INF-4046

GAUHATI UNIVERSITY
Centre for Distance and Online Education

Fourth Semester

(Under CBCS)

M.Sc.- IT

Paper: INF 4046

ARTIFICIAL INTELLIGENCE

CONTENTS:

SLM Development Team:

HoD, Department of Computer Science, Gauhati University
Programme Coordinator, M.Sc.-IT, GUCDOE
Prof. Shikhar Kr. Sarma, Department of IT, Gauhati University
Dr. Khurshid Alam Borbora, Assistant Professor, GUCDOE
Dr. Swapnanil Gogoi, Assistant Professor, GUCDOE
Mrs. Pallavi Saikia, Assistant Professor, GUCDOE
Dr. Rita Chakraborty, Assistant Professor, GUCDOE
Mr. Hemanta Kalita, Assistant Professor, GUCDOE

Course Coordination:

Dr. Debahari Talukdar	Director, GUCDOE
Prof. Anjana Kakoti Mahanta	Programme Coordinator, GUCDOE Dept. of Computer Science, G.U.
Dr. Khurshid Alam Borbora	Assistant Professor, GUCDOE
Dr. Swapnanil Giogoi	Assistant Professor, GUCDOE
Mrs. Pallavi Saikia	Assistant Professor, GUCDOE
Dr. Rita Chakraborty	Assistant Professor, GUCDOE
Mr. Hemanta Kalita	Assistant Professor, GUCDOE
Mr. Dipankar Saikia	Editor SLM, GUCDOE

Contributors:

Dr. Khurshid Alam Borbora Assistant Professor, GUCDOE	(Block I : Unit- 1)
Dr. Ganapati Das Teaching Associate Dept. of Computer Science, G.U.	(Block I : Units- 2 & 3)
Dr. Rita Chakraborty Assistant Professor, GUCDOE	(Block II : Units- 1, 2 & 3)
Mrs. Pallavi Saikia Assistant Professor, GUCDOE	(Block II : Units- 4 & 5) (Block III : Unit- 1)
Dr. Pranamika Kakati Assistant Professor Dept. of Computer Science, G.U.	(Block III : Unit- 2)
Mr. Hemanta Kalita Assistant Professor, GUCDOE	(Block III : Unit- 3)
Dr. Ridip Dev Choudhury Associate Professor, KKHSOU	(Block III : Unit- 4)

Content Editor:

Dr. Dwipen Laskar	Assistant Professor Dept. of Computer Science, G.U.

Cover Page Designing:

Bhaskar Jyoti Goswami	GUCDOE
Nishanta Das	GUCDOE

ISBN: 978-81-986642-8-0
June, 2025

Published on behalf of Gauhati University Centre for Distance and Online Education by the Director, and printed at Gauhati University Press, Guwahati-781014.

CONTENTS:

BLOCK- I

BASIC FOUNDATION OF ARTIFICIAL INTELLIGENCE

UNIT 1: INTRODUCTION TO AI

UNIT 2: PROBLEM AND PROBLEM SPACES

UNIT 3: HEURISTIC SEARCH TECHNIQUES

UNIT-1: INTRODUCTION TO AI

Unit Structure:

1.1 INTRODUCTION

In this unit, we will explore the definition, significance, historical background, and various application areas of Artificial Intelligence (AI). We will also gain an overview of some common challenges faced in AI and the techniques used to address them.

Artificial Intelligence (AI) refers to the branch of computer science dedicated to designing systems capable of performing tasks that typically require human intelligence. In simple terms, AI is focused on creating intelligent behavior in machines. These systems are designed to learn new concepts, draw logical conclusions from facts, understand and process natural language, and perform activities that would otherwise require human cognitive skills.

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Define Artificial Intelligence and understand its meaning.
- Explain the significance and role of Artificial Intelligence in today's world.
- Describe the historical foundations and early developments in the field of Artificial Intelligence.
- Identify and explain the key problems and tasks addressed by Artificial Intelligence.

1.3 WHAT IS AI?

Artificial Intelligence (AI) is a branch of computer science that focuses on creating machines or systems capable of performing tasks that normally require human intelligence. These tasks include learning from experience, reasoning, problem-solving, understanding natural language, perceiving the environment, and making decisions. In simple terms, AI is the process of designing and building intelligent systems that can think, learn, and act like humans to some extent. It involves simulating human cognitive abilities through algorithms, data, and computational power.

Let's explore some of the key aspects of Artificial Intelligence:

AI systems are capable of collecting information from multiple sources, storing that information, and utilizing it effectively to address problems or make informed decisions. Similar to how humans learn and apply knowledge, AI uses data and knowledge bases to perform tasks intelligently.

Reasoning plays a crucial role in AI. It involves drawing logical conclusions from available information. AI systems are equipped with the ability to evaluate situations, consider possible outcomes, and make decisions based on logic and analysis.

Learning from experience is another significant feature of AI. With the help of machine learning techniques, AI systems can continuously adapt and enhance their performance by analysing past actions or examples — without needing to be manually reprogrammed each time.

Perception and understanding are also essential components. AI systems are designed to interpret data from various inputs such as text, speech, images, and sensory data. This ability enables them to comprehend natural human language and respond appropriately to different stimuli.

Finally, AI focuses on automating intelligent behaviours typically performed by humans. This includes solving complex problems, planning, diagnosing, and even strategizing in games or business scenarios. These tasks are automated to enhance efficiency and accuracy, reducing the need for constant human supervision.

1.4 HISTORY AND CURRENT TRENDS OF AI

The journey of Artificial Intelligence (AI) began with the ambitious idea of building machines capable of simulating human intelligence. This vision was first conceptualized by Alan Turing, a pioneer in computing, who introduced the concept of thinking machines and proposed the famous Turing Test as a measure of machine intelligence.

In 1956, the field of AI was formally born at the Dartmouth Conference, where the term "Artificial Intelligence" was officially coined. This marked the beginning of focused research in AI as an academic discipline.

In the early years, researchers achieved promising results. Early AI programs demonstrated the ability to solve algebraic problems, prove mathematical theorems, and even play games like checkers. These efforts led to the development of symbolic AI systems, which relied

on logic and predefined rules to simulate human reasoning and problem-solving.

However, as expectations grew, progress could not keep pace. Limitations in computing power, combined with overly optimistic predictions, led to a slowdown. This period of reduced interest and funding became known as the AI Winter.

AI research saw resurgence in the 1980s with the emergence of expert systems. These systems used large rule-based knowledge bases to make intelligent decisions, with systems like MYCIN (used for medical diagnosis) demonstrating the potential of machines to replicate human expertise in specialized areas.

Despite this success, expert systems eventually faced scalability issues and became difficult and costly to maintain. This led to another decline in enthusiasm and funding for AI research.

The field then shifted focus to machine learning in the 1990s, where systems began learning patterns directly from data rather than relying on pre-programmed rules. Key advancements in statistical methods, neural networks, and support vector machines propelled AI forward and laid the groundwork for modern applications.

The 2010s marked a major turning point with breakthroughs in deep learning, fueled by big data and the availability of powerful GPUs. AI applications rapidly expanded into areas like image recognition, natural language processing, and autonomous vehicles. Today, AI plays a crucial role in various industries such as healthcare, business, finance, entertainment, and is embedded in personal assistants like Siri, Alexa, and Google Assistant.

Currently, AI research is focused on tackling advanced challenges, including explainable AI, ethical AI, artificial general intelligence, and combining AI with cutting-edge fields like quantum computing and robotics, setting the stage for even greater advancements in the future.

1.5 APPLICATION AREAS OF AI

Artificial Intelligence has become an integral part of modern life, with its presence felt across numerous fields. It enables machines to perform complex tasks, make decisions, and assist humans in solving problems more efficiently. Let's explore the key areas where AI has made a significant impact:

Healthcare

AI is transforming the healthcare sector with advanced tools and intelligent systems that assist medical professionals.

- AI models help detect diseases through medical imaging analysis (X-rays, MRIs, CT scans).
- AI algorithms suggest customized treatments based on patient data.
- AI accelerates drug discovery and testing by predicting the effectiveness of chemical compounds.
- AI chatbots and virtual nurses offer round-the-clock health advice and reminders.

Education

AI is contributing to personalized and adaptive learning solutions.

- AI guides students through additional learning support.
- AI grades exams, quizzes, and essays automatically.
- AI-based platforms adapt to individual student learning styles.

Agriculture

AI helps improve agricultural productivity and sustainability.

- AI optimizes planting, irrigation, and harvesting times.
- AI systems identify crop diseases and recommend remedies.
- AI models predict crop output based on environmental conditions.

Finance and Business

AI streamlines operations and improves decision-making in business and financial services.

- AI systems identify suspicious activities in transactions.
- AI executes trades at high speed and precision by analysing market data.
- AI-powered chatbots handle customer inquiries efficiently.
- AI tools predict market trends and consumer behaviours.

Entertainment and Media

The entertainment industry uses AI for content curation and production.

- Platforms like Netflix and YouTube suggest content based on user preferences.
- AI is used for generating music, art, and video content.
- AI enhances gaming experiences through smart character behaviours and adaptive gameplay.

Transportation

AI technologies have revolutionized the transportation industry by making it more efficient and safe.

- Self-driving cars and trucks navigate and make decisions using AI.
- AI optimizes traffic signals and reduces congestion.
- AI enhances route planning and package tracking.

Security and Surveillance

AI enhances security measures and real-time monitoring.

- Widely used in security for identification and access control.
- AI detects potential cyber threats and unusual activity in networks.
- AI analyzes crime patterns and helps law enforcement in prevention.

Smart Homes

AI makes daily life easier through smart automation.

- Devices like Siri, Google Assistant, and Alexa control smart home devices and answer queries.

- AI-enabled devices learn user habits to adjust settings automatically.
- AI optimizes energy consumption to reduce costs.

Robotics

AI powers various types of robots for industrial and personal use.

- Used for manufacturing, quality control, and packaging.
- Robots assist in hospitals, restaurants, and homes.
- Deployed in space exploration and underwater research.

Military and Defense

AI contributes to modern defence systems and strategic planning.

- AI assists in surveillance and reconnaissance missions using Drones.
- AI tools help military commanders plan operations.
- AI-based systems create realistic scenarios for military training.

1.6 AI PROBLEMS

In Artificial Intelligence (AI), problems refer to tasks or situations where a machine or computer system is required to make decisions, reason, plan, or learn in order to achieve a particular goal. These problems usually involve complexity, uncertainty, incomplete information, or require intelligent behavior that mimics human cognitive abilities.

AI problems arise when the solution cannot be obtained through simple programming or fixed instructions. Instead, these problems require methods that allow machines to think, adapt, and find solutions based on reasoning, learning, and decision-making.

1.6.1 Problem Characteristics in AI

Artificial Intelligence problems are not simple or linear. They involve multiple factors that make them challenging for machines to handle

without advanced techniques. Below are some key characteristics of AI problems:

- AI problems require dealing with many variables and factors at the same time. Real-world situations are complicated and involve multiple elements that must be considered together. For instance, an autonomous car needs to pay attention to traffic lights, nearby vehicles, pedestrians, road conditions, and weather — all at once. Another example is a game like chess, where the system must think through millions of possible moves and counter-moves before making a decision.
- In many cases, AI systems must work with incomplete or unclear information. This uncertainty may arise from sensor errors, missing data, unpredictable human actions, or sudden changes in the environment. For example, a medical diagnosis system may not have complete information about a patient's condition but still needs to make informed suggestions about possible diseases and treatments.
- AI systems often operate in situations that change over time. Unlike static problems, these systems must respond to real-time changes and adjust their actions accordingly. Predicting stock market trends is one such dynamic problem, as prices and conditions change rapidly. Similarly, a robot moving through a crowded space has to continually adapt its path based on how people around it are moving.
- AI systems must not only store data but also reason logically and improve over time. It's important for these systems to learn from past experiences and refine their actions without needing constant reprogramming. Technologies like machine learning and deep learning help AI systems analyze data, make decisions, and enhance their performance. For instance, recommendation

systems used by platforms like Netflix and Amazon learn from user preferences and feedback to provide smarter suggestions.

AI problems are distinguished by complexity, uncertainty, constantly changing conditions, and the need for logical reasoning combined with learning capabilities. Understanding these characteristics is essential for developing robust, adaptive, and intelligent AI systems that can perform in real-world scenarios.

1.6.2 AI Problems

Artificial Intelligence problems can be broadly categorized based on the nature of tasks they perform, the complexity involved, and the methods used to solve them. Below are some of the most common types of AI problems:

Search Problems

Search problems involve finding a solution by exploring different possible options.

- AI systems use search techniques to navigate large problem spaces and find the best solution.
- Examples include path finding in maps, solving puzzles, and playing games like chess or tic-tac-toe.
- Algorithms used: Breadth-first search, Depth-first search, A* algorithm, etc.

Knowledge-Based Problems

These problems require reasoning with large amounts of stored information (knowledge).

- The system needs to use facts and rules to come to logical conclusions.
- Examples:
    - Expert systems (such as MYCIN for medical diagnosis)
    - Question-answering systems that retrieve relevant answers from databases

- These systems rely on knowledge representation techniques like semantic networks, frames, or ontologies.

Planning Problems

Planning problems involve creating a sequence of actions to achieve a specific goal.

- AI systems must decide what actions to take and in what order to meet the goal efficiently.
- Examples include robot navigation, military mission planning, and scheduling airline flights.
- Planning algorithms help the system figure out future steps while handling constraints and priorities.

Reasoning Problems

Reasoning problems require drawing conclusions from known information.

- The AI system applies logic and inference rules to solve complex problems.
- Examples: legal decision-making systems, theorem provers, and diagnostic systems.
- Types of reasoning:
    - Deductive reasoning (from general rules to specific facts)
    - Inductive reasoning (from specific examples to general rules)

Learning Problems

Learning problems involve improving system performance through experience.

- These problems focus on enabling the system to learn patterns, predict outcomes, and adapt to new data.
- Examples:
    - Spam detection in emails
    - Recommender systems in Netflix or Amazon
    - Image and speech recognition

- Key techniques include machine learning, neural networks, and deep learning.

Perception Problems

These problems involve interpreting sensory data such as images, sounds, or text.

- AI systems are designed to see, hear, read, and understand human input.
- Examples such as Facial recognition systems, Voice assistants like Siri and Alexa, Optical character recognition (OCR) for reading printed text.

Natural Language Processing (NLP) Problems

These problems focus on understanding, interpreting, and generating human language.

- Examples include chatbots, translation tools, and sentiment analysis systems.
- NLP problems involve parsing sentences, extracting meaning, and generating responses that sound natural.

Robotics and Motion Problems

These problems involve AI systems controlling robots or machines that interact with the physical world.

- The robot needs to sense its environment, make decisions, and move accordingly.
- Examples such as Autonomous vehicles, Robotic arms in manufacturing and Drones for surveillance or delivery.

1.7 UNDERLYING ASSUMPTIONS IN AI

When designing and developing Artificial Intelligence systems, certain key assumptions are made to simplify complex real-world problems and make them manageable for machines. These underlying assumptions guide how AI systems operate, reason, and learn. Underlying assumptions in AI help simplify the design and

development of intelligent systems. These assumptions — such as rationality, the availability of models and data, finite problem space, and predictable environments — allow AI systems to reason, learn, and make decisions. However, understanding these assumptions is crucial because deviations from these ideal conditions require additional handling mechanisms like error correction, uncertainty modeling, and adaptive learning. Let's discuss these assumptions in detail:

AI assumes that the real world, despite its complexity, can be represented through models. These models use symbols, logic, rules, or data structures to represent knowledge about the world. For example, an AI system playing chess uses a model of the chessboard, possible moves, and outcomes. Even complex domains like medical diagnosis are modelled using symptoms, diseases, and treatments.

AI systems are built on the assumption that agents (the AI systems) will behave rationally. Rationality means the AI will choose actions that maximize its chances of success or achieving its goal. In uncertain conditions, rational agents use probabilities and reasoning to make the best possible decisions.

AI systems often assume that enough data is available for learning or decision-making. Machine learning and deep learning models, for instance, rely on large datasets to identify patterns. In reality, data may be incomplete or noisy, but AI systems are designed with the assumption that they can learn sufficiently from what they have.

It is assumed that any problem the AI system is trying to solve has a well-defined boundary and a finite set of possible solutions. This allows the AI to search for solutions without being overwhelmed by infinite possibilities. For instance, in a game of chess, the board and the pieces create a defined problem space.

AI algorithms assume that sufficient computational resources (processing power, memory, and time) are available to perform

complex calculations and learning tasks. The development of modern GPUs and cloud computing has helped meet this assumption in practice.

AI systems are often designed assuming they will receive accurate input from sensors or users. For example, an autonomous vehicle assumes its sensors are working perfectly to detect obstacles, though error handling is included as a backup.

AI systems operate under the assumption that the environment in which they function can be understood and that future outcomes can be predicted based on current knowledge. While real-world scenarios may be unpredictable, AI models try to handle uncertainty using probabilistic reasoning or learning from past patterns.

1.8 AI TECHNIQUES

AI techniques are methods and strategies used to design intelligent systems that can solve problems, make decisions, and learn from data. These techniques help AI systems think and act like humans in different situations. Let's discuss the major AI techniques in brief:

Search Techniques

Search is a basic technique used in AI to explore all possible solutions and find the best one. AI systems use search algorithms to solve puzzles, plan routes, play games, and make decisions. There are two types of search - **Uninformed Search** (No prior knowledge is used) and **Informed Search** (uses heuristics or knowledge to guide the search and reach solutions faster).

Knowledge Representation

AI systems need to store and organize knowledge in a form that machines can understand and use. The techniques include:

- Semantic Networks: Represent relationships between concepts.
- Frames: Use structures to represent stereotypical situations.

- Rules and Logic: Represent knowledge through if-then rules and logical statements.

Reasoning Techniques

Reasoning helps AI systems draw conclusions and make decisions based on available data. There are mainly three types of reasoning and they are:

- Deductive Reasoning: Drawing conclusions from known facts.
- Inductive Reasoning: Making generalizations from examples.
- Probabilistic Reasoning: Handling uncertainty using probabilities (e.g., Bayesian networks).

Machine Learning

AI systems learn from data and improve their performance over time without being explicitly programmed each time. Following are the types of machine learning:

- Supervised Learning: Learning from labeled examples.
- Unsupervised Learning: Finding patterns in unlabeled data.
- Reinforcement Learning: Learning through trial and error by receiving rewards or penalties.

Natural Language Processing (NLP)

This technique allows machines to understand, interpret, and respond to human language. Applications: chatbots, voice assistants, machine translation, sentiment analysis.

Expert Systems

These are AI systems designed to simulate the decision-making ability of human experts. They use large knowledge bases and inference rules to solve specific problems (e.g., medical diagnosis, legal advice).

Neural Networks and Deep Learning

Inspired by the human brain, neural networks are used for pattern recognition, speech processing, and image classification. Deep Learning uses multi-layered neural networks, powers applications

like facial recognition, autonomous vehicles, and advanced language models.

Fuzzy Logic

Fuzzy logic deals with reasoning that is approximate rather than fixed or exact. It is useful in situations where information is vague or incomplete, such as in control systems (like washing machines or air conditioners).

Genetic Algorithms

These algorithms are inspired by natural evolution and are used to solve optimization problems by evolving solutions over generations. Applications include scheduling, game strategies, and engineering design.

Check Your Progress-I

1. State True or False:

a) the primary goal of Artificial Intelligence (AI) is to create intelligent systems that can perform human-like tasks.

b) John McCarthy is considered the pioneer of AI and introduced the concept of the Turing Test.

c) Supervised Learning focuses on learning from labelled examples.

d) Lack of computational power and overestimated expectations caused the AI Winter in the past.

2. Fill in the Blanks:

a) The _____ was held in 1956, where the term "Artificial Intelligence" was officially coined.

b) AI systems use _____ reasoning to make logical decisions based on available data.

c) _____ is a technique in AI that helps machines understand and process human language.

d) AI models in healthcare help in _____ through medical imaging analysis.

e) _____ is an AI technique that uses evolutionary principles to find optimal solutions.

1.9 SUMMING UP

- AI focuses on creating machines that mimic human intelligence.
- Key capabilities of AI are - learning from experience, reasoning, problem-solving, understanding natural language, perceiving the environment, and decision-making.
- The application area of AI are:

Healthcare: Medical imaging analysis, personalized treatments, drug discovery, virtual health assistants.

Education: Personalized learning, automated grading, adaptive platforms.

Agriculture: Optimized planting, disease detection, crop yield prediction.

Finance and Business: Fraud detection, automated trading, customer service chatbots, market prediction.

Entertainment and Media: Content recommendations, content creation (music, art), intelligent gaming.

Transportation: Autonomous vehicles, traffic optimization, route planning.

Security and Surveillance: Identity verification, cyber threat detection, crime pattern analysis.

Smart Homes: Smart assistants (Alexa, Siri), habit-based automation, energy optimization.

Robotics: Industrial manufacturing, service robots, space and underwater exploration.

Military and Defence: Surveillance drones, strategic operation planning, realistic training simulations.

- The different AI Techniques are as follows:

Search Techniques: Uninformed and informed searches for finding solutions.

Knowledge Representation: Semantic networks, frames, rules, and logic.

Reasoning Techniques: Deductive, inductive, and probabilistic reasoning.

Machine Learning: Supervised, unsupervised, and reinforcement learning.

Natural Language Processing (NLP): Language understanding and response.

Expert Systems: Simulating decision-making of human experts.

Neural Networks and Deep Learning: Pattern recognition, speech/image processing.

Fuzzy Logic: Approximate reasoning for vague situations.

Genetic Algorithms: Optimization using evolutionary principles.

1.10 ANSWERS TO CHECK YOUR PROGRESS

1. a) True b) False c) True d) True

2. a) Dartmouth Conference b) Probabilistic c) NLP
d) Disease detection e) Genetic Algorithms

1.11 POSSIBLE QUESTIONS

Short Answer Type Questions:

1. What is Artificial Intelligence (AI)?
2. Name two key AI techniques used in problem-solving.
3. What is the purpose of the Turing Test?
4. How does AI help in transportation?
5. Why is reasoning important in AI?

Long Answer Type Questions:

6. Explain the history and evolution of AI.

7. Discuss the key application areas of AI.
8. Explain the different types of AI problems.
9. What are the underlying assumptions in AI?
10. Discuss the impact of AI on automation and industry.

1.12 REFERENCES AND SUGGESTED READINGS

1. Russell, Stuart J., and Peter Norvig. *Artificial intelligence: a modern approach*. pearson, 2016.

*****

UNIT-2: PROBLEM, PROBLEM SPACES

Unit Structure:

2.1 INTRODUCTION

In the last unit, we looked at the challenges that AI usually faces, which can be complex, engaging, and varied. In this unit, we will define these challenges in greater detail, analyze them to determine the several states that a problem might be in, and investigate different approaches to ultimately select the most effective problem-solving strategies that can be used to address the particular situation. The idea of state space search is one of the most well-known methods to arise in the field of artificial intelligence, which has been dealing with the problem of creating efficient problem-solving techniques. This unit will offer a thorough investigation of state space search, exploring its basic concepts along with its various AI applications.

2.2 UNIT OBJECTIVE

After learning this unit, student will be able to-

- Gain a basic understanding of search algorithms and state-space representation.
- Learn to define issues in terms of goals, states, actions, and transitions.
- Gain expertise in using search strategies including Minimax, A*, DFS, and BFS.
- Evaluate the efficiency and effectiveness of different search strategies.

2.3 DEFINING THE PROBLEM AS A STATE SPACE SEARCH

A primary approach in artificial intelligence is state space search, which conceptualizes a problem as a search for a solution inside a specified search space. The aim of the search space, which is

represented as a collection of states, is to find a series of steps that change the starting state into the intended target state. This method has been especially helpful in various fields, including robotics, natural language processing, and puzzle solving.

A problem can be formally represented as a state space search by using the following elements:

1. **State space (S):** Set of all possible states.
2. **Initial State(s_0):** The starting state.
3. **Goal States (G):** The set of acceptable goal state $G \subseteq S$.
4. **Actions(A(s)):** The set of available actions from states.
5. **Transition Models (T(s, a)):** The function that returns a new state when an action 'a' is applied to state 's'.
6. **Path Cost (C(s, a):** The cost of transition from a state 's' when an action 'a' is applied.

Let us understand it with a "Play Chess" problem.

1. **State space:** State space for a chess board consists of all possible configurations of the board. This is an approximate 10^{120} board configurations which drops to 10^{40} when all illegal moves are eliminated.
2. **Initial State:** Initial state is the standard starting position (Figure 2.1)

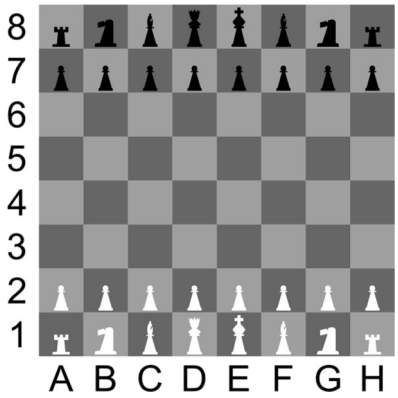


Fig 2.1 Initial State

3. **Goal States:** The set of acceptable chess board configurations that leads to either a win of one player or a draw.
4. **Actions:** Actions refer to any of the valid moves a player can make from a given state. Once a move is made, the board configuration changes according to the rules of chess.
5. **Transition Models:** Input to the transition model is the current board configuration and the move (Action), which results in a new board configuration.
6. **Path Cost:** Each move in a standard chess game is equally weighted, and thus there is no explicit path cost.

As we have seen in the case of chess games, representing approximately 10^{120}states is practically difficult and time-consuming. To overcome this problem, more general rules are used. Another way of representing the rule is as follows:

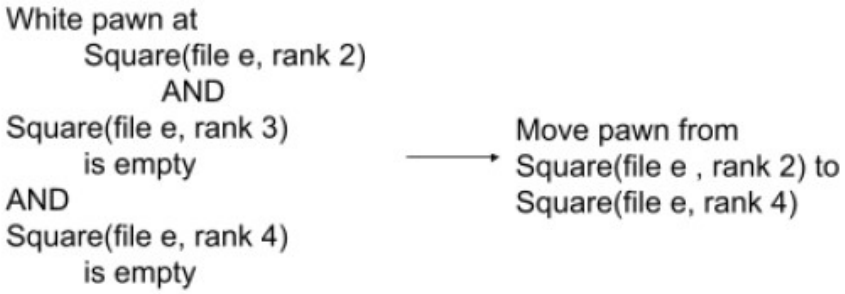


Figure 2.2: Another way to describe chess rule

Stop to Consider

State space search is a fundamental technique in artificial intelligence that frames problem-solving as navigating through a defined search space to reach a solution. It involves identifying steps that transform an initial state into a desired goal state. This approach is applicable across various disciplines, such as robotics, natural language

processing, and puzzle solving. A problem within this framework is formally depicted through several components: the state space, which encompasses all possible states; the initial state; a set of goal states; the available actions at each state; transition models, which describe the outcome of applying an action to a state; and path cost, which quantifies the cost associated with making a transition.

To understand the state space representation, let us explore another classical problem.

The Water Jug Problem: You are given two jugs with capacities ***A*** liters (say $A = 4$) and ***B*** liters (say $B = 3$). The objective is to measure exactly ***C*** liters (say $C=2$) of water in the jug with a capacity of ***A*** liter using the two jugs. You can perform the following actions:

1. Fill a jug completely.
2. Empty a jug.
3. Pour water from one jug into the other until one is empty or the other is full.

State space representation of the problem is as follows:

- **States:** State can be represented as a pair(x, y), where x is the amount of water in jug ***A*** and y is the amount of water in jug ***B.*** In our example, $0 \leq x \leq 4$ and $0 \leq y \leq 3$.
- **Initial State:** $(0\ ,0)$ where both jugs are empty.
- **Goal State:** (c, y), where jug ***B*** can have any amount of water while jug ***A*** will have ***c*** amount of water. In our case, ***C*** is 2 liters i.e. $(2, y)$.
- **Actions:** Various valid operations are presented in Table 2.1. These operations are represented as a rule where the current state will be matched with the left side of the rule, and the corresponding changes are made to the state to match the right

side. The process will be repeated in a cycle until the goal state is reached.

1.	(x, y) if $x < 4$	$(4, y)$	Fill Jug A
2.	(x, y) if $y < 3$	$(x, 3)$	Fill Jug B
3.	(x, y) if $x > 0$	$(x - d, y)$	Pour water from A to B.
4.	(x, y) if $y > 0$	$(x, y - d)$	Pour water from B to A
5.	(x, y) if $x > 0$	$(0, y)$	Empty jug A
6.	(x, y) if $y > 0$	$(x, 0)$	Empty jug B
7.	(x, y) if $x + y \geq 4, y > 0$	$(4, y - (4 - x))$	Pour from jug B in jug A till it is just full.
8.	(x, y) if $x + y \geq 3, x > 0$	$(x - (3 - y), 3)$	Pour from jug A in jug B till it is just full.
9.	(x, y) if $x + y \leq 4, y > 0$	$(x + y, 0)$	Pour all water from Jug B into Jug A
10.	(x, y) if $x + y \leq 3, x > 0$	$(0, x + y)$	Pour all water from Jug A into Jug B.

The approach to solving this problem is to identify the initial state and make that the current state. This current state is compared with the left-hand side. Once a match is found, necessary action is taken to make changes in the state that corresponds to the right-hand side. The process continues in the cycle and every time the resulting state is checked if it is one of the goal states. The efficiency of finding the

right solution depends on the mechanism used. One possible solution using a depth-first search (DFS) is presented below:

1. Initial state: (0,0).
2. Fill A: (4,0).
3. Pour from A to B: (1,3).
4. Empty B: (1,0).
5. Pour from A to B: (0,1).
6. Fill A: (4,1).
7. Pour from A to B: (2,3).

In step 7, we have reached the goal state. Although the solution to the above problem could be easily found, it may take a long search time if the search space is large and lacks an efficient control strategy. We will discuss some of the available mechanisms in the future. It has been found that in the above examples, we solve the problem by searching in the state space. ***Search*** is a general mechanism that can be used when no other direct method is known.

Check Your Progress

1. How will you define a problem as a state space problem?

2.4 PRODUCTION SYSTEM

The production system provides a structure that enables AI programs to describe a problem and perform search operations. The components of a Production system are as follows:

- **A Set of Production Rules:** The rules consist of a left side and a right side.
    - **Left side**: Specifies when the rule can be applied based on the current state.

    - **Right side**: Specifies the new state when the rule is applied.
- **Global Database:** Also known as the working memory, contains the required information for a particular task. Some parts of the memory may be permanent while others are updated continuously as the rule is applied.
- **A Control Strategy:** Control strategy determines the order of application of the rules. It also manages to resolve any conflict that may arise when more than one rule is applicable in a given state.
- **A rule applier**.

2.4.1 Control Strategies

In a production system, a control strategy oversees the selection, application, and execution of rules to successfully address an issue. It has already been discovered that solutions to the problem are dependent on the efficiency of the control strategy. The control strategy determines whether a solution can be reached and, if such solutions are possible, whether this solution is efficient enough. To guarantee that the production system functions effectively and achieves the intended goal state, an optimal control strategy must meet several characteristics.

1. ***A Control strategy must cause motion:*** "Motion" in the context of a production system is the application of production rules to move through the state space. This motion should transform the current state to a new state which is closer to the goal state. The motion should avoid getting trapped in loops. For example, if the rules of the water jug problem are applied in sequence, then it will never lead to a solution as the process

will be trapped in an infinite loop of filling and emptying the 4-litre jug.

2. ***A Control strategy must be systematic:*** One approach to overcome the water jug problem is randomly applying a rule. This approach may prevent the infinite looping issue but it does not ensure the efficiency of the solution as the same rule may get selected several times. Instead of applying rules randomly, a systematic control approach makes sure that the system moves through the state space in a logical and structured way. This helps ensure that the system reaches the desired goal efficiently, avoids redundancy, and doesn't get stuck in loops or irrelevant paths. Some systematic strategies are Breath-First Search (BFS) and Depth-First Search (DFS). These strategies are discussed in detail in later sections.
3. ***A control strategy must be efficient, flexible, and scalable:*** Besides the above-mentioned requirement, a control strategy must minimize computational overhead and avoid applying redundant and irrelevant rules. The control plan should adapt to dynamic changes in the problem or rules and scale effectively as the number of states and rules rises.

2.4.2 Breadth-First Search

Breadth-First Search (BFS) is an algorithm used to search for a state space or graph. It systematically explores all possible states level by level, starting from the initial state and gradually expanding outward. In BFS, it begins by creating a tree with the initial state as the root of the tree. Taking the root node as level 0, it will generate all its offspring at each level 1 by applying all applicable rules. In the case of the water jug problem, the root node is the initial state, (0,0), and generates the nodes of level 1 by applying the rules (Fig. 2.2). At

this state, only two rules are applicable, i.e. Rule 1(where we can fill Jug A) and Rule 2(where we can fill Jug B). The process continues to generate the next level (Fig. 2.3). To generate the next level, let us consider the left side, where the state is (4,0); three rules apply to this state, i.e., Rule 2 (fill Jug B), Rule 5 (empty Jug A) and Rule 3 (pour water from Jug A to Jug B). You may continue to create a complete tree.

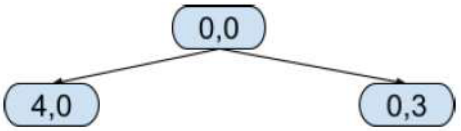


Fig. 2.2 First Level BFS Tree

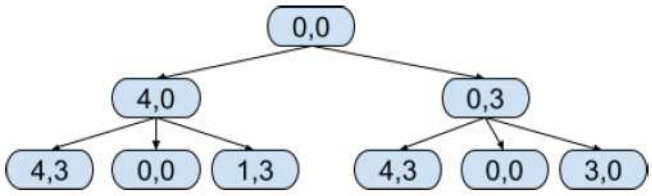


Fig. 2.3 BFS tree up to Second level

Steps in BFS:

1. Initialize the NODE-LIST (queue) with the starting state.
2. Initialize a visited list to keep track of states that have already been explored.
3. While the NODE-LIST is not empty:
    - Remove the first state from the queue (the state to explore).
    - If the state is the goal state, return the solution.
    - Otherwise, generate the possible successor states (states reachable from the current state), add them to the queue, and mark them as visited.
4. If the goal is reached, return the sequence of moves (the solution).

5. If the frontier is empty and no solution is found, the search fails.

Characteristics of BFS are:

1. **Level-by-Level Exploration**: Before going on to the next level, BFS investigates each state at a single "depth" or level.
2. **Optimality:** BFS will identify the shortest path to the goal (fewest steps) if the cost of shifting between states is constant.
3. **Complete:** If the state space is finite, BFS ensures that it will find a solution if one exists.

Advantages of BFS:

1. BFS will never get trapped in a blind alley. As we can see, BFS won't become stuck investigating an unproductive path endlessly; instead, it will only explore the next level after all the states in the current level have been examined. Later on, when we investigate depth-first search, we will fully understand it.
2. If a solution exists, BFS is guaranteed to find it.
3. BFS ensures that the shortest solution will always be found.

Disadvantages of BFS:

1. In large state spaces, BFS can consume a lot of memory.
2. BFS can also be inefficient when the state space is very large, as it explores every possible state.

Stop to Consider

BFS operates through level-by-level exploration, examining each state at a specific depth before progressing. This algorithm guarantees the identification of the shortest path to the goal when transition costs are uniform. Furthermore, in finite state spaces, BFS ensures solution discovery when one is available.

Check Your Progress

What is breadth-first search? Write down the steps for BFS.

2.4.3. Depth-First Search

Like BFS, another systematic control strategy that can be used to traverse the tree is Depth-first search. Unlike Breadth-First Search (BFS), which explores all states level by level, DFS traverses deeply into one branch of the state space before backtracking and exploring alternative branches. DFS uses a stack to keep track of states to be explored. It continues exploring the deepest level of the current branch until it either finds the goal state or reaches a dead end, at this point, it backtracks to explore other branches. One of the solution paths for the water jug problem could be as shown in Fig 2.4.

Steps of DFS

1. Initialize the stack with the initial state.
2. If the initial state is the goal state, return the solution.
3. While the stack is not empty:
    - Pop the top state from the stack.
    - If the state is the goal state, return the solution.
    - Otherwise, generate successor states and push them onto the stack.
4. If the stack becomes empty without finding the goal, the search fails.

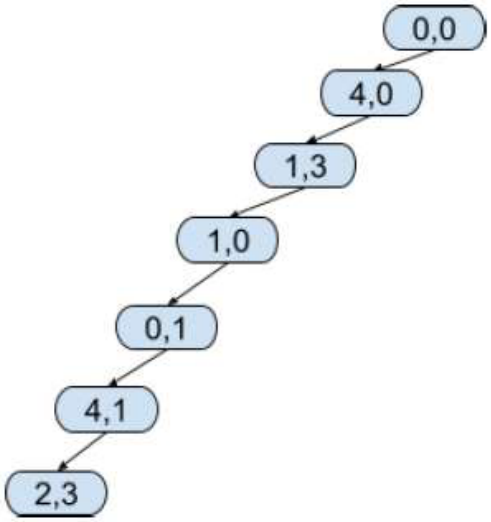


Fig. 2.4 One possible solution using DFS

Characteristics of DFS

1. **Explores Deeply**: Before exploring any alternative path, DFS thoroughly investigates one path as deeply as possible.
2. **Memory Efficient**: Since DFS just needs to store the current path and the next unexplored state, it uses less memory than BFS.
3. **Not Always Optimal**: DFS may discover a solution deep within one branch even if a shorter solution is available in another, hence it cannot guarantee the shortest path to the solution.
4. **Not Always Complete**: DFS may occasionally become trapped in endless loops.

Advantages of DFS

1. Since DFS does not have to keep every state at a given level, it uses less memory than BFS.
2. Thankfully, DFS will be able to locate a solution quickly if it decides to follow a path where one exists. When the answer is deep in the state space, DFS works well.

Disadvantages of DFS

1. There is no guarantee that the best solution will be found. If DFS investigates a deep branch before a shorter branch, it might discover a longer path to the answer.
2. If the state space contains cycles or if repeated states are not managed appropriately, DFS may become trapped in endless loops.

Stop to Consider

Depth-first search (DFS) is a strategy that explores one path extensively before considering alternatives. It is more memory-efficient than breadth-first search (BFS), as it only needs to retain the current path and the next unexplored state. However, DFS does not always guarantee the shortest path to a solution, as it may find a solution in a deeper branch while a shorter alternative exists elsewhere. Additionally, there are instances where DFS may get caught in infinite loops, making it not entirely complete.

Check Your Progress

1. What is depth-first search? Write down the steps for DFS.
2. How is breadth-first search different from depth-first search?
3. What are the advantages of BFS?
4. What are the advantages of DFS?
5. What are the disadvantages of BFS?
6. What are the disadvantages of DFS?

Using the above-mentioned strategies, we could solve the water jug problem. But, it may not be the case for other problems. Let us consider the problem of "The Travelling Salesman Problem".

The Travelling Salesman Problem (TSP): A salesman is required to visit a group of cities precisely once before returning to the

beginning location while reducing the overall cost of travel. Every pair of cities on the list has a direct path between them.

Theoretically, both Depth-First Search (DFS) and Breadth-First Search (BFS) can solve TSP, but they have major practical issues. The number of potential routes increases factorially with the number of cities in the Travelling Salesman Problem (TSP). Let us understand this with an example of five cities, (viz. A, B, C, D, and E) as shown in Fig 2.5. There are four cities left to pick from for the following trip if the starting city is A. There are still three cities to pick from after seeing one of those four, and so on. As a result, the cities' total number of permutations is:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$10! = 3628800$ is a big number if 10 (ten) cities are involved. A salesperson might travel to many more cities. The time needed to do this search is N! if N cities are involved. This phenomenon is called *combinatorial explosion*.

If BFS is used, it requires storing all nodes or states of the current level in the memory, this becomes infeasible due to memory limitation even for a moderate value of N. Similarly, DFS dives deep into one branch before exploring others. If the first branch explored is suboptimal, it will not find the optimal solution until all branches are explored.

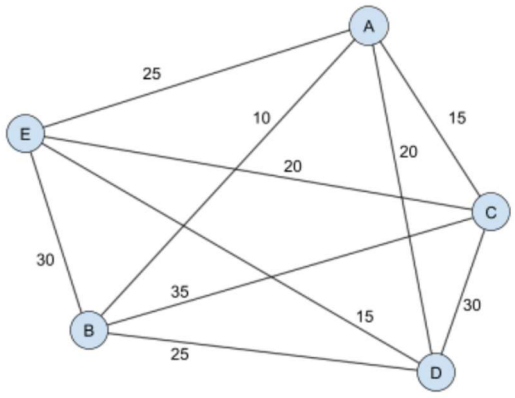


Fig 2.5 Cities to be covered by a salesman

To overcome this issue, several different approaches can be employed, including Branch and Bound, Heuristics (such as Nearest Neighbor), Dynamic Programming (Held-Karp Algorithm), Meta-heuristics, A* Search, etc. This unit will cover the first two options, and the following unit will cover A* Search.

Branch and Bound in TSP: In this simple strategy, we explore all the paths and keep track of the shortest path found so far. If the length of the incomplete path exceeds the path discovered thus far, stop investigating the path (pruning). Time is saved by avoiding the exploration of suboptimal routes. Although the method has several drawbacks, it is far more efficient than the previously stated strategies. Many nodes in the search tree are still explored, and in the worst scenario, it might still evaluate every state that could exist. For issues with a large number of cities, the algorithm might possibly run out of memory. In the example (Fig.2.5), the optimal route is found to be $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow A$ with the cost of 85.

Check Your Progress

1. Explain the Bound and Branch algorithm with an example.

2.5 HEURISTIC SEARCH

Heuristic search is an artificial intelligence (AI) technique that uses domain-specific knowledge to effectively tackle complex search and optimization issues. In many difficult issues, the systematicity and motion of the control strategy must be compromised, resulting in a good search result that may not be the best one. Even if it means sacrificing completeness, the goal of employing a heuristic is to improve search efficiency. Heuristic search uses a heuristic function to direct the search toward a goal state, in contrast to uninformed search techniques (such as Breadth-First Search or Depth-First

Search), which aimlessly traverse the search space. A heuristic is a problem-specific function or rule of thumb that estimates the cost or distance from a given state to the goal state. It helps prioritize which states to explore, reducing the computational effort required to find an optimal or near-optimal solution. Heuristics are frequently used to effectively identify approximations of solutions to the Traveling Salesman Problem (TSP). Among the most widely used heuristics is the Nearest Neighbor (NN) algorithm.

Nearest Neighbor Heuristics in TSP: This is a general purpose heuristics which and be used in travelling salesmen problem (TSP) to deal with the combinatorial explosion problem. In the TSP we can formulate the following procedure:

1. Start at any city at random.
2. Go to the nearest unexplored city (smallest edge cost) at each stage.
3. Continue until every city has been seen, then return to the beginning city.

For example, if city A is selected at first, it will take the path $A \rightarrow B$ (B being nearest to A with a distance of 10). From B, which has options C, D, and E, it will select D with a distance of 25, and the final selected path will be $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow A$ with the cost of 85. In the above solution, it happens to find the optimal solution, but it may not be the case always.

The advantage of using heuristics is that it is computationally efficient and easy to implement. It also offers a speedy solution although, in many situations, it may produce a suboptimal solution. The efficiency is achieved by drastically reducing the search space and therefore it can handle complex problems. A well-designed heuristics can efficiently provide the right direction to the search leading to a solution. Some limitations of heuristic search are inaccurate solutions

if poor heuristics are applied, and the need for domain knowledge, while some heuristics may even prioritize speed over optimality.

Stop to Consider

Heuristics offer significant advantages in computational efficiency and ease of implementation, making them suitable for complex problems by reducing the search space and providing quick solutions. However, they may yield suboptimal outcomes, especially if poorly designed, and often require domain knowledge. Additionally, some heuristics may favor speed over optimality, leading to inaccurate solutions.

Check Your Progress

1. Explain the nearest-neighbor heuristics for the traveling salesman problem.

2.6 PROBLEM CHARACTERISTICS

Applications of the appropriate heuristics methods and algorithms require us to understand the characteristics of the problem to which it will be applied. This categorization of the problems can help match problem-solving strategies to the underlying characteristics of the problem. Problems can be classified according to the following dimensions:

1. Is the Problem Decomposable?
2. Can Solution Steps Be Ignored or Undone?
3. Is the Problem's Universe Predictable?
4. Is a Good Solution Absolute or Relative?
5. Is the Solution a State or a Path?
6. What Role Does Knowledge Play in the Problem?
7. Does the Task Require Interaction with a person?

2.6.1 Is the Problem Decomposable?

Is it possible to divide the problem into more manageable, independent sub-problems that can be resolved independently and then merged to create a solution? Smaller problems can be solved easily as compared to larger complex problems. In the case of a larger problem, it can be broken down into smaller manageable problems. In the beginning, it checks if the problem is solvable. If the problem is solvable, it returns the solution. In case, the problem is not solvable, can the problem be decomposed into smaller problems and check if the resulting problem is solvable. The process is continued recursively until all solutions are found. This is followed by the integration of these solutions.

One example of a decomposable problem is to sort the list [38,27,43,3,9,82,10]. To solve this problem, it can be decomposed into two halves [38,27,43],[3,9,82,10]. Each halves can be recursively sorted as [27,38,43] and [3,9,10,82]. Finally merge the sorted halves to produce the result [2,9,10,27,38,43,82].

Not all problems are decomposable. One example of a non-decomposable problem is the block world problem (Fig. 2.6). Fig. 2.6 (a) shows the initial or start position, while Fig. 2.6(b) shows our Goal. Operation that can performed is to move only one block at a time. The nature of the problem is non-decomposable, as to move block C on A, C must be already in the correct position (i.e. C has no other block on top of it.) To do so, we must first move block D on the table. The moving of block C is dependent on the operation of block B. Because of these interdependencies, a problem cannot be solved by separating and solving its component elements separately.

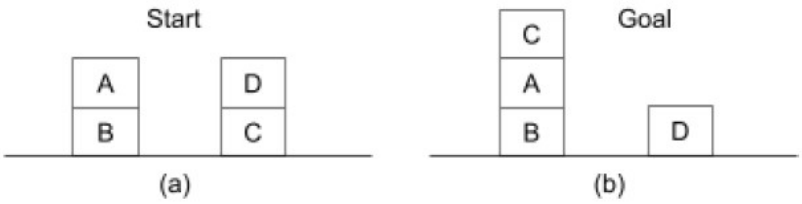


Fig. 2.6:The Block world problem

A precise series of steps must be followed to complete the solution, with each step establishing the prerequisites for the one after it. As an example:

- Step 1: Move D to the table.
- Step 2: Move A to the table.
- Step 3: Move C on top of A.

Skipping or solving steps in isolation leads to failure.

Since actions are closely related, a non-decomposable problem requires a holistic approach as every action causes the world's state to change dynamically. As we can see sub-problems, unlike sorting or factorial computation, cannot be resolved separately and then combined.

2.6.2 Can Solution Steps Be Ignored or Undone?

Let's say we're attempting to demonstrate a mathematical theorem. We start by proving a lemma that we believe would be helpful. We eventually concluded that the lemma is completely useless. Since the theory is still valid and in memory, if it ever existed, we have nothing to lose in this case. It is still possible to apply any rules that were initially applicable. We can simply move forward like we ought to have done initially. The only thing we've lost is the effort required to

investigate the blind alley. In this situation, we can simply ignore the steps taken to prove the lemma.

A well-known example of a problem where solution steps cannot be ignored is the 8-puzzle problem, which occasionally requires undoing to examine alternate options. A 3x3 grid with eight numbered tiles (1–8) and one blank area makes up the 8-puzzle (Fig. 2.7). From an initial configuration, the tiles must be moved by sliding into the vacant space to reach a predetermined goal configuration.

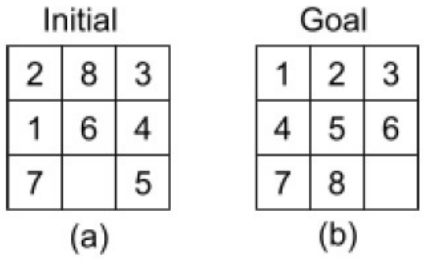


Fig. 2.7 8-Puzzle Problem

If a move is made and the move does not lead to the solution, we may have to backtrack to the initial state. The wrong steps taken have changed the initial configuration, thus we can not ignore the step as in the case of theorem proving. But we can undo the step to recover.

Let's use the example of a game of chess. We cannot reverse the action or deny that the step was never taken if we make a foolish move and later realize it. In this instance, the harm done cannot be undone. Thus it is irrecoverable.

So, we can say that in the theorem-proving problem, the solution steps can be ignored. In the case of 8-puzzle, solution steps can be undone. However, in the case of chess game, solution steps can not be ignored or undone.

2.6.3 Is the Problem's Universe Predictable?

The predictability of a problem's universe refers to whether the outcomes of actions or decisions in the problem-solving process can be determined with certainty. It is an essential characteristic that influences the choice of problem-solving strategies in artificial intelligence. Let's look at the previously described 8-puzzle problem. In this instance, we are aware of the outcome of every step taken. We can prepare ahead of time and take the required action to get to the desired state. There may be several moves that we need to undo to reach the desired goal. Thus the problem-solving strategy applied must have the capability of backtracking.

Similarly, in the game of chess, each move a player makes has a clear and predictable impact on the board. There are specific rules and no room for uncertainty about the movement of the pieces.

The problem has an unpredictable universe, when actions have probabilistic or numerous possible outcomes, the environment is subject to dynamic changes caused by external factors beyond the control of the problem solver, and we lack comprehensive information about the problem and its surroundings. One example of this type of problem is weather prediction, where forecasts are unreliable due to complex relationships between atmospheric variables.

Computational modeling is simpler for predictable problems. They frequently make it possible to solve problems using precise algorithms. Systematic investigation of the solution space is feasible due to the universe's predictability.

In an unpredictable universe, we must deal with uncertainty and incomplete knowledge. Decision-making under uncertainty or probabilistic reasoning is required.

2.6.4 Is a Good Solution Absolute or Relative?

A good solution's absolute or relative quality depends on the problem's characteristics and the criteria by which it is evaluated.

A solution is absolute if it is definitively correct and optimal. It is independent of external factors. For example, a mathematical problem has a definite answer; algorithms like Dijkstra's find the shortest path, which is the absolute optimal solution; sorting a list of numbers in ascending order has a single correct result.

A solution is relative if it is good enough or acceptable based on specific criteria, circumstances, or trade-offs and the solution is dependent on the context. There may be multiple acceptable solutions to a given problem and the "goodness" of a solution is judged in relation to other options or criteria. As seen in the traveling salesman problem, Nearest Neighbor heuristics may provide a relatively good solution within a reasonable time. When buying a car, price may be more important to you than comfort or fuel economy. Your criteria will determine the best answer.

Absolute solutions work best in domains that are well-defined and predictable. Relative solutions, however, are more useful for real-world issues where ambiguity, trade-offs, and limits are common.

2.6.5 Is the Solution a State or a Path?

Depending on the problem's nature and the definition of success, the answer may be a path or a state.

A solution is a state when the primary objective is to reach a specific goal configuration or state. The sequence of steps (path) or actions taken to reach the goal is not important. The end result (final state) fully satisfies the problem requirements. For example, in the 8-puzzle problem, the goal state is important where the tiles are arranged in the

correct order. The sequence of moves made to reach the goal is secondary. Similarly, in a chess game, a checkmate configuration is the solution, irrespective of how the position was achieved.

A solution is a path when a problem calls for determining not only the desired state but also the order in which decisions or activities must be performed to get there. The path or journey is just as significant as the final destination, if not more so. Every activity or step makes a significant contribution to the solution. The path (a series of cities) that minimizes the overall cost or distance of travel is the answer to the Traveling Salesman Problem (TSP).

2.6.6 What Role Does Knowledge Play in the Problem?

When it comes to problem-solving, knowledge is essential because it affects how efficiently and effectively a solution may be found. Consider a scenario in which you are unaware of the intended goal and the permitted moves (such as sliding tiles) in the 8-puzzle problem. Can you come up with a solution to the problem? Obviously not. Knowledge of the valid moves and the goal state is important for solving the 8-puzzle problem. In the Traveling Salesman Problem (TSP), knowledge of heuristics like the Nearest Neighbor helps estimate good solutions quickly. By removing impractical or unnecessary states and actions, knowledge aids in reducing the search space.

The foundation of successful problem-solving is knowledge. It clarifies the issue, directs the quest for answers, simplifies the process, and permits well-informed decision-making. A problem can be solved more quickly and effectively if there is more knowledge available and it is used effectively. Knowledge frequently makes the difference between success and failure when it comes to solving problems in both real-world and artificial intelligence scenarios.

2.6.7 Does the Task Require Interaction with a person?

The nature of the task and the extent to which it incorporates human judgment, preferences, or behaviors determine whether or not human contact is necessary. Consider the following task:

1. Arranging a list of numbers in ascending order.
2. Architects, designers, or software developers need to meet the design expectations of the stakeholders.

In the first task, the solution can be achieved autonomously through algorithms. No individualized or subjective input is required. The task can be repeated and yields reliable results. However, in the second task, decisions rely on human preferences or opinions of the stakeholders to refine requirements and ensure the design meets expectations.

The complexity of the activity, the level of interaction needed, and the importance of human judgment all influence whether human contact is necessary. Some jobs are intrinsically dependent on human involvement to ensure their successful completion, while others can be totally automated.

Check Your Progress

1. Explain the nearest-neighbor heuristics for the traveling salesman problem.
2. Discuss the seven problem characteristics to which a problem can be classified.
3. What are decomposable problems?
4. Give two examples, where solution steps can be ignored or undone.
5. Elaborate with an example where the solution is a path and not a state.
6. Differentiate between absolute and relative solutions.
7. What role does knowledge play in problem-solving?
8. Cite an example where human interaction is required to find a solution.

2.7 PRODUCTION SYSTEM CHARACTERISTICS

Just like the problems production system can also be classified based on the task and how the rules are applied. Production system can be classified into four main types:

2.7.1 Monotonic Production Systems

Monotonic production system is a production system in which application of a rule never prevents the later application of another rule that could have been applied at the time the first rule was selected.

It works well for issues where alterations to the current situation are merely additive and don't require going back and changing previous decisions. For example, each step in the mathematical theorem-proving process builds upon the one before it without making it invalid.

2.7.2 Non-Monotonic Production Systems

Non-Monotonic production system is a production system in which previously applied rules might need to be undone. As new information becomes available, the system permits facts to be retracted or modified. Ideal for situations that call for backtracking or a dynamic environment. For instance, in medical diagnosis systems, a diagnosis may be changed in response to the observation of new symptoms.

2.7.3 Partially Commutative Production Systems

A partially commutative production system is one that has the characteristic that, if a specific set of rules is applied, and state x becomes state y, then each possible permutation of those rules also

causes state x to become state y. The ultimate result is unaffected by the sequence in which the rules are applied. The same solution can be obtained by applying the rules in different sequences. It is employed when the final objective is clear and independent of intermediate steps. For example, consider robot path planning in an open grid, where multiple routes could go to the same location.

2.7.4 Commutative Production Systems

Production systems that are partially commutative and monotonic are known as commutative production systems. Each set of rule applications that results in a goal is legal and equally good. The system prioritizes reaching the objective over the route. Useful in systems where the cost and quality of every solution are the same.

It should be mentioned that any type of production system can address any problem. Some will be more efficient or more natural than others. It is worth mentioning that a commutative production system could be so time-consuming that it is essentially worthless. We have already explored the type of problem that can be solved by different types of production systems. Fig. 2.8 depicts four different types of production systems and the problems that can be solved by them.

	Monotonic	Non-monotonic
Partially Commutative	Theorem proving	Robot Navigation
Non Partially Commutative	Chemical Synthesis	Bridge

Fig. 2.8 Categories of a production system

The upper left corner represents the commutative systems. Theorem proving is partially commutative and monotonic as the sequence of

applicable rules does not affect the end goal and the application of one rule does not prevent the application of another rule. While navigating a robot, it does not matter if the route taken is Go_North->Go_East->Go_North or Go_North->Go_North->Go_East. But if the robot takes the wrong route, it must be able to backtrack. The order in which reactants are added may have a significant impact on the output for chemical synthesis.

2.8 DESIGN ISSUES OF SEARCH PROGRAMS

So far, we have seen that every problem can be seen as a traversal of the tree structure. Therefore, it is important how the tree is represented and structured. Selecting a suitable, effective, and expressive representation for states, activities, and goals is essential for an efficient search program. Inadequate representation may result in irrelevant branches or an excessively wide search area. It is also important to know whether this tree is constructed entirely from the production rules or it implicitly generate only those nodes it is going explore next. Creation of the entire tree and storing them in the memory is impractical for most cases. As search programs are responsible for the creation of these trees, it is important to keep in mind while designing these search programs.

We have seen in the examples above that the search program should find the path or paths from an initial state to the goal state in the search tree. However, we have not explored the direction in which a search is conducted, i.e., from the initial state to the final state (forward reasoning) or from the goal state to the initial state (backward reasoning).

Another issue while designing a search program is to create an efficient procedure to select the applicable rule (matching) as the

production systems typically spend a majority of their time looking for the applicable rule.

For simple problems, nodes can be represented as simple arrays, but for complex problems, the efficient representation (Knowledge representation issue) is critical for the search programs' performance.

At this stage, we should also think about the difference between search trees and search graphs. This may happen that the same nodes are frequently generated as part of many paths during the search process, meaning they are processed more than once. This occurs because the search space might not be a tree but rather a directed graph. Although it may be more efficient to treat the search process as a graph rather than a tree, doing so necessitates additional work each time a node is generated to determine whether it has already been generated.

Having a thorough understanding of these problems aids in choosing or creating the best search approach for a particular issue, guaranteeing that the solution is workable and efficient.

2.9 SUMMING UP

In this unit, we have learned

- How to represent a problem as a state space search problem.
- About the production system and its components.
- About uninformed search strategies like BFS and DFS.
- How to classify a problem based on its characteristics.
- About classification of production systems.
- About the issues encountered while designing the search program.

2.10 MODEL QUESTIONS

1. What is breadth-first search? Write down the steps for BFS.

Answer: Breadth-First Search (BFS) is an algorithm used to search for a state space or graph. It systematically explores all possible states level by level, starting from the initial state and gradually expanding outward. Steps in BFS are as follows:

1. Initialize the NODE-LIST (queue) with the starting state.
2. Initialize a visited list to keep track of states that have already been explored.
3. While the NODE-LIST is not empty:
    a. Remove the first state from the queue (the state to explore).
    b. If the state is the goal state, return the solution.
    c. Otherwise, generate the possible successor states (states reachable from the current state), add them to the queue, and mark them as visited.
4. If the goal is reached, return the sequence of moves (the solution).
5. If the frontier is empty and no solution is found, the search fails.

2.11 REFERENCES AND SUGGESTED READINGS

1. Elaine Rich, Kevin Knight, & Shivashankar B Nair, Artificial Intelligence, McGraw Hill, 3rd ed., 2009
2. Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Third Edition, Pearson, 2016

*****

UNIT-3: HEURISTIC SEARCH TECHNIQUES

Unit Structure:

3.1 INTRODUCTION

In the previous unit, we covered a few general-purpose search algorithms, such as bread-first and depth-first search. Although these straightforward approaches provided a basis for understanding search strategies, they were insufficient to address intricate issues within the realm of artificial intelligence. In this unit, we shall cover the other informed search techniques.

3.2 OBJECTIVES

This unit is an attempt to understand heuristic search techniques. After going through this unit, you will be able to -

- explain the concept of heuristics in search and problem-solving.
- Learn the working principle of heuristic search algorithms.
- Understand and analyze heuristic functions regarding admissibility, consistency, and efficiency.
- Apply Heuristic Techniques to real-world problems.

3.3 INFORMED VERSES UNINFORMED SEARCH

It was also seen in the "Traveling salesman problem" that adopting simple heuristics like nearest-neighbor heuristics can help guide the search towards the goal. Heuristic search is an artificial intelligence technique that uses domain-specific knowledge to solve complex search issues effectively. By prioritizing which states to investigate, it lessens the amount of computation required to find the optimal or nearly optimal answer. Whether or not search techniques incorporate problem-specific knowledge (heuristics) to direct the search classifies them as informed or uninformed search strategies. Uninformed or blind search methods use the problem specification to explore the state space; they lack additional knowledge about the goal state. Only the target condition and the search space structure are used. Informed (or heuristic) search techniques employ heuristics or domain-specific information to direct the search process toward its goal more effectively. Some of these techniques are shown in Fig. 3.1.

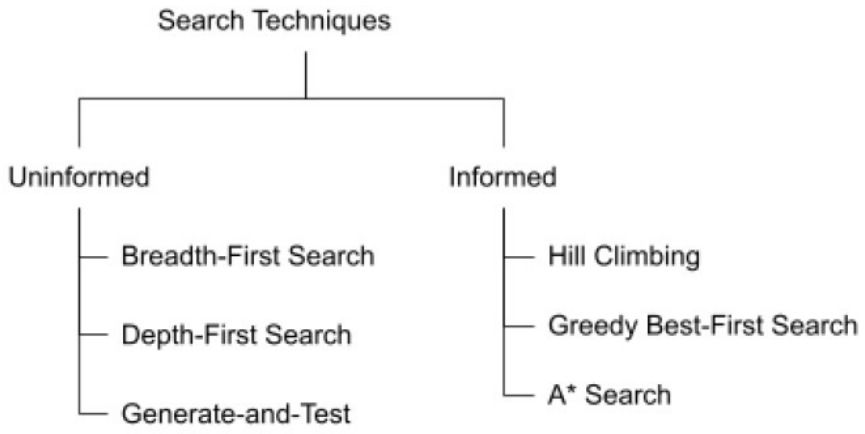


Fig. 3.1 Search Techniques

Among uninformed search techniques, we have already discussed breadth-first search and depth-first search in the previous unit. We will discuss the generate-and-test and other informed search techniques in this unit.

Self-Assessment Questions

Differentiate between Informed and Uninformed Search.

……

……

……

……

3.4 GENERATE AND TEST

The generate-and-test algorithm is a simple problem-solving method in which possible solutions are generated either randomly or systematically and then tested to see if they satisfy the intended target condition. Many AI systems are built using this approach, among the most basic search techniques.

Algorithm: Generate-and-test

1. Generate a new possible solution. The type of problem determines the solution generation process, which might involve methodically investigating every option.
2. Check if the generated solution meets the desired outcome (Goal). Return the solution and end the process if it satisfies the requirements.
3. If the solution is not the goal state, return to step 1 to generate another solution. Continue until a solution is found or all options are exhausted.

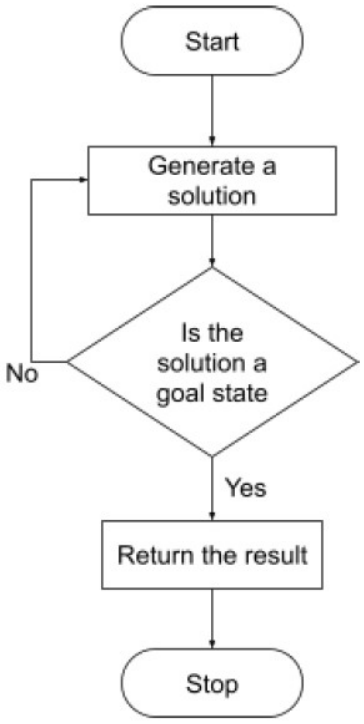


Fig. 3.2 Flow chart of Generate-and-test

Let us understand the generate-and-test algorithm with the help of the 8-puzzle problem discussed earlier. We are given the initial state

in Fig. 3.3(a), and the goal state is in 3.3(b). Operations that can performed is to move the tiles. Each movement of the tile creates a new state. For convenience, instead of moving all the tiles, let us consider only the blank tile. A blank tile can move in four directions if it is at the center, in three directions if it is on any of the sides, and in two directions if it is on any of the corners.

We can proceed to solve the problem by performing the following steps.

1. Generate a new state by arranging the tiles.
2. Check if the present configuration corresponds to the goal state. If it does, return the arrangement as the solution.
3. If not, generate another state using a different arrangement and test again.

The generate-and-test algorithm is similar to a depth-first search with backtracking because a complete solution must be generated before it can be tested. It is merely a thorough search of a problem space in its most systematic form. If states are created at random, a solution may never be found.

This method has the benefit of being simple to comprehend and apply. It doesn't require any heuristics or prior information. Any problem with a well-defined goal can use this strategy. It offers a standard by which more complex search strategies can be measured. The drawback of this technique is that it is computationally expensive because of its exhaustive nature; it frequently involves blindly examining a vast number of states. As no guidance or heuristic is available to lessen the search effort, it does not prioritize a more promising solution over a less promising one. Because of the search space size, this method is not feasible for large or complex issues.

Stop to Consider

The generate-and-test algorithm is a fundamental problem-solving method. Potential solutions are created randomly or systematically and evaluated to check if they meet the criteria of goal state. This approach is frequently utilized in artificial intelligence systems and represents one of the most basic search techniques.

3.5 HILL CLIMBING

In the generate-and-test algorithm, once a new state is generated using any available operations, it checks if the generated state is the goal state. If it is not the goal state, the algorithm cannot check if the generated state is closer to the goal state. In hill climbing, we are provided with feedback from the test procedure, which determines the direction in which we should move. Here, the test function is augmented with a heuristic function that estimates whether the generated state is closer to the goal state. If the generated state is not the goal state but it may be better than the current state, it makes the new state the current state and proceeds. Popularity of Hill Climbing algorithm, resulted to create multiple variants each designed to overcome the shortcomings of the original method and enhance its functionality in different problem scenarios. The predominant versions of the Hill Climbing algorithm are presented below.

3.5.1 Simple Hill Climbing

This is the most fundamental type of hill climbing. Simple to implement. It requires less computation since it stops testing neighbors once the better state is discovered. The algorithm can be presented as follows:

Algorithm: Simple Hill Climbing

1. Evaluate the initial state. If it is also the goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:
    a. Select an operator that has not yet been applied to the current state and apply to produce a new state.
    b. Evaluate the new state.
        i. If it is the goal state, then return it and quit.
        ii. If it is not a goal state but better than the current state, then make it the current state.
        iii. If it is not better than the current state, then continue in the loop.

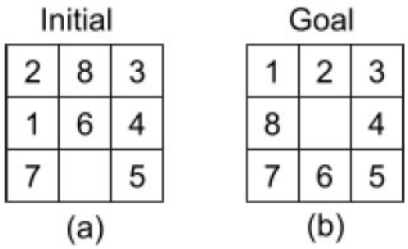


Fig. 3.3 The 8-puzzle problem

Let us solve the 8-puzzle problem discussed above using simple hill climbing. We will use a simple heuristic here; say we give a value of 1 for every incorrectly placed tile and a value of 0 for the correctly placed tile. We add these values to give a heuristic value to the state. Since all the tiles in the goal state are correctly placed, the heuristic value will be 0. In the case of the initial state, four tiles viz., 2, 8, 1, and 6 are incorrectly placed. Therefore, the heuristic value of the initial state is 4.

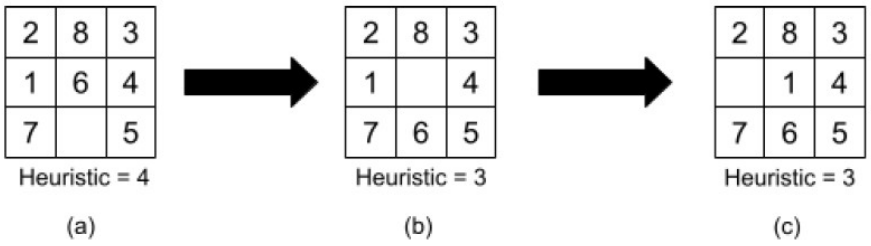


Fig. 3.4 Simple Hill Climbing on 8-puzzle

In simple hill climbing, the initial state is evaluated to see if it is a goal state. In our case, it is not the goal state. Therefore, it produces the next state by moving the blank tile (say, to the top). The new state produced is evaluated. Since the new state is not the goal state, it computes the heuristic value of the new state, which is 3 (At this state, three tiles are incorrectly placed, 2,8 and 1). Since the heuristic value of the new state is closer to the goal state than the initial state, it makes the new state the current state. At this point, it produces the next state by moving the blank tile. It has four different operators at this moment. Suppose it takes the move left option, and then the heuristic value does not improve. Since the new state is not better than the current operation, it returns to the loop and applies the next operator. You will find that none of the new states is better than the current state. The algorithm stops at this stage. This could be because of the heuristic function we use.

We can use another simple heuristic called “Manhattan Distance” to address this problem. Here, we determine the total horizontal and vertical distances between each tile's current location and its desired location. The total Manhattan Distance for every tile provides the heuristic value. The Heuristic value for the initial state (Fig. 3.5 (a)) will be 5 (Manhattan distance for Tile 2 = 1; Tile 8 = 2; Tile 1 = 1; Tile 6 = 1; Total = 5). Suppose it applies the same operation as above; then the heuristic value of the new state (Fig.3.5(b)) will be 4, which is better than the current state, and it makes the new state the current state. The following state produced by moving the blank

tile up again produces a better new state with a heuristic value of 3 (Fig.3.5(c)). Heuristic value improves in the next two steps (Fig. 3.5 (d) &(e)). At this time, Fig.3.5 (e), as the current state, produces a new state that evaluates to be the goal state, and the algorithm terminates.

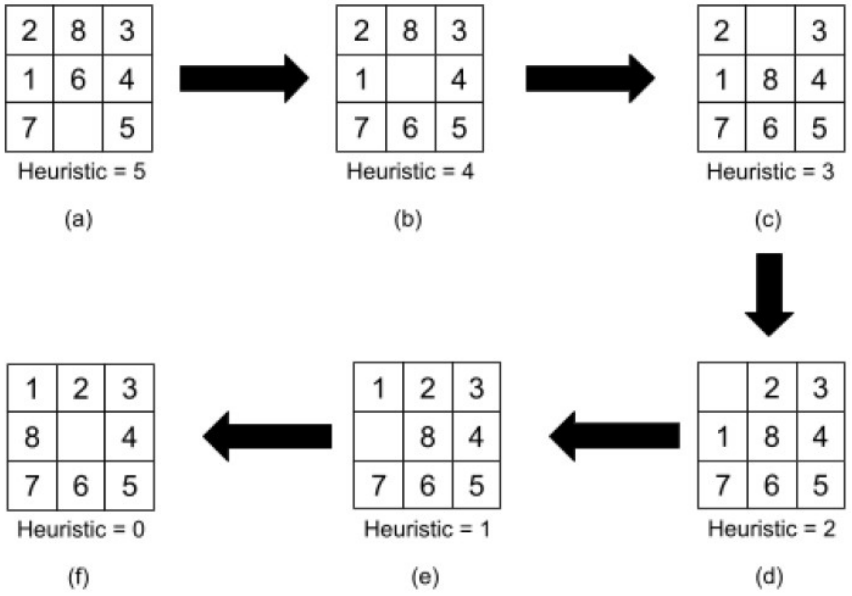


Fig. 3.5 Simple Hill Climbing on 8-puzzle using Manhattan distance heuristics.

Finally, we have used simple hill climbing to solve the 8-puzzle problem. We have also discovered that the algorithms cannot function without a suitable heuristic function.

Stop to Consider

Simple Hill Climbing is a local search algorithm that iteratively moves to a neighboring state with a better heuristic value, stopping when no improvement is possible. It evaluates only one neighbor at a time, making it straightforward but prone to issues like getting stuck in local maxima, plateaus, or ridges.

3.5.2 Steepest-Ascent Hill Climbing

The Steepest-Ascent Hill climbing is a variant of the Simple Hill Climbing algorithm. Steepest-Ascent applies all available operators from the current state and assesses them to choose the one with the greatest improvement in the heuristic value (i.e., the steepest ascent). It contrasts Simple Hill Climbing, which chooses the first better state. The algorithm can be presented as follows:

Algorithm: Steepest-Ascent Hill Climbing

3. Evaluate the initial state. If it is also the goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
4. Loop until a solution is found or until a complete iteration produces no change to the current state:
    a. Let SUCC be a state such that any possible successor of the current state will be better than the SUCC.
    b. For each operator that applies to the current state, do:
        i. Apply the operator and generate a new state.
        ii. Evaluate the new state. If it is the goal state, then return it and quit. If not, compare it to SUCC.

            If it is better, then set SUCC to this state. If it is not better, leave SUCC alone.
    c. If the SUCC is better than the current state, then set the current state to SUCC.

If we revisit the previous example of the 8-puzzle problem, Steepest-Accent Hill climbing will apply all operators to the current state. At this point, the blank tile can be moved three ways (Left, up, and Right). The heuristic values of the new states are shown in

Fig.3.6. The second option of moving the blank tile up produces a state with the heuristic value closest to the goal state, which will be selected as the current state.

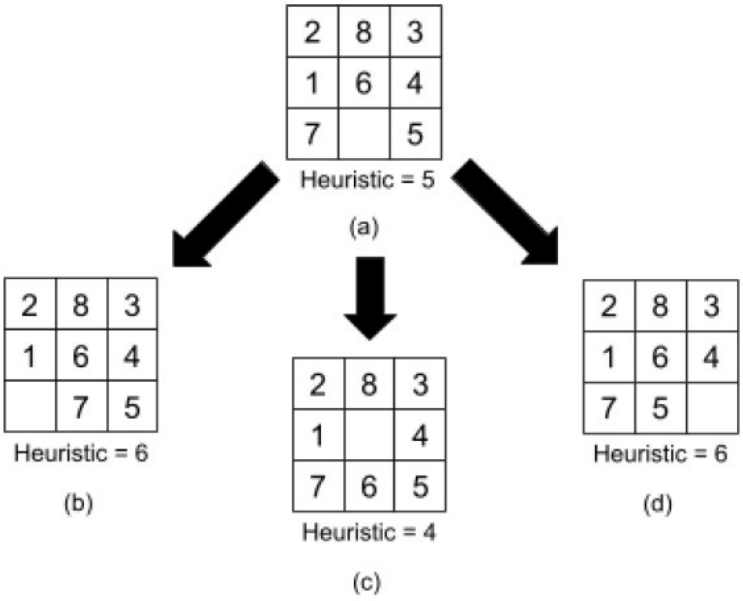


Fig. 3.6 Steepest Hill Climbing on 8-puzzle problem

As it chooses the state with the best heuristic value, it always moves towards the steepest improvement and arrives at an optimal solution (global maximum/minimum). It is computationally expensive as it evaluates all new states in each step. Although the search is thorough compared to simple hill climbing, it is slower.

Stop to Consider

Steepest-Ascent Hill Climbing is an advanced version of the Simple Hill Climbing algorithm. Unlike Simple Hill Climbing, which opts for the first available state that shows improvement, Steepest-Ascent evaluates all possible moves from the current state and selects the one that offers the maximum increase in heuristic value, effectively identifying the steepest ascent.

With basic heuristics, Simple and Steepest-Ascent Hill Climbing are easy to implement and efficient for smaller problems. However, they might not be able to solve a problem. When the program reaches a state where no better states can be produced, either algorithm could terminate without discovering a goal state. This will occur when the program reaches a local maximum, plateau, or ridge. We will try to understand these terms in the next section.

Check Your Progress

1) State True of False

a. Any problem with a well-defined goal can use Generate and Test strategy.

b. Hill climbing technique is very hard to implement and takes a lot of computation time.

c. Heuristic search is an artificial intelligence technique that uses domain-specific knowledge to solve complex search issues effectively.

3.5.2.1 Local Maximum, Plateau, and Ridge

Local Maximum: In the search space, a local maximum is a position where the present solution is better than all its nearby neighbors, but it is not the best option (the global maximum). This is a typical issue with search algorithms, particularly with Hill Climbing and related methods. We faced a similar problem while using Hill Climbing with simple heuristics. Fig.3.7 illustrates the local maximum vs global maximum. When a local maximum occurs, the search process stops before finding the best answer. The solution that was discovered might not be the most optimal one.

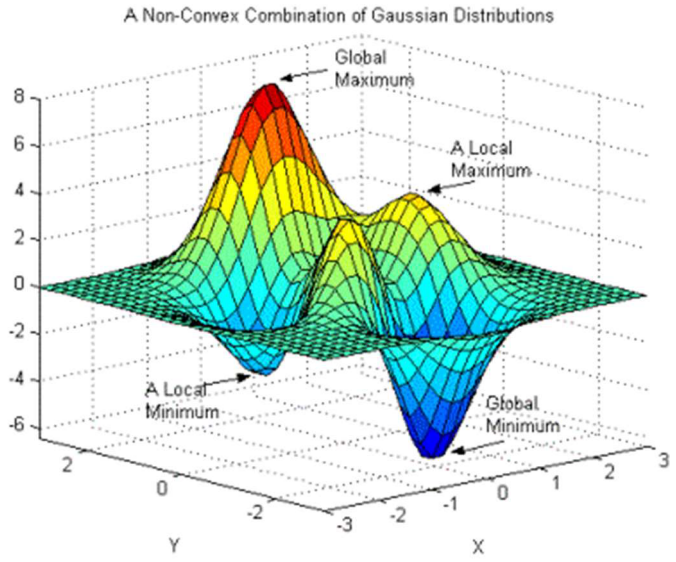


Fig. 3.7 Local Maximum vs Global Maximum [3]

Plateau: A plateau is a flat area in the search space where the heuristic function has the same value for a set of surrounding states. When an algorithm reaches a plateau, all steps seem equally excellent (or equally terrible), making it impossible to tell which path will improve. When a plateau emerges, the algorithm is unable to decide how to proceed; it may wander aimlessly on the plateau or come to a complete halt, failing to locate its goal.

Ridge: Ridge represents a special kind of local maxima. It is a section of the search space that has a slope and is higher than the surrounding areas. However, it is not possible to cross a ridge with a single move due to the high region's orientation in regards to the set of moves that are available and the direction in which they move. The algorithm could waste time looking in other parts of the search space.

3.5.2.2 Dealing with Local Maximum, Plateau, and Ridge

In this section, we will discuss some strategies for dealing with these problems. However, in most situations, these strategies do not ensure success.

- **Backtrack:** Backtracking is a systematic process that involves moving back to the previous state and iteratively trying other alternative paths in the search space that may be equally promising. Backtracking can be helpful since it enables the search to reverse actions and investigate several routes that could ultimately lead to the global maximum. Backtracking works best with local maximum, preventing it from getting trapped in a dead end.
- **Jumping:** An effective strategy to deal with the problem of plateau is to take a big jump to a new section of the search space. Apply the available rules repeatedly in the same direction if they only describe single steps.
- **Applying more rules:** To address the ridge problem, we can apply two or more rules before testing. As movement takes place in more than one direction at once, this prevents the algorithm from wandering down a less promising path.

Check Your Progress

2) Fill in the blanks

a) _____works best with local maximum, preventing it from getting trapped in a dead end.

b) _____represents a special kind of local maxima.

c) A plateau is a flat area in the search space where the_____ has the same value for a set of surrounding states

3.5.2 Simulated Annealing

Simulated Annealing is a variation of hill climbing. It is a probabilistic search technique inspired by the annealing process in metallurgy, which involves heating and then cooling materials gradually to develop a crystalline structure with the least amount of energy. To get around obstacles like local maxima or plateaus, simulated annealing is used to explore the solution space and obtain an approximate global optimum in search problems.

Throughout this section, we make two notational adjustments to conform to the standard usage in discussions of simulated annealing. The phrase "heuristic function" is substituted with "objective function." Additionally, we minimize the objective function's value rather than maximize it. Therefore, instead of a hill-climbing process, we describe a valley-descending process. Before further exploring the algorithm, a few key terms need to be introduced. They are temperature, energy, cooling schedule, and acceptable probability.

This simulated annealing process mimics the physical process of annealing, where the ***temperature*** regulates the likelihood of accepting worse solutions to break out of local maximum.

Energy indicates a solution's quality, often derived from an objective function (minimization problems benefit from reduced energy).

Following a ***cooling schedule***, the temperature begins high and progressively drops. The speed at which the algorithm moves from state to state depends on the cooling schedule.

Simulated annealing allows movement to worse solutions with a probability P based on the temperature and the difference in energy:

$$P = e^{\frac{-\Delta E}{T}}$$

Where, ΔE is the change in energy $(E_{new} - E_{current})$ and T is the current temperature.

Exploration is encouraged by higher temperatures since they make it more likely to accept worse options.

Algorithm: Simulated Annealing

1. Evaluate the initial state. If it is also the goal state, then return it and quit. Otherwise, continue with the initial state as the current state.
2. Initialize *BEST-SO-FAR* to the current state.
3. Initialize *T* according to the annealing schedule.
4. Loop until a solution is found or until there are no new operators left to be applied in the current state.
    a. Select an operator that has not yet been applied to the current state and apply it to produce a new state.
    b. Evaluate the new state. Compute

$$\Delta E = (E_{new} - E_{current})$$

- If the new state is a goal state, then return it and quit.
- If it is not a goal state but is better than the current state, then make it the current state. Also, set *BEST-SO-FAR* to the new state.
- If it is not better than the current state, then make it the current state with probability $P = e^{\frac{-\Delta E}{T}}$.

    c. Revise T as necessary according to the annealing schedule.
5. Return *BEST-SO-FAR* as the answer.

For the algorithm to be successfully implemented, the initial temperature, cooling schedule, and termination condition must be selected.

Stop to Consider

By simulating the annealing process and striking a balance between exploration and exploitation, simulated annealing can uncover near-optimal solutions when local optima causes traditional methods to fail.

3.6 BEST-FIRST SEARCH

In the previous unit, we explored uninformed search techniques such as breadth-first search and depth-first search. We also realized that a breadth-first search has the advantage of never getting trapped in a blind alley. In contrast, a depth-first search is more memory efficient as it can find an optimal solution without exploring all paths. In the Best-first search, we combine the advantages of both these search techniques.

Best-first search is an informed search technique using a heuristic function to determine the most promising state generated so far. It explores the promising path while remembering the other unexplored states generated earlier. It then generates its successors to see if it is the goal state. If it is not the goal state, the generated states are added to the list of unexplored states. The most promising state is selected among the unexplored states. This results in exploring the promising path using depth-first search, and if the path looks less promising in the later state, it has the option of selecting a less promising path that was ignored earlier. This contrasts with the steepest hill climbing algorithm discussed in the previous section,

where it explores the most promising path and ignores the other less promising one.

We can understand the Best-first search with the following example. Let A be the initial state. In the beginning, A is in the list of unexplored states. It checks if A is the goal state. Since A is not the goal state, it generates its successors B, C, and D. The Heuristic function used in the example is the estimated cost from the node to the goal state. The lower value of the heuristic function gives the most promising state. The goal state has a heuristic value of 0 (Zero). As D has the lowest heuristic value, i.e., the lowest estimated cost to reach the goal state, it is selected and expanded. At this stage, another path from B looks more promising. Therefore, the B is expanded to generate G and H. After step 4, E looks promising and expanded to produce I and J. J is the goal state, so the algorithm terminates and returns the path A -> D -> E -> J.

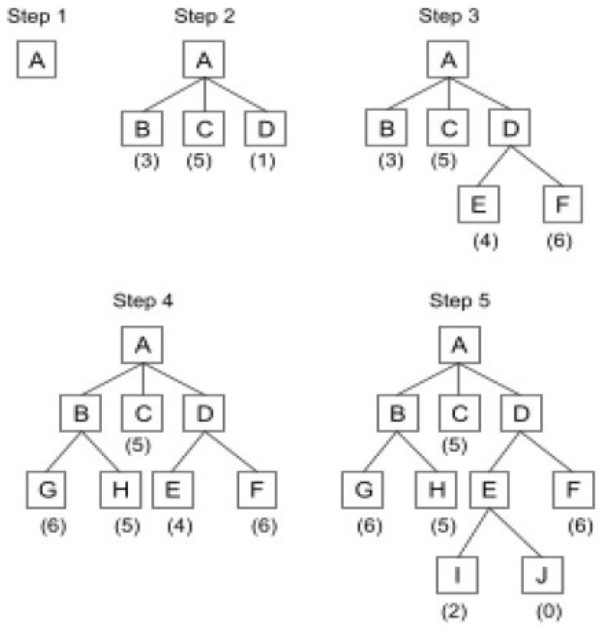


Fig. 3.8 Best-first search

In the above example, we have seen the working of the best-first search algorithm for a search tree. However, the same can be used to

search a directed graph when some nodes appear in multiple paths. To implement a graph search, we need to maintain two lists:

- OPEN list: This list contains all the nodes that have been generated, and the heuristic function is applied but not explored yet. OPEN list is a priority queue where more priority is assigned to the node, which is more promising than the other.
- CLOSED list. This list contains all the nodes that have already been examined. It prevents previously examined nodes from being revisited or expanded, which aids in effectively managing the search.

Algorithm: Best-First Search

1. Start with the initial state and add it to the OPEN list.
2. Continue until the goal is found or no nodes are left in the OPEN.
    a. Remove the node with the lowest heuristic value from OPEN.
    b. If it is the goal, terminate the search and return the path.
    c. If it is not the goal, generate all successor nodes of the current node and add them to the OPEN list after evaluating its heuristic values.

In the following example, we are given a simplified Romania road map (Fig.3.9). Our goal is to reach the city of Bucharest, provided that we are in the town of Arad. The available route and the actual distances between the cities are also shown on the map. We refer to this distance as $g(n)$. It is the exact distance from the starting town to the current one.

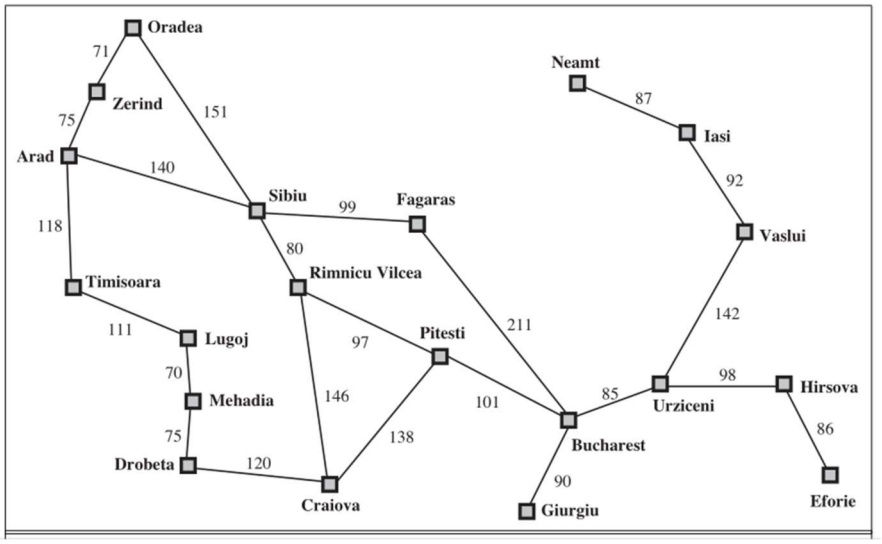


Fig. 3.9 Simplified road map of Romania [2]

Our heuristic function for the route-finding problem is the straight-line distance from the city to the goal state (Bucharest). We refer to it as $h'(n)$. This can be easily calculated using the coordinates of the cities. To keep it simple, we avoid any mathematical calculation and use the heuristic values given in Fig. 3.10. The function $f'(n)$ used in the Best-first algorithm uses only the heuristic function $f'(n) = h'(n)$.

The algorithm starts by expanding Arad (Fig 3.11(a)). Sibiu was selected for expansion as being the most promising of the three (Fig.3.11(b)). The next promising node is Fagaras, which, on expanding, reaches Bucharest, the goal state.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Fig. 3.10 Straight-line distance to Bucharest [2].

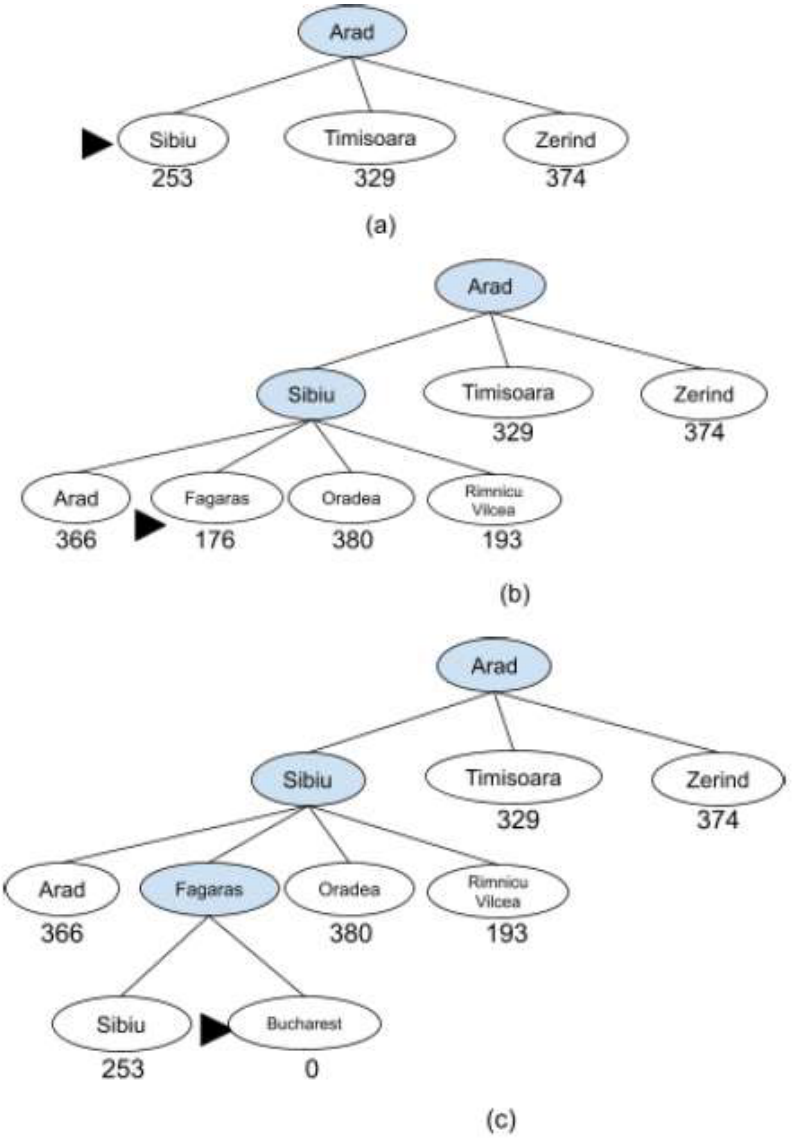


Fig 3.11 Steps of Best-First Search on Route-finding Problem.

In the example, it is working like a depth-first search. Therefore, like the depth-first search, it is incomplete. It might get caught in a dead end. Performance is highly dependent on the quality of the heuristic function. Because of the greedy nature, it does not produce an optimal path either, as we will explore in a later example, that a shorter path does exist.

Stop to Consider

Some characteristics of the Best-First Search are:

1. Its performance is heavily influenced by the heuristic function used.
2. It is incomplete.
3. It is not optimal.

3.7 A* ALGORITHM

A* search, pronounced "A-star search," is the most popular type of best-first search. It assesses nodes by adding the costs of getting to the node ($g(n)$) and the heuristic function ($h'(n)$). As discussed earlier, $g(n)$ this is the exact (actual) cost from the starting node to the node n and $h'(n)$: the estimated cost from the node n to the goal state. Therefore, the combined function $f'(n) = g(n) + h'(n)$.

A* always provides an optimal solution, and it is complete. The algorithm is presented below:

Algorithm: A* Search

1. Start with the initial state and add it to the OPEN list $f'(initial\_node) = g(initial\_node) + h'(initial\_node)$. The CLOSED list is empty.
2. Continue until the goal is found or no nodes are left in the OPEN.
    a. Select the node with the lowest $f'(n)$ from the OPEN list.
    b. If the selected node is the goal, return the path and terminate.
    c. If it is not the goal, generate all successor nodes of the current node
    d. For each successor,

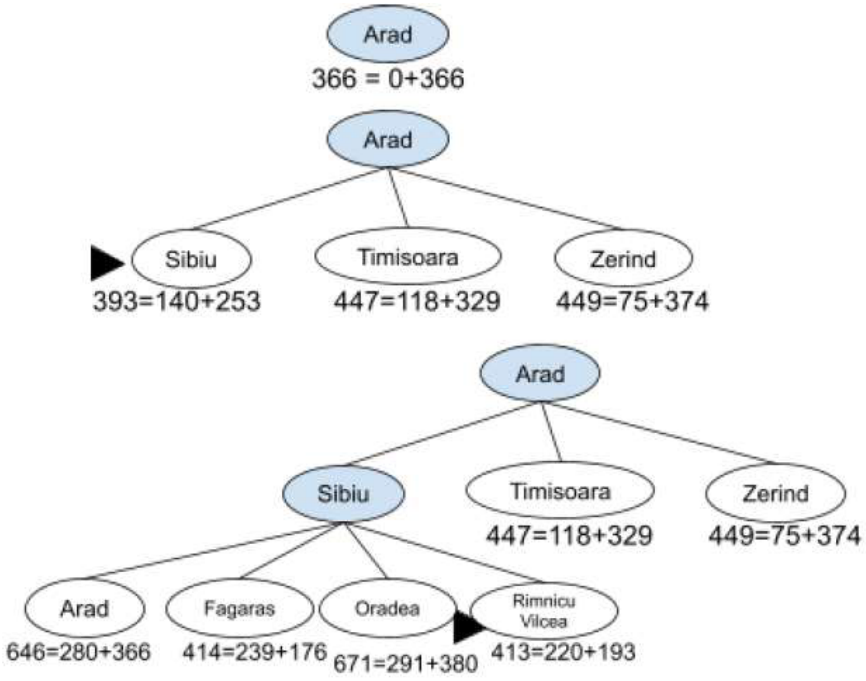
i. Evaluate the $f'(successor) = g(successor) + h'(successor)$.

ii. If the successor is not in the OPEN or CLOSED list, add it to the OPEN list.

iii. If the successor is already in the OPEN list with a higher f'-value, update its cost and parent.

3. Remove the expanded node from the OPEN list and place it in the CLOSED list.

We will demonstrate the use of A* search on our route-finding problem (Fig.3.9). We will calculate $g(n)$ from the exact distance given in the figure. The steps of the algorithm are presented in Fig. 3.12. We begin with the initial state, i.e., Arad $f'(n) = 0 + 366 = 366$. After evaluating Arad, it is expanded and is moved to the CLOSED list. Expanding Arad, Sibiu has been selected as it is the most promising one and evaluated. Since it is not the goal state, its successors are generated. All successor nodes are added to the OPEN list.



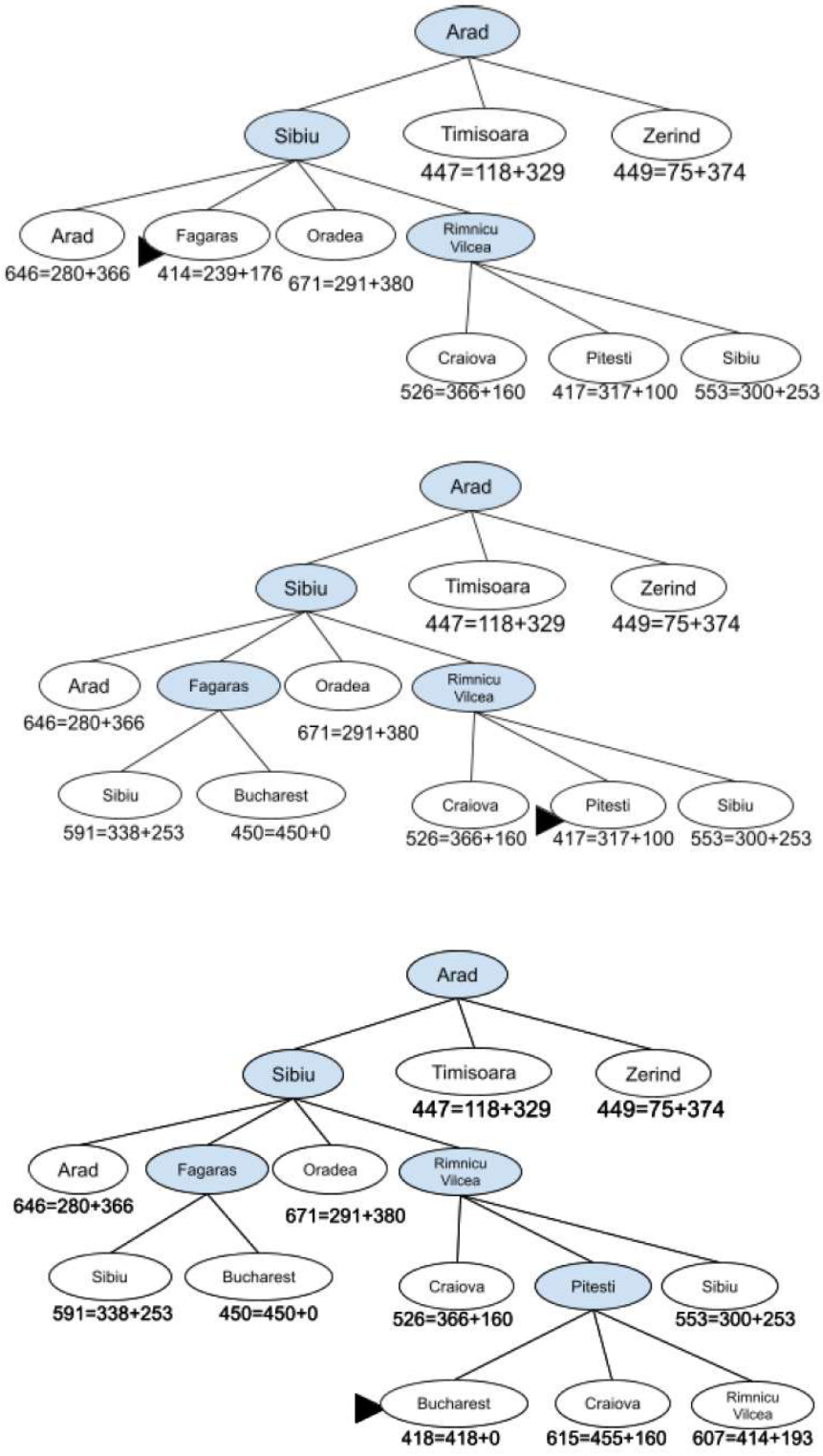


Fig. 3.12 Steps of A* search algorithm on the route-finding problem.

Rimnicu Vilcea, which is the most promising node in the OPEN list, has been selected for evaluation and expanded. However, Fagaras is the most promising node on the OPEN list and has been selected for evaluation. It should be noted that Bucharest, the goal node, has already been generated but has not been selected for evaluation, as Pitesti looks more promising. After expanding Pitesti, Bucharest is the most promising node and is selected for evaluation. On evaluation, it is discovered that it is the goal node, and the algorithm returns the path and terminates.

Combining the benefits of informed and uninformed searches, A* is one of the most influential and popular search techniques. It is a preferred option for resolving various issues due to its completeness and optimality.

Stop to Consider

A* search is the most common kind of best-first search. It evaluates nodes by adding the heuristic function (h'(n)) and the costs of traveling to the node (g(n)) , thus f'(n) = g(n) + h'(n).

3.8 PROBLEM REDUCTION

Search problems we have handled so far are represented using OR graphs. In an OR graph, we must find a single path from the initial to the goal state. However, for problems of decomposable nature, where a significant complex problem is decomposed into many more minor problems, we must solve each of these smaller problems simultaneously. These kinds of problems are represented using an AND-OR graph.

3.8.1 OR Graph versus AND-OR Graph

Artificial intelligence search problems can be represented by OR graphs or AND-OR graphs, depending on whether a solution addresses several subproblems simultaneously (AND) or only one subproblem (OR). An OR graph shows problems for which any one of the graph's branches can be satisfied to provide a solution. Finding a single path from the start node to a destination node is necessary to solve the graph. This was the case in our route-finding problem discussed earlier. There are several routes from Arad to Bucharest. If the optimality of the route is not a concern, then any of these routes will solve the problem.

When complex problems are decomposed or reduced to simpler ones, it is necessary to solve each of these problems simultaneously. Since these problems must be solved simultaneously, arcs are generated called AND arcs. Specific nodes need AND conditions to be met, which means that every branch that emerges from the node must result in a solution. Only one branch (OR condition) may need to be solved for other nodes. Therefore, such graphs are called AND-OR graphs. An example in Fig. 3.13 represents an AND-OR graph.

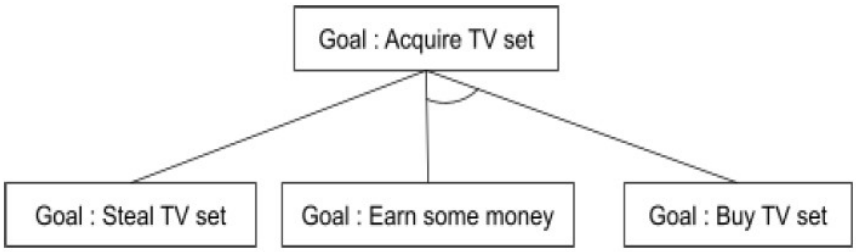


Fig.3.13 An example of an AND-OR graph

Solving an AND-OR graph problem can be solved using algorithms like best-first search that can handle the AND arcs. Martelli and Montanari described an algorithm called the AO* algorithm to deal

with AND arcs. A simplified version of this algorithm is presented below:

Algorithm: Problem Reduction

1. Start with the initial node and set its heuristic value $f'(start)$.
2. Loop until the starting node is labeled SOLVED or until its cost goes above FUTILITY.
    a. Traverse the graph, starting with the initial state, and choose the most promising unexpanded node not labeled SOLVED.
    b. Expand the chosen node. If there is no successor, assign FUTILITY as the value of this node. Otherwise, add the successors to the graph and compute f' for each node. Label the node as solved if the $f' = 0$.
    c. Update the parent nodes' costs recursively to reflect the new information provided by the successors. If any node contains a successor arc whose descendants are all SOLVED, label the node itself as SOLVED. Also update the current best path.

The process is illustrated in Fig. 3.14. Note that an assumption is made that every operation has a uniform cost, so each arc with a single successor has a cost of 1, and each AND arc with multiple successors has a cost of 1 for each of its components. In step 1, A is the only node. When node A is expanded, it yields B, C, and D. Node B is labeled as the most promising with a cost of 5 (4+1) compared to the combined cost of C and D, which is 7 (2+3+1+1). Two new nodes, E and F, are generated on expanding B. Since the new costs of E and F are higher, the algorithm needs to update f' for the latest updated value. Since going through E is cheaper than

F, therefore the updated cost of B is 7. Similarly, the cost of A through B is updated to 8. At this time, the path through C and D looks more promising. New nodes G, H, and I (AND arc between H & I) are generated on expanding C. No cost updation takes place here. Importantly, nodes H and I are the terminal nodes or the goal nodes therefore, they are marked as SOLVED. Moreover, its parent node B is also marked as SOLVED. After expanding D, node J is generated. The new costs of nodes D and A are updated. Node J, also a terminal node, is marked SOLVED along with its parent D. Since C and D are both marked as SOLVED, the parent A can now be marked as SOLVED. As the initial node is marked as SOLVED, the algorithm terminates.

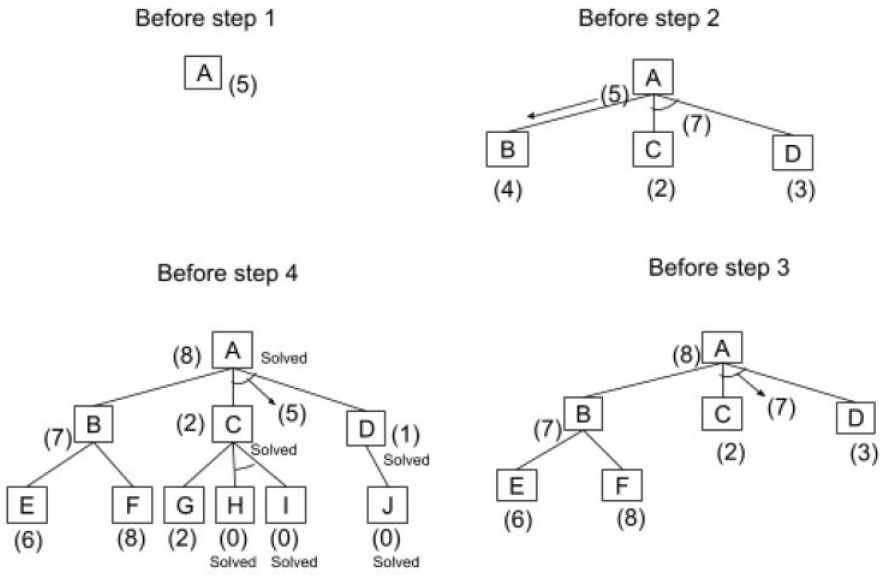


Fig 3.13 Steps of Problem Reduction.

It is ideal for issues that must be solved as subproblems simultaneously, as it effectively directs the search toward the optimal answers. Problem-reduction algorithms are heuristic-dependent and the space and time complexity of managing and storing AND-OR is expensive.

An AND-OR graph problem can be solved using a problem reduction algorithm that can handle the AND arcs.

3.9 SUMMING UP

In this unit, we have,

- We saw the difference between informed and uninformed search techniques.
- Uninformed search technique like Generate-and-test is discussed in detailed.
- The generate-and-test algorithm is similar to a depth-first search with backtracking because a complete solution must be generated before it can be tested.
- We also explored different variations of the Hill Climbing algorithms such as Simple hill climbing. Steepest-ascent hill climbing and simulated annealing move toward the direction of the goal based on a heuristic.
- Best-first search is an informed search technique using a heuristic function to determine the most promising state generated while remembering the other unexplored states generated earlier
- A∗ search is the most common kind of best-first search. It evaluates nodes by adding the heuristic function (h'(n)) and the costs of traveling to the node (g(n)), thus f'(n) = g(n) + h'(n).
- An AND-OR graph problem can be solved using a problem reduction algorithm that can handle the AND arcs.

3.10 ANSWERS TO CHECK YOUR PROGRESS

1. a) True b) False c) True

2. a) Backtracking b) Ridge c) Heuristic function

3.11 POSSIBLE QUESTIONS

1. Write down the algorithm for Simple Hill Climbing.
2. Write down the algorithm for Steepest Ascent Hill Climbing.
3. Compare Basic with Steepest Ascent Hill Climbing.
4. How will you mitigate the problem of Local Maximum, plateau, or Ridge in a Hill climbing algorithm?
5. What is simulated annealing?
6. Explain the Best First Search with an example.
7. Explain A^* algorithm with an example.
8. What do you mean by Problem Reduction?

3.12 REFERENCES AND SUGGESTED READINGS

1. Elaine Rich, Kevin Knight, & Shivashankar B Nair, Artificial Intelligence, McGraw Hill, 3rd ed., 2009
2. Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Third Edition, Pearson, 2016
3. Zachary Kaplan, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

*****

BLOCK- II

KNOWLEDGE REPRESENTATION AND PREDICATE CALCULUS

UNIT 1: CONCEPTS IN KNOWLEDGE REPRESENTATION

Unit Structure:

1.1 INTRODUCTION

Role of knowledge is very important in all AI systems. Up until now, in the previous chapters, we have paid little focus on knowledge representation issues and various techniques to manipulate that knowledge. In this chapter, we shall examine some specific knowledge representation techniques that can be implemented for retrieving and manipulating knowledge. These methods or techniques must be sufficiently general enough to be used in any kind of problem domain without any reference to how the knowledge the problem

needs is to be acquired. We shall also look at some ways of representing knowledge which facilitate more problem-solving capabilities. In this chapter, we shall examine various techniques that can be used for manipulating knowledge within programs.

1.2 UNIT OBJECTIVE

After going through this unit, you will be able to:

- know the definition and importance of knowledge.
- understand various issues related to knowledge representation.
- have an idea of knowledge based systems and their properties.
- have knowledge about facts and representations and their mappings.
- have ideas on knowledge manipulation systems.
- have an idea about simple relational model.
- know how property inheritance technique is implemented and manipulated.
- have an idea about inferential knowledge.
- know about procedural knowledge.

1.3 DEFINITION AND IMPORTANCE OF KNOWLEDGE

Human beings are embodied with the act of knowing which is formed by facts and principles. But sometimes, this is not the adequate definition of Knowledge. Knowledge is much more than this. Any fact or concept is worthless without the ability of acquiring knowledge. It is closely associated with intelligence. In order to perform any task that requires intelligence also needs access to knowledge. Thus, it can be concluded that “Intelligence requires

knowledge". In computers, knowledge is stored as symbolic structures, in the form of magnetic spots and voltage levels.

Knowledge can be defined as the act, fact and skill of knowing. It refers to the information and facts that an AI system uses to interpret, understand, and make decisions about the world. It is the foundation upon which AI models are built with reasoning, learning, and problem-solving capabilities. Knowledge can be concluded as the information, facts, rules, concepts and relationships that an AI system uses to perform tasks, make decisions, or solve problems.

A common way to represent knowledge is in the form of written language, like English. For e.g., the facts and relations cited as below-

Emma is beautiful.

James loves his mother.

The first one is a simple fact which expresses an attribute possessed by a girl named Emma that she is beautiful. The second fact represents a binary relation between a boy names James and his mother. Knowledge may be considered as either declarative or procedural. A declarative knowledge is the passive knowledge expressed as facts. In this kind of knowledge representation system, the knowledge is specified. But to use that knowledge, we must append it with a program that specifies what is to be done and how. For e.g., the employee's data stored in a database. Such kinds of data are the independent pieces of knowledge. In order to do something with this data, we need to write codes which access and manipulate them finally to give something which we desire for. Declarative knowledge also includes knowledge about the world that can be inferred through facts or relationships. On the other hand, in procedural knowledge, the control structure needed to use the knowledge is embedded into the program itself. Such kind of knowledge is the compiled knowledge. For e.g., the steps to be

followed in order to solve an algebraic equation can be expressed as procedural knowledge. Another might be the diagnosis of disease based on the symptoms.

Knowledge and data are two completely different concepts. For instance, we can take an example of a physician who treats the patients with both data and knowledge. The data is the record of the patient which may include the patient's history, drugs prescribed and the responses to drugs etc, whereas the knowledge is what the physician has learnt during his studies in the medical college and in his internship period, specialization or practice. In the game of tic-tac-toe, we have knowledge about the game and the valid move on a particular position. The current position forms data input to the game. Using the knowledge of the game, we can make valid move on each position in order to attain the winning state. Therefore, knowledge requires utilization of both data and information. Knowledge in broader sense can be defined as the belief which has its own justification. It is the foundation of learning, thinking, understanding, manipulating and reasoning. The AI systems respond, interpret and understand inputs in an intelligent way.

Knowledge has got its own meaning and importance in the field of AI. Knowledge is important from the fact that, it is possible to incorporate it into the software that can reason and draw conclusions. AI systems use this knowledge to solve problems, recognize patterns, make predictions, and even engage in complex reasoning activities. We can imagine of some kind of software which can give advice in specialized areas, such as manufacturing techniques, defence mechanism, marketing sectors, financial strategies, medical diagnosis etc.

To build such systems which show intelligence requires vast amount of knowledge to be stored. Artificial Intelligence (AI) focuses on creating systems that can perform tasks that typically require human

intelligence. It is divided into several subfields, including machine learning (ML), deep learning (DL), machine translation, natural language processing (NLP), computer vision, robotics, and expert systems.

Check Your Progress-1

1. Do you think that the data stored in a database is a declarative one?
2. Knowledge can be defined as the true _____ belief.

1.4 ISSUES IN KNOWLEDGE REPRESENTATION

Knowledge representation (KR) in artificial intelligence (AI) plays a crucial role in designing methods to represent and manipulate information about the world. Such representations are processed by machines. However, building AI systems involve several associated challenges and issues. These issues exist in almost all knowledge representation formalisms. Some are discussed below-

- In any problem domain, facts play a very important role. Each problem can be subdivided into some basic level modules. The more basic level we break the problem, solution becomes easier. Each such module can be represented in terms of objects. These objects are associated with attributes. We might have some situation when some attributes fit in almost every problem domain. If yes, they must be handled appropriately in each of the techniques we propose. There are actually two such kinds of important attributes which have general significance- instance and isa. These attributes are important from the perspective that they facilitate property inheritance. Instance and isa represent class membership and class inclusion respectively. In logic-based systems, these relationships may be represented by a set of predicates.

- Another important issue associated with representation of knowledge is relationships among objects. These attributes are used to describe the objects and they possess four major properties irrespective of the specific knowledge they encode. They are-
  - Property of Inverses
  - A hierarchical structure exhibiting an *isa* attribute
  - Techniques for inferring knowledge according to values
  - Attributes that may contain single value.
- A very important aspect of knowledge representation is to break knowledge into basic primitive level. We always split a complex problem into simple operations. The major advantage of this approach is that such simple primitives represent rules that derive inferences. In order to illustrate this problem, let us consider a simple fact:

John had a glass of water.

This could be represented as –

had (agent (John), object (water))

Such type of representation is sufficient to answer the questions such as "Who had a glass of water?" The answer is obviously John. But if the question is asked like "Did John drink a glass of water?", then also the answer is "yes". But we must have an inference rule which says that both having a glass of water and drinking are indifferent. Therefore, in this case, the inference rule would be

had (x, y) -> drank (x, y)

This rule states that if x had y; means x drank y.

Such type of inference will of course give the appropriate answer.

A serious drawback of this approach might be the additional storage space required to store such rules. A simple fact may require more

storage when it is splitted into its constituent primitive assertions. As well as, substantial amount of work has to be done on reducing those high level propositions into low level primitives.

There are several reasons behind the importance of representing sets of objects. One reason for doing this is that there are some properties which are possessed by the whole set of objects but not exhibited by the individual elements of the set. For instance, consider the assertion "Eskimos live in the South Pole". The only way to represent the fact described in this sentence is to attach assertions being made about the set of Eskimos; that they live in the South Pole. This may not be true for individual member of the set; for e.g., No Eskimos live in the South Pole. The other reason for the importance of representing sets of objects is that if a property is true for individual members of the set, then it is more efficient to attach it with the set itself rather than to associate with individual members of the set. There are various ways in which sets must be represented. This may include logical formalisms and slot-and-filler structure.

Suppose, a database containing a large amount of knowledge is provided. Now, it is important to understand that to fulfil our task, we do not require everything stored in the database. Rather, we might need only some part of the same. Now, how can we access those relevant parts? To elaborate this, let us consider the following shopping script which describes the sequence of events that might take place in a typical shop.

> Tom went to the market for shopping yesterday. He searched for a suit for his brother. He found a black one. He really liked it. So, he paid the bill and took it home. And the answer is "yes" to the question "Did Tom buy a black suit yesterday?"

However, the point to be noted here is that there is nowhere mentioned explicitly that Tom had bought anything, though it is mentioned that Tom went for shopping. But going to the market, searching for something which is really required, paying the bill and taking it home means the person had really shopped something. Therefore, we need to know the shopping script to be able to answer such kind of direct questions. But in order to be able to reason a variety of questions, number of scripts need to be used. These scripts must be so basic that it must be able to give variety of information about an event. The appropriate script which provides the best responses be utilized according to the question being asked. Such approach enhances the knowledge base of a system.

Check Your Progress-2

3. The two common attributes which can occur in almost all domains are _____ and _____.

4. Instance means _____

5. Isa means _____

6. Who are the agent and object of the following fact?
"He drinks a cup of coffee".

1.5 KNOWLEDGE BASED SYSTEMS

In the early days of AI research, it was realized that the general purpose problem solving systems were weak to solve more complex problems. This is because; they used a limited number of rules or axioms, which were not very much effective in inferring more information. Eventually this has led to the design and development knowledge based systems that can perform complex tasks on a rich knowledge based systems. Much of the works have been done on

knowledge-based systems, including works in learning, vision or natural language processing. More emphasis has been provided on research related to knowledge representation techniques, memory organization and knowledge manipulation. Knowledge based systems get their power from the knowledge representation formalisms coded into facts, rules and procedures etc. It is a system that uses knowledge and reasoning to solve complex problems typically requiring human expertise. It relies on a knowledge base (KB) and inference mechanisms to derive conclusions or provide solutions by using the rules which define how facts relate with each other in order to derive more new knowledge or new decisions. In addition, it also makes possible to append new knowledge or refine existing knowledge without recompiling the inferencing programs. The core of a good knowledge representation system is the repository of facts that represent knowledge about a domain. This in turn greatly simplifies the construction and maintenance of the knowledge-based systems.

A good knowledge representation system in a particular domain should possess the following four properties:

a) It should possess the ability to represent all kinds of knowledge that are needed in a particular domain. In other words the domain should contain adequate information to define the system. The more information the system incorporates, the system would have more inference capability as well as more decision making ability to draw new conclusions.

b) A good knowledge based system should be able manipulate the knowledge base. The manipulation should occur in such a way that new structures corresponding to new knowledge can be derived from old. It is also regarded as the inference procedure which contains the adequate information to derive knowledge.

c) The knowledge about a domain is very important in creating an intelligent system. We already have discussed that if the core of the system is rich; definitely the system is going to perform better. Integration of more facts into the knowledge structure is referred to as additional information and they can be used to direct the focus of the inference technique towards the most fruitful direction. In other words, this property makes the inference mechanism more efficient.

d) The system should represent the ability to acquire new information easily. As the inference engine derives new knowledge, they may be appended into the core of the system. The simplest case may be the direct insertion of records into a relational database. This enriches the knowledge base of the system; which eventually leads to more efficient inference mechanism.

However, there are some challenges that a knowledge based system always faces. It is not that easy to incorporate human expertise into a knowledge based system. In order to behave intelligently, the system must be able to encode knowledge into a system. It is really a tedious job to perform. When the core of the system grows or the system scales to hold more knowledge, it becomes complex to manage the system as well as the efficiency of the system degrades. Eventually, this leads to the issue of maintainability. Keeping the system well maintained with the incorporation of new knowledge as well as with the conflicts that arise during knowledge acquisition is really a matter of concern.

1.6 KNOWLEDGE REPRESENTATION AND MAPPINGS

Knowledge is important from the fact that it is essential for showing intelligent behaviour. It can be represented in various forms, such as spoken or written words, as graphics or pictures, as string of

characters or collection of words stored in computers etc. These may be defined as facts, rules or concepts. The representations we are concerned here is on the study of written languages only.

Knowledge Representation (KR) is a field of computer science that deals with how knowledge can be represented in a formal system. Such representation is important from the perspective that the computer system must be able to understand and reason the knowledge and make decisions based on the information stored.

In order to solve many AI problems, one should possess a large amount of knowledge and some inference mechanisms for manipulating that knowledge. There are a variety of ways for representing and manipulating knowledge so that machines can understand them. But before we discuss about them, we must have some idea about the following entities-

- Facts – Facts are the things that are needed to be represented. Facts represent the true knowledge about the world. The concepts of real-world domain can be defined by the facts. For e.g., the sentence "Man is mortal" can be considered as a fact.
- Representations of facts in some specific formal way. We will only be able to manipulate the facts only after their representations in the corresponding chosen way. We already have mentioned that knowledge about any domain must be represented in the most basic level. This enables the inference engine to have the capability to derive more new knowledge. There are some formalisms in which such knowledge must be represented: logical propositions like the propositional and predicate logic, semantic networks, frames and rule-based representation.

1. Logical Representation:

- **Propositional Logic**: Represents facts in the form of sentences or propositions. It is the simplest way of representing facts. Let us assume a simple fact “John is a student”. The logical proposition for the same would be STUDENTJOHN. Similarly, the corresponding propositional logic representation of the fact “Joe is a student” would be STUDENTJOE. These are simple and very basic means of representing knowledge and cannot be extensively used to represent all kinds of knowledge.
- **Predicate Logic**: A more expressive form of logic is the predicate logic. It tries to overcome the limitations of propositional logic. It allows for more detailed relationships and structures, including variables, functions, and quantifiers. It’s used to express statements like "Man is mortal" ($\forall x$: Man(x) $\rightarrow$ Mortal(x)).

2. Semantic Networks:

- A **semantic network** is an interconnection of different network nodes. Each node of the network is connected by edges. Nodes represent the concepts or attributes of the objects. Edges represent the relationships among the attributes. For example, "Bird" might be connected to "Has wings" or "Can fly." This approach helps in understanding and visually mapping knowledge.

3. Frames:

- **Frames** are data structures that represent an object or a concept of the real world. These are similar to object-oriented programming, where each frame represents the object or concept, and it includes both attributes and actions. For

example, a "Car" frame might have attributes like color, engine type, engine number, chassis number and model.

4. **Rule-Based Representation**:

- Knowledge can also be represented as a set of rules (if-then statements). It is mainly used in expert systems and decision-making systems. For instance, "If it rains, I shall not go out." Rule-based systems are often used in expert systems and decision-making systems.

There are two levels in which facts and their corresponding representations must be described-

- At knowledge level, only the facts about a domain must be declared. Insertion of more facts at this level corresponds to enriching the knowledge base.
- At symbol level, the objects gathered from the knowledge level can be manipulated by programs. At this level, objects are defined in terms of symbols. The program manipulates them in order to produce the target representation.

The mapping between facts and their representations in a particular language is shown in Figure 1:

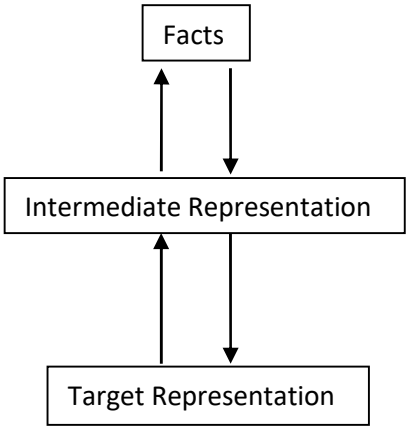


Fig 1: Mapping between facts and representations

Source: Artificial Intelligence by Elaine Rich and Kevin Knight

There is a two way communication between facts and their representations- from facts to their representations known as the forward representation mapping and from representations to facts which we call the backward representation mapping. We call these communication links the representation mappings.

But, there is no direct correspondence between facts and their target representation. Rather, there is an intermediate step which lies between the two. In forward representation mapping, facts are converted into an intermediate representation before the target code is attained. Similarly, in the reverse process, the target representation is again converted into some intermediate form before facts can be attained.

One common representation of facts that needs to be mentioned here specially is the natural language sentences. Here we need to be concerned not only with the representation of facts but also getting information out of those facts. This requires a mapping function from the given facts into their corresponding representations and from the representations back to the facts.

Let us consider an example of simple English sentence-

Bob is a man.

The fact can be represented by means of logic as-

man(Bob)

Suppose, we have the following logical representation of the fact that "All men are mortal"-

∀x : man(x) -> mortal(x)

The two above representations may be used to infer the new knowledge as follows-

mortal(Bob)

Using the backward representation mapping, we will be able to generate the following fact for the above proposition-

Bob is mortal.

It is important to keep in mind that some mapping functions are not always one-to-one. There may exist more than one representation for some facts in a particular domain. For e.g., the sentence "Her friends called her a doctor" may represent either the fact that- her friends used to call her 'a doctor' or the fact that- "her friends called a doctor for her". While trying to convert the English sentences into some other formalism such as logical propositions, it should be kept in mind the facts the sentences are going to represent and then convert them into the new structure. An AI program is provided with the internal representations of facts that it manipulates.

A knowledge base can be described as a mapping between the objects and relations in a problem domain. The computational objects, their relations and the inference rules are mediated by the knowledge representation languages. This may include languages like PROLOG, LISP or languages like C++, JAVA etc.

Check Your Progress3

7. Convert the following facts into logical propositions and try to infer if a new knowledge could be generated from them.

a) John is a man.

b) All men are persons

1.7 KNOWLEDGE MANIPULATION

Manipulations are the computational aspects of reasoning. Decisions and actions in knowledge based systems are specified by the way knowledge is manipulated. The search for a goal is initiated by some

forms of input. The process of decision making may set up other sub goals or may require further inputs until a final solution is found. A form of inference or deduction integrating the knowledge and inferring rules is a necessary component of knowledge manipulation. The two operations, searching and matching are integral to all forms of reasoning. These two operations are known to consume the greatest amount of computational time in most AI systems. That is why; we must have techniques to limit the amount of search and matching necessary for efficient completion of a knowledge production task.

1.8 APPROACHES TO KNOWLEDGE REPRESENTATION

In section 1.5, we had a discussion on various properties that a good knowledge representation system must possess. Unfortunately, no such single system has yet been developed which could incorporate all these capabilities. This results in developing multiple techniques for the representation and manipulation of knowledge. In this section, we shall discuss some important techniques for the representation of knowledge.

1.8.1 Simple Relational Knowledge

One of the simplest way to represent knowledge is as a set of relations, same as the one used in database systems. Figure 2 shows an example of such a relational system.

Player Name	Age	Weight	Height	Bats
Nicholas	28	75	5-9	Left handed
Williams	26	70	5-8	Right handed
Robinson	29	68	5-6	Right handed

Fig 2: Example of a Relational System

This is a simple representation and it can provide very weak knowledge inference capability. If the weight of Nicholas is asked for, then definitely the reply can be provided, given the facts of the above figure. However, it is not possible to get the direct answer to a very simple question, “Who is the oldest player?” However, it should be mentioned here that such kind of knowledge can provide the inputs to many powerful inference engines. If a procedure is written to find the oldest player, the given facts will definitely enable the procedure to work and derive the record of the oldest player.

Database systems are designed to provide support for relational knowledge. However, there are some consequences that may arise from linking a database system in order to provide some other capabilities which need to be discussed in the next section.

1.8.2 Inheritable knowledge:

The relational knowledge discussed in the last section consists of a set of attributes and their associated values. This as a whole describes the objects of the knowledge base and they are simple. We can make the system more effective if we augment this basic representation with inference mechanisms that operate on the structure. The structure must be designed to correspond to inference mechanisms and one of such useful forms of inference is the property inheritance technique, in which elements of specific classes can inherit properties from more general classes.

In this kind of knowledge representation system, objects must be organized into classes and classes must be organized into a hierarchy. Figure 3 shows the structure of a baseball player augmented with some additional knowledge. Lines represent attributes. Boxed nodes represent objects and values of attributes of objects. The arrows use to point from an object to its value along the corresponding attribute

line. Such structure is called the slot-and-filler structure. It may also be called as semantic network or the collection of frames.

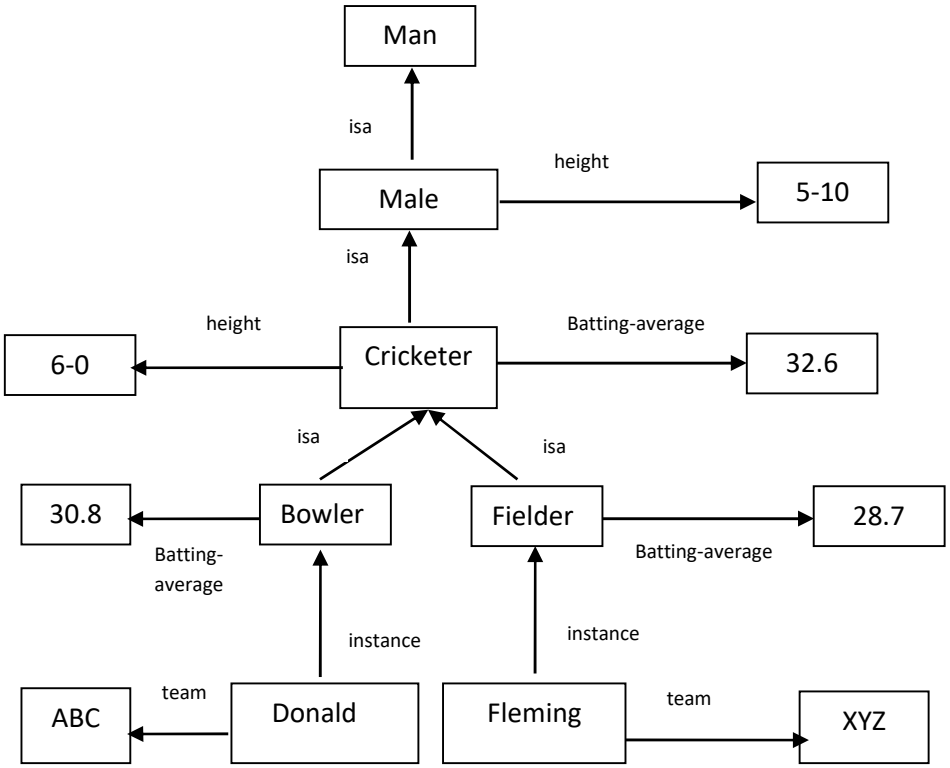


Fig 3: Structure of a baseball player augmented with some additional knowledge.

Each frame represents the collection of attributes and the values associated with a particular node. Frame system is the term which is more structured and inference mechanisms can be applied on them.

Now let us see how such type of structures support inference using the knowledge they contain. The objects and the attributes shown in this structure correspond to the baseball domain. The two attributes instance and isa are being used correspond to class membership and class inclusion respectively. They provide the basis for property inheritance which provides the support for retrieval both of facts that have been explicitly stored and of facts that can be derived from them. A node can be viewed as a frame as follows:

Cricketer

isa	:	*Male*
height	:	*6-0*
batting-average	:	*32.6*

Algorithm: Property Inheritance

To retrieve a value V for the attribute A of an instance object O:

1. Find O in the knowledge base.
2. If there is a value for the attribute A, report it.
3. Otherwise, see if there is a value for the attribute *instance*. If not, then fail.
4. Otherwise, move to the node corresponding to that value, look for the value for the attribute A and report it if found.
5. Otherwise, loop until there is no value for the *isa* attribute or until an answer is found:
    a. Get the value of the *isa* attribute and move to that node.
    b. See if there is a value for the attribute A. If there is, report it.

1.8.3 Inferential Knowledge

Though property inheritance is a powerful form of inferencing knowledge, it is not always the useful form. Sometimes, the power of logic needs to be incorporated into the knowledge structure so that additional information could be generated. Inferential knowledge requires an inference procedure to exploit knowledge. The inference procedure implements the standard logical rules of inference. The forward inferential process reasons forward, from facts to

conclusions. The backward inferential process reasons from the desired conclusion to facts. There is one more commonly used procedure known as resolution, which follows proof by contradiction strategy. Logic provides a powerful structure which describes the relationships among values. In general, it can be said that inferential knowledge can be regarded as the basic building block for a complete knowledge representation system.

1.8.4 Procedural Knowledge:

One important kind of knowledge that specifies what to do when is known as procedural knowledge. Procedural knowledge can be represented in programs in many ways. The most common way is to write simply as a set of codes for doing something. The machine uses the knowledge when it executes the code to perform a task. The most common way to represent procedural knowledge is the use of production rules.

1.9 SUMMING UP

- Knowledge about a particular domain in which the problem occurs is important from the fact that without possessing knowledge no one can solve the problem.
- Techniques to analyze, represent and manipulate knowledge must be general enough for representing all kinds of knowledge irrespective of the domain to which it belongs.
- There are two common attributes found in almost all knowledge representation systems instance and isa.
- Instance represents class membership and isa represents class inclusion.

- A knowledge based system is the one that should be able to map from facts to its corresponding representations.
- A knowledge based system may be built based on relational database systems, inheritable knowledge, procedural knowledge or inferential knowledge etc.

1.10 ANSWERS TO CHECK YOUR PROGRESS

1. Yes, the data stored in a database can be considered as declarative knowledge. This is because; we need an extra procedure to manipulate the knowledge or to find out the required information stored in it.
2. justified.
3. instance , isa.
4. class membership.
5. class inclusion.
6. Agent : - He , Object : - Coffee
7. a) man (John)

   b) ∀x : man(x) -> person(x)

   These two facts imply that

   person(John)

1.11 POSSIBLE QUESTIONS

Short answer type questions:

1. Give the definition of knowledge.
2. What is the importance of knowledge in any domain?
3. What do you mean by fact?
4. What are the two major entities that effect in representations of facts?
5. What do you mean by manipulation of knowledge?
6. What do you mean by procedural knowledge?

7. What is simple relation based knowledge?

Long answer type questions:

1. Why and how does knowledge play significant role in any problem domain? Explain.
2. Explain the various issues related to knowledge representation.
3. What is knowledge based system? Explain its basic properties?
4. Discuss why knowledge is essential for showing intelligent behaviour.
5. Explain the mapping between facts and their corresponding representations.
6. Consider the following sentences and try to find out the ambiguous word and its meanings-
    1. Let's play in the diamond.
    2. Book lives in Australia.
    3. Your diamond earrings are shimmering.
    4. They found churches.
7. Explain the various approaches of representation of knowledge with suitable examples.
8. What is inheritable knowledge? Explain with example.

1.12 REFERENCES AND SUGGESTED READINGS

1. E. Rich & K. Knight, *Artificial Intelligence*, (3rd Edition, 2017, Tata McGraw Hill
2. George Luger, *Artificial Intelligence*: *Structures and Strategies for Complex Problem solving*, (6th Edition), 2008, Pearson Education.
3. Nils J Nisson: Narosa, *Principles of Artificial Intelligence*, 1st Edition, Springer

*****

UNIT 2: PREDICATE CALCULUS - I

Unit Structure:

2.1 INTRODUCTION

An important way of representing facts about the world is the language of logic. The language of logic is a powerful and practical means of representing and manipulating knowledge. It provides a way of deriving new knowledge from old. In this formalism, it can be concluded that a new statement is true because it follows from some statements which already are known to be true. The language of logic can be divided into two major areas- the propositional logic and the predicate logic. However, the predicate logic or predicate calculus or the First Order Predicate Logic (FOPL) is considered to be playing an important role in AI for representation of knowledge. It is widely accepted in AI which offers a formal approach for reasoning. In FOPL, the statements of a natural language like English can be

translated into symbolic representations which closely approximate the meaning of these statements. Another important concept behind the application of logic is that it provides a way of generating answers to questions and solutions to problems.

2.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic concept of propositional and predicate logic.
- represent facts both in propositional and predicate logic.
- learn the syntaxes in First-Order Predicate calculus.
- know how inference rules can be used to produce predicate logic expressions using quantified variables.
- represent predicate logic expressions using instance and isa relationships.

2.3 REPRESENTING FACTS IN LOGIC

2.3.1 Propositional Logic:

As we already have discussed that propositional logic provides a way of representing and manipulating knowledge. Propositional logic is appealing as it is simple and a decision procedure exists for it. Facts in the world can be represented by logical propositions and they are written as well-formed formulas (wff s). Suppose, we want to represent a simple sentence in English

Richards is a student.

This could be written as:

RICHARDS STUDENT

But if we want to represent another fact like

Ricardo is a student.

We would have to write it like

RICARDO STUDENT

Which is a totally different assertion and we will not be able to draw any conclusion from these two assertions. The better representation of these two facts would be:

STUDENT (RICHARDS)

STUDENT (RICARDO)

Now the structures of the representations reflect the structure of the knowledge itself. Propositions are the simple, primitive and atomic sentences. Whereas, compound propositions also exist and they are formed using the logical connectives like NOT, AND, OR, if.....then and if and only if etc. The following are the symbols representing the logical connectives:

" ~ " for NOT or negation.

" ∧ " for AND conjunction.

" ∨ " for OR conjunction

" → " for if ..... then or implication

"↔ " for if and only if or double implication

Now, to represent the compound statement "It is raining and I won't go out", we could write it as (R ∧ W) where R and W stand for "It is raining" and "I won't go out" respectively. Similarly, the statement (R ∨ W) means "It is raining or I won't go out".

Again if we consider the following simple facts:

1. All students are good.
2. All men are mortal.

Which would be represented as:

1. GOOD STUDENT
2. MORTAL MEN

The point to be noted here is that such types of representations are unable to capture any relationship between 1. any student and the same student having the quality of being good and 2. An individual being a man and that individual being mortal. We need variables and quantifiers to write such statement of assertions.

As a result, another logical formalism known as FOPL or predicate calculus has come into existence because it permits representation of things which cannot be represented in propositional logic. Predicate logic allows real-world facts to be represented using wffs. Such kind of logical statements provide a good way of reasoning with statements. Unfortunately, it does not possess any decision procedure. Still FOPL serves as a useful medium of representing knowledge. We shall discuss predicate calculus in more details in the next sections.

2.3.2 Predicate Logic:

All knowledge representation schemes possess the requirement of expressiveness, that all concepts must be accurately stated. Propositional logic has the limitation of not being able to generalize the statements. For e.g., consider the following statements-

All students of Computer Science must opt Artificial Intelligence.

Tom is a student of Computer Science.

From the above statements, it should be concluded that Tom must opt Artificial Intelligence. But using propositional logic, it is not possible to make such conclusion. There should be a valid inference rule which would draw the desired conclusion.

FOPL was developed to extend the expressiveness of propositional logic. It is a generalization of propositional logic which permits reasoning about objects. The syntax for FOPL is determined by symbols and rules of symbols.

Predicate logic also has the capability to reason, provided we have sufficient amount of facts to reason. Consider the following sentences expressed in natural language-

1. Richard is a student.
2. All students are good.

They must be expressed as simple facts in predicate logic-

1. student (Richard)
2. ∀x : student(x) -> good(x)

These two facts can reason a new fact:

Richard is good.

Thus, the predicate logic can infer new knowledge from the factual information which is already provided.

2.3.2.1 Syntax of FOPL:

The symbols and rules of combination permitted in FOPL are as follows-

Connectives: FOPL supports five connective symbols: ¬ (not or negation), ∧ (and or conjunction), ∨ (or or disjunction), → (implication), ↔ (equivalence or if and only if).

Quantifiers: FOPL statements have the feature of being quantified. There are two quantification symbols that an FOPL statement supports- ∃ (existential quantification) and ∀ (universal quantification). If we have the quantification symbols (∃x), then it means for some x or there exists an x and (∀x), which means for all

x. Furthermore, if we have a quantification statement which quantifies more than one variable with the same quantifier, for e.g., (∀x)(∀y)(∀z), then we can drop the parenthesis to get ∀xyz.

Constants: Constants are the terms that can take fixed value and are denoted by numbers, words and small letters such as a, b, 1.6, -18 and Bob etc.

Variables: Variables are the terms that can take different values overtime. They are denoted by words and small letters such as vehicle_type, x, y etc.

Functions: A domain D can have function symbols that denote relations in the domain. They map n-elements to a single element of the domain. An n-ary function declared as $g(t_1, t_2, t_3,.....,t_n)$ where each t_i denotes the terms defined over some domain. A 0-ary function is assumed to be a constant.

Predicates: The relations and functional mappings from the elements of a domain to their corresponding truth values are defined by predicate symbols. Capital letters and capitalized words like P, Q, FATHER are used to denote the predicate symbols. Like the functions, predicates may also take n-arguments. A 0-ary predicate is considered to be a proposition.

Constants, variables and functions are designated as terms, predicates are designated as atoms. We often use the word literal, whenever we refer to an atom or its negation. It is often fruitful to use computable functions or computable predicates to the statements so that we might be able to evaluate their truth values. If we assume the statement such as

gt(5+2, 3)

Then, the computable function gt requires adding 5 and 2 prior to comparing the added value with 3 to test to see whether the value is greater than 3 or not. If it is, it returns true; otherwise false.

Now, if we consider the facts described as follows-

F1: All men are mortal

F2: Some students are sick today.

Both may be represented in terms of wffs in predicate logic as -

$\forall x : man(x) \rightarrow mortal(x)$

$\exists y : student(y) \rightarrow sick(y)$

Check Your Progress-1

1. Decision procedure exists for propositional logic. (True/False)
2. Logical propositions are written using _____.
3. Expressiveness refers to accurately stated concepts or knowledge. (True/False)
4. FOPL can extend the _____ of propositional logic.
5. FOPL can represent both _____ and _____.
6. FOPL statements can have the quality of being _____.
7. There are two quantification symbols in an FOPL-
_____ and _____.

2.4 INFERENCE RULES TO PRODUCE PREDICATE CALCULUS EXPRESSIONS

In this section we are going to have a brief discussion on the representations of inference rules which must correspond to the way the facts are described in terms of sentences. Suppose, we have the following set of sentences-

1. Robin was a man.
2. Robin was an Assamese.
3. All Assamese are Indians.

4. The age of Robin is 30.
5. Robin married Shally.
6. Sue either does not like Tom or hates him.
7. Sue eats everything Tom eats.
8. Every pet is loyal to his master.
9. Everyone who passes this exam is either intelligent or lucky.
10. All employees who are male must visit a particular place.
11. One who loves someone means he does not hate him.

The above described facts are represented accordingly in terms of wffs in predicate logic as below-

1. Robin was a man.
   man(Robin)

   This assertion captures the knowledge of Robin being a man. However, it fails to produce some of the important information regarding tense.

2. Robin was an Assamese.
   Assamese(Robin)
3. All Assamese are Indians.
   ∀ x : Assamese(x) -> Indian(x)

   This assertion produces information about every Assamese being an Indian in a more generalized way. The use of the quantification statement ∀ tries to capture the general information that the assertion is true for all Assamese people.

4. The age of Robin is 30.
   age (Robin, 30)

   This wff takes two arguments, the name of the person and his age. Therefore, it is a binary assertion which shows the relationship between a person and his age.

5. Robin married Shally.

married (Robin, Shally)

This statement also follows the same explanation because it follows from the binary relationship between two persons, that they are married to each other.

6. Sue either does not like Tom or hates him.

   ¬like (Sue, Tom) ∨ hate (Sue, Tom)

7. Sue eats everything Tom eats.

   ∀x : eats (Tom, x) -> eats (Sue, x)

   This assertion is somewhat different from (3) since the terms here are following binary relationship. The assertion is a generalized one since it tries to exploit the fact that if Tom eats something, then it is eaten by Sue also.

8. Every pet is loyal to his master.

   ∀x : ∃y : pet(x) -> loyalto (x, y)

   This wff contains two quantified variables x and y, where x represents every or all pets and y represents x's master. Therefore, x is universally quantified and y is existentially quantified. This statement expresses the fact of x being loyal to y.

9. Everyone who passes this exam is either intelligent or lucky.

   ∀x : ∃y : exam (y) ∧ pass (x, y) -> intelligent(x) ∨ lucky(x)

   This wff has two variables x and y being quantified universally and existentially representing human beings and exams respectively. The predicates are associated with each other by connectives. The statement says that one should pass an exam, and then only he will be regarded as an intelligent or a lucky person.

10. All employees who are male must visit this place.

∀x : ∃y : employee (x) ∧ male(x) ∧ place (y)-> visit (x, y)

This statement contains the information about all x being an employee and that particular employee being a male and there must exist a particular place y, then only x is allowed to visit place y.

11. One who loves someone means he does not hate him.

    ∀x : ∃y : love (x, y) -> ¬ hate (x, y)

    Here, the "¬" symbol signifies a not operation. It defines the relationship between x and y representing the fact that loving someone does not mean hating someone.

2.5 REPRESENTATION OF INSTANCE AND ISA RELATIONSHIPS

In the previous unit, we had a discussion on the instance and isa attributes, which play very important roles. They are the useful forms of reasoning as far as the property inheritance is concerned. Now, if we consider the assertions 1, 2 and 3 declared in the previous section, they do not seem to have used these two attributes at all. Instead, they represent the knowledge about Robin. We have not used predicates with these names. It should be mentioned here that, although we have not used the predicates instance and isa explicitly, we have been able to capture the relationships they express, namely class membership and class inclusion.

The first three sentences described in the last section can be represented in three different ways. The first one contains the representations we already have discussed. Class membership is represented using unary predicates (such as Assamese). The assertion P (x) is true, which asserts that x is an instance or element of P. The second one contains the representations using instance predicate

explicitly. Instance is a binary predicate, whose first argument is an object and second one is a class to which the object belongs. If an object is an instance of a subclass, then the implication rule in sentence 3 states that the object is also an instance of its super class. These representations do not use the isa predicate. The third part contains representations that use both instance and isa relationships explicitly. The isa predicate simplifies the representation of assertion 3, but requires providing one general, additional axiom. The axiom describes that the instance and isa relation can be combined to derive a new instance relation. Let us see how these three parts can be represented according to the ways we have just discussed.

1. man(Robin)
2. Assamese(Robin)
3. ∀ x : Assamese(x) -> Indian(x)

1. instance (Robin, man)
2. instance (Robin, Assamese)
3. ∀ x : instance (x, Assamese) -> instance (x, Indian)

1. instance (Robin, man)
2. instance (Robin, Assamese)
3. isa (Assamese, Indian)
4. ∀ x : ∀ y : ∀ z : instance (x, y) ∧ isa (y, z) -> instance (x, z)

It is not necessary to represent the membership between class and its superclass using the predicates instance and isa. Though this is an important relationship and needs to be represented; instead in a logical framework, unary predicates are sufficient to represent such relationship.

Check Your Progress-2

8. Logic is one form in which real world _____ can be represented.
9. Logical propositions represent the elementary _____ assertions.
10. Do you think that propositional logic is sufficient enough to represent all kinds of knowledge?
11. The predicates instance and isa are useful in _____ _____ technique.
12. The predicates instance and isa represent _____ _____ and _____ respectively.

2.6 SUMMING UP

- Logical representations of real-world facts are important from the point that it is one of the common ways of presenting knowledge.
- Basically two major kinds of logical formalisms are available- 1) The propositional logic and 2) The predicate logic.
- The propositional logic is simple and has a decision procedure as the facts can be expressed using logical assertions.
- The propositional logic differs from predicate calculus from the fact that the former captures only the facts which are specialized ones. Whereas, the later tries to capture more generalized knowledge structures.
- In predicate logic, the factual information about a particular domain is represented by First Order Predicate Logic (FOPL).
- The logical assertions in FOPL must be written using logical connectives.

2.7ANSWERS TO CHECK YOUR PROGRESS

1. True
2. well-formed formulas
3. True
4. expressiveness.
5. facts, rules.
6. Quantified
7. ∃ (existential quantification), ∀ (universal quantification)
8. facts.
9. atomic.
10. No, propositional logic is not sufficient enough to represent all kinds of knowledge. Therefore, predicate calculus has been introduced which can exploit more knowledge than the propositional logic.
11. property inheritance
12. class membership, class inclusion.

2.8 POSSIBLE QUESTIONS

Short answer type questions:

1. Explain in brief the language of logic.
2. What are ways of representing facts in logic? Mention.
3. What are the different connectives used in propositional calculus?
4. Mention the different quantifiers used in predicate calculus?
5. What do you mean by computable functions?
6. What do you mean by inference rule?
7. What is resolution?
8. Write down some features of PROLOG.
9. What is declarative knowledge?
10. What is procedural knowledge?
11. What is predicate calculus?

Long answer type questions:

1. How facts can be represented using language of logic?
2. What is propositional logic? How does it differ from predicate logic?
3. What are the syntaxes used in First Order predicate Logic?
4. How FOPL expressions can be converted into instance and isa relationships?
5. What is logic programming? Explain.
6. Consider the following set of facts and convert them into their corresponding predicate calculus expressions:
   a) All dogs bark at night.
   b) Birds fly.
   c) Piper does not fly.
   d) All judges are not crook.
   e) The sky is blue.
   f) John is unmarried.
   g) Both Ann and Sue are students.
   h) If Joe is a politician, then he is crook.
   i) Anyone who passes exam is either happy or excited.
   j) To be a parent, one must be a father or mother of someone.

2.9 REFERENCES AND SUGGESTED READINGS

1. Luger G. F. 2002. *Artificial Intelligence Structures and Strategies for Complex Problem Solving*. Pearson Education, Ltd. and Dorling Kindersley Publishing.
2. Patterson, D. W. 1990. *Introduction to Artificial Intelligence and Expert Systems.* New Jersey: Pearson Education.
3. Rich, A., and Knight K. 1991. *Artificial Intelligence.* New York: Tata McGraw-Hill.

*****

UNIT-3: THE PREDICATE CALCULUS-II

Unit Structure:

3.1 INTRODUCTION

Language of logic is a powerful method of representing facts about the world. It is also a means of manipulating knowledge as well as deriving new knowledge from old. Here, the known statements or facts are already assumed to be true. Therefore, the new knowledge derived from them is also considered as true. In the previous unit, we had studied the two major areas of language of logic - the

propositional logic and the predicate logic. We also had learnt that the propositional logic has a major drawback of not being able to represent all kinds of knowledge; except the simplest one. Then, predicate logic or predicate calculus or the First Order Predicate Logic (FOPL) has come for the rescue and is widely accepted as a formal approach of reasoning with logic. We got to know various aspects of representations of knowledge using FOPL. In this unit, we shall try to learn some major and advanced topics reasoning with FOPL. We shall see how computable functions play an important role in logical language. We shall also see how the logical statements are converted into some standard format.

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- know the relationship between computable functions and predicate logic expressions.
- define resolution principle.
- know the conversion of predicate logic expressions to clausal forms.
- understand the basis of resolution.
- have an idea about the unification process and the use of resolution technique in a Question-Answering system.
- understand what logic programming is and its use in PROLOG platform.
- define procedural and declarative knowledge.
- explain the logic behind matching.

3.3 COMPUTABLE FUNCTIONS AND PREDICATE LOGIC

So far, we have discussed the fact that sentences written in natural language can be converted into predicate calculus; such as man(Robin) ; corresponding to the sentence that "Robin was a man".

If the facts or the knowledge that a sentence can express is not very large, then such type of representation is good. But, some facts may need to compare values and evaluate to either true or false, then the situation would be different. Suppose we have some facts which incorporates greater-than or less-than relationships, like-

gt(1, 0)
lt (0, 1)

Then it is useful to augment the representations with computable predicates. The benefit of such kind of procedure is that we can simply invoke them just by making a call. Suppose, we want to evaluate the truth value of the computable predicate:

gt(4+2, 3)

This requires the value of the addition operation to be computed first and then send the arguments 6 and 3 to gt(). Then the final outcome will be evaluated in terms of either true or false depending on the arguments. The next few examples show how the computable functions and predicates are useful for representing facts. It also makes use of equality and allows substitution for equal objects whenever useful.

Now let us consider some set of facts which involves use of computable predicates.

1. If someone is dead at some time, he is considered to be dead at later times.
   $\forall x : \forall t_1 : \forall t_2 : \text{died}(x, t_1) \wedge \text{gt}(t_2, t_1) \rightarrow \text{dead}(x, t_2)$

This statement tries to say that someone (x, in this case) had died at time t_1 and if we assume a time t_2 which is greater than t_1, then it is obvious that the person is not alive at time t_2.

2. It is now 2013.

   now= 2013

   Here, the idea of equal quantities has been exploited and substituted.

3. No man remains alive for longer than 130 years.

   $\forall x : \forall t_1 : \forall t_2 : man(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 130) \rightarrow \neg alive(x, t_2)$

   This is not the only way to represent this sentence. For e.g., we could have introduced a predicate age and assert that the value is never greater than 130. Our representation tries to deduce that if any person x, who was born during time t_1 and the difference between an assumed time t_2 and t_1 is greater than 130 years, then x is not considered to be alive. The symbol '¬' designates the negation of a statement.

The use of computable predicates allows us to derive answers to questions. They can be used to generate answers to most direct questions whose responses involve truth values.

3.4 RESOLUTION AND ITS BASIC PRINCIPLE

Sometimes it is required to carry out a variety of statements within a single operation and reasoning with the assertions in predicate logic. Resolution is such a kind of technique and it is efficient from the fact that it operates on statements that are converted into convenient clausal forms. Resolution attempts to prove by confutation; that is, the negation of a statement tries to disprove the known true statements or makes the statements unsatisfiable. This approach contrasts with

the backward chaining approach which generates proofs from the theorem to be proved to its axioms. As we already have said that the process of resolution involves axioms which are already in clausal form. There are some sequences of steps which need to be involved in order to convert the predicate logic statements into clausal form.

3.4.1 Algorithm: Convert to Clausal Form:

The input to the algorithm is the set of wffs (well formed formulas) in predicate calculus for which the conversion will take place. Now, the steps are mentioned below-

1. Eliminate ->, of the fact a -> b, by making it is equivalent to ¬a ∨ b. Consider the following wffs which needs the conversion:
   a. ∀ x: ∀ y: [P(x) ∧ Q(x)] -> R(x, y)
   b. ∀ x: ∀ y: ∃z: (R (y, z)) -> F(x, y)

   After transformation the statements become:
   a. ∀ x : ∀ y: ¬[P(x) ∧ Q(x)] ∨ R(x, y)
   b. ∀ x : ∀ y: ¬(∃z : R (y, z)) ∨ F(x, y)
2. Apply deMorgan's laws [which says that ¬ (a ∧ b) = ¬a ∨ ¬b and ¬(a ∨ b) = ¬a ∧ ¬b] and if necessary do the transformations between quantifiers [¬∀x: P(x) = ∃x: ¬P(x) and ¬∃x: P(x) = ¬∀x: P(x)] and reduce each ¬ to a single term [using ¬(¬P) = P]. After performing these transformations of the wff generated in step 1, the statements become:
   a. ∀ x : ∀ y: [¬P(x) ∨ ¬Q(x)] ∨ R(x, y)
   b. ∀ x : ∀ y: (¬∃z : R (y, z) ∨ F(x, y))]
3. Each quantifier is bound to a single variable. Since change in variable names do not affect the truth value of the wff, we can just replace one name with another. For e.g., the formula
   ∀ x : F(x) ∨ ∀ x : G(x)

Can be converted to

$\forall x : F(x) \vee \forall y : G(y)$

4. Move all the quantifiers to the left of the formula. While moving them, it should be kept in mind that the order in which the quantifiers appear in the wff is maintained. Performing this operation on the formula gained in step 2, we get
   a. $\forall x : \forall y : [\neg P(x) \vee \neg Q(x)] \vee R(x, y)$
   b. $\forall x : \forall y : \forall z : (\neg R(y, z)) \vee F(x, y)$
5. Eliminate all existential quantifiers. A formula that contains an existentially quantified variable means that the variable can contain a particular value which makes the formula true. We can substitute the variable with a function which produces the value. Therefore, the formula

   $\forall y : \exists z :$ grandfather (z, y)

   Can be transformed into

   $\forall y :$ grandfather $(S(y), y)$
6. If prefixes exist, then drop all of them. The prefixes now contain some variables which are universially quantified and they can be dropped, assuming the fact that all variables that appear in the formula are quantified universally. The formula generated in step 4 now would be
   a. $[\neg P(x) \vee \neg Q(x)] \vee R(x, y)$
   b. $(\neg R(y, z)) \vee F(x, y)$
7. Convert the formula into a conjunction of disjuncts. Since our formula contains no conjunctive (or and), therefore, we can remove the parenthesis to get the clause
   a. $\neg P(x) \vee \neg Q(x) \vee R(x, y)$
   b. $\neg R(y, z) \vee F(x, y)$
8. Create a separate clause for each conjunct. A wff is assumed to be true, if all the clauses it contains are true.

3.4.2 The Basis of Resolution:

Resolution is a simple process which tries to resolve a problem by comparing two clauses at each step. The clauses are known as the parent clauses. The comparison process yields a new clause inferred from the parent clauses. Suppose, we have the following two clauses

love ∨ hate

¬love ∨ affection

As we already have discussed in section 5.7 that both the clauses must be true. Though the clauses look separate and independent, they are related or conjoined.

Now, it should be observed that love and ¬love cannot be true at the same time. At one point, if love is true, then affection must be true in the second clause. And if ¬love is true in one clause, then hate must be true to guarantee the truth of both the clauses. Therefore, these two parent clauses deduce

hate ∨ affection

The resolution procedure follows such type of deduction. Resolution operates on two clauses that each clause must contain at most one same literal. One must occur in positive form and the other is in negative form. The resolvent is obtained by combining the literals of the parent clauses.

So far whatever we have discussed is the resolution in propositional logic. In predicate logic, the resolution procedure is slightly more complicated. Resolution in predicate logic attempts to find the answer to questions by substituting the values of the variables. The substitution process requires matching the arguments of the predicates. For e.g., Roman(Joe) and ¬Roman(Tom) is not a contradiction. We need a matching process to compare two literals and substitute one with another to make them identical. The

procedure which does this is known as the unification process which we shall discuss in the following section.

3.4.3 The Unification Process:

The objective of unification process is to unify two literals which have their predicate symbols same. If so, we will proceed by checking their arguments in pair. Otherwise, we would not be able to unify them. For e.g., the two literals

Roman(Joe)
Greek(Joe)

cannot be unified. But, if the first pair matches, we can continue by checking the second pair and so on. We can call the unification procedure to match the arguments by applying the matching rules. A variable can be matched with another variable, with a constant or with a predicate expression. Here, we will see how unification can be applied on variable or constant. The substitution process must be consistent for each literal. For e.g., suppose we have the following two axioms to be unified:

know(Tom, Joe)
know(Ben, Sue)

The predicate know matches in both the axioms. Next, we will compare the first pair of arguments which are Tom and Ben and look for a substitution Ben/Tom (which means replace Tom with Ben). Now if we continue with the next pair of arguments and apply the substitution Sue/Joe (which replaces Joe with Sue). The composition of substitutions for entire unification process would be:

(Sue/Joe) (Ben/Tom)

But, if the same variable is replaced by more than one substitution, then the entire unification process will go wrong. Therefore, the

unification process is considered successful if we can have a single, consistent substitution for each variable.

3.4.4 The Resolution Process in a Question-answering System:

Resolution attempts to give response to questions by considering the set of axioms which are converted to a clausal form. We have already discussed the conversion process of predicate logic statements in section 5.3.1. Now after the transformation, the statement which is to be proved must be negated as well as converted into clausal form. Apply unification as and when required and repeat until we get either a contradiction situation or no progress can be made. At each step, two clauses will be chosen and resolved. If the resolvent is an empty clause, we will get a contradiction. Let us now consider the following set of axioms:

1. Julioclaudian (Agrippina)
2. Lugdunumish(Claudius)
3. ∀ x : Lugdunumish (x) -> Gaulish(x)
4. emperor(Claudius)
5. ∀ x : Gaulish (x) -> admired(x, Claudius) ∨ tyrant(x, Claudius)
6. ∀ x : ∃y : admired(x, y)
7. ∀ x : ∀y : Julioclaudian (x) ∧ emperor(y) ∧ triedtoexecute (x, y) -> ¬admired(x, y)
8. ∀ x : ∀y : tyrant(x, y) -> triedtoexecute(x, y)
9. triedtoexecute (Agrippina, Claudius)

This knowledge base must be converted into the following clausal form:

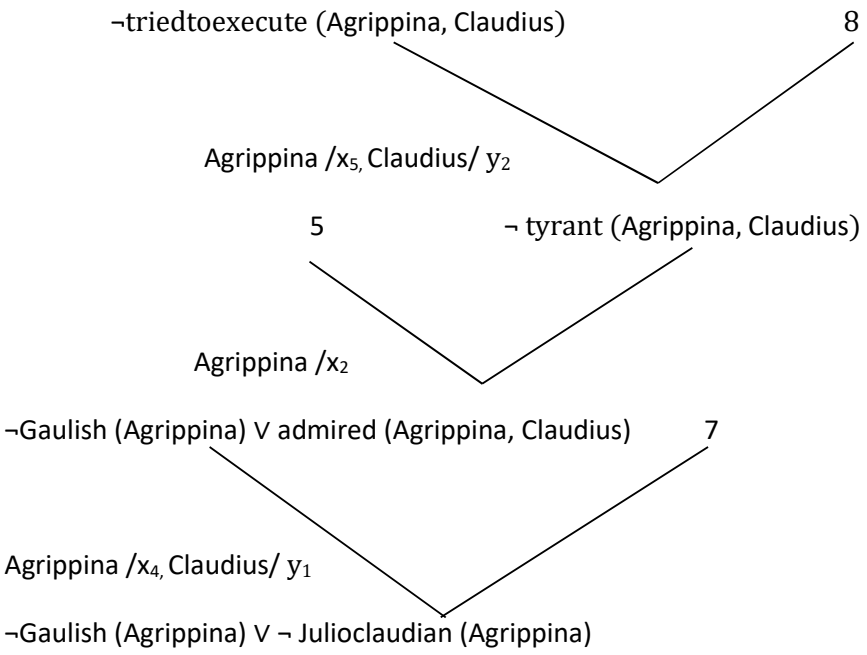
1. Julioclaudian (Agrippina)
2. Lugdunumish(Claudius)

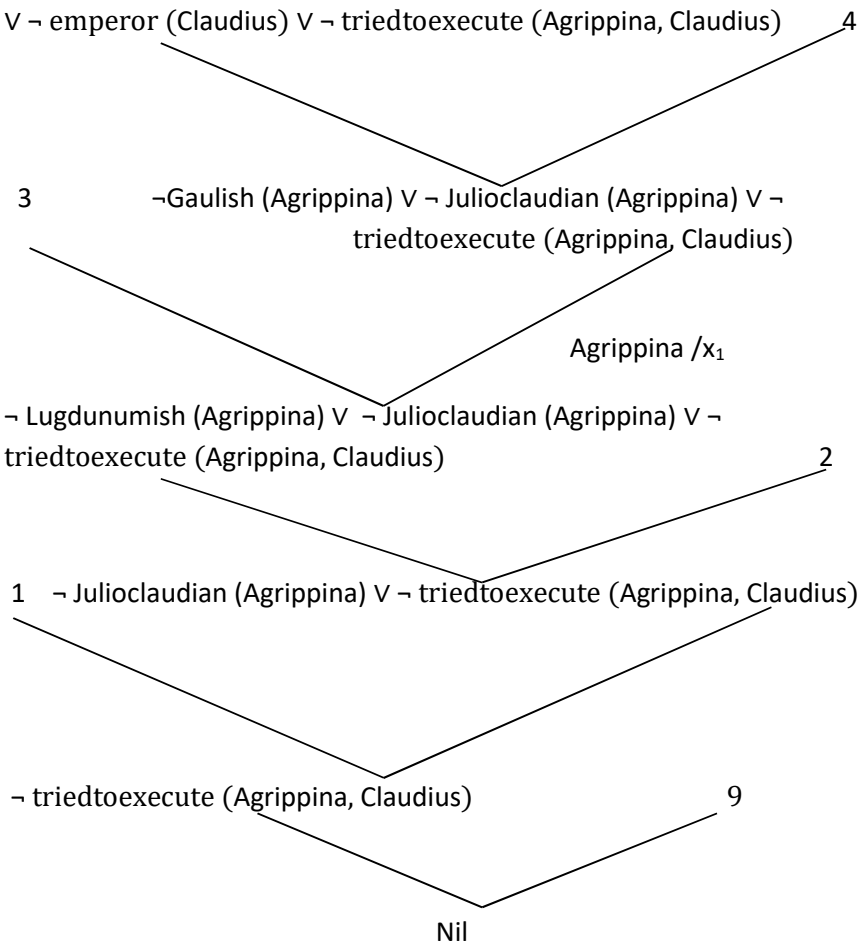
3. ¬ Lugdunumish (x_1) ∨ Gaulish(x_1)
4. emperor (Claudius)
5. ¬Gaulish (x_2) ∨ admired (x_2, Claudius) ∨ tyrant (x_2, Claudius)
6. admired (x_3, f(x_3))
7. ¬ Julioclaudian (x_4) ∨ ¬ emperor (y_1) ∨ ¬ triedtoexecute (x_4, y_1) ∨ ¬ admired (x_4, y_1)
8. ¬ tyrant(x_5, y_2) ∨ triedtoexecute(x_5, y_2)
9. triedtoexecute (Agrippina, Claudius)

Now, suppose, we want to answer the question "Did Agrippina tried to execute Claudius?" For this, the resolution theorem prover must start to prove the statement by negating the assertion:

¬ triedtoexecute (Agrippina, Claudius)

The proof procedure tries to prove this statement in the following manner:





Thus, we can conclude that the resolution theorem prover for the assumed statement has produced a contradiction with the statements provided. It should be noted that our assumption of negating the assertion has proven to be wrong. Therefore, we can deduce that "Agrippina tried to execute Claudius". The response to our query would be "Yes".

Check Your Progress-I

1. The return type of computable function is always _____.
2. State whether true or false:
   a) Resolution attempts to prove a statement by converting it into clausal form.
   b) Resolution attempts to prove a statement by negating it.
   c) Resolution tries to prove that the negation of a statement is unsatisfiable.

3.5 NATURAL DEDUCTION

We have got to know that resolution is a technique in which a proof procedure exists for a problem statement. The procedure is simple and unifies two clauses at each step. The clauses are required to have uniform representation of the statements. This technique raises a serious issue regarding selection of clauses. If the knowledge base is a larger one and since the statements are uniform, it is difficult to select those statements which are most likely to lead to solution of the problem. In addition to this, this process involves an overhead of converting the statements in clausal form. This has the serious drawback of losing valuable information contained in the original representation. For example, considering the following representation:

$$\forall x : \text{student}(x) \rightarrow \text{good}(x)$$

This assertion depicts the fact that all students are good. The equivalent clausal form of this statement would be:

$$\neg\text{student}(x) \vee \text{good}(x)$$

This implies the deduction of the fact that someone is not a student by showing that he is not good. But, it is obviously not the best way to show that someone is not a student.

We already have mentioned that resolution proves a statement by refutation. It is not easy for a person to think and interact according to resolution theorem prover. People find it impractical to think in terms of resolution and accordingly attempt to solve a problem. Therefore, it is important to find a way that facilitates such interaction as well as the proof procedure seems to be very natural. In other words, it corresponds more closely to the way a human theorem prover attempts to find solution to a problem. This is natural deduction which consists of a combination of techniques; rather than a single one. While solving a problem, all these techniques work

together. Natural deduction does not consider predicates. Instead, it considers only the objects involved in these predicates. Rules are written in a manner that not only describe the condition-action pair or implications; but provides suggestion about the manner in which these rules are applied in proofs.

3.6 LOGIC PROGRAMMING

Logic programming is a programming paradigm which provides the facility of being able to write logical assertions as programs. There are several logic programming systems available today like PROLOG, LISP etc. A PROLOG program may consist of a series of several logical statements known as Horn Clauses. A Horn Clause contains at most one positive literal, like p, ¬p ∨ q and p ->q. A logical assertion can be converted into Horn Clause. Below, we shall see some logical assertions and the same assertions transformed into Horn Clauses.

∀ x : cube(x) ∧ small(x) -> dice(x)

∀ x : cat(x) ∨ dog(x) -> pet(x)

∀ x : leopard(x) -> carnivorous(x) ∧ has_spot(x)

bischon_frise(piper)

Representations in Logic

dice(X) : - cube(X) , small(X).

pet(X) : - cat(X).

pet(X) : - dog(X).

carnivorous(X) : - leopard(X).

has_spot(X) : - leopard(X).

bischon_frise(piper).

Representations in PROLOG

Since the PROLOG programs consist of Horn Clauses only, because of the uniformity, a simple and efficient interpreter can be designed to take all the charges. The goal to be attained is fed as the input into a PROLOG program. The PROLOG interpreter tries to attain the goal state just by chaining backward; from the goal to the initial configuration. The program is read in sequence from the top, left-to-right and the depth first search with backtracking is performed. The PROLOG program may contain two types of statements- facts, which contains only constants not variables and rules, which contains only variables. It needs to be mentioned here that there are some rule-based differences that exist between PROLOG statements and logical assertions.

1. As seen above, logical propositions contain variables that are explicitly quantified. In PROLOG, variables are declared implicitly. The variables in PROLOG statements are written in uppercase whereas the constants are in lowercase or numbers.
2. In logic, there exists symbols designating and (∧) and or (∨). But in PROLOG, and is designated by (,) and none for or.
3. In logic, implication statement such as “p implies q“ is written as “ p -> q “ symbol; whereas the PROLOG implication statement is written backward like “q : - p”.

A PROLOG interpreter has a fixed control strategy; so, the assertions follow a particular search strategy to find the answers to questions. Whereas, the logical assertions do not say anything about how to choose the most appropriate answer when more than one are available. The control strategy of PROLOG always starts with the problem definition; that is, with the goal to be proved. Then it searches for the set of assertions that can satisfy the goal. For this, it

looks for the fact that can prove the goal directly or the rule whose head matches with the goal. The decision of whether the rule or fact can be applied is left to the unification procedure. Then it reasons backward from the goal until a path is found which terminates with the initial assertions of the program. The search path may also follow the backtracking strategy.

Now let us consider the following PROLOG program and see how the PROLOG interpreter works to find the responses to queries.

```
grandchild (X, Y)      : - child(X, Z), child(Z, Y).
child (X, Z)           : - son (X, Z).
child (X, Z)           : - daughter (X, Z).
child (Z, Y)           : - son (Z, Y).
child (Z, Y)           : - daughter (Z, Y).
son(bob, ann).
daughter(ann, john).
```

Now, suppose, we have the following query :

```
?- grandchild (bob, john).
```

This assertion is fed as the input to the program. The interpreter tries to look for the fact with the predicate grandchild or a rule whose head matches with that predicate. A PROLOG program is written with the facts coming prior to the rules so that the fact with a particular predicate can be used first. And if there is no such appropriate fact found, then the rule containing that predicate is chosen. In this program, we can see that there is no such fact present which matches its head with the predicate “grandchild”. However a rule containing grandchild as its head can be used. The rule can be applied successfully if its right hand side can also be satisfied. Next the interpreter will try to prove each of them. The first predicate that needs to be proved child (X, Z). Now, the interpreter will search for the appropriate fact or rule whose head matches the predicate child

because there is no such fact containing the predicate child exists in the program. There are four such rules available, but based on the arguments we must apply only the first two to get the most appropriate direction towards the solution. Applying the first rule requires to prove son(X, Z). This in turn applies the fact with the predicate son (bob, ann). Now, the values for X and Z corresponding to this fact will be X=bob and Z=ann. Now, the second child predicate of the first rule needs to be proved. Based on the arguments, the PROLOG interpreter again tries to use the third rule containing son (Z, Y) predicate. But, this will no longer help because already the son predicate has been attained. Now, applying the fourth rule with head child (Z, Y), the fact with daughter predicate is used and the values for Z and Y will be Z=ann and Y=john. The child predicate is now satisfied and the interpreter attains the values for X, Z and Y are X=bob, Z=ann and Y=john. Now, we have attained a situation where the predicate grandparent has been proved and satisfied. Therefore, the result of the query would be- yes.

3.7 DECLARATIVE VS. PROCEDURAL KNOWLEDGE

A declarative knowledge is a kind of knowledge represented in a specific format that is analyzed, decomposed or manipulated by the reasoners. This kind of representation specifies the knowledge; but does not say anything about how to achieve a particular goal using that representation. We need to augment a declarative representation within a program which specifies what is to be solved or achieved and how.

Procedural knowledge is also known as imperative knowledge. The procedural knowledge is the one which encodes how to achieve a particular goal. The control structure that is needed to achieve the goal

must be incorporated into the knowledge base itself. An interpreter must be augmented to manipulate and reason all kinds of knowledge defined in the knowledge base. The interpreter searches the assertions in sequence from top to bottom and proceed in depth-first fashion to establish a new sub goal. If the new sub goal is not satisfiable, then an alternative and effective route is considered and followed. To see how the declarative and procedural representations work on a set of knowledge base, let us consider the following knowledge base:

1. Julioclaudian (Agrippina)
2. Julioclaudian (Claudius)
3. ∀ x : Julioclaudian (x) -> Roman(x)
4. Julioclaudian (Nero)

Now, based on the above knowledge base, let us try to answer the question:

∃y : Roman(y)

If we consider the declarative representation, the assertions do not say anything about how they will be examined. Since, there are more than one value that satisfies the query predicate, the answer we get are:

Y = Agrippina
Y = Claudius
Y = Nero

But, if we follow the procedural representation, only one value we will generate; because the responses are based on the sequence in which the assertions are being examined. Thus the above knowledge base will generate the answer:

Y = Agrippina

This is because the statement which actually achieves the goal to find the Roman is

∀ x : Julioclaudian (x) -> Roman(x)

This in turn again sets up a sub goal to find a Julioclaudian. The fact which satisfies this sub goal is :

Julioclaudian (Agrippina)

Therefore, the answer that is returned is Agrippina.

3.7.1 Fundamentals of Matching:

So far, we have discussed that search helps in achieving solutions to problems. Upon the application of the most appropriate rule to the individual problem state, the search process can generate a new state and so forth until a solution is found. An efficient search process can choose the best one, which most likely can lead to success, when more than one applicable rules are available. To do so, what we need is to collect the entire sequence of rules that can be applied at a given point. And this is done by the matching process which matches the current state with the heads of the rules. Matching involves some techniques to extract the applicable rules. One of them is discussed below:

3.7.1.1 Indexing:

It is a technique which does a simple search over the set of rules and selects the appropriate ones. It does so by comparing each rule's preconditions with the current state and extracting the ones that match. But, this may lead to two important consequences. First, to solve s problem, sometimes it is necessary to use a huge set of rules. Extracting the important ones is really a big deal and may make the system inefficient. Second, it is not necessary that a rule's preconditions will always match the current state. To deal with such kind of situation, a new approach that has been taken is to use the current state as an index into the set of rules and select the ones that

match. The index can be generated by a perfect hashing function. Thus the matching process has become easy with this method.

Check Your Progress3

3. a -> b = ?
4. Can know(Tom, Joe) and student (Tom, Joe) be unified?
5. Unification process can unify two_____in propositional logic.
6. In resolution procedure, one predicate must be the negation of the other one. True or false?
7. In PROLOG, the logical assertions are converted into _____ clauses.
8. PROLOG interpreter uses depth-first search with _____.
9. The PROLOG interpreter tries to attain the goal state by _____ backward.
10. p -> q in logic implies _____ in PROLOG.
11. Procedural knowledge is also known as _____ knowledge.
12. Do we need an extra program to reason with declarative knowledge?
13. An _____ is sufficient to manipulate procedural knowledge.

3.8 SUMMING UP

- Logical representations of real-world facts are important from the point that it is one of the common ways of presenting knowledge.
- Basically two major kinds of logical formalisms are available- 1) The propositional logic and 2) The predicate logic.
- The propositional logic is simple and has a decision procedure as the facts can be expressed using logical assertions.

- The propositional logic differs from predicate calculus from the fact that the former captures only the facts which are specialized ones. Whereas, the later tries to capture more generalized knowledge structures.
- In predicate logic, the factual information about a particular domain is represented by First Order Predicate Logic (FOPL).
- The logical assertions in FOPL must be written using logical connectives.
- The logical assertions can also be written using computable functions like gt(), lt(), which may evaluate to either true or false.
- For deriving new knowledge or for reasoning about the statements in logic or for generating answers to questions, resolution procedure may be applied.
- Resolution attempts to prove a statement by starting with the negation of that statement.
- If the statement evaluates to a contradiction, then it must be concluded that our assumption of negating the statement was wrong.
- It should be noted that here that during the resolution proof procedure, the unification procedure may be applied.
- Resolution embodied with unification tries to prove the statements or attempts to derive answers to questions.
- Two very important things that should to be kept in mind during the resolution proof procedure are that 1) At every step literals involved must have one in common and one should be the negation of the other and 2) The logical assertions must be converted into a convenient clausal form so that an efficient and smooth resolution process could be executed.
- The PROLOG interpreter tries to find responses to questions by starting with the goal, and then chains backward using the

matching procedure to get into the position where the results can be generated.

- Knowledge may be in one of two basic forms- declarative and procedural.
- In declarative knowledge representation formalism, only the knowledge is provided; but it does not provide any information regarding how to get responses to queries.
- On the contrary, the procedural knowledge follows a specific control mechanism to get the solution and therefore, a decision procedure exists for such kind of knowledge.

3.9 ANSWERS TO CHECK YOUR PROGRESS

1. boolean.
2. a) True
   b) True
   c) True.
3. a -> b = a ∨ b
4. No, know (Tom, Joe) and student (Tom, Joe) cannot be unified. This is because, their literals are not same.
5. literals.
6. In resolution procedure, one predicate must be the negation of the other one. This statement is true.
7. Horn.
8. backtracking.
9. chaining.
10. q : - p.
11. imperative.
12. Yes, we need an extra program to reason with declarative knowledge.
13. interpreter.

3.10 POSSIBLE QUESTIONS

Short answer type questions:

1. Explain in brief the language of logic.
2. What are ways of representing facts in logic? Mention.
3. What are the different connectives used in propositional calculus?
4. Mention the different quantifiers used in predicate calculus?
5. What do you mean by computable functions?
6. What do you mean by inference rule?
7. What is resolution?
8. Write down some features of PROLOG.
9. What is declarative knowledge?
10. What is procedural knowledge?
11. What is predicate calculus?

Long answer type questions:

1. How facts can be represented using language of logic?
2. What is propositional logic? How does it differ from predicate logic?
3. What are the syntaxes used in First Order predicate Logic?
4. How FOPL expressions can be converted into instance and isa relationships?
5. How and in what situations can computable functions be used in predicate logic expressions?
6. Define resolution. What is the basis of resolution?
7. With the help of an example, show the process of converting a predicate logic expression into clausal form.
8. Describe the unification process.
9. What is logic programming? Explain.
10. Consider a set of statements in logic and convert it into the corresponding PROLOG statements. Also write down the differences between representations in logic and that of PROLOG.

11. How does a PROLOG interpreter derive responses to queries? Explain with an example.
12. Differentiate between declarative and procedural knowledge. Explain the differences with the help of an example.
13. Consider the following set of facts and convert them into their corresponding predicate calculus expressions:
    a) All dogs bark at night.
    b) Birds fly.
    c) Piper does not fly.
    d) All judges are not crook.
    e) The sky is blue.
    f) John is unmarried.
    g) Both Ann and Sue are students.
    h) If Joe is a politician, then he is crook.
    i) Anyone who passes exam is either happy or excited.
    j) To be a parent, one must be a father or mother of someone.
    k) What is natural deduction? Explain.

3.11 REFERENCES AND SUGGESTED READINGS

1. Bratko I. 2001. *PROLOG programming for Artificial Intelligence.* Pearson Education, Ltd.
2. Luger G. F. 2002. *Artificial Intelligence Structures and Strategies for Complex Problem Solving*. Pearson Education, Ltd. and Dorling Kindersley Publishing.
3. Patterson, D. W. 1990. *Introduction to Artificial Intelligence and Expert Systems.* New Jersey: Pearson Education.
4. Rich, A., and Knight K. 1991. *Artificial Intelligence.* New York: Tata McGraw-Hill.

*****

UNIT 4: KNOWLEDGE REPRESENTATION USING RULES I

Unit Structure:

4.1 INTRODUCTION

In this unit we are going to discuss the use of rules to encode knowledge. We have already discussed about rules as the basis for a search program. Here we will consider a set of rules to represent both knowledge about relationships in the world as well as knowledge about how to solve problems using the content of rules.

4.2 OBJECTIVES

After going through this unit, you will be able to-

- Understand Procedural and Declarative Knowledge

- Understand the difference between Procedural and Declarative Knowledge.
- Understand logic programming
- Understand Forward and Backward Reasoning

4.3 PROCEDURAL VERSUS DECLARATIVE KNOWLEDGE

A procedural knowledge is a representation in which the information required to use the knowledge is embedded in the knowledge itself. Thus only an interpreter is necessary that will follow the instructions given in the knowledge. For example computer programs, directions, recipes which indicate specific use and implementation. Let's understand procedural knowledge elaborately with the help of an example.

Example: *man (Marcus)*

man (Caesar)

person (Cleopatra)

$$\forall x: man(x) \longrightarrow person(x)$$

Now let us try to answer the question

$$\exists y: person(y)$$

The knowledge base justifies any of the following answer:

y = Marcus

y = Caesar

y = Cleopatra

We get more than one value that satisfies the predicate.

If only one value needed, then the answer to the question will depend on the order in which the assertions examined during the search for a response.

If the assertions are declarative then they do not themselves say anything about how they will be examined. In case of procedural representation, they say how they will examine.

Examples of Procedural Knowledge :

a) Knowledge involved in writing a sorting algorithm that requires understanding the specific steps such as bubble sort or quick sort, to sort a number of elements.

b) Knowledge involved in training a neural network that requires understanding of the architecture, activation functions, back propagation and optimization techniques.

On the other handa declarative knowledge is a representation in which the knowledge is specified but the application of this knowledge as where to utilise it is not specified. Universal truths, laws are some of the facts that can stand alone and does not depend on other knowledge. For using this knowledge, it needs to be augmented with a program that will specify what and how this knowledge is to be used. For example, a set of logical assertions can combine with a resolution theorem prover to give a complete program for solving problems but in some cases, the logical assertions can view as a program rather than data to a program. Thus define the legitimate reasoning paths are defined by the implication statements and automatic assertions provide the starting points of those paths. These reasoning paths define the execution path similar to “if-then-else” execution.

The real difference between declarative and procedural views of knowledge lies in where the control information resides.

Let us consider the same example discussed above but in a different order:

man (Marcus)

man (Caesar)

$\forall x : man(x) \rightarrow person(x)$

person (Cleopatra)

In this case, if declarative knowledge is considered then all the answers are supported by the system and none of them is explicitly selected. But if procedural knowledge is considered and as the knowledge base is bit different from the earlier one, here we will get the answer as *Marcus* but not *Cleopatra*. This happens as the first statement that satisfies the person goal is the inference rule $\forall x: man(x) \rightarrow person(x)$. The sub goal of this rule is to find a man. As the statements are once again examined from the initial statement, Marcus first satisfies both the sub goal as well as the goal. So we get the answer as *Marcus.*

Examples of Declarative Knowledge:

a) Knowledge involved in medical diagnosis which requires understanding the symptoms, diseases and their relationships enabling the system for accurate diagnosis.

b) Knowledge involved in recommender systems which requires understanding of user preferences, item attributes and historical data to provide personalized recommendations.

Difference between Procedural and Declarative Knowledge Representation

Procedural Knowledge	Declarative Knowledge
1. High efficiency	1. Higher level of abstraction
2. Low modifiability	2. Suitable for independent facts
3. Low cognitive adequacy	3. Good cognitive matching

4. Procedural knowledge means to incorporate on AI systems through procedures like LISP and PROLOG languages	4. Declarative knowledge means to incorporate on AI systems through Declarative mechanisms like Semantic Nets, CD diagrams, frames and scripts
5. Object facts	5. Rule procedure

STOP TO CONSIDER

- Procedural is also referred to as Imperative knowledge.
- Declarative knowledge is referred to as Functional knowledge

4.4 LOGIC PROGRAMMING

Logic programming is an effort made by computer scientist to make machines able to decide or to reason so that it becomes useful for knowledge representation. Two main components of logic programming are logic and inference. Logic is used to represent knowledge and inference is used to manipulate it. Logic programming is a paradigm which helps in building logical assertions as programs. Although there are various logic programming languages, PROLOG is the most popular one.

A PROLOG program is a collection of facts and rules. It is a series of logical assertions each of which is a Horn clause. A *Horn Clause* is a clause with at most one positive literal, it is thus either:

1. A single positive literal, which is regarded as a *fact*,
2. One or more negative literals, with no positive literal, or
3. A positive literal and one or more negative literals which is a *rule*.

Thus p, $\neg p$ V q and p $\rightarrow q$ are all horn clauses.

In the example below we simply show the difference between Logic representation and PROLOG representation.

$$\forall x: pet(x) \wedge small(x) \rightarrow apartmentpet(x)$$

$$\forall x : rabbit(x) \vee dog(x) \rightarrow pet\ (x)$$

$$\forall x: pygmy(x) \rightarrow rabbit(x) small(x)$$

pygmy(tomy)

Logic Representation

apartmentpet (X):- pet (X), small (X)

pet (X):- rabbit (X)

pet (X):- dog(X)

rabbit (X):- pygmy (X)

small (X) :- pygmy (X)

pygmy (tomy)

PROLOG Representation

PROLOG programs are composed of only Horn Clauses and hence has a uniform representation. So designing the interpreter becomes simple and efficient. Another important fact is that the logic of the Horn clause is decidable. The input to a PROLOG program is the goal to be proved. The PROLOG interpreter tries to attain the goal by applying backward reasoning, provided the assertions are given in the program. The program is read top to bottom, left to right and search is performed depth-search with backtracking. There are two types of statements in PROLOG program: facts and rules. Facts contain only constants and represents statements about specific objects. On the other hand rules contain variables and represents statements about classes of objects.

Let's now see the rule-based differences between logic and the PROLOG representations:

1. Variables in logic representations are explicitly quantified whereas in PROLOG variables are declared implicitly. All variables in PROLOG begin with upper case letters and all constants begin with lower case letters or numbers.

2. Symbols that exists in logic are *and* (∧) and *or* (∨). In PROLOG symbol used for **and** is (,) whereas there is no symbol representing *or*.

3. In logic implication statement of the form "p implies q" are written as p →q. On the other hand the same implication in PROLOG is written as q: - p as the interpreter always work backwards from a goal.

Logic and PROLOG mainly differs in the way they are represented. The interpreter in PROLOG has a fixed control strategy and hence the assertions in PROLOG program define a particular search path to an answer to any question. On the other hand logic representations only justifies the answers they define but says nothing about how to choose the answer when there are more than one.

Let's now discuss in details about the control strategy in PROLOG. It always starts with the problem statement. The problem statement is nothing but the goal to be proved. Next is to look for the assertions that could prove the goal. The facts that prove the goal directly as well as the rule whose head matches the goal are considered. The decision to apply a fact or a rule is accomplished by the unification procedure. Now it is reasoned backward from the goal to find out whether there is a path that terminates with assertions in the program. The procedure of searching the path continues until it suffices the condition. The paths are considered using a depth first search and backtracking.

Conventions used for generating proof/ search tree of a PROLOG query

a) Start generating tree with root as goal G to be satisfied

b) Downward arrow indicates the reduction of goal.

c) Clause number on the left side of an arrow.\

d) Possible bindings enclosed within curly brackets (if necessary) on right of an arrow.

e) Leaf of the tree is labelled either succeeds or fails.

The following example will illustrate the execution of a PROLOG query

The example consists of one rule for defining grandfather and five facts about father relations. A rule is defined as: X is a grandfather of Y if X is father of Z and Z is father of Y.

grandfather(X, Y):- father(X,Z), parent(Z,Y)

(1)

parent(X, Y) :- father(X, Y)

(2)

parent(X, Y) :- mother(X, Y)

(3)

father (james, robert)

(4)

father (mike,william)

(5)

father (william, james)

(6)

father (robert, hency)

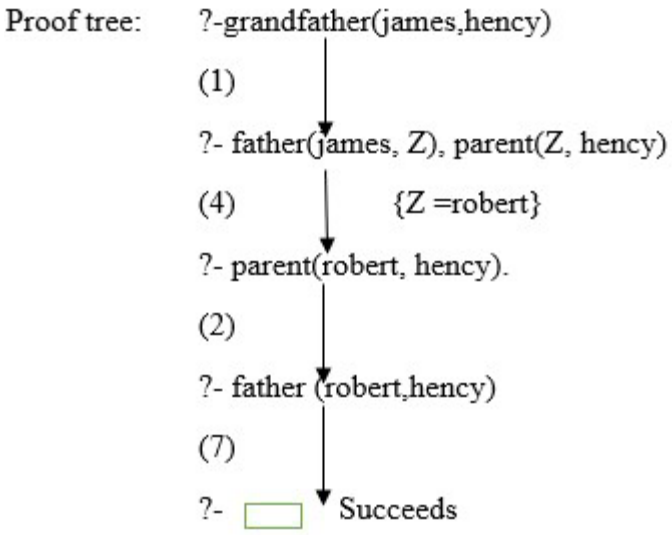
(7)

father (robert, cris)

(8)

Suppose we have the query

?-grandfather(james,hency)



Here the interpreter tries to look for the fact grandfatType equation here.her or a rule which head matches with the predicate. A PROLOG program is written with the facts coming prior to the rules so that the fact with a particular predicate can be used first. And if no such appropriate fact is found, then the rule containing the predicate is chosen.

Answer: Yes

Let's consider another query:

?-grandfather(james, william)

Answer: No

Check Your Progress

1. Knowledge involved in training a neural network requires _____knowledge.
2. A_____ program is a collection of facts and rules.
3. A Horn Clause is a clause with at most _____ positive literal
4. PROLOG programs are composed of only _____ and hence has a uniform representation.
5. Variables in logic representations are _____ quantified

4.5 FORWARD VERSUS BACKWARD REASONING

A search procedure must always discover a path through a problem space from an initial configuration to a goal state. There are actually two directions in which a search procedure can proceed. They are:

- Forward search which starts from the start state
- Backward search which starts from the goal state

The production system provides an easy way of viewing forward and backward reasoning as symmetric processes. Let us consider a game of playing 8 puzzles. Assuming that the areas of the tray are numbered shown in Fig 1, the rules defined for the puzzle are depicted in Fig 2

1	2	3
4	5	6
7	8	9

Fig 1: The areas of the tray are numbered

Square 1 empty and Square 2 contains tile n →

Square 2 empty and Square 1 contains tile n

Square 1 empty and Square 4 contains tile n→

Square 4 empty and Square 1 contains tile n

Square 2 empty and Square 1 contains tile n→

Square 1 empty and Square 2 contains tile n

Fig 2: A sample of the rules for solving the 8-puzzle

There are two ways in which the puzzle can be solved:

1. Reasoning forward from the initial state:

Step 1: Start building a tree of move sequences that might be the solution with the initial configuration at the root of the tree.

Step 2: Generate the next level of the tree by finding all the rules whose left sides match the root node. The right side is used to create the new configurations.

Step 3: The next level is generated by taking each node generated at the previous level and the rules whose left side matches are applied to it. This is continued until a configuration that matches the goal state is generated.

2. Reasoning backward from the goal state:

Step 1. Begin building a tree of move sequences by starting with the goal node configuration at the root of the tree.

Step 2. Generate the next level of the tree by finding all rules whose right-hand side matches against the root node. The left-hand side used to create new configurations.

Step 3. Generate the next level by considering the nodes in the previous level and applying it to all rules whose right-hand side match. So, the same rules can use in both cases.

Also, in forwarding reasoning, the left-hand sides of the rules are matched against the current state and right sides used to generate the new state. Moreover, in backward reasoning, the right-hand sides of the rules matched against the current state and left sides are used to generate the new state. This process is continued until one of these goal states is matched by an initial state. Sometimes searching in one direction is significantly better than the other direction. The searching depends on the topology of the problem space.

The four factors that influence the question of reasoning forward or backward are:

- First we need to observe that whether the number of possible states is more or the goal states. Then we would prefer to move from smaller set of states to the larger set of states.
- Next is to observe the direction of the branching factor. We would like to proceed in the direction with the lower branching factor.
- To check whether the program will be asked to justify it's reasoning process to a user. If yes, then it is important to proceed in the direction that corresponds more closely with the way the user will think.
- Lastly to find the kind of event that is going to trigger a problem-solving episode. If it is the arrival of a new fact, forward reasoning makes sense. If it is a query to which a response is desired, backward reasoning is more natural.

Let us consider some examples for understanding the concept of Forward versus Backward reasoning.

Example 1: It is always easier to drive to home from an unfamiliar place rather than to drive to an unfamiliar place from home. Now if we consider home as the starting place and unfamiliar place as goal state then we have to backtrack from unfamiliar place to home.

Example 2: Let us consider the problem of symbolic integration where the problem space is the set of formulas. Some of this formula might contain integral expressions. Here START is one of these formula containing integrals and the GOAL state is equivalent to the expression of the formula without any integral. Here we start from the formula with some integrals and proceed to an integral free expression rather than starting from an integral free expression.

Example 3: The third factor is nothing but deciding whether the reasoning process can justify its reasoning. If it justifies then it can apply. For example, doctors are usually unwilling to accept any advice from diagnostics process because it cannot explain its reasoning.

Although the underlying principle or the set of rules that can be used for both forward and backward reasoning are same, two classes of rules are proved to be useful, each of which encodes a particular kind of knowledge.

- Forward rules, to respond to certain input configurations
- Backward rules, to achieve particular goals

4.5.1 Backward-Chaining Rule Systems

PROLOG is a good example of backward chaining rule system. PROLOG rules are restricted to Horn clauses. This allows for rapid indexing because all of the rules for deducing a given fact share the same rule head. Rules matched with unification procedure. Unification tries to find a set of bindings for variables to equate a sub-goal with the head of some rule. Rules in the PROLOG program matched in the order in which they appear.

4.5.2 Forward-Chaining Rule Systems

In forward-chaining systems, left sides of rules are matched against the state description. The rules that match dump their right-hand side assertions into the state and the process repeats. The matching process in forward chaining system is more complex than backward ones. For example, let's consider a rule that checks for some condition in the state description and adds an assertion. The rule fires and could fire again immediately as it's condition are still valid. To prevent repeated

firings, we need some mechanism to prevent especially when the state remains unchanged. Most forward-chaining systems implement highly efficient matchers and supply several mechanisms for preferring one rule over another.

4.5.3 Combining Forward and Backward Reasoning

A combination of both forward and backward reasoning methods are frequently employed on AI systems to tackle complex problems and make intelligent decisions. This hybrid approach takes advantage of the strengths of each reasoning method and allows AI systems to complement one another, resulting in more efficient and effective problem-solving processes. Let us go through some examples where a combination of both the approaches is used.

- Medical Diagnosis: Backward reasoning can be used to to identify the necessary symptoms and tests required to diagnose a specific condition while forward reasoning can suggest the most likely diagnosis based on available data.
- Route Planning: Backward reasoning can be used for establishing the destination as the goal and identifying the intermediate waypoints in route planning. On the other hand forward reasoning can be used to calculate the most efficient route based on current traffic and road conditions.
- Theorem Proving: In mathematical theorem proving, backward reasoning starts with the desired theorem to be proven and works backward to identify the axioms and logical steps required. Forward reasoning is then used to perform the actual proof.

4.6 SUMMING UP

1. A procedural knowledge is a representation in which the information required to use the knowledge is embedded in the knowledge itself.

2. A declarative knowledge is a representation in which the knowledge is specified but the application of this knowledge as where to utilise it is not specified.

3. Procedural knowledge means to incorporate on AI systems through procedures like LISP and PROLOG languages.

4. Declarative knowledge means to incorporate on AI systems through Declarative mechanisms like Semantic Nets, CD diagrams, frames and scripts.

5. Logic programming is an effort made by computer scientist to make machines able to decide or to reason so that it becomes useful for knowledge representation.

6. PROLOG is the most popular logic programming language.

7. A PROLOG program is a collection of facts and rules. It is a series of logical assertions each of which is a Horn clause.

8. Logic and PROLOG mainly differs in the way they are represented.

9. A search procedure must always discover a path through a problem space from an initial configuration to a goal state.

10.There are actually two directions in which a search procedure can proceed. They are:

- Forward search which starts from the start state
- Backward search which starts from the goal state

11. A combination of both forward and backward reasoning methods are also frequently employed on AI systems to tackle complex problems and make intelligent decisions

4.7ANSWERS TO CHECK YOUR PROGRESS

1. Procedural
2. PROLOG
3. One
4. Horn clauses
5. Explicitly

4.8 POSSIBLE QUESTIONS

1. Consider the following knowledge base:

$$\forall x: \forall y: cat(x) \wedge fish(y) \rightarrow likes - to - eat(x, y)$$

$$\forall x: calico(x) \rightarrow cat(x)$$

$$\forall x: tuna(x) \rightarrow fish(x)$$

tuna (Charlie)

tuna (Herb)

calico (Puss)

Write a PROLOG query corresponding to the question," What does Puss like to eat?" and show how it will be answered by your program?

2. A problem solving search can proceed either forward or backward. What factors determine the choice of direction for a particular problem?
3. Mention the differences between Procedural and Declarative knowledge.
4. How does logic representation differ from PROLOG representation?
5. Explain Forward and Backward reasoning with the help of examples.
6. State some examples where a combined approach for Forward and Backward reasoning is used.

4.9 REFERENCES AND SUGGESTED READINGS

[1]. https://rsisinternational.org/journals/ijrias/DigitalLibrary/Vol.4&Issue12/78-81.pdf

[2]. https://www.youtube.com/watch?v=dNTfeHOqS5A

[3]. https://home.agh.edu.pl/~wojnicki/phd/node24.html

[4]. Rich, E., Knight, K., & Nair, S. B. (2010). Artificial intelligence.

[5]. Kaushik, S. (2007). *Logic and prolog programming*. New Age International.

[6]. https://www.almabetter.com/bytes/tutorials/artificial-intelligence/forward-and-backward-reasoning-in-ai

*****

UNIT 5: KNOWLEDGE REPRESENTATION USING RULES II

Unit Structure:

5.1 INTRODUCTION

In this unit we will continue to learn another set of knowledge rules like matching, indexing, nonmonotonic reasoning and logic, depth first and breath first search. Here we will find out how to select the rule applicable for a particular problem from the set of rules available.

5.2 OBJECTIVES

After going through this unit, you will be able to:

- Understand the concept of matching and indexing
- Understand nonmonotonic reasoning
- Understand depth first and breadth first search

5.3 MATCHING

So far we have only used the process of searching to find out solutions to solve problem, so that solving it may generate it to a new state and then again applying rules until a solution is found. A clever search would be to choose among the rules that applied at a particular point leads to a solution. To choose a rule to be applied at a particular point of time for solving a problem from the entire collection of rules requires some kind of matching between the current state and the preconditions of rules. Now the question is how the rules are being extracted that are applied? There are different matching techniques to extract the applicable rules. One of these is discussed below:

5.3.1 Indexing

Indexing is a technique that does a simple search over the set of rules and selects the appropriate ones. It accomplishes this task by comparing each rule's preconditions with the current state and extracting the ones that match. But this may lead to two important consequence. First, it will be necessary sometimes to go through the whole set of rules to find out solutions to problems. This process will be inefficient as scanning through all of them will simply waste time. Secondly, it is not always obvious that a rule's precondition will always match the current state. To overcome such type of problems the current state can be used as an index into the set of rules and select the matching ones immediately. This index can be generated by any reasonable hashing function. Let us understand with the help of an example. Consider the legal move generation for a chess game. To access the rules appropriate for the legal moves can be done by simply assigning an index to each board position. This is accomplished by simply treating the board description as a large number. After this a suitable hash function is applied to that number as an index into the

rules. Thus all the rules that describe the given board position will be stored under the same key and so will be found together.

5.3.2 Matching with Variables

To discover whether there is a match between a particular situation and the preconditions of a given rule involve a significant search process. In many rule-based system, we need to compute the whole set of rules that match the current state description. So it is efficient to consider the many-many match problem, in which many rules are matched against many elements in the state description simultaneously. One such many many match algorithm is RETE. RETE is a very efficient algorithm as it gains efficiency from three major sources:

- The temporal nature of data. The state description is usually not altered by rules rather it adds one or two new elements or it deletes one or two of it. So the state description remains unaltered. But RETE maintains a network of rule conditions and uses the temporal nature of data i.e the state description to determine which new rules might be applied. The candidates which are more likely to be affected by incoming or outgoing data, full matching can be followed for them.
- Structural similarity in rules. A large number of preconditions may be shared by different rules. So when we try to match two rules independently and they have some common preconditions it may so happen that a lot of work has to be done unnecessarily as it does not have any way to store such rules and share it. RETE tackles this issue easily as it stores rules so that they can share structures in memory. Hence the set of preconditions that appear in several rules are matched once per cycle.

- Persistence of variable binding consistency. A rule might not get fired due to some variable binding conflicts. But this can be avoided as RETE can remember it's previous calculations and is able to merge new binding information efficiently.

5.4 NONMONOTONIC REASONING AND LOGIC

Nonmonotonic reasoning is a technique where we can solve problems with incomplete and uncertain models. In nonmonotonic reasoning the axioms or rules of inference are extended to make it possible to reason with incomplete information. These systems are based on the property that at any given moment a statement is believed either to be true, believed to be false or not believed to be either. Some conclusions in nonmonotonic reasoning can be invalidated if some more information is added to the knowledge base. Let us suppose we are given this knowledge base which contains the following knowledge:

- **Birds can fly**
- **Penguins cannot fly**
- **Pitty is a bird**

So from the above sentences, we can conclude that **Pitty can fly**.

However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.Non-monotonic reasoning is used for real world systems such as Robot navigation.

The conventional reasoning system has the three main properties:

- It is complete with respect to the domain of interest as all the facts that are necessary in solving a problem is present in the system or can be derived from the conventional rules of first order logic.

- It is consistent.
- In order to make a change in this kind of system is to add new facts as they become available.

The main setback of this kind of system is that if any of these properties is not satisfied, the reasoning system becomes inadequate.

But nonmonotonic reasoning can solve problems even if the above said properties are missing. In order to do this, several key issues must be considered given below:

1. How can the knowledge base be extended to allow inferences to be made on the basis of lack of knowledge as well as the presence of it?

For example if we have no reason to suspect whether a person committed a crime then we need to assume that he didn't or if we have no reason to believe that someone is not getting along with her relatives then we have to assume that her relative will try to protect her.So there should be a clear distinction between:

- It is known that $\neg P$
- It is not known whether P

In the first case, reasoning is allowed by first order predicate logic.

But we also need an extended system that allows reasoning for the second case also.

2. How can the knowledge base be updated properly when a new fact is added to the system?

In nonmonotonic systems addition of new facts may invalid earlier discovered proofs. Now how to find out those proofs and the conclusions that depended on them. One usual solution is to keep track of all the proofs which are termed as justifications. This makes suitable to find all the justifications that depended on the absence of

the new fact and mark those proofs as invalid. Thus such mechanisms supports conventional and monotonic reasoning as well.

3. How can knowledge be used to help resolve conflicts when there are several in consistent nonmonotonic inferences that could be drawn?

Sometimes when inferences are based on lack of knowledge as well as it's presence, contradictions are likely to occur. In nonmonotonic systems we can observe that there are certain parts of knowledge base which are consistent and some parts which are mutually inconsistent. To overcome this problems additional methods are required for resolving conflicts in ways that are appropriate for the particular problem being solved. Suppose Sam, John and Harry are suspected for committing a crime but at the end after going through all the facts we conclude that they didn't commit the crime but still we know that one of them have committed it, then there is a contradiction. In such case we resolve the conflict by finding the person with the weakest alibi and conclude that he committed the crime

Now let us dive into the concept of nonmonotonic logic. Nonmonotonic logic provides a basis for default reasoning. Here the language of first order predicate logic is augmented with a modal operator M, which can be read as "is consistent". For example, the formula:

$$\forall x, y: Related\ (x, y) \wedge M\ GetAlong(x, y) \rightarrow \neg WillDefend(x, y)$$

which is read as "For all x and y, if x and y are related and if the fact that x gets along with y is consistent with everything else that is believed, then conclude that x will defend y". The first issue here is that we have to define what consistency means as consistency in this system as in first order predicate logic is undecidable and hence we require some approximation. It's necessary to define consistency on some heuristic basis, such as failure to prove inconsistency within some fixed level of effort.

The second problem is that if multiple nonmonotonic statements are taken aloneand it suggests ways of augmenting our knowledge which if taken together would be inconsistent. Let us consider the following set of assertions:

$\forall x: Republican\ (x) \wedge M \neg\ Pacifist \rightarrow \neg Pacifist(x)$

$\forall x: Quaker\ (x) \wedge M\ Pacifist(x) \rightarrow Pacifist(x)$

Republican (Dick)

Quakev(Dick)

So there are two different ways of augmenting this knowledge base. Now if we apply the first assertion it allows us to conclude $\neg Pacifist(Dick)$. After applying the first assertion, we cannot apply the second assertion as it concludes *Pacifist (Dick)*.So it's difficult to conclude about the theory it supports. In order to deal with these type of problems there are two types of nonmonotonic reasoning techniques:

- Abduction: Abduction is a form of nonmonotonic reasoning where assumptions are made to explain observations. Suppose we have an axiom like:

  $\forall x: Pneumonia(x) \rightarrow Lung\ infection(x)$

  The axiom says that having pneumonia implies lung infection. But suppose if we noticed lung infection first then we might like to conclude as pneumonia. This conclusion might be wrong but it may be the right guess based on whatever is going on. This type of conclusions is another form of default reasoning and is termed as abductive reasoning.

- Inheritance: Nonmonotonic reasoning also forms the basis for inheriting attribute values from a prototype description of a class to the individual entities that belong to the class. This is nothing but the concept of inheritance. Let's consider the following knowledge: dogs have lungs because they are

mammals and mammals have lungs. However, there may be exceptions. For example, mammals, by and large, do not fly; Since bats are mammals, in the absence of any information to the contrary, we are justified in inferring that bats do not fly. But if we learn that bats are exceptional mammals in that they do fly, the conclusion that they do not fly is retracted, and the conclusion that they fly is drawn instead. There are more complicated scenarios. For example, baby bats are exceptional bats in that they do not fly. Our example gives rise to potentially conflicting inferences. When we infer that Stellaluna, being a baby bat, does not fly, we are resolving all these potential conflicts based on the Specificity Principle. *Non-monotonic inheritance networks* were developed for the purpose of capturing taxonomic examples such as the one above. Such networks are collections of nodes and directed ('*is-a*') links that represent taxonomic information.

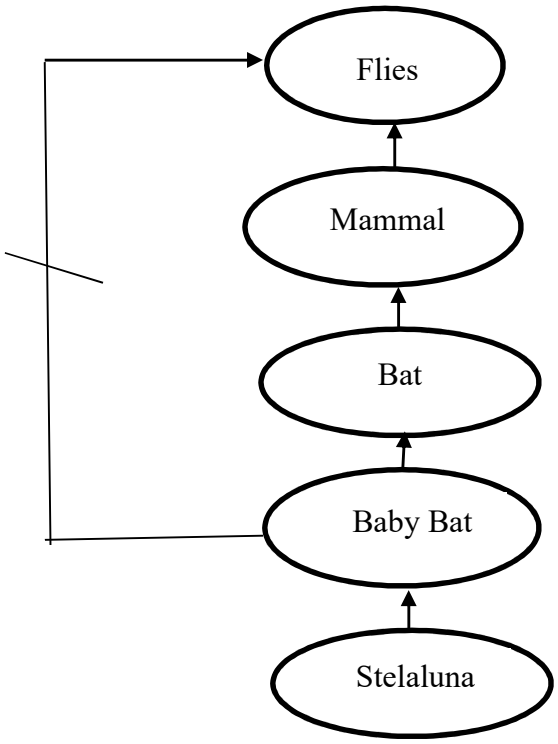


Fig 1: Inheritance

In figure 1 we can see that Stelaluna is a baby bat which belongs to the class Bat. Now if we move straight we conclude that Stelaluna can fly. But on the other hand if we go through the left branch as Stelaluna is a baby bat, it cannot fly. Thus if we consider both the branches we arrive in different conclusions and arises conflicts. But as we know that bats are mammals and mammals can fly but there may be some exceptions like baby bats does not fly.

5.5 DEPTH FIRST AND BREATH FIRST SEARCH

Non-monotonic reasoning is not enough for problem solving as it has some weakness and fails to deal with four important problems that arise in real systems.

Let's go through the four main problems:

1. First is how to derive exactly those nonmonotonic conclusions that are necessary to solve the problem at hand rather than going through those that are not necessary and useful although they may licensed by logic.
2. Secondly there must be a way to update our knowledge incrementally as problem solving progresses.
3. Thirdly it often happens in nonmonotonic reasoning system that more than one interpretation of the known facts is licensed by the available inference rules.
4. Lastly the theories used in nonmonotonic reasoning is neither effective nor efficient as most of these are not decidable and only some are semi-decidable.

The solutions to these problems regarding the reasoning process can be divided into two parts:

1. A problem solver that uses whatever mechanism it happens to have, to draw conclusions as necessary.

2. A truth maintenance system whose job is to maintain consistency in knowledge representation of a knowledge base.

So now we are going to discuss two techniques which helps in tracking the nonmonotonic inferences so that the changes made in knowledge base is handled properly.

5.5.1 Depth First Search

Depth-first search are most likely to follow a single path until some new piece of information forces us to give up this path and find a new path. Now if we take depth first search approach for nonmonotonic reasoning, the following scenario might likely occur. Suppose we have derived a fact F after taking assumption A. Later on we derived some other facts like G and H from F. After that some more facts like M and N are derived but these facts are completely independent of A and F. It may happen that after sometime a new fact might come and invalidate A. Then we have to override the proofs of F as well as G and H as all these facts are dependent on F. But what to do with M and N as they are logically independent of A. By following the conventional backtracking we have to back up past M and N thus undoing them in order to get back F,G,H and A. These problems are solved with dependency directed backtracking which is based on logical dependencies with slightly different notion of backtracking. Dependency directed backtracking is not only used in handling nonmonotonic reasoning but also is used in many conventional search programs. Let us take an example where we need to find a solution to a simple problem. The problem is that we need to fix a time for three busy people to attend a meeting. To solve this problem, first we need to make an assumption that the meeting is held on a particular day, say Thursday. Now an assertion is added to this effect and then we proceed to find a time, checking for any inconsistencies in the

people's schedule. If any conflict arises, then we have to discard the assumption made and replace it by another which must not be contradictory. We must also discard any statements that has been generated along the way and was dependent on the now discarded assumption. By withdrawing statements, based on the order in which they were generated by the search process rather than on the basis of responsibility for inconsistency, a great deal of effort is wasted.

If we use dependency-directed backtracking, then we need to do the following things while solving any problem:

1. Each node is associated with one or more justifications. Each justification corresponds to a derivation process that led to a node and each justification must contain a list of all nodes on which it's derivation depended.

2. A mechanism should be provided so the given contradiction node and it's justification computes the "no good" set of assumptions that underlie the justification. The no-good set is the minimal set of assumptions where removal of any element from the set will make the justification invalid and the inconsistent node will no longer be believed.

3. A mechanism should be provided for considering a no-good set and choosing an assumption to retract. Also a mechanism should be provided to propagate the results of retracting an assumption. This mechanism will cause all of the justifications dependent on the retracted assumption invalid.

5.5.2 Breadth-First Search

The assumption-based truth maintenance system (ATMS) is an alternative way of implementing nonmonotonic reasoning. In an ATMS, backtracking is avoided and alternative paths are maintained

in parallel. As we proceed in ATMS based system reasoning, the universe of consistent contexts are pruned as soon as contradictions are discovered. The context that remain consistent are used to label assertions and thus each assertion has a valid justification. So the assertions that do not have valid jurisdiction are pruned from consideration by the problem solver. Now the size of the consistent contexts as well as the set of assertions gets smaller and it is consistently believed by the problem solver. The breadth first search technique is applied to ATMS.

The job of problem solver that is used in conjunction with the ATMS is:

1. First to create nodes that corresponds to assumptions.

2. To associate with each such node one or more justifications, each describing the reasoning chain that led to the node.

3. The ATMS should be informed about the inconsistent contexts.

Thus the role of ATMS can be described as below:

1. The inconsistencies should be propagated ruling out the contexts including the set of assertions that are known to be inconsistent.

2. Each node in the problem solver should be labelled with the context with which it has valid jurisdictions. In particular, given a jurisdiction of the form

$$A1 \wedge A2 \wedge \ldots \wedge An \rightarrow C$$

assign as a context for the node corresponding to C the intersection of the contexts corresponding to the nodes A1 through An.

Thus contexts gets eliminated due to inconsistencies and new nodes are created by the problem solver to represent possible components of a problem solution. They may be considered to be pruned if all their context labels get pruned. Thus a possible solution gradually evolves in this process.

Check Your Progress

1. The index in Indexing technique can be generated by any reasonable _____function

2. In many_____, we need to compute the whole set of rules that match the current state description.

3. _____maintains a network of rule conditions and uses the state description to determine which new rules might be applied

4. Some conclusions in nonmonotonic reasoning can be invalidated if some more information is added to the_____

5. _____provides a basis for default reasoning

6. _____is a form of nonmonotonic reasoning where assumptions are made to explain observations.

5.6 SUMMING UP

1. To choose a rule to be applied at a particular point of time for solving a problem from the entire collection of rules requires some kind of matching between the current state and the preconditions of rules

2. Indexing is a technique that accomplishes this task by comparing each rule's preconditions with the current state and extracting the ones that match.

3. To discover whether there is a match between a particular situation and the preconditions of a given rule involve a significant search process.

4. RETE is an efficient algorithm where many rules are matched against many elements in the state description simultaneously.

5. The efficiency of RETE algorithm due to the temporal nature of data, structural similarity of rules and persistence of variable binding consistency.

6. Nonmonotonic reasoning is a technique that can solve problems with incomplete and uncertain models.

7. In nonmonotonic reasoning the axioms or rules of inference are extended to make it possible to reason with incomplete information.

8. Abduction is a form of nonmonotonic reasoning where assumptions are made to explain observations.

9. Nonmonotonic reasoning supports the concept of inheritance by inheriting attribute values from a prototype description of a class to the individual entities that belong to the class.

10. Depth first search and Breadth first search helps in tracking the nonmonotonic inferences so that the changes made in knowledge base is handled properly.

5.7ANSWERS TO CHECK YOUR PROGRESS

1. Hashing
2. Rule-based system
3. RETE
4. Knowledge base
5. Nonmonotonic logic
6. Abduction

5.8 POSSIBLE QUESTIONS

1. What is matching?
2. What is the purpose of indexing?
3. Explain indexing with the help of an example.
4. What is a RETE algorithm?
5. What are the three major sources by which RETE gains efficiency?
6. Why is nonmonotonic reasoning required?
7. What are the three main properties of a conventional reasoning system?
8. What are the two nonmonotonic reasoning techniques? Explain with the help of examples.
9. What are the main problems that arise in nonmonotonic reasoning? Explain.
10. What are the two techniques two techniques that helps in tracking the nonmonotonic inferences so that the changes made in knowledge base is handled properly? Explain.

5.9 REFERENCES AND SUGGESTED READINGS

1. Rich, E., Knight, K., & Nair, S. B. (2010). Artificial intelligence.

2. https://www.javatpoint.com/reasoning-in-artificial-intelligence

*****

BLOCK- III
DIFFERENT DOMAINS
OF
ARTIFICIAL INTELLIGENCE

UNIT 1: INTRODUCTION TO STATISTICAL REASONING

Unit Structure:

1.1 INTRODUCTION

In the earlier chapters we have learnt several representation techniques that can be used to model belief system in which at any moment, a particular fact is believed to be true, false or not considered in one way or the other way. Sometimes for certain problems it is important to be able to describe facts that may not have certainty but might have supporting evidence. For such type of problems, statistical measures serve a useful tool in describing the facts. In this unit we will learn how statistical measures can be augmented with knowledge representation techniques to describe the levels of evidence and belief.

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to-

- Understand the concept of uncertainty in knowledge
- Understand what is probability
- Learn prior, conditional and posterior probability
- Understand the Bayes theorem
- Learn to solve problems using Bayes theorem
- Learn the different applications of Bayes Theorem
- Understand the Bayesian Network.

1.3 PROBABILITY

Let us first try to understand the concept of uncertainty before discussing probability. Suppose we have two statements A and B. Now if the if-then rule is implemented then we might write like this A→ B which implies that if A is true then B is also true or if A is false then B is false or if A is true then B is false or if A is false then B is true. This can only be implemented if we know the state of A. But suppose we don't know about A then we cannot say any of these. Such kind of situation in which we can't take a firm decision is called uncertainty. To represent such type of uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning. The different causes of uncertainty might be any one of the following reasons:

1. Information occurred from unreliable sources.

2. Experimental Errors

3. Equipment fault

4. Temperature variation

5. Climate change.

The need of probabilistic reasoning in AI is required due to unpredictable outcomes, predicates are too large to handle and also for occurrence of unknown errors.

Let's now define probability

Probability: The way of representing knowledge by using the concept of probability to indicate the uncertainty in knowledge is called probability. Here in statistical reasoning, probability theory is combined with logic to handle the uncertainty. We can have a lot of examples in real life scenarios where certainty of something is not confirmed such as predicting the weather like "It will rain today" or behaviour of someone in a particular situation or predicting the outcome of a match.

Probability can be defined as chance of occurrence of an uncertain event. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1. $0 \leq P(X) \leq 1$, where P(X) is the probability of an event X.

- P(X) = 0, indicates total uncertainty in an event X.
- P(X) =1, indicates total certainty in an event X.

Probability of an uncertain event can be found out by using the formula

$$\text{Probability of occurrence} = \frac{Number\ of\ desired\ outcomes}{Tota\ \ number\ of\ outcomes}$$

Before we learn about Baye's theorem, it is important to learn the concepts given below:

Event: The set of each possible outcome of a variable is called event.

Sample space: Sample space is the collection of all possible events.

Random variables: To represent the events and objects in the real world, random variables

Prior probability: The probability that is computed before observing new information is called prior probability.

Posterior probability: The probability that is calculated after all evidence or information are considered. It is a combination of prior probability and new information.

Conditional probability: The conditional probability of an event B is the probability that the event will occur if and only if it is given that event A has already occurred.

Probability of event B given A is written as $P(B|A) = \frac{P(A \wedge B)}{P(B)}$

Where $P(A \wedge B)$= Joint probability of A and B

P(B)= Marginal probability of B

Probability of event A given B is written as $P(A|B) = \frac{P(A \wedge B)}{P(A)}$

Example: In a party, there are 70% of the children who like pizza and 40% of the children who likes pizza and burger, and then what is the percent of students those who like pizza also like burger?

Solution: Let, A is an event that a child likes burger

B is an event that a child likes pizza

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like pizza also like burger.

Check Your Progress

1.Probability is the numerical measure of the likelihood that an ____will occur.

2. The value of probability always remains between __ and __

3. Define sample space

4. What is the difference between prior and post probability?

1.4 BAYE'S THEOREM

Baye's theorem also known as Bayes' rule, Bayes' law, or Bayesian reasoning in AI, is a fundamental concept in probability theory and statistics. In probability theory, it relates the conditional probability and marginal probabilities of two random events. Bayes' theorem was named after the British mathematician Thomas Bayes. Here the value of P(B|A) is calculated with the knowledge of P(A|B).By observing new information of the real world the Baye's theorem allows updating the probability prediction of an event.

Baye's theorem can be derived using product and the of event A with known event B

Using product rule we can write:

$P(A \wedge B) = P(A|B)\ P(B)$ ----------(1)

Again the probability of event B with known event A:

$P(A \wedge B) = P(B|A)\ P(A)$----------(2)

Now equating both the right hand sides of the equations 1 & 2 we get

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$----------(3)

This equation is called Baye's rule or Baye's theorem. This forms the base of all modern AI systems for probabilistic inference. The relationship between the joint and conditional probabilities is shown in the simple equation above.

P(A|B) which we need to calculate is known as posterior and it will be read as Probability of hypothesis A when we have occurred an evidence B. P(B|A) is called the likelihood, in which we consider the hypothesis to be true and then we proceed to calculate the probability of evidence.P(A) is called the prior probability. It is the probability of hypothesis before the evidence is being considered. P(B) is called marginal probability which is the pure probability of an evidence. In equation 3 we can write P(B) as

$$P(B) = P(A) * P(B|A_i)$$

Hence the Baye's rule can be written as

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{K} P(A_i)*P(B|A_i)}$$

where $A_1, A_2, A_3, \ldots\ldots\ldots A_n$ is a set of mutually exclusive and exhaustive events.

If we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes

$$P(\text{cause} \mid \text{effect}) = \frac{\boldsymbol{P}(\boldsymbol{effect}|\boldsymbol{cause}).\boldsymbol{P}(\boldsymbol{cause})}{\boldsymbol{P}(\boldsymbol{effect})}$$

Example: In a clinic 10% of patients are having liver disease and and 5% of the patients are alcoholic. Out of the patients suffering from liver disease 7% are alcoholic. Find out the probability of a patient having liver disease if they are alcoholic?

Ans: Let A be the event " Patient having liver disease" .

So P(A) = 10% = 0.10

B be the event " Patient who is alcoholic"

So P(B) = 5% = 0.05

As 7% of patients suffering from liver disease are alcoholic

Hence we have P(B|A) = 7%

Probability of a patient having liver disease if they are alcoholic

$$P(A|B) = \frac{\boldsymbol{P}(\boldsymbol{B}|\boldsymbol{A}).\boldsymbol{P}(\boldsymbol{A})}{\boldsymbol{P}(\boldsymbol{B})}$$

$$= (0.07 * 0.1)/0.05 = 0.14$$

Example: John is planning a picnic for his family. He is trying to decide whether to postpone the picnic due to rain. The chance of rain on any day is 15%. Now it's cloudy in the morning of the picnic. The probability of it being cloudy is 25% and on days where it rains, it's cloudy in the morning 80% of time. What should John do now?

Ans: Probability of rain on any day P(rain) = 0.15

Probability of being cloudy P (cloudy) = 0.25

Probability of being cloudy on days when it rains P(cloudy|rain) = 0.80

Hence probability of rain during picnic

$$P(\text{rain} \mid \text{cloudy}) = \frac{P(\,cloudy\,|rain).P(rain)}{P(cloudy)}$$

$$= \frac{0.80\ .0.15}{0.25}$$

$$= 0.48$$

Self-Assessment Question

From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

……
……
……

1.4.1 Applications of Bayes theorem in AI

- **Spam Filtering**: The use of electronic messaging system to send unwanted or unrequested messages is called a spam. The nature of a spam resembles with an advertisement or a promotional campaign but in reality it is a way to deceive the users to acquire their personal confidential information. The phrases used in spam are recognized by the computer algorithm and can hence judge the authenticity of the mail.
- **Image Classification**: In Image recognition systems Bayes'theorem can be used to assign probabilities to place them in the correct category.
- **Recommendation Systems**: By studying the user's past behavior and preferences, recommendation engines can utilize Bayes' theorem to personalize suggestions.
- **Financial Modelling:** Bayes' theorem is used in financial institutions to assess creditworthiness of loan applicants or predict market trends by incorporating historical data and economic indicators to calculate the probability of different financial outcomes.
- **Robot Navigation**: Robots navigation uses Bayes' theorem to update their understanding of the surroundings based on sensor data. This helps them adapt to changes and avoid obstacles more effectively.
- **Self-Driving Cars**: Autonomous vehicles or self-driving cars have a very complex decision-making process by sensing the data from the surroundings and then analyzing it. The sensors use Bayes theoremto interpret data it has collected and make real-time decisions about steering, braking, and lane changes while considering uncertainties in the environment.

- **Anomaly Detection**: Bayes' theorem helps to calculate the likelihood of an event being anomalous by identifying the unusual patterns in data.
- **Sentiment Analysis**: Using Bayes theorem in analyzing the sentiment of text data (positive, negative, neutral) can be enhanced by considering the context and prior knowledge about sentiment-related words.
- **Natural Language Processing (NLP):** Bayes theorem in NLP helps in accomplishing tasks like machine translation and part-of-speech tagging. It can be used for predicting the most probable part of speech for a word based on surrounding words and context.
- **Medical Diagnosis**: Bayes theorem can be helpful in predicting a particular disease with the use of patient data and medical history. This is definitely not a replacement for medical expertise but can be used as an aid to help medical professionals to diagnose the disease.

Let's just go through a particular example to understand Bayes theorem in details. We are going to see the use of Bayes theorem in image classification. Image classification is a task in computer vision. It helps to categorize images under a specific label. Bayes theorem used in classification is named as Bayes optimal classifier. Suppose we have a classification problem with idifferent classes. Here the main concern is to find out the class probability for each class wi. So the prior class probabilities will be p(wi) and posterior class probabilities**,** after using data or observations will be *p(wi|x)*. Therefore Bayes formula for Bayes optimal classifier is

$$P(wi|x) = \frac{\boldsymbol{P}(\boldsymbol{x}|\boldsymbol{wi}).\boldsymbol{P}(\boldsymbol{wi})}{\boldsymbol{P}(\boldsymbol{x})} \text{----------(4)}$$

Here P(x) is the density function common to all the data points, P(x|wi) is the density function of the data points belonging to class wi, and P(wi) is the prior distribution of class wi. P(x|wi) is calculated from the training data, assuming a certain distribution and calculating a mean vector for each class and the covariance of the features of the data points belonging to such class. The prior class distributions P(wi) are estimated based on domain knowledge, expert advice or previous works, like in the regression example.

1.4.2 Bayesian Networks

The main idea of Bayesian network is that to describe the real world, it is not necessary to use a huge joint probability table in which we list the probabilities of all conceivable combination of events. The interaction of events that are conditionally independent of each other actually need not be considered. Instead a more local representation can be used that describe the cluster of events that interact. In this representation we construct a model from data and expert opinion consisting of two parts: Direct Acyclic Graph (DAG) and Conditional Probability Table (CPT). In DAG, nodes correspond to random variables which can be continuous or discrete. Each node is assigned with a CPT. For problem-solving, this DAG is converted into the undirected graph in which the arcs can be used to transmit probabilities in either direction, depending on where evidence is coming from. The only constraint or arcs allowed in Bayesian Network is that there must not be any directed cycles. DAG mainly represents causality relationships among variables.

Now let us understand Bayesian Network with the help of a Figure 1: Burglars and earthquake problem. Let us assume that a house has an alarm system against burglary. The house is situated in a seismically active area and the alarm system can get occasionally set off by an

earthquake. The house has two neighbors who don't know each other namely Mary and John. If the alarm is heard by them they will call you.

So for constructing the Bayesian Network first we need to identify the variables:

Burglary (B), Earthquake (E), Alarm system (A), John calls (J) , and Marry calls (M)

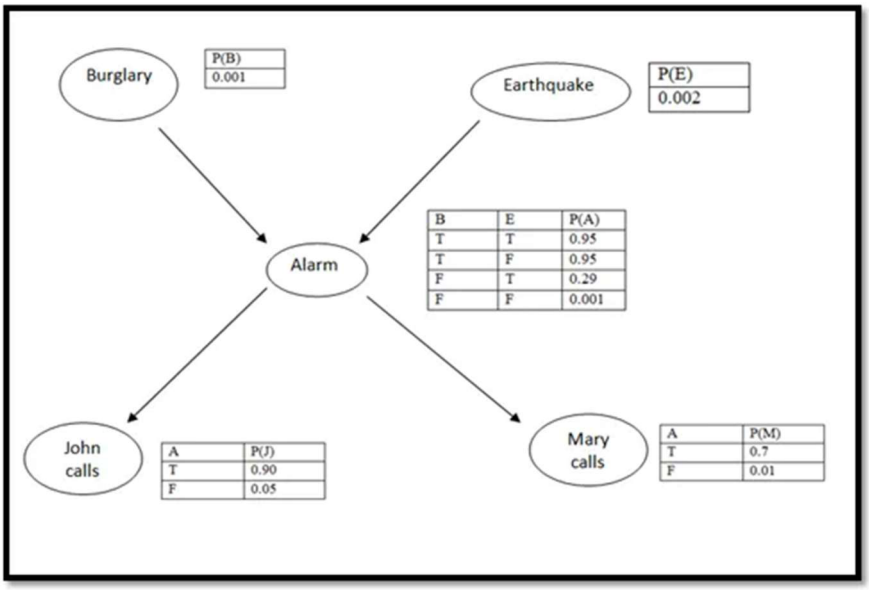


Fig 1: Burglar and Earthquake Problem

(Source:https://miro.medium.com/v2/resize:fit:640/format:webp/1*KAeo07om2Ehaa_dPHeJeqQ.png)

In the graph showed in Fig 1, there are two nodes Burglary and Earthquake whose parent is Alarm. Earthquake is the ancestor of both John calls and Mary calls. Again, John calls and Mary calls are the children of Alarm and are descendants of both Burglars and Earthquake.

Thus, Burglary and Earthquake are both root nodes, John calls and Mary calls are leaf nodes while Alarm is an intermediate node.

Suppose we need to find the probability when John calls and Mary calls are true when the Alarm rang but no Burglary and Earthquake occurred.

Now here we have three true events: J,M and A and two false events ∼B and ∼E

Going through the Fig 1 we get the probabilities as

P(J) = 0.90

P(M) = 0.7

P(A) = 0.001

P(∼B) = 1- 0.001 = 0.999

P(∼E) = 1 – 0.002 = 0.998

So, the final probability will be like

= P (J ∩ M ∩ A ∩∼B ∩∼E)

= P(J/A) * P(M/A)*P(A/∼B, ∼E) * P(∼B) * P(∼E)

= 0.90 * 0.7*0.001*0.999*0.998

=0.000628

1.4.3 Role of Bayesian Networks in AI

Addressing uncertainty is the most significant impact of Bayesian network in AI. Most of the models in AI suffer from uncertainty. Thus the Bayesian network provides the necessary framework for modelling and reasoning under uncertainty. Bayesian networks represents explicitly the dependencies between variables, thus incorporating probabilistic information. With the help of Bayesian networks, complex systems can be made more more accurate and robust. Thus, the application of Bayesian networks makes more informed predictions and decisions, accounting for the inherent uncertainty in the data.

Bayesian networks can also aid in optimal decision making particularly when the decisions are to be made on limited or noisy data. Another significant characteristic of Bayesian network is it's ability to learn from data. It accomplishes by augmenting the prior knowledge it has and the present observed data. Thus, by doing this it enhances it's knowledge and also updates it's belief based on the new information. This increases the adaptability as well as also the performance of the AI models.

Thus, Bayesian networks can be used in a wide variety of applications thus making AI models robust, accurate, and interpretable.

1.5 SUMMING UP

1. The need of probabilistic reasoning in AI is required due to unpredictable outcomes, predicates are too large to handle and also for occurrence of unknown errors.

2. The way of representing knowledge by using the concept of probability to indicate the uncertainty in knowledge is called probability.

3. An event is a set of each possible outcome of a variable is called event.

4. Sample space is the collection of all possible events.

5. Random variables are used to represent the events and objects in the real world.

6. Prior probability is the probability that is computed before observing new information.

7. Posterior probability is the probability that is calculated after all evidence or information are considered. It is a combination of prior probability and new information.

8. The conditional probability of an event B is the probability that the event will occur if and only if it is given that event A has already occurred.

9.Bayes theorem relates the conditional probability and marginal probabilities of two random events.

10. The main idea of Bayesian network is that to describe the real world, it is not necessary to use a huge joint probability table in which we list the probabilities of all conceivable combination of events.

1.6 ANSWERS TO CHECK YOUR PROGRESS

1. Event

2. 0,1

3. Sample space is the collection of all possible events

4. The probability that is computed before observing new information is called prior probability and conditional probability is the probability that is calculated after all evidence or information are considered. It is a combination of prior probability and new information.

1.7 PROBABLE QUESTIONS

1. What is statistical reasoning?

2. What is the necessity of probability in AI?

3. Define probability with the help of an example.

4. Define: event, sample space and random variable.

5. Explain prior, posterior and conditional probability with the help of an example.

6. Explain Bayes theorem with the help of an example.

7.Mention some applications of Bayes theorem in AI.

8. Explain how Bayes theorem is applied in image classification.

9. Explain Bayesian network with the help of an example.

10. Describe the role of Bayesian network in AI.

1.8 REFERENCES AND SUGGESTED READINGS

1. https://www.cecmohali.org/public/documents/cse/material/ppt/ai-ppt-3.pdf

2.https://www.lkouniv.ac.in/site/writereaddata/siteContent/20200402191015875 8chandrabhan_Artificial_Intelligence_Probabilistic_reasoning.pdf

3. https://rashandeepsingh.medium.com/bayes-theorem-and-bayesian-network-14b5a614ca26

4. https://www.leewayhertz.com/bayesian-networks-in-ai/#indispensable-role-of-Bayesian-networks-and-probabilistic-inference-in-machine-learning

5.https://www.lkouniv.ac.in/site/writereaddata/siteContent/20200402191015875 8chandrabhan_Artificial_Intelligence_Probabilistic_reasoning.pdf

6. Rich, E. L. A. I. N. E., Knight, K., & Nair, S. B. (2009). Artificial intelligence third edition.

*****

UNIT-2: FUZZY LOGIC CONCEPT

Unit Structure:

2.1 INTRODUCTION

A computer consists of circuits and devices. There is no intelligence of its own, but we can make it intelligent artificially. We can make it expert in areas like medical diagnosis, robot movement, location of mineral deposits, etc. Therefore, for this, a mathematical movement of vague concepts or vague knowledge or imprecise data is required. Zadeh introduced Fuzzy set concept in 1965. Since Zadeh initiated Fuzzy sets, many approaches and theories treating imprecision and uncertainty have been proposed. The term **fuzzy** implies things that are not clear or are vague. In the real world many times we meet a situation when we can't decide whether the state is true or false, there fuzzy logic gives very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation. Fuzzy Logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1, instead of just

the traditional values of true or false. It is useful for dealing with imprecise or uncertain information and is a mathematical method for representing vagueness and uncertainty in decision-making problems.

2.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand the Fuzzy set concept and Fuzzy membership function
- Explain the basic terms used in Fuzzy concept
- Analyze the basic operations of Fuzzy set

2.3 INTRODUCTION TO FUZZY CONCEPT

A computer consists of circuits and devices. There is no intelligence of its own, but we can make it intelligent artificially. We can make it expert in areas like medical diagnosis, robot movement, location of mineral deposits, etc. Therefore, for this, a mathematical movement of vague concepts or vague knowledge or imprecise data is required. In classical set theory, the "integer numbers which are greater than or equal to 7 and less than or equal to 13" can be represented as:

$S = \{x \in Z \mid 7 \leq x \leq 13\}$

i.e., $S = \{7, 8, 9, 10, 11, 12, 13\}$

This representation is insufficient in the sense that we are not in a position to answer somewhat vague concept e.g., "integers which are more or less equal to 7." The source of vagueness is "more or less.". In conventional set theory, we have the characteristic function defined as:

$$\mu : X \rightarrow \{0, 1\}$$

which associates with each element of a universe of discourse X either 1 or 0 which means that a particular element either belongs to the set or does not, respectively. As a result there is a clear difference of the elements 'belonging and not belonging to the set', or equivalently the conversion from 'belonging' to 'not belonging' to the set is unexpected. But for the

"integers being more or less equal to 7", such a exact distinction is artificial. Here, it is impossible to fix any clear border line. Therefore, one of the most important method for such information processing is fuzzy set theory.

In fuzzy set theory, we say the classical sets as crisp sets, in order to differentiate them from fuzzy sets. Let C be a crisp set defined on the universe X. Then for any element x of X, either $x \in C$ or $x \notin C$. In fuzzy set theory, this property is generalized, therefore in a fuzzy set F, it is not necessary that either$x \in F$ or $x \notin F$. In fuzzy set theory, the characteristic function $\mu_c : X \rightarrow \{0, 1\}$ defined in a crisp set is generalised to a membership function that assign to every $x \in X$ a value from the unit interval [0, 1] instead from the two-element set {0,1}. The set that is defined on the basis of such a membership function is called a fuzzy set.

2.3.1 Fuzzy Membership Function

A fuzzy set is made of elements and their respective membership grades in the set. This membership grade refers to the degree to which that individual is similar or well-matched with the concept represented by the fuzzy set. The grade of membership of an element can be obtained by a subjectively defined membership function. The value of the grade of membership of an element can range from 0 to 1 .Here the value 1 means full membership, and the closer the value is to 0, the weaker is the element having its membership in the fuzzy set.

Definition 5.1: The **membership function** $\mu_F$of a fuzzy set F is a function:

$$\mu_F : X \rightarrow [0, 1]$$

Thus, every element x from X has a membership degree$\mu_F(x) \in [0, 1]$, and F is completely determined by the set of tuples:

$$F = \{(x, \mu_F(x)) \mid x \in X\}$$

Therefore a fuzzy set is a set of pairs having the particular element of the universe and their membership grades. For example we can define a

possible membership function for the set of real numbers close to 0 as follows:

$$\mu_F(x) = \frac{1}{1 + 10x^2}$$

and graphically represent it as in Fig 5.1:

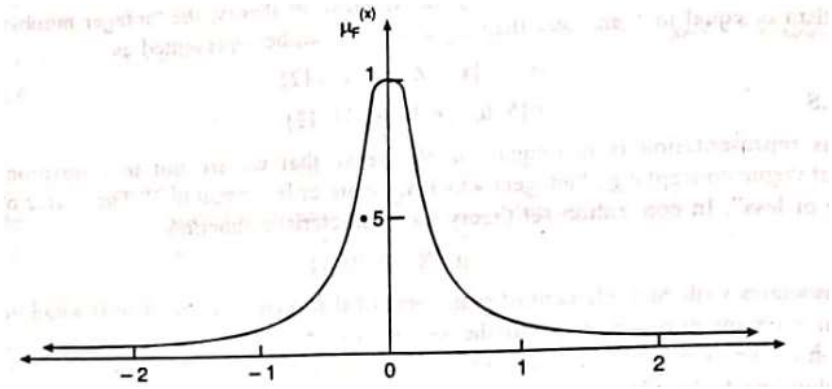


Figure 5.1 Possible membership function of the fuzzy set of real member close to zero

If $X = \{x_1, x_2, \ldots, x_n\}$,then a fuzzy set A of X can be written as:

$A = \{(x_1, \mu_{A(x_1)}), (x_2, \mu_{A(x_2)}), \ldots, (x_n, \mu_{A(x_n)})\}$

which can sometimes be written as:

$A = \left\{\frac{\mu_{A(x_1)}}{x_1}, \frac{\mu_{A(x_2)}}{x_2}, \ldots, \frac{\mu_{A(x_n)}}{x_n}\right\}$

For example, if the real numbers 5, 1, and 0 have membership grades of 0.1, 0.09, and 1 can be written as:

A = {(5, 0.1),(1, 0.09),(0, 1)}

Or

$$A = \left\{\frac{0.1}{5}, \frac{0.09}{1}, \frac{1}{0}\right\}$$

Stop to Consider

Fuzzy Membership functions were first introduced by Lofti A. Zadeh in his first research paper fuzzy sets in 1965. Fuzzy Membership functions characterize fuzziness or vagueness (i.e., all the information in fuzzy set), ie.whether the elements in fuzzy sets are discrete or continuous. Fuzzy Membership functions can be defined as a method

to solve practical problems by experience rather than knowledge. Fuzzy Membership functions can be represented by graphical forms. It can be understood as Fuzzy rules for defining fuzziness are fuzzy too.

2.4 FUZZY BASIC TERMINOLOGY

Some important terms used in Fuzzy Logics are discussed in this section.

Definition 5.2: **Subset**

Let X be a set ($\neq \emptyset$) and let A and B be two fuzzy sets on X with membership functions μ_A and μ_B respectively. We say that the fuzzy set A is contained in the fuzzy set B iff:

$$\mu_{A(x)} \leq \mu_{B(x)} \forall x \in X$$

Example 5.1: If X = {1, 2, 3} and A, B, C are three fuzzy sets given by:

$A = \left\{\frac{.1}{1}, \frac{.5}{2}, \frac{1}{3}\right\}$, $B = \left\{\frac{.1}{1}, \frac{.3}{2}, \frac{0.6}{3}\right\}$, $C = \left\{\frac{.1}{1}, \frac{.6}{2}, \frac{.5}{3}\right\}$ then B ⊆ A, but C ⊈ A.

Definition 5.3: **Normal Fuzzy Set**

A fuzzy set A on the set X is called a normal fuzzy set if and only if:

$$\max_{x \in X} \mu_{A(x)} = 1$$

i.e., $\mu_A(x) = 1$ for at least one $x \in X$ otherwise A is subnormal .

Example 5.2: If X = {1, 3, 5, 7} and A = $\left\{\frac{.1}{1}, \frac{1}{3}, \frac{.2}{5}, \frac{.5}{7}\right\}$ then we can say A is a normal fuzzy set.

Definition 5.4: **Support of a Fuzzy Set**

The support of a fuzzy set A on the set X is the crisp set that contains all the elements of X that have a nonzero membership grade in A and denoted by supp(A) i.e.,

$$\text{supp}(A) = \{x \in X \mid \mu_A(x) > 0\}$$

The element x in X at which we get $\mu_A(x) = 0.5$ is called the **crossover point.** A fuzzy set whose support is a single element in X with $\mu_A(x) = 1$ is called **a fuzzy singleton**.

Example 5.3: Let X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and let the fuzzy set A on X be:

A = {(1, 0), (2, 0), (3, 0.5), (4, 0.7), (5, 0.2), (6, 0.8), (7, 0), (8, 0), (9, 0)}

Then:

supp(A) = {3, 4, 5, 6}

x = 3 is the crossover point. A special notation is often used for defining fuzzy set with a finite support:

$$A = \mu_1|x_1 + \mu_2|x_2 + \ldots + \mu_n|x_n$$

For the case in which a fuzzy set A is defined on a universal set that is finite and countable, one may write

$A = \sum \frac{\mu_i}{x_i}$

Definition 5.5: **α-Cut or α-Level Set**

The α-level set is the crisp set of elements that belong to the fuzzy set A at least to the degree α. Mathematically, we can write,

$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\}$$

The **strong α-level set or strong α-cut** is defined as:

$$A'_\alpha = \{x \in X | \mu_A(x) > \alpha\}$$

Example 5.4: Suppose X = { 2, 3, 4, 5, 6, 7}.Consider the fuzzy set A of X given by:

A = {(2, 0.2), (3, 0.5), (4, 0.7), (5, 1), (6, 0.8), (7, 0.3)}

Then all possible α-level sets are:

$$A_{0.2} = \{2, 3, 4, 5, 6, 7\}, \quad A_{0.3} = \{3, 4, 5, 6, 7\},$$
$$A_{0.5} = \{3, 4, 5, 6\}, \quad A_{0.7} = \{4, 5, 6\},$$

$A_{0.8} = \{5, 6\}$, $A_1 = \{5\}$

Clearly, $A_0 = X$ and $A_\alpha = \emptyset$ for all $\alpha > 1$.

2.5 BASIC FUZZY OPERATIONS

Generally, the basic operations in the theory of fuzzy sets are the complement, union and intersection. In short, the definitions can be given in terms of the respective membership functions.

Definition 5.6: **Equal Fuzzy Sets**

Two fuzzy sets A and B are equal if:

$$\mu_{A(x)} = \mu_{B(x)} \forall x \in X$$

Definition 5.7: **Absolute and Relative Complements**

The **absolute complement** of a fuzzy set A is represented by $\overline{A}$and is defined by:

$$\mu_{\overline{A}}(x) = 1 - \mu_{A(x)}, \qquad \forall x \in X$$

Thus, if an element has a membership grade of 0.6 in a fuzzy set A, its membership grade in the complement of A will be 0.4.

The **relative complement** of A with respect to B, denoted by B − A, is defined by:

$$\mu_{B-A}(x) = \mu_B(x) - \mu_A(x) \text{ provided that } \mu_B(x) \geq \mu_A(x).$$

Example 5.5: Let A = {(0, 0.3),(1, 0.5),(2, 0.7),(3, 0.8)} and B = {(0, 0.4),(1, 0.6),(2, 0.8),(3, 0.8)}. Find $\overline{A}$ and B − A.

Solution:

$$\overline{A} = \{(0, 1 - 0.3), (1, 1 - 0.5), (2, 1 - 0.7), (3, 1 - 0.8)\}$$

$$= \{(0, 0.8), (1, 0.6), (2, 0.4), (3, 0.3)\}$$

$$B - A = \{(0, 0.1), (1, 0.1), (2, 0.1), (3, 0)\}.$$

Definition 5.8: **Union of Fuzzy Sets**

The union of two fuzzy sets A and B is a fuzzy set C given by:

C= A ∪ B where $\mu_C(x) = \max(\mu_A(x), \mu_B(x)), \ \forall x \in X.$

Definition 5.9: **The Intersection of Fuzzy Sets**

The intersection of two fuzzy sets A and B is a fuzzy set C given by: C = A∩B

Where $\mu_C(x) = \min(\mu_A(x), \mu_B(x)), \ \forall x \in X.$

Example 5.6:

Let: A = {(4, 0.2),(6, 0.2),(8, 0.4),(10, 0.5)},

B = {(0, 0.3),(2, 0.5),(4, 0.7),(5, 0.9),(8, 0.7)} are two fuzzy sets.

Then we have A ∩ B = min[0, 0.3]/0 + min[0, 0.5]/2 + min[0.2, 0.7]/4 + min[0, 0.9]/5 + min[0.2, 0]/6 +min[0.4, 0.7]/8 + min[0.5, 0]/10

= 0/0 + 0/2 + 0.2/4 + 0/5 + 0/6 + 0.4/8 + 0/10

= {(4, 0.2),(8, 0.4)}

We can give the graphs that represent the membership function as in Fig. 5.2:

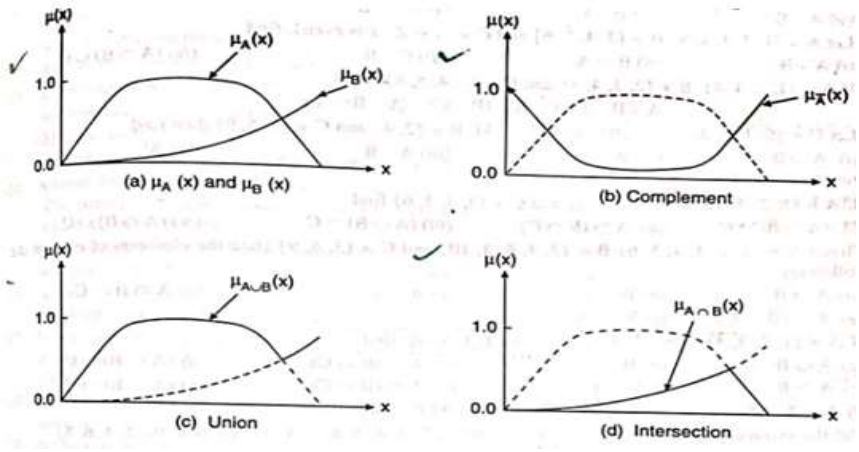


Figure: 5.2. Fuzzy Basic Operations

Stop to Consider

Fuzzy logic is a generalization from standard logic, in which all statements have a truth value of one or zero. In fuzzy logic, statements can have a value of partial truth, such as 0.9 or 0.5. Theoretically, this gives the approach more opportunity to mimic real-life circumstances, where statements of absolute truth or falsehood are rare.

2.6 SUMMING UP

In summary, Fuzzy Logic is one of the most important method for processing vagueness and uncertainty in decision-making is fuzzy set theory mathematical method for representing, it allows for partial truths, and it is used in a wide range of applications. It is based on the concept of membership function and its implementation is done using Fuzzy rules.

2.7 CHECK YOUR PROGRESS

1. State True or False:

a) The fuzzy set A is contained in the fuzzy set B iff:$\mu_{A(x)} \leq \mu_{B(x)} \forall x \in X$

b) An empty Fuzzy set has an empty support.

c)The crossover point has membership function value equals to 1.

d) There is no difference between Absolute and Relative complement of a Fuzzy set.

e)Two fuzzy sets are called equal if they have equal number of elements

2. Fill in the blanks:

a) In Fuzzy set theory classical sets are called _____.

b)The grade of membership of an element is given by _____ function.

c)A fuzzy set which is not normal is called _____.

d) A_α = ____ for all α> 1.

e) A fuzzy set consists of elements and their respective grades of_____ in the set.

Answers to check your progress

1. a) True **b)** True **c)** False **d)** False **e)** False

2. a) Crisp sets **b)** Membership **c)** Subnormal **d)** ∅ **e)** membership

2.8 POSSIBLE QUESTIONS

1. Describe Fuzzy Membership function. Give example with graphical representation.

2. Define the terms:

i) Crossover point

ii) Fuzzy Singleton

iii) α-level set

3. Discuss the basic operations of Fuzzy set with examples

4. Let X = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } and let the fuzzy set A on X be:

A = {(1, 0.1), (2, 0.2), (3, 0.3), (4, 0.4), (5, 0.5), (6, 0), (7, 0), (8, 0), (9, 0)} Find supp(A) . Also find the crossover point.

5. Suppose X = {3, 4, 5, 6, 7, 8}.Consider the fuzzy set A of X given by:

A = {(3, 0.3), (4, 0.6), (5, 0.7), (6, 1), (7, 0.1), (8, 0.9)}

Find α-level sets: $A_{0.6}$, $A_{0.9}$, A_0 .

6. Let A = {(1, 0.3),(2, 0.5),(3, 0.7),(4, 0.8)} and B = {(1, 0.4),(2, 0.6),(3, 0.8),(4, 0.8)}. Find $\overline{A}$, A^C and B − A.

7. Let: A = {(4, 0.1),(6, 0.5),(8, 0.6),(10, 0.7)},

B = {(0, 0.4),(2, 0.6),(4, 1),(6, 1),(8, 0.6),(10, 0.5)} are two fuzzy sets. Then find A ∪ B and A ∩ B.

8. If X= {51, 52, 53, 54}

A = {(51, 0.1),(52, 0.5),(53, 0.6),(54, 0.7)} and B = {(51, 0.4),(52, 0.6),(53, 1),(54, 1}. Then find A ∪ B and A ∩ B.

2.9 REFERENCES AND SUGGESTED READINGS

[1]. https://www.techtarget.com/searchenterpriseai/definition/fuzzy-logic

[2]. James K. Peckol (2021). Introduction to Fuzzy Logic

[3]. https://plato.stanford.edu/entries/logic-fuzzy/

*****

UNIT 3: FUNDAMENTAL OF NATURAL LANGUAGE PROCESSING

Unit Structure:

3.1 INTRODUCTION

Language is the communicating medium in the world. By studying or learning languages one can understand more about the world. Natural language processing is a subfield of Artificial Intelligence. It studied the automatic generated problems in natural language processing and understanding the natural human languages. Here in this unit we will discuss the different language processing problems like processing written text and spoken languages. Here we also discuss about several components of the natural language understanding process like

morphological analysis, syntactic analysis and semantic analysis, discourse integration and pragmatic analysis.

3.2 OBJECTIVES

After going through this unit learner will able to

- Understand the basic concept of natural language processing
- Learn about different natural language understanding process
- Understand the morphological analysis and syntactic analysis of language
- Understand the concept of Semantic analysis of a sentence in a language
- Learn about the discourse integration and pragmatic analysis of a sentence

3.3 WHAT IS NATURAL LANGUAGE PROCESSING (NLP)?

A natural language is a language that is spoken, written by humans for general purpose communication. Natural language is different from the other formal language like computer-programming language or the language which are used in the study of formal logic. The term natural language refers to the language that people speak, like English, Assamese, Hindi etc. The goal of natural language processing is to design and build software that will understand, analyze and generate languages that human use naturally. The computational activities required for enabling a computer to carry out information processing using natural language is called natural language processing. There are many challenges are involves in NLP such as enabling the computer to derive meaning from human or some

natural language input. In computer science taxonomy, NLP can be categorized as in the Figure 3.1

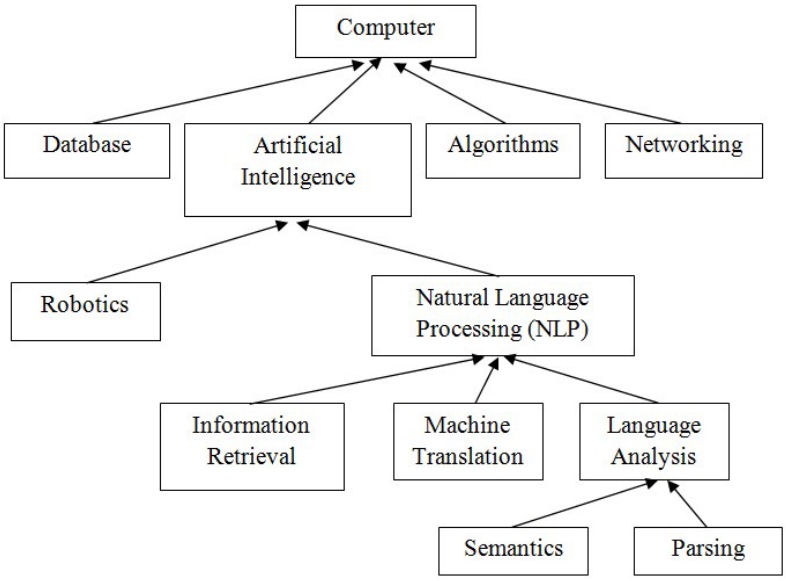


Figure 3.1 Natural Language Processing in Computer Science Taxonomy

General people have not any problem for understanding a language because of the following reasons

- Common sense knowledge
- Reasoning capacity
- Experience

But computer have no common sense knowledge and reasoning capacity. An application may require NLP for processing natural language input or producing natural language output or both. For achieving or gaining human like language processing capabilities is a difficult task for a machine and these difficulties are as follows

- Ambiguity
- Interpreting or adding partial information in the language.
- There are many inputs can mean same thing.

The application of natural language can be divided into two categories and these are as follows

- Text based applications: The text based involves the processing of written text, such as books, newspapers, reports, manuals, email-messages etc. These all are the reading based tasks.
- Dialogue based applications: It involves the communication between machine and the human like a spoken language.

3.4 KNOWLEDGE REQUIRED FOR NLP

Natural language uses the knowledge about the structure of the language itself which includes the structure of the sentences, structure of the words and how these words forms a sentence, how the meaning of individual word helps to form the sentence meaning etc. The different forms of knowledge related to natural language are

Phonological knowledge: This includes how words are related to the sounds that realized them.

Morphological knowledge: It concerns how words are formed using the morphemes. Morpheme is the primitive unit of a meaning in a language. For example the meaning of the word friendly is derivable from the meaning of the word friend which is a noun and its suffix –ly, which transform the word from noun to adjective.

Syntactic knowledge: The syntactic knowledge concerns how words can be put together to form a sentence and determine what type of structure role played by each word in the sentence. Some word sequences may be rejected if it break or violate the word formation rules of the language. For example an English syntactic analyzer would reject the sentence “Girl the go the to school”.

Semantic knowledge: It concerns what is the meaning of a word and how these meaning combine in sentence meaning. The most famous example "Colorless green ideas sleep furiously" is syntactically correct but rejected semantically.

Discourse knowledge: It concerns how the immediately preceding sentences affect the interpretation of the next sentence. For example the word "it" in the sentence, "Ram wanted it", depends on the prior discourse context, while the word "Ram" may influence the meaning of the later sentences such as "He always had".

Pragmatic Knowledge: It concerns how sentences are used in different situations and how use affects the interpretation of the sentence. For example, the sentence "Do you know what time it is?" should be interpreted as a request to be told the time.

The following examples may help you to understand the distinction between syntax, semantics, and pragmatics.

(1) Language is one of the fundamental aspects of human behavior and is a crucial component of our lives.
(2) Green frogs have large ears.
(3) Green ideas have large ears.
(4) Large have green ideas ear.

The above sentence (1) "Language is one of the fundamental aspects of human behavior and is a crucial component of our lives is semantically, syntactically and pragmatically correct. The sentence (2) "Green frogs have large ears" is both syntactically and semantically correct, but not pragmatically. The sentence (3) "Green ideas have large ears" is in ill-formed in pragmatically and semantically, but the sentence maintain a structure. Now let us discuss what is wrong in the sentence: the idea cannot be green, even if they could, they cannot have large ears. The sentence (4) "Large

have green ideas ear" uses the same words as the sentence (3) but it is ill-formed in syntactically. Thus the sentence (4) is syntactically, semantically and pragmatically incorrect.

Check Your Progress - I

1. **State True or False**
   (i) Artificial Intelligence is the sub-field of natural language.
   (ii) Natural language and formal language both are same.
   (iii) Phonological knowledge includes how words are related to the sounds that realized them.
   (iv) Morphological knowledge concerns how words are formed using the morphemes.
   (v) Syntactic and semantic knowledge are same.

3.5 MORPHOLOGY

Morphology is a process that deals with the study of form and structure. In terms of language morphology refers to the structure of word on that language. Morphology is the study of the way, where words are built up from smaller meaning bearing units called morphemes. For example the word "foxes" is the combination of two morphemes fox and es. The problems recognizing, foxes breaks down into two words fox and es is known as morphological parsing or morphological analysis. If we consider the two word "dog" and "dogs" , dog has only one morpheme and dogs has two morphemes "dog" and "s". Morpheme can be classified into two categories

(i) Stem - the main morpheme of the word and

(ii) Affixes - add additional meaning to the main morpheme.

Affixes can further divided into the following categories

(i) Prefixes: prefixes precede the stem. For example the word unhappy is composed of a stem “happy” with the prefix “un”.

(ii) Suffixes: suffixes follow the stem. For example the word “eats” is composed of a stem “eat” with the suffix “s”.

(iii) Infixes: in infixes the morpheme is inserted in the middle of the word. This occurs very commonly, for example in the Philippines language Tagalog.

(iv) Circumfixes: In circumfixes both prefix and suffix occurs in the stem. *The word unbelievable is composed of the stem believe; prefix un- and suffix –able.*

3.6 SYNTACTIC PROCESSING

In the syntactic processing steps a flat input sentence is converted into hierarchical structure which corresponds to the units of meaning in the sentence. This process is called the parsing. Parsing plays an important role in many natural language understanding systems for two reasons

- Semantic processing must operate on sentence constituents. If there is no any syntactic parsing step, then the semantic system of the language must decide on its own language structure. Syntactic parsing is computationally less expensive than semantic processing. Thus it plays a significant role in reducing overall complexity of the system.
- Although it is possible to extract the meaning of sentence without using the grammatical facts, but it is not possible to do always.

3.6.1 Parsing of Natural Language

Parsing is a process which automatically building syntactic analysis of a sentence in terms of a given grammar. So in the parsing analysis, it produces grammatical structure according to the input sentences. A parser is a computer program that carries the parsing task. The output of parsing is logically equivalent to a tree. The sentence is grammatically correct if parsing of a sentence successfully generates a parse tree and the leaf nodes of the tree are the tokens of input sentence, otherwise there are some grammatical errors in the sentence.

There are two requirements to examine the syntactic structure of a sentence. These two requirements are as follows

- The grammar: This is the formal specification of the structures which are allowed in the language.
- Parsing Technique: Parsing is a method that analyzing sentence to determine its structure according to its grammar. Context free grammar is used in natural language processing.

For example " I saw a man in the road". Consider the following context free grammar

1. S → NP VP
2. S → S PP
3. NP → n
4. NP → art n
5. NP → NP PP
6. PP → p NP
7. VP → v NP
8. n → I
9. n → man
10. n → road

11. v → saw
12. art → a
13. art → the
14. p → in

In the above grammar the left hand side of each rule is called “non terminals” and the symbols that are not appeared in the left hand side are called “terminal symbols”. The non terminal symbols which directly produce terminals called “Pre terminal” symbol. Here in this above example “S” is the start symbol and the parse tree generating for the sentence “I saw the man in the road” is shown in the Figure 3.2.

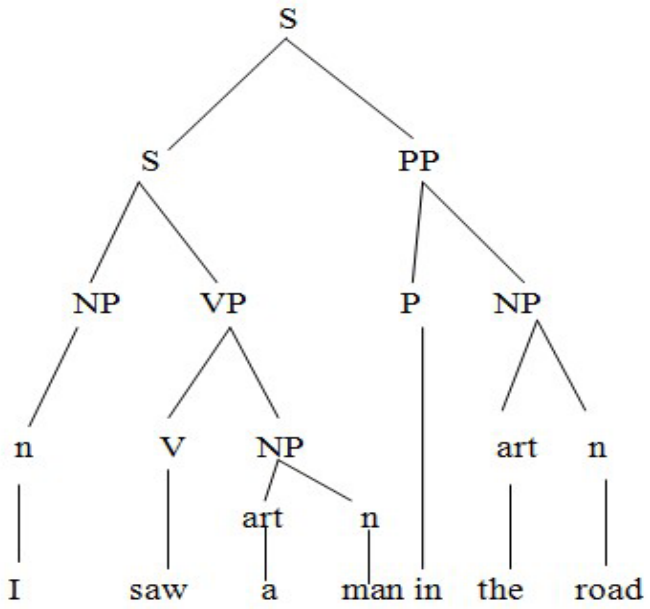


Figure 3.2: Example of parse tree

There are two basic techniques of parsing which are as follows

- Top-Down Parsing- Top-down parsing always begins with the start symbol and applies the grammar rules forward until the symbols at the terminals of the tree correspond to the components of the sentence being parsed.

- Bottom-Up Parsing- Bottom-Up parsing always begins with the sentence to be parsed and apply the grammar rule back-ward until a single tree whose terminals are the words of the sentence and whose top node is the start symbol has been produced.

If we consider the same grammar rule, the possible top-down parser action for the sentence "I saw a man in the road" is as follows

Grammar rule:

1. S → NP VP
2. S → S PP
3. NP → n
4. NP → art n
5. NP → NP PP
6. PP → p NP
7. VP → v NP
8. n → I
9. n → man
10. n → road
11. v → saw
12. art → a
13. art → the
14. p → in

S	[S → S PP]
S PP	[S → NP VP]
NP VP PP	[NP → n]
n VP PP	[n → I]
"I" VP PP	[VP → v NP]
"I" v NP PP	[v → saw]
"I" "saw" NP PP	[NP → art n]
"I" "saw" art n PP	[art → a]

“I” “saw” “a” n PP	[n → man]
“I” “saw” “a” “man” PP	[PP→ p NP]
“I” “saw” “a” “man” p NP	[p → in]
“I” “saw” “a” “man” “in” NP	[NP → art n]
“I” “saw” “a” “man” “in” art n	[art → the]
“I” “saw” “a” “man” “in” the n	[n → road]
“I” “saw” “a” “man” “in” “the” “road”	

The possible bottom-up parser action for the sentence “I saw a man in the road” is as follows

“I” “saw” “a” “man” “in” “the” “road”	
n “saw” “a” “man” “in” “the” “road”	[n → I]
NP “saw” “a” “man” “in” “the” “road”	[NP→ n]
NP v “a” “man” “in” “the” “road”	[v → saw]
NP v art “man” “in” “the” “road”	[art → a]
NP v art n “in” “the” “road”	[n→ man]
NP v NP “in” “the” “road”	[NP → art n]
NP VP “in” “the” “road”	[VP → v NP]
S “in” “the” “road”	[S → NP VP]
S p “the” “road”	[p → in]
S p art “road”	[art → the]
S p art n	[n →road]
S p NP	[NP →art n]
S PP	[PP → p NP]
S	[S → S PP]

The selection between top-down and bottom-up parsing is similar to the choice between forward and backward reasoning as other problem solving task. The branching factor is the important consideration for this purpose. In some cases these two parsing techniques combined into a single method which is generally termed as bottom-up parsing with top-down filtering. In this method the grammar rules are applied in backward that means it proceeds essentially bottom-up parsing.

The process of understanding a sentence is a search process to find one that meets all constraints imposed by a particular sentence. There are four ways of handling sentence and these are as follows

- All paths- here follow all possible paths and construct all the possible intermediate components. Many of the interpretations ignored because these interpretations will not appear at the later time. The major disadvantage of this approach is that many spurious constitutions being built and many deadend paths being followed. It is not an efficient way of handling sentences.
- Best path with Backtracking- In this handling, follow only one path at a time but record every choice point. The recorded choice point will be considered if the previous chosen path fails to interpretation of the sentence. The implementation of the parser is more complex in this handling approach.
- Best Path with Patchup- It will follow only one path at a time but error is occurred, it will explicitly shuffle around the components that have already been formed. This approach is usually more efficient than the previous two techniques. The major disadvantage of this approach is that, it requires interactions among the rules of the grammar to be made explicit in the rules for moving components from one place to another.

- Wait and see- In this approach follow only one path. But making a final decision about the function of each component it will wait until enough information is available to make the decision correctly. It uses a small fixed size buffer in which the constituents can be stored until the final decision can be taken. This approach is very efficient in comparison to the other approaches. The main disadvantage of this approach is that if the amount of storage of constituents is greater than the size of the buffer, then the interpreter will fail.

3.7 SEMANTIC ANALYSIS

The first step of understanding a language is producing the syntactic parse on that sentence. After that we still produce a representation of the meaning of the sentence. There is no single, definitive language in which all the sentence meaning can be described. The main purpose of semantic processing is the creation of a target language representation of a sentence's meaning.

3.7.1 Lexical Processing:

The first step of semantic processing is to look up individual words in a dictionary or lexicon and extract their meaning. But in a language one word have different meanings and it is may not be possible to identify the correct word only by seeing the word. For example the word "diamond" might have the following meaning

- A geometrical shape
- A baseball field
- A valuable gemstone

To select the correct meaning of the “diamond” in the sentence,

“Hari buy a diamond ring for her”. Here it is necessary to know that neither geometrical shape nor base ball field is appropriate for the sentence whereas a valuable gemstone is perfect meaning for the sentence. So there is lexical ambiguity is often in everyday English. If we consider the word “mean” – the word is ambiguous at least three ways. It can be “to signify” as verb meaning, as adjective meaning “unpleasant” or “cheap” and as noun meaning “statistical average”. So lexical ambiguity is a serious problem, even domain of discourse is severely constrained. The process of determining the correct meaning of an individual word is called lexical disambiguation or word sense disambiguation.

3.7.2 Sentence Level Processing:

In sentence level processing, several approaches to the problem of creating a semantic representation of sentence have been developed, including the following:

- Semantic grammars, the semantic grammar combine the syntactic, semantic and pragmatic knowledge into single set of rules in the form of a grammar. The semantic grammar is a context context-free grammar. Here the choices are the non terminal and the production rules are controlled by semantic and syntactic functions. In a sentence applying all the associated semantic actions reflects the meaning of the sentence.
- Case grammar, case grammar contains some semantic information although it will require further interpretation may also be necessary. Parsing using a case grammar is usually exception-driven.

- Conceptual parsing, in conceptual parsing syntactic and semantic knowledge are combined into a single interpretation system that is driven by the semantic knowledge. Conceptual parsing is done on the basis of dictionary that describes the meaning of words as conceptual dependency structure. It is similar to the process of parsing using a case grammar.
- Approximately, compositional semantic interpretation, in this section semantic processing is applied to the result of performing a syntactic parse. If a syntactic parse of a sentence is produced, then the way of generating semantic interpretation is as following-
    - Check each word in a lexicon that contains one or more definitions for that particular word. These definitions must describe how the idea that corresponds to the word is to be represented. It also described how the idea represented by the word may combine with ideas represented by other words in the sentence.
    - Use the structure information contained in the output of the parser to give additional restriction beyond those extracted from the lexicon, on the way that a single word meaning may combine to form larger meaning units.

Stop to Consider

Parsing into a case representation, directed by the lexical entries associated with to each verb.

3.8 DISCOURSE AND PRAGMATIC PROCESSING

It is necessary to consider the discourse and pragmatic context to understand a sentence. The discourse and pragmatic processing is more important when one needed to understand the text and

dialogues. There are different relationships that may hold between the phrases and parts of their discourse context. The following are the some examples in English language

- Identical entities. For this purpose consider the text
  -Ram had a blue balloon
  -Hari wanted it
  The word "it" should be identified as referring the blue balloon. Such type of references is called *anaphoric references* or *anaphora.*

- Parts of entities. Consider the following sentences
  - Ram opened the book he just bought
  - The title page was torn

  The "the title page" should be recognized as the part of the book that was just bought.

- Parts of actions. Consider the text
  - Pranab went on a business trip to Guwahati
  - He left on an early morning bus.

  Here taking a bus should be recognized as part of going on a trip.

- Entities involved in actions. Consider the text
  - My car was broken on last week.
  - they took the head light and stereo

  The pronoun "they" should be recognized as the persons who broke the car on last week.

In order to be able these types of relationships among sentences, a vast discussion is required about the knowledgebase. The way this knowledge is organized is important to the success of the understanding a sentence. For this purpose we always focus on the use of the following kinds of knowledge

- The current focus of the dialogue
- A model of each participant's current beliefs
- The goal-driven character of dialogue and
- The rules of conversation shared by all participants

Check Your Progress - II

1. State True or False

(i) The main morpheme is called the stem.

(ii) Prefixes always follow the stem.

(iii) Suffixes always precede the stem.

(iv) Parsing produces grammatical structure according to the input sentences.

(v) In top-down parsing, grammar rules are applies in forward directions.

2. Fill in the blanks

(i) _____ processing must operate on sentence constituents.

(ii) A _____ is a computer program that carries the parsing task.

(iii) _____is the formal specification of the structures which are allowed in the language.

(iv) _____ parsing always begins with the sentence to be parsed.

(v) The process of determining the correct meaning of an individual word is called_____.

3.9 SUMMING UP

- Natural language processing is a subfield of Artificial Intelligence.
- Natural language studied the automatic generated problems in natural language processing and understanding the natural human languages.
- The term natural language refers to the language that people speak, like English, Assamese, Hindi etc.
- The computational activities required for enabling a computer to carry out information processing using natural language is called natural language processing.
- Phonological knowledge includes how words are related to the sounds that realized them.
- Morphological knowledge concerns how words are formed using the morphemes.
- The syntactic knowledge concerns how words can be put together to form a sentence and determine what type of structure role played by each word in the sentence.
- Semantic knowledge concerns what is the meaning of a word and how these meaning combine in sentence meaning.
- Discourse knowledge concerns how the immediately preceding sentences affect the interpretation of the next sentence.
- Morphology is the study of the way, where words are built up from smaller meaning bearing units called morphemes.
- Stem is the main morpheme of the word and affixes add additional meaning to the main morpheme.

- Parsing is a process which automatically building syntactic analysis of a sentence in terms of a given grammar.
- The discourse and pragmatic processing is more important when one needed to understand the text and dialogues in a natural language.

3.10 ANSWER TO CHECK YOUR PROGRESSES

1. (i) False (ii) False (iii) True (iv) True (v) False

2. (i) True (ii) False (iii) False (iv) True (v) False

3. (i) semantic (ii) parser (iii) grammar (iv) bottom-up
(v) lexical disambiguation

3.11 POSSIBLE QUESTIONS

1. What is NLP? Explain its benefits.
2. What is the goal of NLP?
3. Explain the different applications of natural language.
4. What is morphological knowledge related to natural language?
5. What is Syntactic knowledge related to natural language?
6. Differentiate between syntactic and semantic knowledge related to natural language.
7. Explain the syntax, semantics, and pragmatics in a language with some suitable examples
8. What is morpheme? Explain its classifications.
9. What is parsing in natural language?
10. Explain the top-down and bottom-up parsing techniques.
11. What is lexical processing? Explain it with a suitable example.

12. Explain the discourse and pragmatic processing in a sentence.

3.12 REFERENCES AND SUGGESTED READINGS

- Rich, E., & Knight, K. Artificial intelligence, Second Edition, Ed.
- Luger, G. F. (2004). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 5/e*. Pearson Education India.
- Nilsson, N. J. (2014). *Principles of artificial intelligence*. Morgan Kaufmann.
- NOC: Natural Language Processing https://nptel.ac.in/courses/106105158

*****

UNIT 4: CONCEPT OF EXPERT SYSTEMS

Unit Structure:

4.1 INTRODUCTION

An expert system is a branch of Artificial Intelligence that uses specialized knowledge to solve problems in a specific area. It is an intelligent computer program designed to make decisions and solve complex issues that usually require human expertise. Essentially, an expert system tries to mimic the decision-making abilities of a human expert in a particular field.

The knowledge used by an expert system is gathered from various sources, such as books, magazines, and conversations with experienced professionals. This knowledge is not general but focused on solving problems within a specific domain.

So, an expert system aims to replicate some of the intelligent behaviours of a human expert to provide solutions in a specific area of expertise. In this unit, we will briefly explore the concept of expert systems.

4.2 OBJECTIVE

After going through this unit, you will be able to

- Understand the history and applications of expert systems.
- Learn the principles of expert systems (ES).
- Understand the architecture of expert systems and its different variations.
- Gain knowledge about the techniques used for knowledge acquisition and representation in expert systems.
- Understand various inference techniques used in expert systems.
- Learn about expert system shells and their uses in building expert systems.

4.3 EXPERT SYSTEM HISTORY

Expert systems that capture the knowledge of human experts in their own fields of expertise were a success story for artificial intelligence research in the 1970s and 1980s. Early, successful expert systems were built around rules (sometimes called heuristics) for medical diagnosis, engineering, chemistry, and computer sales. One of the early expert system successes was MYCIN2, a program for diagnosing bacterial infections of the blood. Expert systems had a number of perceived advantages over human experts. For instance, unlike people, they could perform at peak efficiency, 24 hours a day, forever. Over time, of course, the drama receded, and it became clear that researchers had vastly underestimated the complexity of the common-sense knowledge that underpins general human reasoning. Nevertheless, excellent applications for expert systems remain to this day. Modern expert systems advise sales people, scientists, medical technicians, engineers, and financiers, among others.

4.4 ADVANTAGES OF EXPERT SYSTEM

An expert system and a conventional programme can both present to the user a series of options on the screen in the form of questions, and answers, which would be given depending on the user responses to those answers. However, there are a number of advantages of expert systems over conventional programs:

(i) Expert systems usually deal with large amounts of knowledge since it has the ability to handle qualitative information.

(ii) The knowledge of multiple experts can be made available to work simultaneously and continuously on a problem at any time. The level of expertise combined from several experts may exceed that of a single human expert.

(iii) Increased availability: Web based expert system can give expertise to the end user removing any physical constraints if the system is made available online through Internet.

(iv) Expert systems are not confined by rigid mathematical or analogue schemes and can handle factual or heuristic knowledge;

(v) The knowledge base can be continuously augmented as necessary with accumulating experience.

4.5 PRINCIPLES OF EXPERT SYSTEM

The principles of a knowledge-based expert system are described in figure 4.1. Most expert systems operate through the user supplying facts and different information to the expert system and in return the user obtains expert advice. The expert system consists of two main internal parts, the knowledge base and the inference engine; the knowledge-base which contains the knowledge helps the inference engine in drawing the related conclusions. These conclusions are considered to be the expert system's responses to the user's queries for expertise.

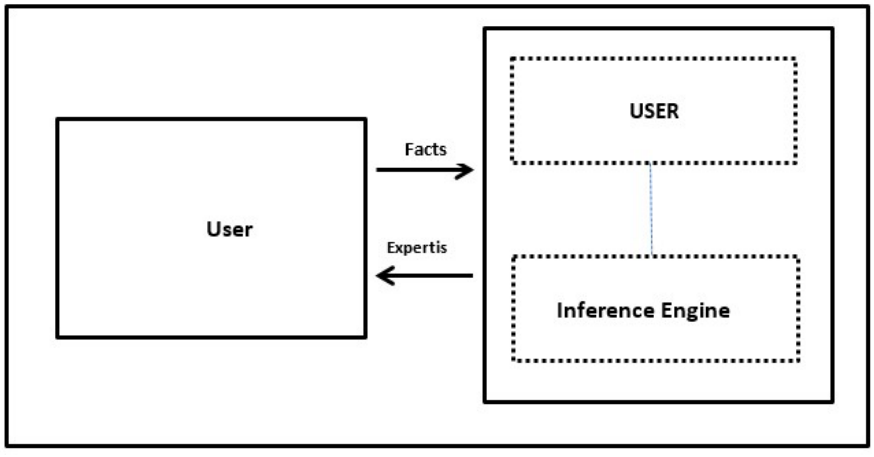


Figure 4.1: Basic concepts of an expert system function

Another vital factor to understand in any expert system is the relationship between the problem domain and the knowledge domain. Giarratano and Riley (1989) discuss this relationship as follows: all knowledge domains are usually presented within the problem domain. As can be seen from figure 4.2 the section outside the knowledge domain symbolises an area in which there is inadequate knowledge about the problem. The knowledge acquired in most expert systems are usually obtained from published materials, project records and directly from experts in the field that the expert system is being built.

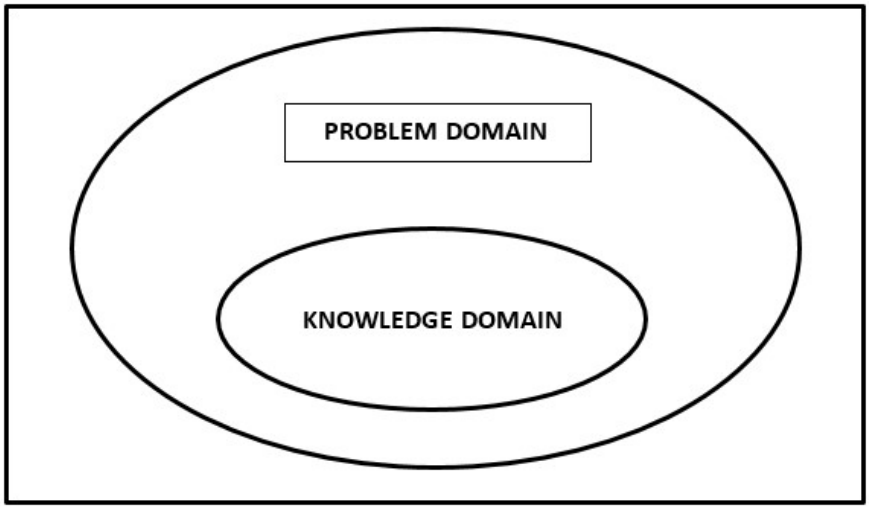


Figure 4.2: A possible problem and domain knowledge Relationship

4.6 EXPERT SYSTEM ARCHITECTURE

In order to get an understanding of how expert systems function, it is appropriate to look at the expert system architecture and examine the different components that contribute to presenting the expert's knowledge in such a system. The architecture of an expert system is difficult to define, as languages and system build up tools can vary in their development stage and are therefore illustrated through examples. An example of expert system architecture is shown below

in figure 4.3. The example illustrates all different components of the expert system architecture. The basic architecture shows a separation of domain knowledge, control knowledge and knowledge which deals with the problem in hand which needs to be solved. This highlights three important components of an expert system which are: the knowledge base, the context, and the inference mechanism. Other components that can be part of the expert system architecture are user interface and an explanation facility. Finally, a knowledge acquisition facility is also considered to be useful in many expert systems. Maher (1987) explains further the expert system components as:

(a) Knowledge Base: The knowledge base is the component of an expert system that contains the facts about the subject which is being dealt with. The facts could be presented in the form of rules and sub rules. Since knowledge is continually changing and expanding it is considered to be important that the knowledge base is clearly structured and can easily be modified if required to do so.

(b) Context: The context is the component which is responsible for providing the information on the problem which is being solved. As the problem solving procedure continues the context will expand to provide more information on the problem in order to continue to solve it.

(c) Inference mechanism: The inference mechanism is the part of the expert system which contains the control information. It does that by using the knowledge base to modify and expand the context. The inference mechanism's main task is to relate rules or sub-rules to the facts and execute the most appropriate rules that can satisfy the facts.

(d) Explanation facility: The explanation facility in an expert system varies from a trace of execution to the ability to give the user the reasons behind reaching a particular solution. It will demonstrate this

by showing the user the path that was followed in order to reach a certain conclusion.

(e) Knowledge acquisition: The knowledge acquisition facility in an expert system is the component which is responsible for entering the knowledge to the knowledge base. This facility acts as an editor and knowledge is entered directly in a form acceptable to the way in which the expert system was structured. Editing the knowledge can be carried out in two ways: either the knowledge engineer uses a screen editor to create and modify a file of rules, or the editor is itself an expert system, and that would be used in building more complex expert systems.

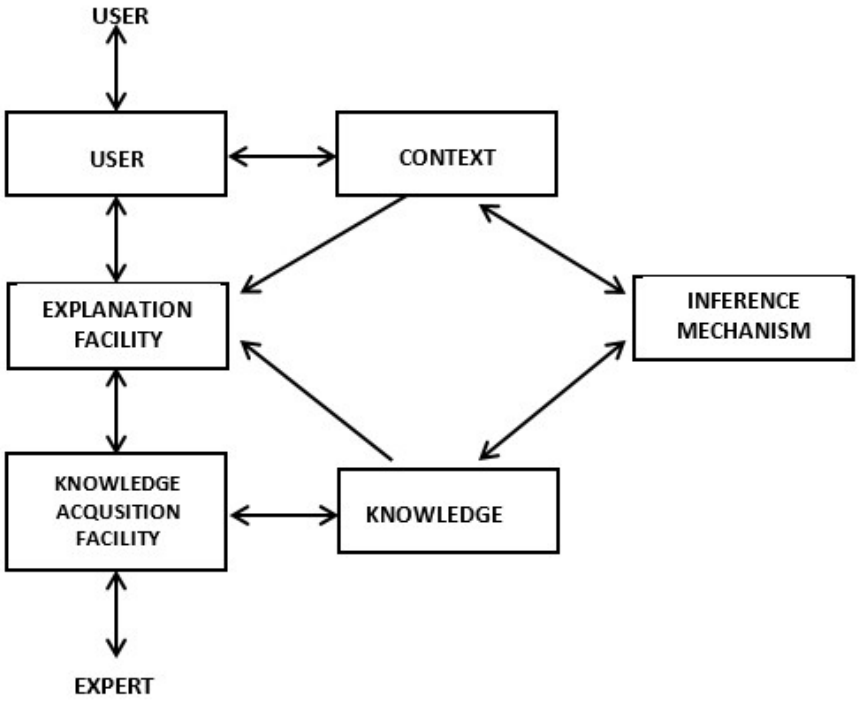


Figure 4.3: Architecture of an expert system

(f) User interface: The expert system user interface is the component which is responsible for the communication mechanism between the user and the system. In addition to being highly interactive, an expert system interface requires a transparency of dialogue, whereby some

form of an explanation facility indicates the inference process that is being used.

4.7 ARCHITECTURAL VARIATIONS

Two of the most commonly used variations on the basic architecture are the blackboard model and the production system model. The two models will be looked at in more detail below.

(a) Blackboard model:

As can be seen in figure 4.4, the blackboard model is based upon the separation of the knowledge base into knowledge sources. It also provides a means of communication between knowledge sources. The circles and lines represent the communication between knowledge sources and the current state of the problem solved which takes place within the blackboard (Maher, 1987). Blackboard architectures are explicitly designed to permit multiple knowledge sources to address a problem simultaneously. Provided its pre-conditions are satisfied, each knowledge source can post recommendations for action to a shared data termed a "blackboard".

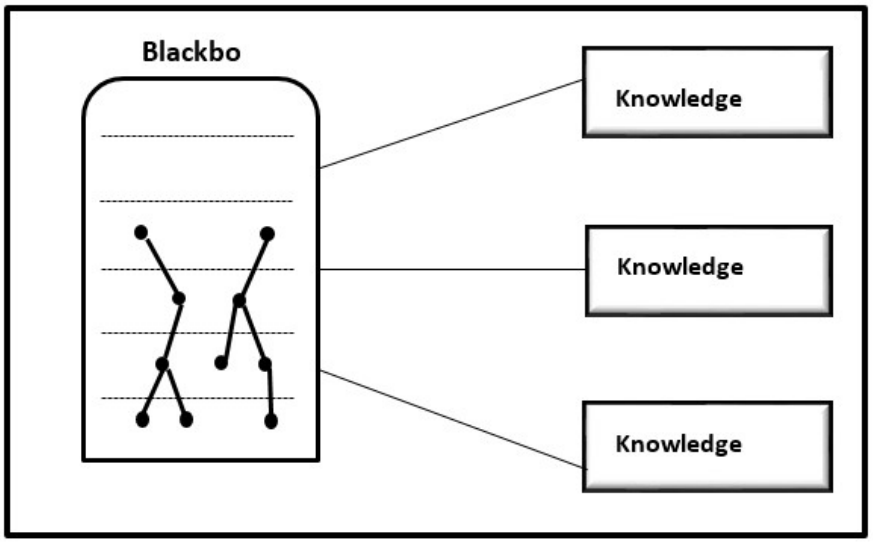


Figure 4.4: Black board Model

(b) **Production system model**

The production system model classifies the knowledge base as a series of rules, usually known as production memory. Normally the rules would be developed by the expert, and there would be no need to specify them in any certain order. The context in a production system model is known as the working memory. The inference mechanism in such a system is expected to identify the production rules that would be executed and perform the selection operation to choose the most suitable rule to solve the problem. The production system model is illustrated in figure 4.5 below.

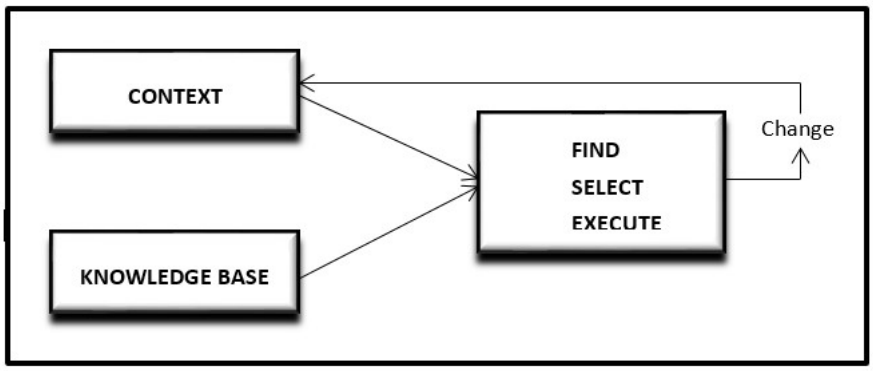


Figure 4.5: Production system Model

4.8 EXPERT SYSTEM VS ALGORITHMIC PROGRAMS

The important features of expert systems when compared to other mathematical models are pointed out by Jackson (1986) as:

(a) Expert systems are not confined by rigid mathematical or analogue schemes and can handle factual or heuristic knowledge.

(b) The knowledge base can be continuously augmented as necessary with accumulating experience. As the knowledge advances the expert system can be easily upgraded to cope with the changes in technology.

(c) Ability to handle qualitative information. This can be clearly experienced if the system to be built is expected to contain a large amount of information as for example in many management areas.

(d) Coping with uncertain, unreliable or even missing data. Dealing with uncertainty in data and inference is a feature of expert systems. When pieces of the knowledge base and context are less than certain, then a new level of complexity is introduced into expert systems. Some expert system shells have adopted Bayesian probability in coping with uncertainties within the knowledge base.

(e) The reflection of decision patterns of the users. This is a function which is sometimes referred to as an explanation facility. It explains the reasons behind giving a certain conclusion that has been reached by the expert system.

4.9 KNOWLEDGE ACQUISITION

There are several steps in knowledge engineering process. All the steps need to be completed for an effective expert system design. Domain identification is the first step. It is an important step as better domain knowledge may yield a better expert system design. In this process, Knowledge Acquisition process is the next important step. This step is not only time consuming but also known as the greatest bottleneck in the expert system development process. This step requires knowledge engineers to extract data, information and knowledge from experts of the identified domain or from other resources like books and journals. After obtaining data, information and knowledge, knowledge engineering present them in a machine-usable form. All these steps, starting from domain identification to presentation is referred to as knowledge acquisition and presentation process. How to acquire knowledge efficiently from a domain and represent it in an appropriate computer format is an extremely

difficult and challenging task. As a result, huge efforts can be seen in this research, and over the years many knowledge acquisition methods have been discovered. Although, studies suggest that there is no perfect method but there are suitable methods for almost every domain. However, a technique that is suitable for one domain may not necessarily work for other domains. As a knowledge engineer one must discover a suitable method for his application domain.

Moreover, we must understand that knowledge elicitation has certain issues we must be aware of. Such as that even if we extract knowledge from books, journals, and other written resources, it is not enough, our knowledge base will still be lacking the knowledge that remains locked up inside the experts' heads'. We must also keep in mind that experts time is valuable thus it is hard to keep him away from his work for long periods of time to extract the knowledge he has.

Another couple of issues we must understand are that even though experts have immense knowledge, yet they do not have all the knowledge of that domain, and that knowledge when extracted has a "shelf-life". As if to say, the knowledge has an expiration date in certain domains, where the information must be updated or it is of no use to the system.

Another important issue in this process is the unclear or tacit knowledge that we must extract from the experts, who cannot explain to us this type of knowledge, it is a hard process and in some cases impossible.

With these issues in mind, engineers must learn to adapt or work around them. For example, since experts are too busy, why not take them out of their current jobs for a period of time, also, do not depend on a single expert, interview more than one expert, and try and extract as much knowledge as possible, also make sure the knowledge is well maintained and validated on regular basis.

4.9.1Knowledge Acquisition Techniques

Some of the knowledge acquisition techniques are:

(a) Protocol-generation techniques which include a variety of types that range from interviews (unstructured, semi-structured and structured), to reporting techniques and observational techniques.

(b) Protocol analysis techniques are used with transcripts of interviews or other text-based information to discover the different types of knowledge, such as goals, decisions, relationships and attributes.

(c) Hierarchy-generation techniques, such as laddering, are used to build taxonomies and other hierarchical structures such as goal trees and decision networks. Laddering means the creation, reviewing, and modification of hierarchal knowledge.

(d) Matrix-based techniques are about constructing grids in which they indicate things such as problems encountered against possible solutions or hypotheses against diagnostic techniques. Frames are considered an important type used with these types of techniques because we can represent properties of concepts. Matrices are mostly used to validate knowledge rather than elicit it.

(e) Sorting techniques are used for capturing how people compare and order concepts, and can lead to the revelation of knowledge about classes, properties and priorities.

(f) Limited-information and constrained-processing tasks are techniques that either limits the time and/or information available to the expert when performing tasks.

(g) Diagram-based techniques include the generation and use of concept maps, state transition networks, event diagrams and process maps. The use of these is particularly important in capturing the “what, how, when, who and why" of tasks and events.

4.10 KNOWLEDGE REPRESENTATION

Knowledge Representation (KR) can be defined as "The encoding and storage of knowledge in computational models of cognition."

Algorithms + Data Structures = Programs

For expert system, we can say that

Knowledge + Inference = Expert Systems

Representation of Knowledge in computational models is a complex problem. Its complexity makes it difficult to devise good KR techniques. However, there are criteria for judging their goodness. The knowledge representation technique should not only be functional but also should able to explain the functionality and have provisions to store the pertinent data that may be needed to justify decisions at a given point of time. Among many, semantic networks, production rules and frames are the three main structures that more or less meet these criteria. Their properties and usage have been thoroughly investigated for encoding and storage of knowledge which is referred to Knowledge Representation techniques in computational models of cognition.

A good system for the representation of knowledge in a particular domain should possess the following four properties:

(a) **Representation Adequacy:** the ability to represent all kinds of knowledge that are needed in the domain.

(b) **Inferential Adequacy:** the ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.

(c) **Inferential efficiency:** the ability to incorporate additional information into the knowledge structure that

can be used to focus the attention of the inference mechanisms in the most promising directions.

(d) **Acquisitional Efficiency:** the ability to acquire new information easily. The simplest case involves direct insertion, by a person, of new knowledge into the database. Ideally, the program itself would be able to control knowledge acquisition.

Unfortunately, no single system that optimizes all of the capabilities for all kinds of knowledge has yet been found. As a result, multiple techniques for knowledge acquisition exist.

The prevalent KR techniques are briefly reviewed next.

4.10.1 Semantic Networks

It has been used mainly to transform the natural language into a graph structure where nodes and edges correspond to concepts and relationships respectively. In this representation, nodes represent entities and classes of entities as well.

It is easy to use semantic representation if we are implementing simple relationships between objects and classes.

4.10.2 Frames

Frames are useful for simulating common sense knowledge, which is a very difficult area for computers to masters. Semantic nets are basically two-dimensional representations of knowledge; frames add a third dimension by allowing nodes to have structures. These structures can be simple values or other frames. While semantic nets lack description of objects, frame representation fills that void. While Frames represent objects, the slots represent the description. Frames

are known for their perfect reflection of domain knowledge, efficiency, default reasoning, and support for procedural knowledge.

4.10.3 Production Rules

The simplest form of representation made of simple IF-THEN rules. They are condition-action, if a condition is met, corresponding rule is fired and action is taken. If more than one condition is met, corresponding rules are fired, and due to this conflict, no action is taken until a conflict resolution method result in selecting one rule, and then it performs the action of that rule. Their modularity, simplicity, and good performance are what make them most often used in simple domains. Moreover, when dealing with domains that contain complex relationships, rule base is not the best representation type since it is hard to deal with uncertainty as well.

Check Your Progress 1:

I. Multiple-Choice Questions (MCQs)

1. What was one of the early successful expert systems used for medical diagnosis?
   a) DENDRAL
   b) MYCIN
   c) PROSPECTOR
   d) XCON
2. Which component of an expert system is responsible for drawing conclusions based on the knowledge base?
   a) User Interface
   b) Knowledge Base
   c) Inference Mechanism
   d) Context

3. Which of the following is an advantage of expert systems over conventional programs?
   a) They require constant human supervision
   b) They cannot handle heuristic knowledge
   c) They can incorporate the knowledge of multiple experts
   d) They work only with rigid mathematical models
4. What is the main function of the explanation facility in an expert system?
   a) To acquire knowledge from experts
   b) To communicate between users and the system
   c) To explain the reasoning behind the conclusions
   d) To expand the knowledge base
5. What is a key characteristic of the blackboard model in expert systems?
   a) It relies on a single knowledge source
   b) It allows multiple knowledge sources to address a problem simultaneously
   c) It follows a strict sequence of rules
   d) It does not support communication between knowledge sources

II. True/False Questions

1. Expert systems were a major success in AI research during the 1970s and 1980s.
2. The knowledge base of an expert system remains static and cannot be modified over time.
3. The production system model organizes the knowledge base as a set of predefined rules without requiring a specific execution order.
4. Expert systems cannot handle uncertain, unreliable, or missing data.
5. Knowledge acquisition is considered one of the most challenging steps in developing an expert system.

4.11 INFERENCE ENGINES

In expert systems, an inference engine deduces new knowledge from available knowledge and observations. In principle, an inference engine accepts observations in the form of user inputs and using its knowledge-base it deduces new knowledge by applying logical rules as illustrated in figure 4.6. In addition to knowledge discovery, an expert system is also expected to provide a justification and explanation about its decision. Therefore, an inference engine should not only produce expert-level decisions but also back it up with the reasoning process that leads to such decisions.

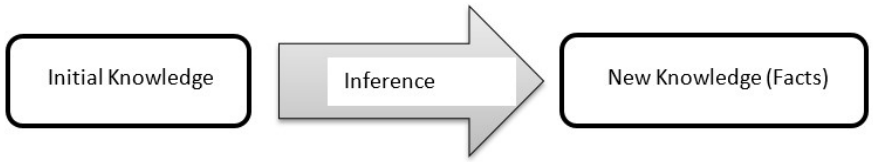


Figure 4.6: Inference Engine infers new facts from available knowledge

Knowledge bases are an essential component in an expert system as they contain facts and rules about the knowledge domain. The knowledge core is acquired from experts in the domain, and additional knowledge can be obtained from text books, manuals and other resources. The acquired knowledge is then organized as a collection of rules known as production rules. The production rules are in an IF-THEN format known as condition and action. A simple example:

IF (traffic_light is red) THEN (slowdown)

To explain this simple rule, traffic_light is the premise, which means the hypothesis, and slowdown is the consequent, known as the action to perform. So if the traffic light is red, the rule is fired, and the action is completed, i.e. the car will slow down.

The inference engines are built using different reasoning techniques. The backward-chaining and forward-chaining are the most frequently

used techniques. However, hybrid techniques that use a mixture of both techniques have been developed for improved accuracy and efficiency.

4.11.1 Forward-chaining Technique

In a rule-based expert system, the domain knowledge is represented by a set of IF-THEN production rules and data is represented by a set of facts about the current situation. The inference engine compares each rule stored in the knowledge base with facts contained in the database. When the IF (condition) part of the rule matches a fact, the rule is fired and its THEN (action) part is executed. The fired rule may change the set of facts by adding a new fact as shown in figure 4.7. Letters in the database and the knowledge base are used to represent situations or concepts.

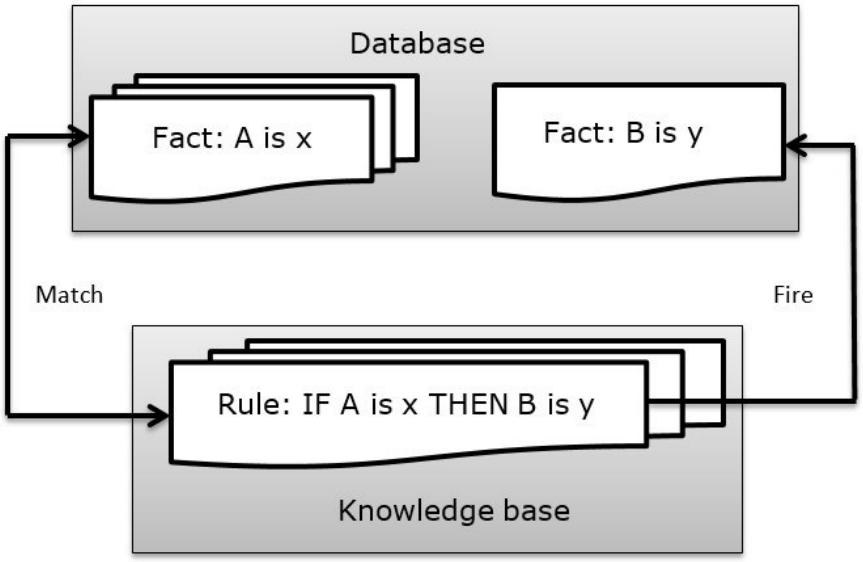


Figure 4.7: The Inference Engine cycles via a match-fire procedure

The matching of the rule IF parts to the facts produce inference chains. The inference chain indicates how an expert system applies the rules to reach a conclusion. To illustrate chaining inference techniques, consider a simple example.

Suppose the database initially includes facts A, B, C, D and E and the knowledge base contains only three rules:

Rule 1: IF Y is true
AND D is true
THEN Z is true

Rule 2: IF X is true
AND B is true
AND E is true
THEN Y is true

Rule 2: IF A is true
THEN X is true

The inference chain shown in figure 4.8 indicates how the expert system applies the rules to infer fact Z. First Rule 3 is fired to deduce new fact X from given fact A. Then Rule 2 is executed to infer fact Y from initially known fat B and E, and already known fact X. Finally, Rule 1 applies initially known fact D and just-obtained fact Y to arrive at conclusion Z.

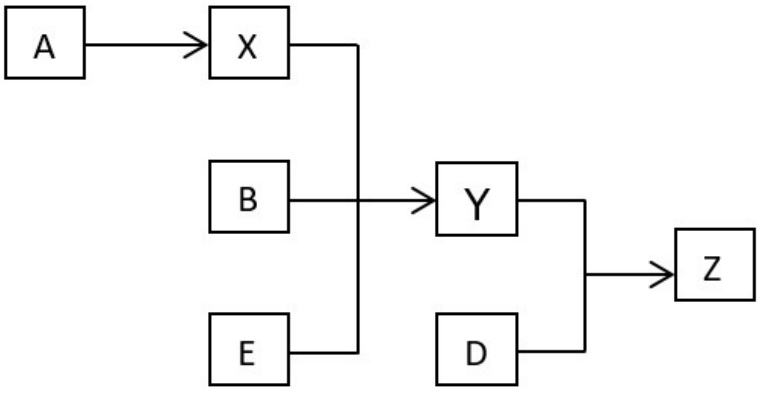


Figure 4.8: An example of inference chain

There are two principal ways in which rules are executed: (a) forward chaining and (b) backward chaining

The example discussed above, uses forward chaining. Forward chaining is the data driven reasoning. The reasoning starts from the known data and proceeds forward with that data. Each time only the top-most rule is executed. When fired, the rule adds a new fact in the database. Any rule can be executed only once. The match-fire cycle stops when no further rules can be fired.

4.11.2 Backward-chaining Technique

Backward chaining is the goal-driven reasoning. In backward chaining, an expert system has the goal (a hypothetical solution) and the inference engine attempts to find the evidence to prove it. First, the knowledge base is searched to find rules that might lead to the desired solution. Such rules must have the goal in their THEN (action) parts. If such a rule is found and the IF (condition) part matches data in the database, then the rule is fired and the goal is proved. However, this is rarely the case. Thus the inference engine puts aside the rule it is working with (the rule is said to be in the stack) and sets up a new goal, a sub-goal, to prove the IF part of this rule. Then the knowledge base is searched again for rules that can prove the sub-goal. The inference engine repeats the process of stacking the rules until no rules are found in the knowledge base to prove the current sub-goal.

4.11.3 Hybrid Technique

Hybrid techniques combine forward and backward chaining techniques. This indicates that the inference engine will perform forward-chaining and then backward-chaining. This is used to confirm a diagnosis or a hypothesis which has reached through forward chaining.

4.12 EXPERT SYSTEM SHELLS/TOOLS

Designing an Expert System (ES) from scratch is a time consuming and costlier process. The alternative approach is the use of expert system shells/tools. Expert System shells are the tools for construction of expert systems which provides knowledge representation facilities and inference mechanisms. The knowledge engineer must gain detailed knowledge about a particular problem domain from an expert and other information source. Hence, ES shell can be considered as an expert system with all the domain specific knowledge removed and a facility for entering a new knowledge base provided. An ES shell is the software skeleton which provides an inference engine and reasoning techniques and can be customized through user interface to add knowledge of a domain. It results of a tailored expert system that matches user requirements.

Inference methods vary significantly from one domain to another and expert system shells have developed to allow the designer more flexibility during the development of the expert system.

Bourbakis (1993) states some of the advantages that arise from using an ES tool include the following:

- Use of an ES tool can improve the quality and reliability of the resulting expert system
- Tools relieve the expert system builder from having to deal with low level programming.
- It allows the expert system builder to focus on the modelling of the expert system domain.
- Tools offer facilities for the acquisition and modification of the expert system's knowledge.

4.12.1 ES Shell Components

The generic components of ES shell: the knowledge acquisition, the knowledge base, the reasoning engine, the explanation and the user interface are shown in figure 4.9

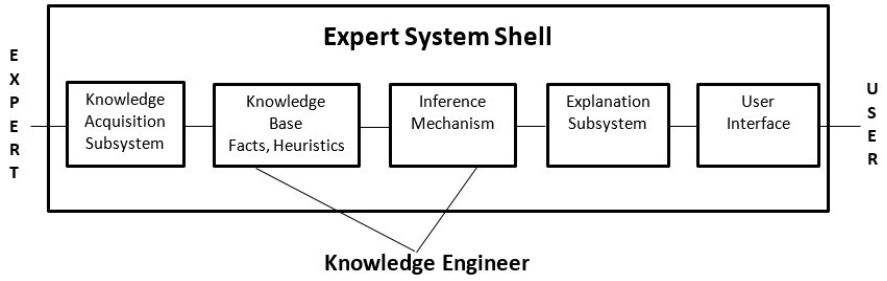


Figure 4.9 Components of ES Shell

There are several approaches in developing expert systems in terms of what kind of ES tools to use. These tools are normally classified as: languages, environments or shells, while some tools may fall between any two categories.

(a) Languages can be either special purpose languages for symbolic programming, such as LISP or PROLOG, or a conventional one, such as PASCAL, C or Java.

(b) Environments contain various types of knowledge representation, inference mechanisms, user interface and development aids. These tools also give access to the underlying language the environment is written in. This enables the developer to incorporate special tasks. KEE, ART and EDSS are all environments.

(c) Shells provide a more specific set of knowledge representation languages and inference mechanisms, geared to handle a particular class of problems.

Both shells and environments differ from the languages in the fact that they already contain control mechanisms that determine how they reach conclusions.

ES shells differ in many aspects such as programming language knowledge, domain knowledge, and development time.

4.12.2 Examples of ES Shells

Lots of ES shells are available, some are commercial and some are open source and free. Let us take an overview of some of the ES shells that are used worldwide in most of the research and development projects.

CLIPS: CLIPS is a productive development and delivery expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. Created in 1985, CLIPS is now widely used throughout the government, industry, and academia. Its key features are:

Knowledge Representation: CLIPS provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. Rule-based programming allows knowledge to be represented as heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. Object-oriented programming allows complex systems to be modelled as modular components (which can be easily reused to model other systems or to create new components). The procedural programming capabilities provided by CLIPS are similar to capabilities found in languages such as C, Java, Ada, and LISP.

Portability: CLIPS is written in C for portability and speed and has been installed on many different operating systems without code changes. Operating systems on which CLIPS has been tested include Windows XP, MacOS X, and Unix. CLIPS can be ported to any system which has an ANSI compliant C or C++ compiler. CLIPS

comes with all source code which can be modified or tailored to meet a user's specific needs.

Integration/Extensibility: CLIPS can be embedded within procedural code, called as a subroutine, and integrated with languages such as C, Java, FORTRAN and ADA. CLIPS can be easily extended by a user through the use of several well-defined protocols.

Interactive Development: The standard version of CLIPS provides an interactive, text oriented development environment, including debugging aids, on-line help, and an integrated editor. Interfaces providing features such as pull down menus, integrated editors, and multiple windows have been developed for the MacOS, Windows XP, and X Window environments.

Verification/Validation: CLIPS includes a number of features to support the verification and validation of expert systems including support for modular design and partitioning of a knowledge base, static and dynamic constraint checking of slot values and function arguments, and semantic analysis of rule patterns to determine if inconsistencies could prevent a rule from firing or generate an error.

Fully Documented: CLIPS comes with extensive documentation including a Reference Manual and a User's Guide.

Low Cost: CLIPS is maintained as public domain software.

JESS: Jess is a rule engine and scripting language developed at Sandia National Laboratories in Livermore, California in the late 1990s. It is written in Java, so it is an ideal tool for adding rules technology to Java-based software systems. The CLIPS expert system shell, an open-source rule engine written in C, was the original inspiration for Jess. Jess and CLIPS were written by entirely different groups of people; however their implementations have always been very different. Jess is dynamic and Java centric, so it automatically gives you access to all of Java’s powerful APIs for networking, graphics,

database access, and so on; CLIPS has none of these facilities built in. Still, there is a strong similarity between the rule languages supported by these two systems. Many of the core concepts of Jess were originally derived from CLIPS, which was itself influenced by early rule engines like OPS5 and ART.

Check Your Progress 2

Choose the correct option from the followings:

1. What is the primary function of an inference engine in an expert system?
   a) Storing large amounts of data
   b) Applying logical rules to deduce new knowledge
   c) Performing low-level programming tasks
   d) Managing the user interface
2. Which of the following best describes forward chaining in an expert system?
   a) A goal-driven reasoning technique
   b) A method that starts with known data and proceeds forward
   c) A hybrid reasoning approach combining multiple techniques
   d) A process that ignores new facts after the first rule is fired
3. In the context of expert systems, what is the purpose of production rules?
   a) To store large datasets efficiently
   b) To establish logical IF-THEN conditions for decision-making
   c) To enhance the graphical interface of the system
   d) To replace human decision-making entirely
4. What distinguishes expert system shells from languages used in expert system development?
   a) Shells contain predefined control mechanisms for inference
   b) Shells require extensive low-level programming
   c) Languages are more specific to a single problem domain
   d) Expert system shells do not include inference mechanisms

5. Which expert system shell is written in Java and is ideal for integrating with Java applications?
   a) CLIPS
   b) PROLOG
   c) JESS
   d) LISP

4.13 SUMMING UP

- Expert system is a branch of Artificial Intelligence that makes extensive use of specialised knowledge to solve a problem in a specific domain.
- Expert system exhibits some intelligent behaviour of a human expert while taking some kind of decisions to solve a problem in a specific domain.
- The knowledge base is the component of an expert system that contains the facts about the subject which is being dealt with.
- The inference mechanism's main task is to relate rules or sub-rules to the facts and execute the most appropriate rules that can satisfy the facts.
- The explanation facility in an expert system varies from a trace of execution to the ability to give the user the reasons behind reaching a particular solution.
- The knowledge acquisition facility in an expert system is the component which is responsible for entering the knowledge to the knowledge base.
- The expert system user interface is the component which is responsible for the communication mechanism between the user and the system.

- Expert systems are not confined by rigid mathematical or analogue schemes and can handle factual or heuristic knowledge.
- Knowledge Representation (KR) can be defined as the encoding and storage of knowledge in computational models of cognition.
- In expert systems, an inference engine deduces new knowledge from available knowledge and observations.
- Expert System shells are the tools for construction of expert systems which provides knowledge representation facilities and inference mechanisms.

4.14 ANSWERS TO CHECK YOUR PROGRESS

I.

1.(c) 2.(b) 3.(d) 4.(a) 5.(b)

II.

1.True 2.False 3.True 4.False

5.True

III. 1. b) Applying logical rules to deduce new knowledge

2. b) A method that starts with known data and proceeds forward

3. b) To establish logical IF-THEN conditions for decision-making

4. a) Shells contain predefined control mechanisms for inference

5. c) JESS

4.15 POSSIBLE QUESTIONS

Short answer type questions:

1. What is expert system?
2. How expert system differs from any algorithmic approach?
3. What is fact?
4. What is rule?
5. What are the advantages of expert system?
6. What is inference engine?
7. What is agenda?
8. What is conflict resolution?
9. What is ES shell?

Long answer type questions:

1. Explain the architecture of expert system.
2. What are the different knowledge acquisition techniques used in ES?
3. How knowledge is represented in ES? Explain.
4. Explain backward chaining technique?
5. Explain forward chaining technique?
6. Explain the architecture of ES tool? Compare different ES tools.

4.16 REFERENCES AND SUGGESTED READINGS

- Expert Systems Principles and Programming, Giarratano & Riley, China Machine Press
- http://clipsrules.sourceforge.net/

*****