM Watson help diagnose cancer using data.
- Self-driving Cars: Tesla's AI makes decisions based on sensor data and neural networks.
- Voice Assistants: Siri and Alexa understand and respond to voice commands.
- Recommendation Systems: Netflix or Amazon uses AI to suggest content/products.

**1c.Explain the following:**  
**i)PEAS**  
**ii)Simple reflex agent**  
**iii)Model based agent**  
**(9 Marks | L2 | CO1)**

i) Task environments, which are essentially the “problems” to which rational agents are the “solutions.”

To specify the performance measure, the environment, and the agent’s actuators and sensors called the PEAS (Performance, Environment, Actuators, Sensors) description.

In designing an agent, the first step must always be to specify the task environment as fully as possible.

PEAS description of an automated taxi driver.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

**Figure 2.4** PEAS description of the task environment for an automated taxi.

The performance measure to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

**Figure 2.4** PEAS description of the task environment for an automated taxi.

What is the driving environment that the taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 12-lane freeways. The roads contain other traffic, pedestrians, stray animals, road works, police cars, puddles, and potholes. The taxi must

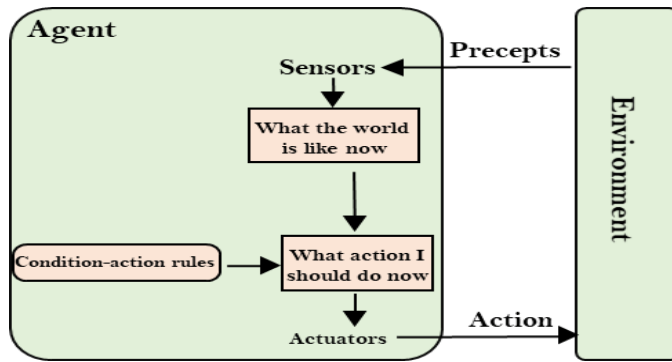
also interact with potential and actual passengers.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

**Figure 2.5** Examples of agent types and their PEAS descriptions.

## ii) Simple reflex agents

- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
  - They have very limited intelligence
  - They do not have knowledge of non-perceptual parts of the current state
  - Mostly too big to generate and to store.
  - Not adaptive to changes in the environment.



```

function REFLEX-VACUUM-AGENT(location, status) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left

```

**Figure 2.8** The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in Figure 2.3.

### iii) Model-based reflex agent

The Model-based agent can work in a partially observable environment, and track the situation.

A model-based agent has two important factors:

- **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
- **Internal State:** It is a representation of the current state based on percept history.

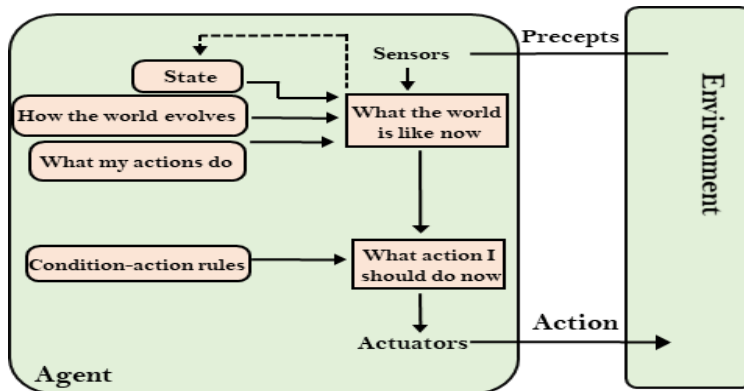
These agents have the model, "which is knowledge of the world" and based on the model they perform actions.

Updating the agent state requires information about:

- How the world evolves



- How the agent's action affects the world.



- For the braking problem, the internal state is not too extensive— just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle

```

function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
               model, a description of how the next state depends on current state and action
               rules, a set of condition-action rules
               action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action

```

**Figure 2.12** A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

go on or off simultaneously.

- For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once. And for any driving to be possible at all, the agent needs to keep track of where its keys are.
- Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program.
- First, we need some information about how the world evolves independently of the agent—for example, that an overtaking car generally will be closer behind than it was a moment ago.
- Second, we need some information about how the agent's own actions affect the world—for example, that when the agent turns the steering wheel clockwise, the car turns to the

right, or that after driving for five minutes northbound on the freeway, one is usually about five miles north of where one was five minutes ago.

- This knowledge about “how the world works”—whether implemented in simple Boolean circuits or in complete scientific theories—is called a model of the world. An agent that uses such a model is called a model-based agent.

## OR

2a. What is AI? List out the applications of AI, state the characteristics of AI problem. (8 Marks | L1 | CO1)

1.Views of AI fall into four categories:

- . Thinking humanly
- . Thinking rationally
- . Acting humanly
- . Acting rationally

<p>Thinking Humanly</p> <p>“The exciting new effort to make computers think... <i>machines with minds</i>, in the full and literal sense.” (Haugeland, 1985)</p> <p>“[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ...” (Bellman, 1978)</p>	<p>Thinking Rationally</p> <p>“The study of mental faculties through the use of computational models.” (Charniak and McDermott, 1985)</p> <p>“The study of the computations that make it possible to perceive, reason, and act.” (Winston, 1992)</p>
<p>Acting Humanly</p> <p>“The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil, 1990)</p> <p>“The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight, 1991)</p>	<p>Acting Rationally</p> <p>“Computational Intelligence is the study of the design of intelligent agents.” (Poole <i>et al.</i>, 1998)</p> <p>“AI . . . is concerned with intelligent behavior in artifacts.” (Nilsson, 1998)</p>
Figure 1.1 Some definitions of artificial intelligence, organized into four categories.	

## 2.Applications of AI:

AI is widely used across various domains. Some common applications include:

1. Healthcare: AI helps in disease diagnosis, robotic surgeries, and drug discovery (e.g., IBM Watson).
2. Finance: Fraud detection, stock market prediction, and algorithmic trading.
3. Retail: Chatbots, recommendation systems (like Amazon or Flipkart).
4. Transportation: Self-driving cars, traffic prediction, and route optimization.
5. Manufacturing: Predictive maintenance, automation using robots.
6. Agriculture: Crop monitoring using drones, yield prediction.
7. Education: Intelligent tutoring systems, personalized learning.
8. Security & Surveillance: Face recognition, behavior prediction.
9. Gaming: Game AI that can compete with or assist players (e.g., AlphaGo).
10. Smart Assistants: Siri, Alexa, Google Assistant use AI for natural language understanding.

## 3. Characteristics of AI Problems

AI problems differ from traditional computational problems. The key characteristics of AI problems are:

1. Perception and Sensing:
  - AI systems need to interpret complex sensory inputs like images, speech, and gestures.
  - Ex: Autonomous cars must analyze road scenes using cameras and sensors.
2. Incomplete and Uncertain Information:
  - AI often operates in incomplete, noisy, or dynamic environments.
  - Ex: Speech recognition in noisy rooms or planning with missing data.
3. Heuristic Search:
  - Many AI problems are solved using heuristics or "rules of thumb" instead of exact algorithms.
  - Ex: Game playing agents like chess use heuristics for move evaluation.

4. Knowledge Representation:

- AI must represent knowledge in a structured form (facts, logic, graphs).
- Ex: Expert systems use knowledge bases to simulate reasoning.

5. Reasoning and Decision Making:

- AI systems must make logical decisions even with limited information.
- Ex: Medical diagnosis systems suggest treatments based on symptoms.

6. Learning from Experience:

- Many AI systems improve over time using machine learning.
- Ex: A recommendation engine refines suggestions based on user feedback.

7. Goal-Oriented Behavior:

- AI agents are designed to achieve goals in an environment.
- Ex: A robot vacuum cleans rooms autonomously based on a goal.

8. Dealing with Natural Language:

- AI systems must understand and generate human language.
- Ex: Translation systems, chatbots, sentiment analysis.

**2b. Analyze and generalize what is a rational agent.**  
**(6 Marks | L4 | CO1)**

A rational agent is an intelligent system that always tries to do the right action to achieve the best possible outcome, based on what it knows and what it observes from the environment. It does not need to be perfect, but it should make the best decision using the information available at that time.

□Definition:

A rational agent is one that, for every situation (called a percept sequence), selects an action that is expected to maximize its performance, based on:

- What it has observed (percepts)
- What it already knows about the environment
- What actions it can perform

□PerformanceMeasure:

This tells the agent what is considered "good" or "successful."

For example, in a vacuum cleaner agent, a good performance measure could be "how many squares are clean over time" rather than just "how many times dirt was sucked."

□Omniscient:

A rational agent doesn't know the future or see everything (not omniscient). It acts based on what it knows now, not what will happen later. For example, if a person crosses a road and gets hit by falling debris, it doesn't mean the decision was irrational—it was based on the information available.

□GathersInformation:

A rational agent will take actions just to get more information when needed. Example: Before crossing the road, a person looks both ways to ensure it's safe. That "looking" is a rational step.

□Learns and Adapts:

A good rational agent can learn from its past experiences and improve its future actions. Example: A vacuum cleaner learns where dirt usually appears and cleans those areas first.

□Autonomy:

A rational agent should not only rely on its designer's instructions. It should learn from its environment and act independently. More autonomy means better performance in new or unknown situations.

Example: Vacuum Cleaner Agent

Let's say we have a vacuum-cleaning robot that:

- Gets 1 point for every square that stays clean
- Can move left, right, and clean

If the robot knows where dirt is, and it cleans efficiently, it is rational. But if the robot keeps moving back and forth after cleaning everything—just wasting energy—it becomes irrational if movement costs points.

A truly rational vacuum cleaner would stop moving once everything is clean, or check occasionally if new dirt appears.

A rational agent is one that acts wisely to get the best possible result, based on:

- Its observations
- Its knowledge
- Its abilities



- And the goal it has to achieve (performance measure)

It learns, adapts, gathers useful information, and works independently to perform well in different environments. Rationality is about making smart choices, not being perfect.

**2c. Explain the structure of agents and analyze characteristics of intelligent agents. (6Marks | L2 | CO1)**

1. The structure of agents

Agent = Architecture + Program

- AI Job: design an agent program implementing the agent function
- The agent program runs on some computing device with physical sensors and actuators: the agent architecture
- All agents have the same skeleton:
  - Input: current percepts
  - Output: action
  - Program: manipulates input to produce output.
- The agent function takes the entire percept history as input
- The agent program takes only the current percept as input.
- if the actions need to depend on the entire percept sequence, the agent will have to remember the percepts

The Table-Driven Agent

The table represents explicitly the agent function Ex: the simple vacuum cleaner

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
               table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

**Figure 2.7** The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

CHARACTERISTICS of intelligent agents:

Omniscience vs Rationality:

- Omniscience means knowing the actual outcome of all actions in advance.
- In reality, no agent can be omniscient, because it cannot predict the future with 100% certainty.
- Instead, rational agents must act based on available information (percept history) and make decisions that maximize expected performance.
- Example: A human crossing a street may get hit by a falling object unexpectedly. That doesn't mean the person was irrational—just not omniscient.

2. Learning:

- A rational agent must learn from experience to improve its behavior.
- It starts with some built-in or prior knowledge, but as it gathers percepts over time, it should update its internal model.
- Agents that do not learn may repeat failed behaviors. Example: Insects like the sphex wasp and dung beetle fail to adapt when their plan is disrupted—they act without learning.
- Learning enhances adaptability, making the agent more intelligent over time.

### 3. Autonomy:

- An agent is autonomous if it relies more on its own experiences rather than pre-programmed knowledge.
- Agents should learn and adapt in new or changing environments rather than just following fixed rules.
- Example: A vacuum cleaner that learns where dirt usually appears will clean more effectively than one that follows a fixed path.
- True autonomy develops over time: an agent may need some initial guidance but should become increasingly self-reliant.

## MODULE 2

**3. a. You are given two jugs, 5 liters one and 4 liters one. A pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 (two) liters of water in the 5 (five) liters of jug?**

**Unit: Apply water jug problem algorithm.**

**(10 Marks | L3 | CO2)**

The water Jug Problem, as the name suggests, is a problem where two jugs of water are given, say one is a 4-litre one, and the other one is a 3-litre one, but none of the measuring markers is mentioned on any of it. There is a pump available to fill the jugs with water. How can you exactly pour 2 litres of water into a 4-litre jug? Assuming that both the jugs are empty, the task is to find a solution to pour 2-litre water into a 4-litre jug. Production

**Rules for the Water Jug Problem** To solve the water jug problem, many algorithms can be used. These include:

- **Breadth-First Search:** BFS or Breadth First Search visits the nodes in order of their distance from the starting node. This implies that it will visit the nearest node first.
- **Depth First Search:** DFS or Depth First Search visits the nodes in order of their depth.

In production rules for the water jug problem, let x denote a 4-litre jug, and y denote a 3-litre jug, i.e.  $x=0,1,2,3,4$  or  $y=0,1,2,3$

Start state (0,0)

Goal state (2,n) from any n

Start from the start state and end up at the goal state. Production rules for the water jug problem in AI are as follows:

1.	$(x,y) \text{ is } X < 4 \rightarrow (4, Y)$	Fill the 4-litre jug
2.	$(x, y) \text{ if } Y < 3 \rightarrow (x, 3)$	Fill the 3-litre jug

3.	$(x, y) \text{ if } x > 0 \rightarrow (x-d, d)$	Pour some water from a 4-litre jug
4.	$(x, y) \text{ if } Y > 0 \rightarrow (d, y-d)$	Pour some water from a 3-litre jug
5.	$(x, y) \text{ if } x > 0 \rightarrow (0, y)$	Empty 4-litre jug on the ground
6.	$(x, y) \text{ if } y > 0 \rightarrow (x, 0)$	Empty 3-litre jug on the ground
7.	$(x, y) \text{ if } X+Y \geq 4 \text{ and } y > 0 \rightarrow (4, y-(4-x))$	Pour water from a 3-litre jug into a 4-litre jug until it is full
8.	$(x, y) \text{ if } X+Y \geq 3 \text{ and } x > 0 \rightarrow (x-(3-y), 3)$	Pour water from a 3-litre jug into a 4-litre jug until it is full



9.	$(x, y)$ if $X+Y \leq 4$ and $y > 0 \rightarrow (x+y, 0)$	Pour all the water from a 3-litre jug into a 4-litre jug
10.	$(x, y)$ if $X+Y \leq 3$ and $x > 0 \rightarrow (0, x+y)$	Pour all the water from a 4-litre jug into a 3-litre jug
11.	$(0, 2) \rightarrow (2, 0)$	Pour 2-litre water from 3-litre jug into 4-litre jug
12.	$(2, Y) \rightarrow (0, y)$	Empty 2-litre in the 4-litre jug on the ground.

### Problem Representation:

- Jug A (5-liter jug)
- Jug B (4-liter jug)
- Allowed operations:
  1. Fill any jug completely

2. Empty any jug
3. Pour water from one jug to another until one is full or the other is empty

We want exactly 2 liters in Jug A (5-liter jug).

Initial State:

- $(0, 0) \rightarrow$  Jug A = 0L, Jug B = 0L

Goal State:

- $(2, \_)$   $\rightarrow$  Jug A = 2L, Jug B can have any amount

Steps Using Algorithm (BFS/DFS-style approach):

We use state representation as (Jug A, Jug B).

Step	1:	Fill	4L	Jug
$(0, 0) \rightarrow (0, 4)$				
Step	2:	Pour	4L Jug into 5L Jug	
$(0, 4) \rightarrow (4, 0)$				
Step	3:	Fill	4L Jug again	
$(4, 0) \rightarrow (4, 4)$				
Step	4:	Pour 4L Jug into 5L Jug (only 1L fits, rest remains)		
$(4, 4) \rightarrow (5, 3)$				
Step	5:	Empty	5L Jug	
$(5, 3) \rightarrow (0, 3)$				
Step	6:	Pour	4L Jug into 5L Jug	
$(0, 3) \rightarrow (3, 0)$				
Step	7:	Fill	4L Jug again	
$(3, 0) \rightarrow (3, 4)$				

Step 8: Pour 4L Jug into 5L Jug (only 2L fits)  
 $(3, 4) \rightarrow (5, 2)$

Step 9: Empty 5L Jug  
 $(5, 2) \rightarrow (0, 2)$

Step 10: Pour 4L Jug into 5L Jug  
 $(0, 2) \rightarrow (2, 0)$

Final Answer (Goal Reached):

- $(2, 0)$ : Now the 5-liter jug has exactly 2 liters of water.

Water Jug Algorithm Concept Explanation:

The Water Jug Problem is a classic AI problem used to demonstrate:

- State space representation
- Search algorithms (DFS/BFS)
- Problem-solving under constraints
- Goal-based agent behavior

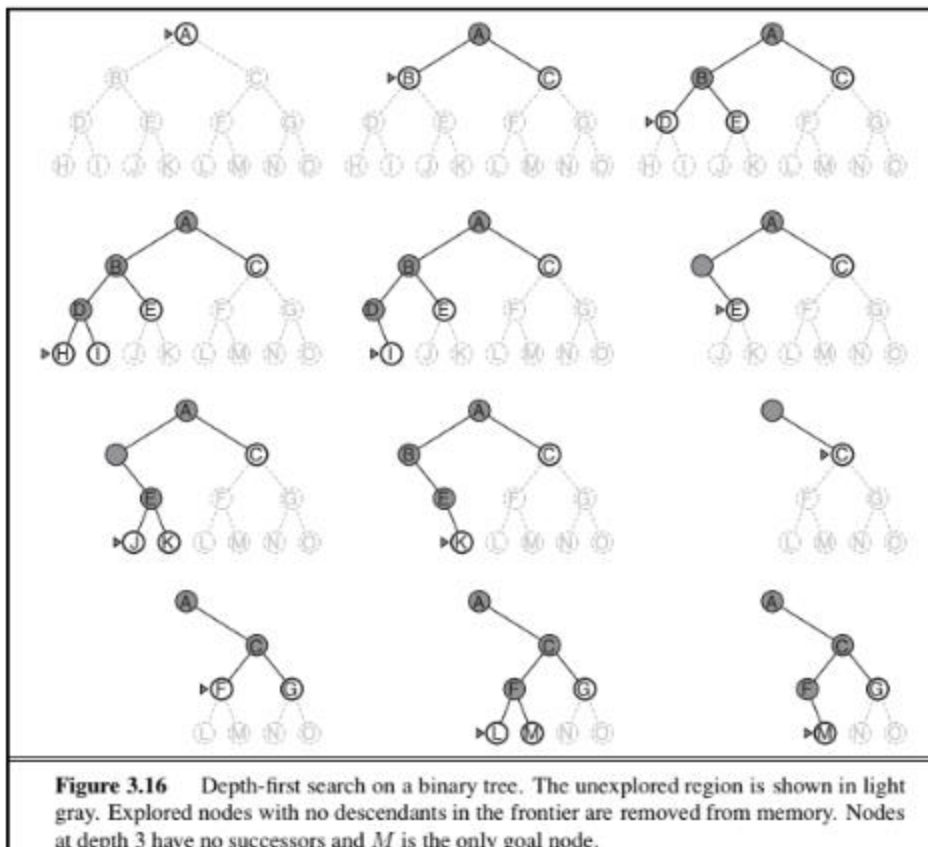
**b. Describe Depth First Search (DFS) search algorithm with an example.**

**(10 Marks | L2 | CO2)**

Depth-first search always expands the deepest node in the current frontier of the search tree. The progress of the search is illustrated in Figure 3.16. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors. The depth-first search algorithm is an instance of the graph-search algorithm in Figure 3.7; whereas breadth-first-search uses a FIFO queue, depth-first search uses a LIFO queue. A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn, was the deepest unexpanded node when it was selected. As an alternative to the GRAPH-SEARCH-style implementation, it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn. (A recursive depth-first algorithm incorporating a depth limit is shown in Figure 3.17.) The properties of depth-first

search depend strongly on whether the graph-search or tree-search version is used. The graph-search version, which avoids repeated states and redundant paths, is complete in finite state spaces because it will eventually expand every node. The tree-search version, on the other hand, is not complete—for example, in Figure 3.6 the algorithm will follow the Arad–Sibiu–Arad–Sibiu loop forever. Depth-first tree search can be modified at no extra memory cost so that it checks new states against those on the path from the root to the current node; this avoids infinite loops in finite state spaces but does not avoid the proliferation of redundant paths. In infinite state spaces, both versions fail if an infinite non-goal path is encountered. For example, in Knuth’s 4 problem, depth-first search would keep applying the factorial operator forever. For similar reasons, both versions are nonoptimal. For example, in Figure 3.16, depth-first search will explore the entire left subtree even if node C is a goal node. If node J were also a goal node, then depth-first search would return it as a solution instead of C, which would be a better solution; hence, depth-first search is not optimal.

The properties of depth-first search depend strongly on whether the graph-search or tree-search version is used. The graph-search version, which avoids repeated states and redundant paths, is complete in finite state spaces because it will eventually expand every node. The tree-search version, on the other hand, is not complete—for example, in Figure 3.6 the algorithm will follow the Arad–Sibiu–Arad–Sibiu loop forever. Depth-first tree search can be modified at no extra memory cost so that it checks new states against those on the path from the root to the current node; this avoids infinite loops in finite state spaces but does not avoid the proliferation of redundant paths. In infinite state spaces, both versions fail if an infinite non-goal path is encountered. For example, in Knuth’s 4 problem, depth-first search would keep applying the factorial operator forever. For similar reasons, both versions are nonoptimal. For example, in Figure 3.16, depth-first search will explore the entire left subtree even if node C is a goal node. If node J were also a goal node, then depth-first search would return it as a solution instead of C, which would be a better solution; hence, depth-first search is not optimal.



The time complexity of depth-first graph search is bounded by the size of the state space (which may be infinite, of course). A depth-first tree search, on the other hand, may generate all of the  $O(bm)$  nodes in the search tree, where  $m$  is the maximum depth of any node; this can be much greater than the size of the state space. Note that  $m$  itself can be much larger than  $d$  (the depth of the shallowest solution) and is infinite if the tree is unbounded. So far, depth-first search seems to have no clear advantage over breadth-first search, so why do we include it? The reason is the space complexity. For a graph search, there is no advantage, but a depth-first tree search needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path. Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored. (See Figure 3.16.) For a state space with branching factor  $b$  and maximum depth  $m$ , depth-first search requires storage of only  $O(bm)$  nodes. Using the same assumptions as for Figure 3.13 and assuming that nodes at the same depth as the goal node have no successors, we find that depth-first search would require 156 kilobytes instead of 10 exabytes at depth  $d = 16$ , a factor of 7 trillion times less space. This has led to the adoption of depth-first tree search as the basic workhorse of many areas of AI, including constraint satisfaction (Chapter 6), propositional satisfiability (Chapter 7), and logic programming (Chapter 9). For the remainder of this section, we focus primarily on the tree-search version of depth-first search. A variant of depth-first search called backtracking search uses still less memory. In backtracking, only one successor is generated at a time rather than all



successors; each partially expanded node remembers which successor to generate next. In this way, only  $O(m)$  memory is needed rather than  $O(bm)$ . Backtracking search facilitates yet another memory-saving (and time-saving) trick: the idea of generating a successor by modifying the current state description directly rather than copying it first. This reduces the memory requirements to just one state description and  $O(m)$  actions. For this to work, we must be able to undo each modification when we go back to generate the next successor. For problems with large state descriptions, such as robotic assembly, these techniques are critical to success.

**OR**

**4. a. Explain Breadth First Search (BFS) algorithm and apply BFS to find the solution for the above graph. Also find the optimum path and cost for the above graph.**

**(10 Marks | L3 | CO2)**

**(Fig Q4(a): Graph with nodes S, A, B, C, D, E, G and edges with weights)**

Breadth-First Search (BFS) is a search strategy that explores all nodes at the current depth before moving to the next level, using a First-In-First-Out (FIFO) queue to manage the frontier. This ensures that the shallowest unexpanded nodes are expanded first. One important feature of BFS is that the goal test is applied when nodes are generated rather than when they are expanded, allowing the algorithm to find a solution more efficiently by stopping as soon as the shallowest goal is found. BFS is complete, meaning it is guaranteed to find a solution if one exists at a finite depth and the branching factor is finite. It is also optimal when all actions have the same cost or when path cost increases with depth, as it always returns the shallowest (and hence least costly) solution. However, the major drawbacks of BFS lie in its time and space complexity. In the worst case, it generates  $O(b^d)$  nodes, where  $b$  is the branching factor and  $d$  is the depth of the shallowest solution. Its space complexity is also  $O(b^d)$ , as it stores all generated nodes in memory. This exponential growth makes BFS impractical for large-depth problems, where memory becomes a greater constraint than time, emphasizing the need for more memory-efficient strategies.

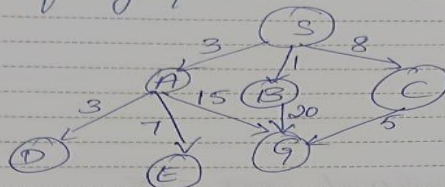
```

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier  $\leftarrow$  a FIFO queue with node as the only element
  explored  $\leftarrow$  an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier  $\leftarrow$  INSERT(child, frontier)

```

Figure 3.11 Breadth-first search on a graph.

Q1a Find solution for graph. Also find optimal path by cost for graph.



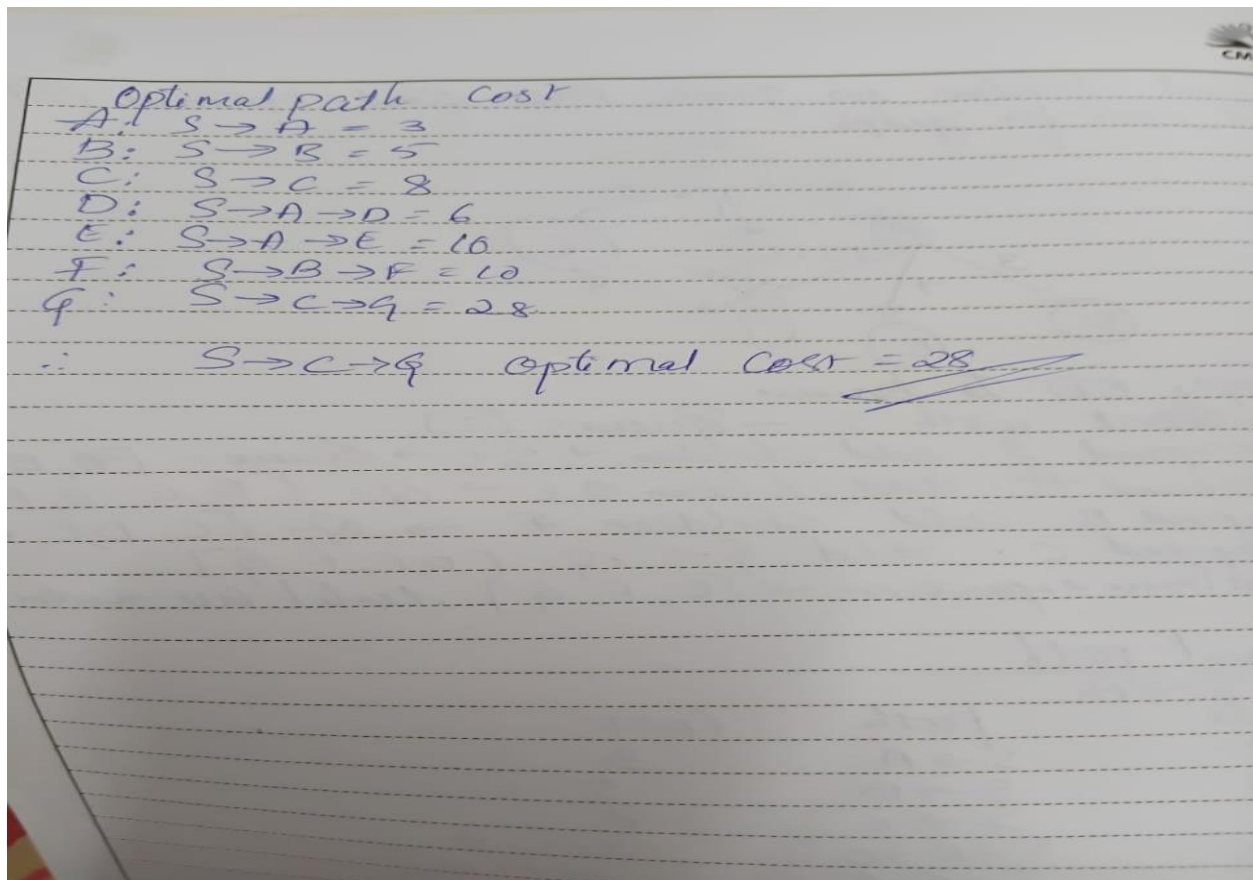
Soln

- Apply BFS to traverse
- start with S → Queue = [S]
- expand S: add children A, B, C → Queue = [A, B, C]
- expand A: add children D, E → Q = [B, C, D, E]
- expand B: add children F → Q = [C, D, E, F]
- expand C: add G → Q = [D, E, F, G]
- Continue expansion → [E, F, G], until all nodes visited

optimal path

Node	path	cost
A	S → A	3
B	S → B	1
C	S → C	8
D	S → A → D	3 + 3 = 6
E	S → A → E	3 + 1 = 4
F	S → B → F	1 + 5 = 6
G	S → C → G	8 + 5 = 13

Traversal order: S → A → B → C → D → E → F → G



**b. Describe the iterative deepening depth first search with an example.**  
**(10 Marks | L2 | CO2)**

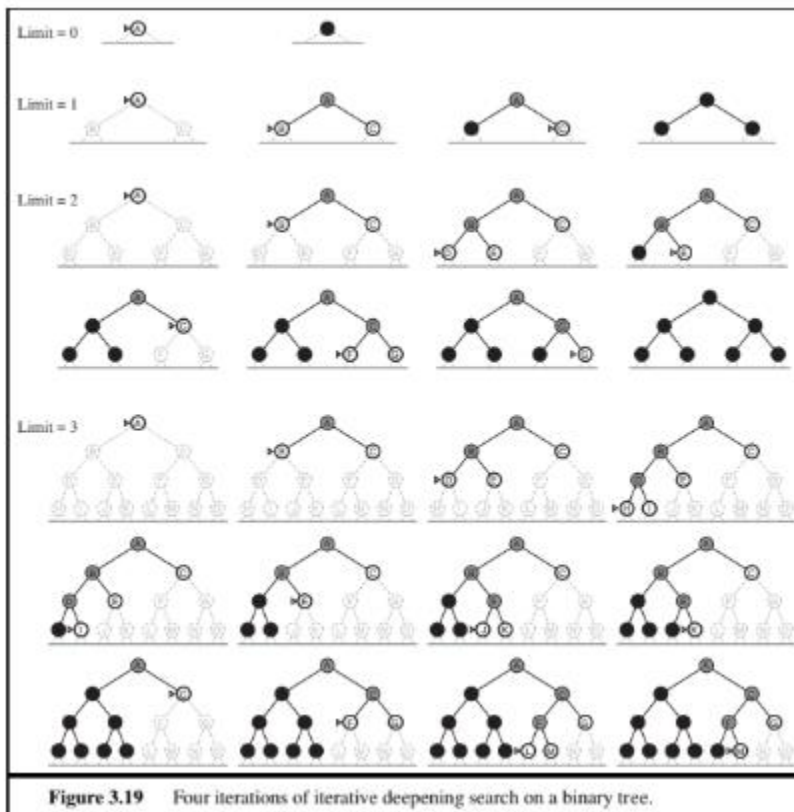
IDS is a general strategy often used in combination with depth-first tree search, that finds the best depth limit. It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found. This will occur when the depth limit reaches  $d$ , the depth of the shallowest goal node. The algorithm is shown in Figure 3.18. Iterative deepening combines the benefits of depth-first and breadth-first search. Like depth-first search, its memory requirements are modest:  $O(bd)$  to be precise. Like breadth-first search, it is complete when the branching factor is finite and optimal when the path cost is a nondecreasing function of the depth of the node. Figure 3.19 shows four iterations of ITERATIVE-DEEPENING-SEARCH on a binary search tree, where the solution is found on the fourth iteration. Iterative deepening search may seem wasteful because states are generated multiple times. It turns out this is not too costly. The reason is that in a search tree with the same (or nearly the same) branching factor at each level, most of the nodes are in the bottom level, so it does not matter much that the upper levels are generated multiple times. In an iterative deepening search, the nodes on the bottom level (depth  $d$ ) are generated once, those on the

```

function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result

```

**Figure 3.18** The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.



next-to-bottom level are generated twice, and so on, up to the children of the root, which are generated  $d$  times. So the total number of nodes generated in the worst case is

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \cdots + (1)b^d,$$

which gives a time complexity of  $O(b^d)$ —asymptotically the same as breadth-first search.

There is some extra cost for generating the upper levels multiple times, but it is not large. For

example, if  $b = 10$  and  $d = 5$ , the numbers are

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$

$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110.$$

If you are really concerned about repeating the repetition, you can use a hybrid approach that runs breadth-first search until almost all the available memory is consumed, and then runs iterative deepening from all the nodes in the frontier. In general, iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known. Iterative deepening search is analogous to breadth-first search in that it explores a complete layer of new nodes at each iteration before going on to the next layer. It would seem worthwhile to develop an iterative analog to uniform-cost search, inheriting the latter algorithm's optimality guarantees while avoiding its memory requirements. The idea is to use increasing path-cost limits instead of increasing depth limits. The resulting algorithm, called iterative lengthening search, is explored in Exercise 3.17. It turns out, unfortunately, that iterative lengthening incurs substantial overhead compared to uniform-cost search.

## Module – 3

### 5. a. Compare blind search and heuristic search algorithm in detail. (6 Marks | L4 | CO3)

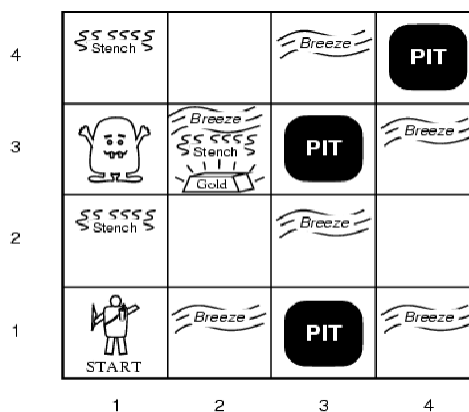
Feature	Blind Search	Heuristic Search
Definition	Blind (or uninformed) search strategies have no additional information about states beyond the problem definition.	Heuristic (or informed) search uses domain-specific knowledge to guide the search process.
Knowledge Used	Only uses the information in the problem definition (initial state, actions, goal test).	Uses a heuristic function ( $h(n)$ ) that estimates the cost from a node to the goal.



Examples	Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS).	Greedy Best-First Search, A* Search.
Efficiency	Less efficient — may explore irrelevant or longer paths.	More efficient — guides the search toward goal-relevant paths.
Completeness & Optimality	Some blind searches are complete and optimal (e.g., BFS), but may be slow.	Heuristic search may not always be optimal (Greedy), but A* is optimal with admissible heuristics.
Use Case	Suitable for small or simple state spaces.	Suitable for large or complex problems where guidance is crucial.

**5.b. Write a note on Wumpus world problem.**  
**(6 Marks | L2 | CO3)**

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow.



- The agent explores a cave consisting of rooms connected by passageways.

- Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room.
- Some rooms contain bottomless pits that trap any agent that wanders into the room.
- Occasionally, there is a heap of gold in a room.
- The goal is to collect the gold and exit the world without being eaten.

#### PEAS description of Wumpus world:

##### Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

##### Environment:

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

##### Actions/Actuators:

- The agent can move *Forward*, *TurnLeft* by 90°, or *TurnRight* by 90°.
- The agent dies a miserable death if it enters a square containing a pit or a live wumpus.
- If an agent tries to move forward and bumps into a wall, then the agent does not move.
- The action *Grab* can be used to pick up the gold if it is in the same square as the agent.
- The action *Shoot* can be used to fire an arrow in a straight line in the direction the agent is facing.
- The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first *Shoot* action has any effect.
- Finally, the action *Climb* can be used to climb out of the cave, but only from square [1,1].

##### Sensors:

The agent has five sensors, each of which gives a single bit of information:

- – In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a *Stench*.
- – In the squares directly adjacent to a pit, the agent will perceive a *Breeze*.
- – In the square where the gold is, the agent will perceive a *Glitter*.
- – When an agent walks into a wall, it will perceive a *Bump*.
- – When the wumpus is killed, it emits a woeful *Scream* that can be perceived anywhere in the cave.

- The percepts will be given to the agent program in the form of a list of five symbols;  
For example: if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get  
[Stench, Breeze, None, None, None].

The Wumpus agent's first step

The first step taken by the agent in the wumpus world.

- (a) The initial situation, after percept [None, None, None, None, None].  
(b) After one move, with percept [None, Breeze, None, None, None].

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	2,1	3,1	4,1
OK	OK		

**A** = Agent  
**B** = Breeze  
**G** = Glitter, Gold  
**OK** = Safe square  
**P** = Pit  
**S** = Stench  
**V** = Visited  
**W** = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK	P?		
1,1	2,1	3,1	4,1
V	V	P?	
OK	OK		

- Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?
- Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.
- At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].
- At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Two later stages in the progress of the agent.

- (a) After the third move, with percept [Stench, None, None, None, None]  
(b) After the fifth move, with percept [Stench, Breeze, Glitter, None, None].

- The agent perceives a stench in [1,2], resulting in the state of knowledge. The stench in [1,2] means that there must be a wumpus nearby. But the wumpus cannot be in

[1,1], by the rules of the game, and it cannot be in [2,2] (or the agent would have detected a stench when it was in [2,1]). Therefore, the agent can infer that the wumpus is in [1,3]. The notation W! indicates this inference. The lack of a breeze in [1,2] implies that there is no pit in [2,2].

- The agent has now proved to itself that there is neither a pit nor a wumpus in [2,2], so it is OK to move there. assume that the agent turns and moves to [2,3]. In [2,3], the agent detects a glitter, so it should grab the gold and then return home.

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 <b>A</b> S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

**A** = Agent  
**B** = Breeze  
**G** = Glitter, Gold  
**OK** = Safe square  
**P** = Pit  
**S** = Stench  
**V** = Visited  
**W** = Wumpus

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 <b>A</b> S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(a)
(b)

### 5.c. Write the connectives used to form complex sentence of propositional logic.

Given example for each.

(8 Marks | L2 | CO3)

Syntax

- o The syntax of propositional logic defines the allowable sentences.
- o The atomic sentences consist of a single proposition symbol.
- o Each such symbol stands for a proposition that can be true or false. Use symbols that start with an uppercase letter and may contain other letters or subscripts, for example:  $P$ ,  $Q$ ,  $R$ ,  $W_{1,3}$  and  $North$ .
- o Complex sentences are constructed from simpler sentences, using parentheses and logical connectives.
- o There are five connectives in common use:
  - $\neg$  (not). A sentence such as  $\neg W_{1,3}$  is called the negation of  $W_{1,3}$ . A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal).
  - $\wedge$  (and). A sentence whose main connective is  $\wedge$ , such as  $W_{1,3} \wedge P_{3,1}$ , is called a conjunction.
  - $\vee$  (or). A sentence using  $\vee$ , such as  $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$ , is a disjunction of the disjunction

( $W_{1,3} \wedge P_{3,1}$ ) and  $W_{2,2}$ .

- $\Rightarrow$  (implies). A sentence such as  $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$  is called an implication. Implications are also known as rules or if–then statements. The implication symbol is sometimes written in other books as  $\supset$  or  $\rightarrow$ .
- $\Leftrightarrow$  (if and only if). The sentence  $W_{1,3} \Leftrightarrow \neg W_{2,2}$  is a biconditional. Some other books write this as  $\equiv$ .

<i>Sentence</i>	$\rightarrow$	<i>AtomicSentence</i>   <i>ComplexSentence</i>
<i>AtomicSentence</i>	$\rightarrow$	<i>True</i>   <i>False</i>   <i>P</i>   <i>Q</i>   <i>R</i>   ...
<i>ComplexSentence</i>	$\rightarrow$	( <i>Sentence</i> )   [ <i>Sentence</i> ]
		$\neg$ <i>Sentence</i>
		<i>Sentence</i> $\wedge$ <i>Sentence</i>
		<i>Sentence</i> $\vee$ <i>Sentence</i>
		<i>Sentence</i> $\Rightarrow$ <i>Sentence</i>
		<i>Sentence</i> $\Leftrightarrow$ <i>Sentence</i>
<b>OPERATOR PRECEDENCE</b> : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$		

**Figure 7.7** A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

## Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- In propositional logic, a model simply fixes the **truth value**—*true* or *false*—for every proposition symbol.

For example,

If the sentences in the knowledge base make use of the proposition symbols  $P_{1,2}$ ,  $P_{2,2}$ , and  $P_{3,1}$ , then one possible model is

$$m_1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}.$$

The semantics for propositional logic must specify how to compute the truth value of any sentence, given a model.

Atomic sentences are easy:

- True is true in every model and False is false in every model.
- The truth value of every other proposition symbol must be specified directly in the model.

For example, in the model  $m_1$  given earlier,  $P_{1,2}$  is false.

For complex sentences, we have five rules, which hold for any subsentences  $P$  and  $Q$  in any model  $m$  (here “iff” means “if and only if”):

- $\neg P$  is true iff  $P$  is false in  $m$ .
- $P \wedge Q$  is true iff both  $P$  and  $Q$  are true in  $m$ .
- $P \vee Q$  is true iff either  $P$  or  $Q$  is true in  $m$ .
- $P \Rightarrow Q$  is true unless  $P$  is true and  $Q$  is false in  $m$ .
- $P \Leftrightarrow Q$  is true iff  $P$  and  $Q$  are both true or both false in  $m$ .

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

**Figure 7.8** Truth tables for the five logical connectives. To use the table to compute, for example, the value of  $P \vee Q$  when  $P$  is true and  $Q$  is false, first look on the left for the row where  $P$  is true and  $Q$  is false (the third row). Then look in that row under the  $P \vee Q$  column to see the result: *true*.

## OR

### 6. a. Describe A\* search algorithm with an example.

(10 Marks | L3 | CO3)

The most widely known form of best-first search is called A\* search (pronounced “A-star search”). It evaluates nodes by combining  $g(n)$ , the cost to reach the node, and  $h(n)$ , the cost to get from the node to the goal:

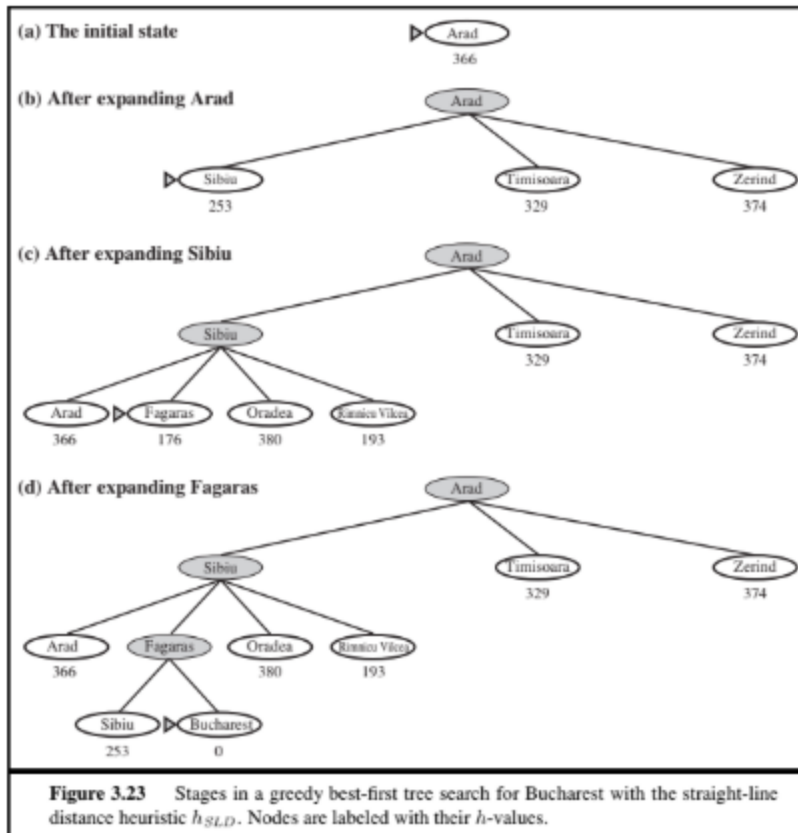
$$f(n) = g(n) + h(n).$$

Since  $g(n)$  gives the path cost from the start node to node  $n$ , and  $h(n)$  is the estimated cost of the cheapest path from  $n$  to the goal, we have

$$f(n) = \text{estimated cost of the cheapest solution through } n.$$

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of  $g(n) + h(n)$ . It turns out that this strategy is more than just reasonable: provided that the heuristic function  $h(n)$  satisfies certain conditions, A\* search is both complete and optimal.

Example:



To ensure optimality in A\* search, two key conditions on the heuristic function  $h(n)$  are required: admissibility and consistency.

An admissible heuristic is one that never overestimates the true cost to reach the goal. This means that for every node  $n$ , the heuristic estimate  $h(n)$  is always less than or equal to the actual minimum cost from  $n$  to the goal. Admissible heuristics are optimistic and ensure that A\* will find an optimal solution. A classic example is the straight-line distance (hSLD) in the Romania map problem, which always underestimates or equals the true path cost since it represents the shortest possible (Euclidean) route.

A consistent heuristic (also called monotonic) satisfies a stronger condition: for every node  $n$  and its successor  $n'$ , the estimated cost from  $n$  should be no greater than the cost of reaching  $n'$  plus the estimated cost from  $n'$  to the goal, i.e.,

$$h(n) \leq c(n, a, n') + h(n').$$

This follows the triangle inequality, ensuring that the estimated cost along a path doesn't decrease unexpectedly. All consistent heuristics are admissible, but not all admissible heuristics are consistent. However, in practice, most commonly used admissible heuristics (like hSLD) are also consistent, making them suitable for graph-based A\* searches.

A\* search is optimal when it uses a good heuristic. Specifically:

- In tree search, A\* is optimal if the heuristic  $h(n)$  is admissible, i.e., it never overestimates the cost to reach the goal.
- In graph search, A\* is optimal only if  $h(n)$  is consistent, meaning it satisfies the triangle inequality:  

$$h(n) \leq c(n, a, n') + h(n').$$

When  $h(n)$  is consistent, the  $f(n) = g(n) + h(n)$  values never decrease along any path. This ensures that A\* always expands the node with the lowest estimated total cost first and never overlooks a cheaper path.

Example: Romania Map

In the path from Arad to Bucharest, the straight-line distance heuristic (hSLD) is both admissible and consistent.

For instance, when Bucharest first appears on the frontier (with  $f = 450$ ), A\* does not immediately expand it. Instead, it continues with nodes like Pitesti ( $f = 417$ ) because there might be a cheaper path through Pitesti. Eventually, A\* finds the optimal path Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest with the least cost.

Thus, by using a consistent heuristic, A\* guarantees it will always find the shortest-cost (optimal) path, without revisiting nodes or missing better routes.

## 6.b. Compare proposition logic and predicate logic in detail with example. (4 Marks | L4 | CO3)

Aspect	Propositional Logic	Predicate Logic (First-Order Logic)
Definition	Deals with simple, atomic propositions that are either true or false.	Extends propositional logic by using quantifiers, variables, and predicates to express complex facts.
Expressiveness	Limited — cannot represent relationships or internal structure.	Highly expressive — can represent relationships among objects and general rules.



Syntax Elements	Uses propositions (e.g., <b>Rain</b> , <b>Snow</b> ) and logical connectives ( $\neg$ , $\wedge$ , $\vee$ , $\rightarrow$ ).	Uses predicates, constants, variables, quantifiers ( $\forall$ , $\exists$ ).
Example	$\text{Rain} \rightarrow \text{WetGround}$ (If it rains, the ground is wet)	$\forall x (\text{Human}(x) \rightarrow \text{Mortal}(x))$ (All humans are mortal)
Inference Power	Less powerful; suitable for simple domains.	More powerful; supports deduction over complex domains.

### 6.c. Explain the following concepts with example:

i) Heuristic function

ii) Atomic sentence

iii) Complex sentence

(6 Marks | L2 | CO3)

#### i) Heuristic function

We look at heuristics for the 8-puzzle, in order to shed light on the nature of heuristics in general.

- The average solution cost for a randomly generated 8-puzzle instance is about 22 steps.
- The branching factor is about 3. (When the empty tile is in the middle, four moves are possible; when it is in a corner, two; and when it is along an edge, three.)
- This means that an exhaustive tree search to depth 22 would look at about  $3^{22} \approx 3.1 \times 10^{10}$  states.
- A graph search would cut this down by a factor of about 170,000 because only  $9!/2 = 181,440$  distinct states are reachable.

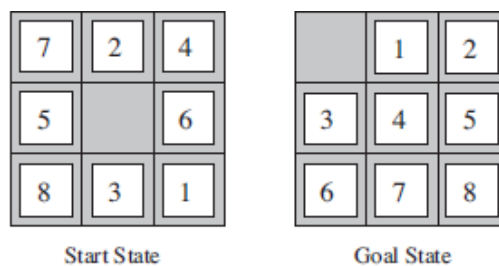


Figure 3.28 A typical instance of the 8-puzzle. The solution is 26 steps long.

Here are two commonly used candidates:

- $h_1$  = the number of misplaced tiles.

For Figure 3.28, all of the eight tiles are out of position, so the start state would have  $h1 = 8$ .  $h1$  is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once.

- $h2$  = the sum of the distances of the tiles from their goal positions.

Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the city block distance or Manhattan distance.  $h2$  is also admissible because all any move can do is move one tile one step closer to the goal. Tiles 1 to 8 in the start state give a Manhattan distance of

$$h2 = 3+1 + 2 + 2+ 2 + 3+ 3 + 2 = 18 .$$

As expected, neither of these overestimates the true solution cost, which is 26.

The effect of heuristic accuracy on performance

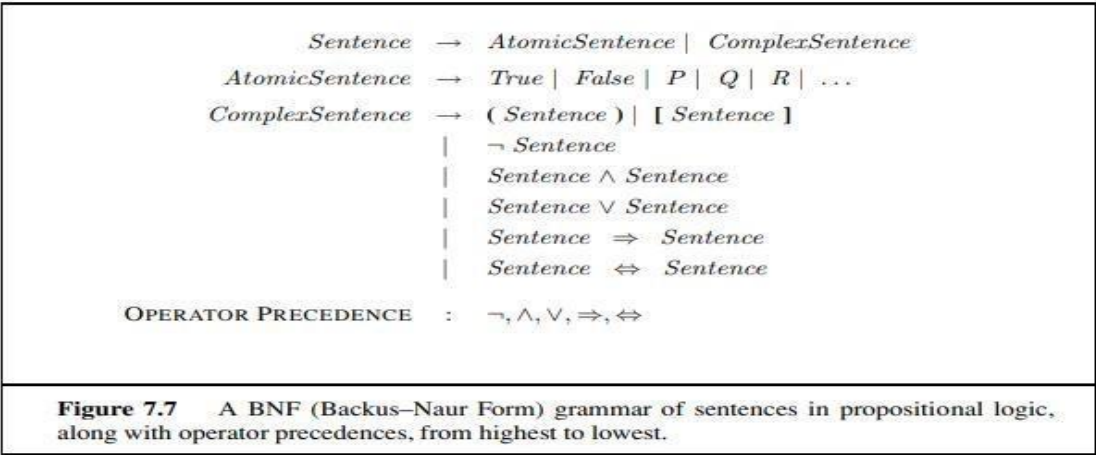
Generating admissible heuristics from relaxed problems

Generating admissible heuristics from subproblems: Pattern databases

Learning heuristics from experience

ii) Atomic sentence

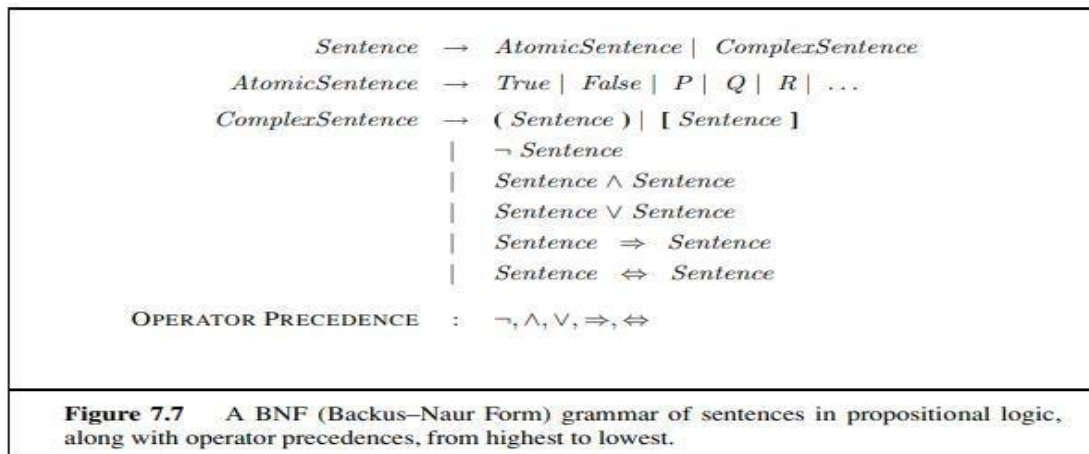
- o The atomic sentences consist of a single proposition symbol.
- o Each such symbol stands for a proposition that can be true or false. Use symbols that start with an uppercase letter and may contain other letters or subscripts, for example:  $P$  ,  $Q$ ,  $R$ ,  $W_{1,3}$  and  $North$ .



iii) Complex sentence

- o Complex sentences are constructed from simpler sentences, using parentheses and logical connectives.
- o There are five connectives in common use:
  - $\neg$  (not). A sentence such as  $\neg W_{1,3}$  is called the negation of  $W_{1,3}$ . A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal).
  - A (and). A sentence whose main connective is A, such as  $W_{1,3} A P_{3,1}$ , is called a conjunction.

- $\vee$  (or). A sentence using  $\vee$ , such as  $(W_{1,3} \vee P_{3,1}) \vee W_{2,2}$ , is a disjunction of the disjunction  $(W_{1,3} \vee P_{3,1})$  and  $W_{2,2}$ .
- $\Rightarrow$  (implies). A sentence such as  $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$  is called an implication. Implications are also known as rules or if–then statements. The implication symbol is sometimes written in other books as  $\supset$  or  $\rightarrow$ .
- $\Leftrightarrow$  (if and only if). The sentence  $W_{1,3} \Leftrightarrow \neg W_{2,2}$  is a biconditional. Some other books write this as  $\equiv$ .



## Module – 4

### 7. a. What are predicates? Explain its syntax and semantics.

(5 Marks | L2 | CO4)

Predicates are functions in First-Order Logic (FOL) that represent properties of objects or relationships between objects. A predicate takes one or more arguments and returns either true or false.

#### Syntax and Semantics of First-Order Logic

##### i. Models for first-order logic

The models of a logical language are the formal structures that constitute the possible worlds under consideration. Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined. Thus, models for propositional logic link proposition symbols to predefined truth values. Models for first-order logic are much more interesting. First, they have objects in them! The domain of a model is the set of objects. The domain is required to be nonempty—every possible world must contain at least one object.

1. Richard the Lionheart, King of England from 1189 to 1199;
2. The evil King John, who ruled from 1199 to 1215;
3. The left legs of Richard
4. The left legs of John;
5. A crown

The objects in the model may be related in various ways. In the figure, Richard and John are brothers. A relation is just the set of tuples of objects that are related. (A tuple is a collection of objects arranged in a fixed order and is written with angle brackets surrounding the objects.) Thus, the brotherhood relation in this model is the set

$$\{ \langle \text{Richard the Lionheart, King John} \rangle, \langle \text{King John, Richard the Lionheart} \rangle \} \quad (8.1)$$

The crown is on King John's head, so the "on head" relation contains just one tuple,

$\langle \text{the crown, King John} \rangle$ . The "brother" and "on head" relations are binary relations—that is, they relate pairs of objects. The model also contains unary relations, or properties: the "person" property is true of both Richard and John; the "king" property is true only of John (presumably because Richard is dead at this point); and the "crown" property is true only of the crown. Certain kinds of relationships are best considered as functions, in that a given object must be related to exactly one object in this way. For example, each person has one left leg, so the model has a unary "left leg" function that includes the following mappings:

$$\langle \text{Richard the Lionheart} \rangle \rightarrow \text{Richard's left leg} \quad (8.2)$$

$$\langle \text{King John} \rangle \rightarrow \text{John's left leg}$$

## ii. Symbols and interpretations

The basic syntactic elements of first-order logic are the symbols that stand for objects, relations, and functions. The symbols, therefore, come in three kinds: Constant symbols, which stand for objects; Predicate symbols, which stand for relations; and Function symbols, which stand for functions. We adopt the convention that these symbols will begin with uppercase letters. For example, we might use constant symbols Richard and John; predicate symbols Brother, OnHead, Person, King, and Crown; and function symbol LeftLeg.

As in propositional logic, every model must provide the information required to determine if any given sentence is true or false. Thus, in addition to its objects, relations, and functions, each

model includes an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols. One possible interpretation for our example—which a logician would call the intended interpretation—is as follows:

- Richard refers to Richard the Lionheart and John refers to the evil King John.
- Brother refers to the brotherhood relation, that is, the set of tuples of objects given in Equation (8.1); OnHead refers to the “on head” relation that holds between the crown and King John;
- LeftLeg refers to the “left leg” function, that is, the mapping given in Equation (8.2).

There are many other possible interpretations, of course. For example, one interpretation maps Richard to the crown and John to King John’s left leg. There are five objects in the model, so there are 25 possible interpretations just for the constant symbols Richard and John.

### iii. Terms

- A term is a logical expression that refers to an object. Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object. For example, in English we might use the expression “King John’s left leg” rather than giving a name to his leg.
- This is what function symbols are for: instead of using a constant symbol, we use

LeftLeg(John).

- In the general case, a complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol.
- The formal semantics of terms is straightforward. Consider a term  $f(t_1, \dots, t_n)$ . The function symbol  $f$  refers to some function in the model (call it  $F$ ); the argument terms refer to objects in the domain (call them  $d_1, \dots, d_n$ ); and the term as a whole refers to the object that is the value of the function  $F$  applied to  $d_1, \dots, d_n$ . For example, suppose the LeftLeg function symbol refers to the function shown in Equation (8.2) and John refers to King John, then LeftLeg(John) refers to King John’s left leg. In this way, the interpretation fixes the referent of every term.

### iv. Atomic sentences

Atomic sentence is formed from a predicate symbol optionally followed by a parenthesized list of terms, such as Brother (Richard, John). This states, under the intended interpretation given earlier, that Richard the Lionheart is the brother of King John. Atomic sentences can have

complex terms as arguments. Thus,  $\text{Married}(\text{Father}(\text{Richard}), \text{Mother}(\text{John}))$  states that Richard the Lionheart's father is married to King John's mother (again, under a suitable

interpretation). An atomic sentence is true in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.

#### v. Complex sentences

We can use logical connectives to construct more complex sentences, with the same syntax and semantics as in propositional calculus. Here are four sentences that are true in the model of Figure 8.2 under our intended interpretation:

$\neg \text{Brother}(\text{LeftLeg}(\text{Richard}), \text{John})$

$\text{Brother}(\text{Richard}, \text{John}) \wedge \text{Brother}(\text{John}, \text{Richard})$

$\text{King}(\text{Richard}) \vee \text{King}(\text{John})$

$\neg \text{King}(\text{Richard}) \Rightarrow \text{King}(\text{John})$

#### 7.b. Define universal and existential instantiation and give example for both.

(5 Marks | L1 | CO4)

Universal Instantiation (UI):

The rule says that we can infer any sentence obtained by substituting a ground term (a term without variables) for the variable. Let  $\text{SUBST}(\theta)$  denote the result of applying the substitution  $\theta$  to the sentence  $a$ . Then the rule is written

$$\frac{\forall v \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

For any variable  $v$  and ground term  $g$ .

For example, there is a sentence in knowledge base stating that all greedy kings are Evils

$$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x).$$

For the variable  $x$ , with the substitutions like  $\{x/\text{John}\}, \{x/\text{Richard}\}$  the following sentences can be

inferred.

$$\begin{aligned}
& King(John) \wedge Greedy(John) \Rightarrow Evil(John) \\
& King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard) \\
& King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John)) . \\
& \vdots
\end{aligned}$$

Thus a universally quantified sentence can be replaced by the set of all possible instantiations.

Existential Instantiation (EI):

The existential sentence says there is some object satisfying a condition, and the instantiation process is just giving a name to that object, that name must not already belong to another object.

This new name is called a Skolem constant. Existential Instantiation is a special case of a more general process called “skolemization”.

For any sentence  $\alpha$ , variable  $v$ , and constant symbol  $k$  that does not appear elsewhere in the knowledge base,

$$\frac{\exists v \alpha}{\text{SUBST}(\{v/k\}, \alpha)} .$$

For example, from the sentence

$$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$$

we can infer the sentence

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

As long as  $C_1$  does not appear elsewhere in the knowledge base. Thus an existentially quantified sentence can be replaced by one instantiation

Elimination of Universal and Existential quantifiers should give new knowledge base which can be shown to be inferentially equivalent to old in the sense that it is satisfiable exactly when the original knowledge base is satisfiable.

**7.c. Consider the following knowledge base:**

**i) Gita likes all kinds of food**

**ii) Mango and chapatti are food**

**iii) Anything anyone eats and is still alive**

**Goal: Gita ate mango and anyone and is still alive is food**

**(10 Marks | L3 | CO4)**



Given Knowledge Base (KB):

1. Gita likes all kinds of food
2. Mango and chapatti are food
3. Anything that anyone eats and is still alive is food

Goal:

Prove (or infer):

Gita ate mango

Anyone who is still alive is food

Step 1: Represent the KB in First-Order Logic (FOL)

Let's define the predicates:

- $\text{Food}(x)$  —  $x$  is food
- $\text{Likes}(x, y)$  —  $x$  likes  $y$
- $\text{Eats}(x, y)$  —  $x$  eats  $y$
- $\text{Alive}(x)$  —  $x$  is alive
- $\text{Person}(x)$  —  $x$  is a person

Now express each sentence:

1. Gita likes all kinds of food  
 $\forall x (\text{Food}(x) \rightarrow \text{Likes}(\text{Gita}, x))$
2. Mango and chapatti are food  
 $\text{Food}(\text{Mango})$   
 $\text{Food}(\text{Chapatti})$
3. Anything that anyone eats and is still alive is food  
 $\forall x \forall y ((\text{Eats}(x, y) \wedge \text{Alive}(x)) \rightarrow \text{Food}(y))$

Step 2: State the Goal in FOL

We need to derive or support the conclusion that:

- $\text{Gita ate mango} \rightarrow \text{Eats}(\text{Gita}, \text{Mango})$
- $\text{Anyone who is alive is food} \rightarrow (\text{We'll clarify this below})$

Step 3: Inference / Reasoning

We need to use resolution or chaining to show what can be inferred.

From the given KB:

- We already know  $\text{Food}(\text{Mango})$  from statement 2
- And from 1:  $\forall x (\text{Food}(x) \rightarrow \text{Likes}(\text{Gita}, x))$   
 $\Rightarrow$  Apply Modus Ponens with  $\text{Food}(\text{Mango})$   
 $\Rightarrow \text{Likes}(\text{Gita}, \text{Mango})$
- So, we've shown Gita likes mango. Now, consider statement 3:  
 $\forall x \forall y ((\text{Eats}(x, y) \wedge \text{Alive}(x)) \rightarrow \text{Food}(y))$   
This implies: If someone eats something and is still alive, that something is food.

We can't reverse this directly to say: "If someone is alive, then they are food" — this is a logical fallacy.

So, the correct derivable conclusions from the KB are:

1. Gita likes mango (since mango is food)
2. Gita likes chapatti (since chapatti is also food)
3. If Gita eats mango and is still alive, then mango is food (but we already know that)
4. We cannot prove from the KB that "anyone who is alive is food" — this is not supported by FOL semantics.

Final Answer Summary (FOL-based reasoning):

- Step 1: From  $\text{Food}(\text{Mango})$ ,  $\text{Food}(\text{Chapatti})$
- Step 2: From  $\forall x (\text{Food}(x) \rightarrow \text{Likes}(\text{Gita}, x))$ , we infer:  
 $\rightarrow \text{Likes}(\text{Gita}, \text{Mango})$   
 $\rightarrow \text{Likes}(\text{Gita}, \text{Chapatti})$

- Step 3: From  $\forall x \forall y ((\text{Eats}(x, y) \wedge \text{Alive}(x)) \rightarrow \text{Food}(y))$ , we know:  
If Gita eats mango and is alive  $\rightarrow$  Mango is food (already known)

**OR**

**8. a. Write appropriate expressions for the following:**

- Some students read well**
- Some students like some books**
- Some students like all books**
- All students like some books**
- All students like no books**

**Explain the concept of resolution in first order logic with appropriate procedure. (8 Marks | L3 | CO4)**

Let the predicates be:

- $\text{Student}(x)$  —  $x$  is a student
- $\text{ReadsWell}(x)$  —  $x$  reads well
- $\text{Likes}(x, y)$  —  $x$  likes  $y$
- $\text{Book}(y)$  —  $y$  is a book

i) Some students read well

$$\exists x (\text{Student}(x) \wedge \text{ReadsWell}(x))$$

ii) Some students like some books

$$\exists x \exists y (\text{Student}(x) \wedge \text{Book}(y) \wedge \text{Likes}(x, y))$$

iii) Some students like all books

$$\exists x (\text{Student}(x) \wedge \forall y (\text{Book}(y) \rightarrow \text{Likes}(x, y)))$$

iv) All students like some books

$$\forall x (\text{Student}(x) \rightarrow \exists y (\text{Book}(y) \wedge \text{Likes}(x, y)))$$

v) All students like no books

$$\forall x (\text{Student}(x) \rightarrow \forall y (\text{Book}(y) \rightarrow \neg \text{Likes}(x, y)))$$

Concept of resolution in FOL

## Resolution in First-Order Logic

Resolution is a rule of inference used for automated theorem proving. It works by refutation: we assume the negation of the goal, add it to the knowledge base, and apply resolution repeatedly to derive a contradiction (empty clause  $\perp$ ).

### Resolution Procedure Steps:

1. Convert all FOL sentences to clause form:
  - Eliminate implications ( $\rightarrow$ )
  - Move negations inward (using De Morgan's laws)
  - Standardize variables
  - Skolemize existential quantifiers
  - Drop universal quantifiers
  - Convert to conjunctive normal form (CNF)
  - Represent each conjunct as a clause
2. Negate the query (goal) and add it to the KB.
3. Apply unification to find matching literals and resolve them.
4. Repeat resolution until:
  - The empty clause ( $\perp$ ) is derived  $\Rightarrow$  goal is proven.
  - No new clauses  $\Rightarrow$  goal not provable.

Example (from textbook):

KB:

1.  $\forall x (\neg \text{Human}(x) \vee \text{Mortal}(x))$
2.  $\text{Human}(\text{Socrates})$

Goal: Prove  $\text{Mortal}(\text{Socrates})$

- Negate goal:  $\neg \text{Mortal}(\text{Socrates})$
- Convert KB and goal to clauses:
  - Clause 1:  $\neg \text{Human}(x) \vee \text{Mortal}(x)$
  - Clause 2:  $\text{Human}(\text{Socrates})$
  - Clause 3:  $\neg \text{Mortal}(\text{Socrates})$
- Apply resolution:
  - From Clause 1 and Clause 2  $\rightarrow \text{Mortal}(\text{Socrates})$
  - Resolving with Clause 3  $\rightarrow \perp$  (empty clause)

Goal is proved by resolution.

**8.b. Write and explain simple backward chaining algorithm and forward chaining algorithm for first-order knowledge bases with example. Also explain the process of unification.**

**(12 Marks | L3 | CO4)**

Forward Chaining

First-Order Definite Clauses:

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

“The law says that it is a crime for an American to sell weapons to hostile nations. The country

Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.”

We will represent the facts as first-order definite clauses

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

Example Knowledge Base:

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

Rule 1.  $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

Nono . . . has some missiles,

i.e.,  $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$ :

Rule 2.  $\text{Owns}(\text{Nono}, M1)$  and

Rule 3.  $\text{Missile}(M1)$

. . . all of its missiles were sold to it by Colonel West

Rule 4.  $\forall x \text{ Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

Missiles are weapons:

Rule 5.  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

An enemy of America counts as “hostile”:

Rule 6.  $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

West, who is American . . .

Rule 7.  $\text{American}(\text{West})$

The country Nono, an enemy of America . . .

Rule 8.  $\text{Enemy}(\text{Nono}, \text{America})$

A simple forward-chaining algorithm:

Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered or no new facts are added. Notice that a fact is not "new" if it is just renaming of a known fact. We will use our crime problem to illustrate how FOL-FC-ASK works. The implication sentences are (1), (4), (5), and (6). Two iterations are required: On the first iteration, rule (1) has unsatisfied premises. Rule (4) is satisfied with  $\{x/M1\}$ , and *Sells* (West, M1, Nono) is added. Rule (5) is satisfied with  $\{x/M1\}$  and *Weapon* (M1) is added. Rule (6) is satisfied with  $\{x/Nono\}$ , and *Hostile* (Nono) is added. On the second iteration, rule (1) is satisfied with  $\{x/West, Y/M1, z/Nono\}$ , and *Criminal*(West) is added. It is sound, because every inference is just an application of Generalized Modus Ponens, it is complete for definite clause knowledge bases; that is, it answers every query whose answers are entailed by any knowledge base of definite clauses

```

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

repeat until new is empty
  new  $\leftarrow \{\}$ 
  for each rule in KB do
     $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-VARIABLES}(\text{rule})$ 
    for each  $\theta$  such that  $\text{SUBST}(\theta, p_1 \wedge \dots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \dots \wedge p'_n)$ 
      for some  $p'_1, \dots, p'_n$  in KB
         $q' \leftarrow \text{SUBST}(\theta, q)$ 
        if  $q'$  does not unify with some sentence already in KB or new then
          add  $q'$  to new
           $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
          if  $\phi$  is not fail then return  $\phi$ 
  add new to KB
return false

```

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

## Backward Chaining

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

are added to the stack and the goal is solved. The algorithm for backward chaining and proof tree for finding criminal (West) using backward chaining are given below.

```

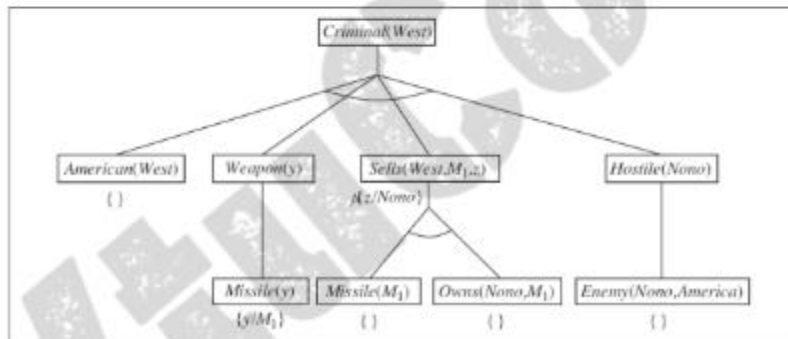
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

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

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

```

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



## Unification:

It is the process used to find substitutions that make different logical expressions look identical. Unification is a key component of all first-order Inference algorithms.  $\text{UNIFY}(p, q) = \theta$  where  $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$   $\theta$  is our unifier value (if one exists). Ex:—Who does John know?||

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(\text{John}, \text{Jane})) = \{x/\text{Jane}\}$ .  $\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Bill})) = \{x/\text{Bill}, y/\text{John}\}$ .

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Mother}(y))) = \{x/\text{Bill}, y/\text{John}\}$   $\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{Elizabeth})) = \text{FAIL}$

The last unification fails because both use the same variable, X. X can't equal both John and Elizabeth. To avoid this change the variable X to Y (or any other value) in  $\text{Knows}(X, \text{Elizabeth})$



$\text{Knows}(X, \text{Elizabeth}) \rightarrow \text{Knows}(Y, \text{Elizabeth})$

Still means the same. This is called standardizing apart. sometimes it is possible for more than one unifier returned:

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, z)) = ???$

This can return two possible unifications:  $\{y/\text{John}, x/z\}$  which means  $\text{Knows}(\text{John}, z)$  OR  $\{y/\text{John}, x/\text{John}, z/\text{John}\}$ . For each unifiable pair of expressions there is a single most general unifier (MGU), In this case it is  $\{y/\text{John}, x/z\}$ . An algorithm for computing most general unifiers is shown below.

```
function UNIFY( $x, y, \theta$ ) returns a substitution to make  $x$  and  $y$  identical
inputs:  $x$ , a variable, constant, list, or compound
          $y$ , a variable, constant, list, or compound
          $\theta$ , the substitution built up so far (optional, defaults to empty)

if  $\theta = \text{failure}$  then return failure
else if  $x = y$  then return  $\theta$ 
else if VARIABLE? $(x)$  then return UNIFY-VAR( $x, y, \theta$ )
else if VARIABLE? $(y)$  then return UNIFY-VAR( $y, x, \theta$ )
else if COMPOUND? $(x)$  and COMPOUND? $(y)$  then
    return UNIFY(ARGs $[x]$ , ARGs $[y]$ , UNIFY(OP $[x]$ , OP $[y]$ ,  $\theta$ ))
else if LIST? $(x)$  and LIST? $(y)$  then
    return UNIFY(REST $[x]$ , REST $[y]$ , UNIFY(FIRST $[x]$ , FIRST $[y]$ ,  $\theta$ ))
else return failure



---


function UNIFY-VAR( $var, x, \theta$ ) returns a substitution
inputs:  $var$ , a variable
          $x$ , any expression
          $\theta$ , the substitution built up so far

if  $\{var/val\} \in \theta$  then return UNIFY( $val, x, \theta$ )
else if  $\{x/val\} \in \theta$  then return UNIFY( $var, val, \theta$ )
else if OCCUR-CHECK? $(var, x)$  then return failure
else return add  $\{var/x\}$  to  $\theta$ 
```

## Module – 5

### 9. a. Explain the impact of uncertainty in probabilistic reasoning.

(5 Marks | L2 | CO5)

Uncertainty in AI arises from partial observability, nondeterminism, or incomplete knowledge about the environment. Traditional logical agents struggle in such environments because they

must account for all possible scenarios, leading to overly complex belief states and unmanageably large contingency plans.

In contrast, probabilistic reasoning allows agents to represent degrees of belief about the world using probability theory. Instead of needing complete certainty, agents can make informed decisions based on likelihoods of outcomes.

For example, an automated taxi agent can't guarantee arriving on time due to possible delays. However, it can select the plan with the highest expected utility, balancing risks like traffic delays against rewards like punctuality. This addresses the qualification problem, where it's impossible to list all conditions that affect success.

Decision theory, which combines probability theory with utility theory, allows agents to choose actions that maximize expected utility, making them rational even when outcomes are uncertain.

**9.b. Explain Bayes' rule and its utilization in probabilistic reasoning.**  
**(5 Marks | L2 | CO5)**

It can actually be written in two forms:

$$P(a \wedge b) = P(a | b)P(b) \text{ and } P(a \wedge b) = P(b | a)P(a) .$$

Equating the two right-hand sides and dividing by  $P(a)$ , we get

$$P(b | a) = P(a | b)P(b)/P(a) . \quad (13.12)$$

This equation is known as Bayes' rule (also Bayes' law or Bayes' theorem). This simple equation underlies most modern AI systems for probabilistic inference.

The more general case of Bayes' rule for multivalued variables can be written in the P notation as follows:

$$P(Y | X) = P(X | Y)P(Y)/P(X) ,$$

As before, this is to be taken as representing a set of equations, each dealing with specific values of the variables. We will also have occasion to use a more general version conditionalized on some background evidence  $e$ :

$$P(Y | X, e) = P(X | Y, e)P(Y | e)/P(X | e) .$$

Applying Bayes' rule: The simple case

On the surface, Bayes' rule does not seem very useful. It allows us to compute the single term  $P(b \mid a)$  in terms of three terms:  $P(a \mid b)$ ,  $P(b)$ , and  $P(a)$ . That seems like two steps backwards, but Bayes' rule is useful in practice because there are many cases where we do have good probability estimates for these three numbers and need to compute the fourth. Often, we perceive as evidence the effect of some unknown cause and we would like to determine that cause. In that case, Bayes' rule becomes

$$P(\text{cause} \mid \text{effect}) = P(\text{effect} \mid \text{cause})P(\text{cause})/P(\text{effect}) .$$

The conditional probability  $P(\text{effect} \mid \text{cause})$  quantifies the relationship in the causal direction, whereas  $P(\text{cause} \mid \text{effect})$  describes the diagnostic direction. In a task such as medical diagnosis, we often have conditional probabilities on causal relationships (that is, the doctor knows  $P(\text{symptoms} \mid \text{disease})$ ) and want to derive a diagnosis,  $P(\text{disease} \mid \text{symptoms})$ . For example, a doctor knows that the disease meningitis causes the patient to have a stiff neck,

say, 70% of the time. The doctor also knows some unconditional facts: the prior probability

that a patient has meningitis is  $1/50,000$ , and the prior probability that any patient has a stiff neck is 1%. Letting  $s$  be the proposition that the patient has a stiff neck and  $m$  be the proposition that the patient has meningitis, we have

$$P(s \mid m) = 0.7$$

$$P(m) = 1/50000$$

$$P(s) = 0.01$$

$$P(m \mid s) = P(s \mid m)P(m)$$

$$P(s) = 0.7 \times (1/50000)/0.01 = 0.0014 .$$

Using Bayes' rule: Combining evidence

We have seen that Bayes' rule can be useful for answering probabilistic queries conditioned on one piece of evidence—for example, the stiff neck. In particular, we have argued that probabilistic information is often available in the form  $P(\text{effect} \mid \text{cause})$ . What happens when we have two or more pieces of evidence? For example, what can a dentist conclude if her nasty steel

probe catches in the aching tooth of a patient? If we know the full joint distribution (Figure 13.3), we can read off the answer:

$$P(\text{Cavity} | \text{toothache} \wedge \text{catch}) = \alpha 0.108, 0.016 \approx 0.871, 0.129 .$$

We know, however, that such an approach does not scale up to larger numbers of variables. We can try using Bayes' rule to reformulate the problem:

$$P(\text{Cavity} | \text{toothache} \wedge \text{catch}) = \alpha P(\text{toothache} \wedge \text{catch} | \text{Cavity}) P(\text{Cavity}) .$$

**9.c. Write the representation of Bayes Theorem. In a class, 70% children were fall sick due to viral fever and 30% due to bacterial fever. The probability of observing temperature for viral is 0.78 and bacterial is 0.31. If a child develops high...  
(10 Marks | L3 | CO5)**

Bayes' Theorem – Formula

$$P(H | E) = P(E | H) \cdot P(H) / P(E)$$

Where:

- $P(H | E)$ : Posterior probability (Hypothesis given evidence)
- $P(E | H)$ : Likelihood (Probability of evidence given hypothesis)
- $P(H)$ : Prior probability
- $P(E)$ : Total probability of the evidence (marginal likelihood)

Given Data

Let:

- Viral fever (V)
- Bacterial fever (B)

$$P(V)=0.70, P(B)=0.30$$

$$P(\text{Temp} | V)=0.78, P(\text{Temp} | B)=0.31$$

We need to find:

$P(V| \text{Temp})=?$

Apply Bayes' Theorem

$$P(V| \text{Temp})=P(\text{Temp}| V) \cdot P(V)/P(\text{Temp})$$

We must compute  $P(\text{Temp})$  using Law of Total Probability:

$$P(\text{Temp})=P(\text{Temp}| V) \cdot P(V)+P(\text{Temp}| B) \cdot P(B)$$

$$=(0.78)(0.70)+(0.31)(0.30)=0.546+0.093=0.639$$

Now compute:

$$P(V| \text{Temp})=0.78 \cdot 0.70/0.639=0.546/0.639 \approx 0.854$$

Final Answer

$$P(\text{Viral Fever}| \text{Temperature}) \approx 85.4\%$$

So, if a child develops high temperature, there is about an 85.4% probability it is due to viral fever.

**OR**

**10. a. Write short notes on:**

**i) Expert systems**

**ii) Knowledge acquisition**

**(8 Marks | L2 | CO5)**

i) Expert systems

Expert systems are computer programs designed to simulate the decision-making ability of human experts. They use a large amount of domain-specific knowledge, often encoded in the form of rules. These systems solve complex problems in areas like medicine, engineering, or mineral exploration by using reasoning techniques such as forward chaining or backward chaining.

Several expert systems have been developed, each with its unique rule base and reasoning method:

- **RI (or XCON):** Used for configuring computer systems (e.g., DEC VAX systems). It applies a set of production rules to recommend configurations, such as selecting disk

drives or cables based on current system status. It uses forward chaining, starting from known conditions and applying rules to reach conclusions.

- MYCIN: A medical diagnosis system that used backward chaining. It worked by reasoning from potential diagnoses back to symptoms and test results, helping determine bacterial infections and recommending treatments.
- PROSPECTOR: Used in geology for mineral exploration. It assigned confidence measures to hypotheses using numerical certainty factors. It combined geological evidence with probabilistic reasoning to suggest the presence of minerals.
- DESIGN ADVISOR: Assisted chip designers by advising on circuit designs. If its suggestion was rejected, it used a justification-based truth maintenance system to revise its reasoning. It relied on checking conditions like "resetability" based on sequential element count and would query the user if a rule was violated.

## ii) Knowledge acquisition

Knowledge acquisition is the process by which a knowledge engineer extracts expert knowledge from a domain expert and translates it into a usable form for an expert system. It typically involves an interview-based method where rules are initially built and refined using iterative expert-level validation. This process is slow and error-prone, especially when no automated systems are involved.

To support knowledge acquisition, several systems have been developed:

- MOLE: A system that acquires knowledge by refining hypotheses through expert consultation. It works by suggesting explanations and refining them based on expert feedback and problem-solving experience. MOLE uses networks where each symptom may have multiple causes and refines the knowledge base with each error made.
- SALT: Focuses on design configuration problems and uses parameter-value dependencies to build networks. It checks constraints and dependencies and updates explanations when violations occur. SALT is especially useful in tasks such as elevator design where solutions are constructed incrementally.
- META-DENDRAL: The first system to use machine learning for rule generation. It learned rules from chemical data and used them to deduce molecular structures accurately, showing that expert knowledge can be automatically learned.

**10.b. Suppose a doctor is trying to find out if a patient is suffering from some type of cancer. If the cancer is only found on average in 2 out of every 1000 people, the doctor's initial beliefs can be expressed as  $P(\text{cancer}) = 0.002$ .**

**There is a laboratory test to determine if the patient has cancer. Unfortunately this test is 100% accurate. The test comes back positive in 98% of cases where the patient has cancer. Also, the test comes out negative only in 97% of cases, where the patient does not have a cancer. If the doctor orders a test and it comes back positive what is the probability that the patient indeed has cancer?**

**(12 Marks | L3 | CO5)**

- Prior probability of cancer:  
 $P(\text{Cancer}) = 0.002$
- Test result is positive
- Test accuracy:
  - $P(\text{Pos} | \text{Cancer}) = 0.98$
  - $P(\text{Neg} | \neg \text{Cancer}) = 0.97$   
 $\Rightarrow \text{So, } P(\text{Pos} | \neg \text{Cancer}) = 1 - 0.97 = 0.03$

We are asked to compute:

$$P(\text{Cancer} | \text{Positive Test Result})$$

Step 1: Apply Bayes' Theorem

$$P(\text{Cancer} | \text{Positive}) = P(\text{Positive} | \text{Cancer}) \cdot P(\text{Cancer}) / P(\text{Positive})$$

We need to calculate  $P(\text{Positive})$ :

$$P(\text{Positive}) = P(\text{Positive} | \text{Cancer}) \cdot P(\text{Cancer}) + P(\text{Positive} | \neg \text{Cancer}) \cdot P(\neg \text{Cancer})$$

Substitute the known values:

$$P(\text{Positive}) = (0.98)(0.002) + (0.03)(0.998) = 0.00196 + 0.02994 = 0.0319$$

Step 2: Now calculate the posterior probability

$$P(\text{Cancer} | \text{Positive}) = 0.98 \cdot 0.002 / 0.0319 = 0.00196 / 0.0319$$

$$P(\text{Cancer} | \text{Positive}) \approx 0.0614 \approx 6.14\% \text{ Final Answer:}$$

Even though the test is highly accurate, the probability that the patient actually has cancer after a positive test is only about 6.14%.

Why is the result low despite a positive test?

Because cancer is very rare (0.2%), even a small false positive rate (3%) causes many non-cancer patients to test positive. This leads to a low posterior probability even after a positive result. This is a classic case of Bayesian diagnosis in medical reasoning.