

PRINCIPLES OF ARTIFICIAL INTELLIGENCE

Sathish S. Nadig



Himalaya Publishing House

ISO 9001:2015 CERTIFIED

PRINCIPLES OF ARTIFICIAL INTELLIGENCE

Sathish S. Nadig

Assistant Professor,
Canara Engineering College,
Mangaluru.



Himalaya Publishing House

ISO 9001:2015 CERTIFIED

© **Author**

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise without the prior written permission of the publisher.

First Edition : 2023

Published by	: Mrs. Meena Pandey for Himalaya Publishing House Pvt. Ltd. , “Ramdoot”, Dr. Bhalerao Marg, Girgaon, Mumbai - 400 004. Phone: 022-23860170, 23863863; Fax: 022-23877178 E-mail: himpub@bharatmail.co.in; Website: www.himpub.com
Branch Offices	:
New Delhi	: “Pooja Apartments”, 4-B, Murari Lal Street, Ansari Road, Darya Ganj, New Delhi - 110 002. Phone: 011-23270392, 23278631; Fax: 011-23256286
Nagpur	: Kundanlal Chandak Industrial Estate, Ghat Road, Nagpur - 440 018. Phone: 0712-2721215, 2721216
Bengaluru	: Plot No. 91-33, 2nd Main Road, Seshadripuram, Behind Nataraja Theatre, Bengaluru - 560 020. Phone: 080-41138821; Mobile: 09379847017, 09379847005
Hyderabad	: No. 3-4-184, Lingampally, Besides Raghavendra Swamy Matham, Kachiguda, Hyderabad - 500 027. Phone: 040-27560041, 27550139
Chennai	: No. 34/44, Motilal Street, T. Nagar, Chennai - 600 017. Mobile: 09380460419
Pune	: “Laksha” Apartment, First Floor, No. 527, Mehunpura, Shaniwarpet (Near Prabhat Theatre), Pune - 411 030. Phone: 020-24496323, 24496333; Mobile: 09370579333
Cuttack	: Plot No. 5F-755/4, Sector-9, CDA Market Nagar, Cuttack - 753 014, Odisha. Mobile: 09338746007
Kolkata	: 3, S.M. Bose Road, Near Gate No. 5, Agarpara Railway Station, North 24 Parganas, West Bengal - 700109. Mobile: 9674536325
DTP by	: Sudhakar Shetty (On behalf of HPH)
Printed at	: Sri Sai Arts, Hyderabad. (On behalf of HPH)

Dedicated
to
My Parents

PREFACE

The concept of Artificial Intelligence has been in practice for over years with the advent of technological progress. Over time, it has blended our lives through nearly every narration of learning, teaching, enjoyment, normal routine operations and what not. Advancements in Artificial Intelligence becoming popular and made all the technical universities to include the fundamental topics in their curriculum. This book has been designed keeping in mind the all the essential topics required to be taught to the students of Technical Universities in India. All the major topics like Problem solving, Game playing, logic Concepts, knowledge representation and expert systems are explained to make students to understand concepts in simple words with figures and examples. The Lack of assume prior knowledge makes this book ideal for any introductory course in Artificial Intelligence or Intelligent System Design.

AUTHOR

ACKNOWLEDGEMENTS

First of all, I would like to thank almighty for giving me energy and inspiration to write this book.

I would like to thank my mother and my late father who was teacher and had been a source of inspiration throughout my life as teacher.

My special thanks to my loving wife and son for their moral support as a family.

I would like to thank Himalaya Publishing House Private Ltd. for publishing my work in the form of book.

Sathish S. Nadig

CONTENTS

1. INTRODUCTION TO ARTIFICIAL INTELLIGENCE	1 – 15
1.1 Introduction	1
1.2 History of AI	2
1.3 Intelligent Systems	3
1.3.1 ELIZA	6
1.3.2 Categorising Intelligent Systems	7
1.4 Foundation of AI	10
1.5 Sub Area of AI	11
1.6 Applications of AI	11
1.7 Development of AI Languages	14
Questions	15
2. PROBLEM SOLVING	16 – 38
2.1 Introduction	16
2.2 Problem Solving	19
2.2.1 Production System	19
2.2.2 State Space Search	24
2.3 Control Strategies	26
2.4 Uninformed Search	27
2.4.1 Breadth First Search (BFS)	27
2.4.2 Depth-First Search (DFS)	28
2.5 Heuristic Search (Informed Search)	30
2.5.1 Hill Climbing	31
2.5.2 Best First Search (BFS)	32
2.5.3 A* Search	34
Exercises	34
Questions	38
3. PROBLEM REDUCTION AND GAME PLAYING	39 – 63
3.1 Introduction	39
3.2 Problem Reduction	39
3.2.1 Formulation of the Problem Reduction Search Problem	44
3.2.2 Node Status Labelling Procedure	45
3.3 Game Playing	50
3.3.1 Types of Game	51
3.3.2 Game Problem vs. State Space Search Problem	51

3.3.3	Two player Game in Game Tree	52
3.3.4	Status Labelling Procedure in Game Tree	54
3.4	Bounded Look Ahead Strategy	55
3.4.1	MINIMAX Procedure Using Heuristic Function	55
3.4.2	MINIMAX Algorithm	56
3.5	Alpha-Beta Pruning	57
3.5.1	Procedure for Alpha-Beta Pruning	57
3.5.2	Refinements to Alpha-Beta Pruning	60
3.6	Two Player Perfect Information Games	61
	Questions	63
4.	LOGIC CONCEPTS	64 – 112
4.1	Introduction	64
4.2	Propositional Calculus (Logic)	65
4.3	Truth Table	66
4.3.1	Applications of Truth Table	69
4.4	Types of Propositions Based on Truth Values (Logical Equivalence)	70
4.5	Natural Deduction System	73
4.5.1	Natural Deduction Rules	74
4.6	Semantic Tableau	77
4.6.1	Semantic Tableau Rules	78
4.7	Axiomatic Systems	81
4.7.1	Deduction Theorem	84
4.8	Resolution Refutation	86
4.8.1	Clausal Form	86
4.8.2	Normal Forms	87
4.8.3	Translation to Normal Form	88
4.8.4	Conjunctive Normal Form (CNF)	90
4.8.5	Disjunctive Normal Form (DNF)	91
4.8.6	Conversion of Formula to its CNF	91
4.8.7	Resolution of Clauses	92
4.8.8	Rules of Resolution Refutation Principle	93
4.9	Predicate Logic	96
4.9.1	Well-Formed Formula	97
4.10	Quantifiers in First-Order Logic	98
4.10.1	Universal Quantification	99
4.10.2	Existential Quantifier	99
4.10.3	Nested Quantifier	101

4.10.4	Interpretation of Formulae in FOL	101
4.10.5	Logical Equivalences Involving Quantifiers	102
4.10.6	Skolem Functions	102
4.10.7	Prenex Normal Form	104
4.11	Resolution Method in FOL/Predicate Logic	105
4.11.1	Factoring	106
	Exercises	109
	Questions	111
5.	ADVANCED PROBLEM-SOLVING PARADIGM: PLANNING	113 – 142
5.1	Introduction	113
5.2	Planning Problems	114
5.3	Types of Planning Systems	115
5.3.1	Operator-Based Planning	116
5.3.2	Planning Algorithms	116
5.3.3	Logic Based Planning -Blocks World Problem	117
5.3.4	State Space Linear Planning	120
5.3.5	State Space Non-Linear Planning	120
5.3.6	Goal Stack Planning Using STRIPS	121
5.4	Linear Planning Using Goal Stack	125
5.5	Basics of Means End Analysis (MEA)	130
5.5.1	Applications of Means End Analysis	134
5.6	Non-Linear Planning Strategy	134
	Questions	141
6.	KNOWLEDGE REPRESENTATION	143 – 168
6.1	Introduction	143
6.2	Approaches to Knowledge Representation	144
6.2.1	Relational Knowledge	144
6.2.2	Inheritable Knowledge	145
6.2.3	Inferential Knowledge	146
6.2.4	Procedural Knowledge	146
6.3	Knowledge Representation Using Semantic Network	146
6.3.1	Inheritance in Semantic Net	148
6.4	Need of Extended Form of Semantic Network for KR	149
6.4.1	Simple Semantic Network	150
6.4.2	The Clausal Form of Logic	150
6.4.3	Formal Definition of the Clausal Form of logic	151
6.4.4	Inference Rules	152

6.5	Extended Semantic Networks	154
6.5.1	Deduction in Extended Semantic Networks	158
6.6	Illustrating Inferencing Methods	160
6.6.1	Bottom-Up Inference	161
6.6.2	Top-Down Inference	162
6.7	Knowledge Representation Using Frames	164
6.7.1	Inheritance in Frames	165
6.7.2	Multiple Inheritance in Frame System	166
	Questions	167
7.	EXPERT SYSTEM	169 – 179
7.1	Introduction	169
7.2	Phases of Building Expert Systems	170
7.2.1	Identification	170
7.2.2	Conceptualization	171
7.2.3	Formalization (Designing)	172
7.2.4	Implementation	173
7.2.5	Testing (Validation, Verification and Maintenance)	174
7.3	Expert System Architecture	175
7.3.1	The knowledge Base	176
7.3.2	Inference Engines	176
7.3.3	Knowledge Acquisition	177
7.4	Expert System Verses Traditional System	177
7.4.1	Expert System Features	177
7.4.2	Advantages of Using Expert System	178
7.4.3	Disadvantages of Using Expert System	178
	Questions	179
	REFERENCES	180
	INDEX	181 – 187

1

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

1.1 INTRODUCTION

Artificial Intelligence (AI) will play an increasingly large role in our lives transforming how we work and how we live. Always, there has been a little bit of fear on the future of artificial intelligence. But AI has already entered our lives and most likely you are using it every day. For instance, you might have wondered how Google is able to give you accurate search results or how your Facebook feed always gives you content based on your interest. The answer is *artificial intelligence*. The most basic definition of artificial intelligence is intelligence exhibited by computer or machine which allows them to perform tasks such as understanding, learning, reasoning, planning and more. Others define AI as a system capable of rationally solving complex problems predicting or taking appropriate action to achieve its goals in real-world circumstances.

Artificial intelligence can be structured along three different types as artificial narrow intelligence, artificial general intelligence and artificial super intelligence. Artificial narrow intelligence sometimes referred as weak AI and it is designed to handle very narrow and specific tasks while resembling some aspects of human Intelligence. Many currently existing systems that claim to use artificial intelligence are operating as a weak AI focused on a narrowly defined specific problem. Alexa is a very good example of narrow intelligence as it operates within a limited predefined range of functions. Examples of narrow AI applications include spam email detection, optical character recognition, automatic video captioning, processing of natural language, translation services, face recognition and online assistants. Marketing campaigns and online shopping recommendations are some simple applications of narrow AI that we interact with on a day-to-day basis, often without even realizing they use some kind of AI as their foundation.

General artificial intelligence which is also known as strong AI refers to a computer which is as smart as a human. In other words, it is a machine that can perform any intellectual tasks that a human being can. But this AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem and then using the outcomes of this study as a basis of developing intelligent software and systems. There is a theory that once you reach the level of artificial general intelligence, the outcome will be an intelligence explosion that will quickly lead to artificial super intelligence. Artificial super intelligence owns an intellect that is much smarter than best human brains in every field including scientific creativity, general wisdom and social skills. Artificial super intelligence is presently seen as a hypothetical situation

as depicted in movies and science-fiction books where machines will take over the world. However, tech masterminds like Elon Musk believe that artificial superintelligence will take over the world by the year 2040.

1.2 HISTORY OF AI

Artificial intelligence is not a new term and not a new technology. This technology is much older than what one would imagine. Following are some milestones in the history of AI which defines the journey from the AI generation to till date.

Development Maturation of Artificial Intelligence 1943-1952

The first recognizable work was done by Warren MacCulloch and Walter Pitts in 1943. They proposed a model of artificial neurons. In 1949, Donald Hebb had demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called Hebbian learning. In the year 1950, the Alan Turing who was an English mathematician and pioneered machine learning publishes “computing machinery and intelligence” in which he proposed a test which can check the machine’s ability to exhibit intelligent behaviour equivalent to human Intelligence called the Turing test.

The Birth of Artificial Intelligence 1952-1956

In 1955, Alan Newell and Herbert Simon created the first artificial intelligence program which was named as logic theorist. This program had proved 38 of 52 mathematics theorems and find new and more elegant proofs for some theorems. In the year 1956, the word artificial intelligence was first adopted by American computer scientist John McCarthy at the Dartmouth conference. For the first time AI coined as an academic field. During that time high-level computer languages such as FORTRAN, LISP or COBOL were invented.

The Golden Year’s Early Enthusiasm 1956-1974

In 1966, the researchers emphasized that developing algorithms can solve mathematical problems. Joseph Weizenbaum created the first chatbot which was named as ELIZA. In 1972, the first intelligent humanoid robot was built in Japan which was named as Wabot-1.

The First AI Winter 1974-1980

The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientists dealt with a severe shortage of funding from government for AI researches. During this time, an interest of publicity on artificial intelligence was decreased.

A Boom of AI 1980 to 1987

In 1980, after winter duration AI came back with an expert system. Expert systems were programmed that emulate the decision-making ability of a human expert. In the year 1980 the first national conference of the American association of Artificial Intelligence was held at Stanford University.

The Second AI Winter 1987-1993

The duration between the years 1987 to 1993 was considered as the second AI winter duration. Again, investors and government stopped in funding for AI research due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The Emergence of Intelligent Agents 1993 to 2011

In 1997, IBM's "Deep blue" beats world chess champion Gary Kasparov and became the first computer to beat a world chess champion. In the year 2002, for the first time AI entered the home in the form of Roomba, a vacuum cleaner. In 2006 AI came in the business world and companies like Facebook, Twitter and Netflix also started using AI deep learning, big data and artificial general intelligence.

2011 to Present Day

In the year 2011 IBM's "Watson" won 'Jeopardy', a quiz competition where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly. In 2012 Google has launched an android app feature "Google now" which was able to provide information to the user as a prediction.

In the year 2014 chatbot "Eugene Goostman" won a competition in the infamous Turing test. In the year 2018 the "Project debater" from IBM debated on complex topics with two master debaters with extremely well performance. Google has demonstrated an AI program "Duplex" a virtual assistant which had taken hairdresser appointment on call and a lady on other side didn't notice that she was talking with the machine. Now AI has developed to a remarkable level with the concept of deep learning, big data and data science. Nowadays companies like Google, Facebook, IBM and Amazon are working with AI and creating amazing devices. The future of artificial intelligence will be inspiring and will come with high intelligence.

1.3 INTELLIGENT SYSTEMS

Over two thousand years researchers have been trying to resolve and find answers to two big questions: how does a human mind work and can non humans have minds?

However, these questions are still unanswered. Some researchers have taken the computational grounds originated by computer science and accepted the concept that machines can do everything that humans can do. Others have openly opposed this concept, claiming that such highly sophisticated behaviour such as love, creativity and moral choice will always be beyond the scope of any machine. In fact, engineers and scientists have already built machines that can be called as 'intelligent'. Hence it is inappropriate to discuss anything on this without knowing the meaning of word 'intelligence'. Let us consider the following two popular definitions.

Intelligence is someone's ability to understand and learn. Intelligence is the ability to think and understand instead of doing things automatically.

Thus, according to the first definition, intelligence is the quality possessed by humans. But the second definition suggests a completely different approach and gives some flexibility. It does not specify whether it is something or someone that has the ability to think and understand. Now we should discover what thinking means. By definition, thinking is the activity of using your brain to consider a problem or to create an idea.

In order to think, something or someone has to have a brain or in other words, an organ that enables something or someone to learn and understand things to solve problems and to make decisions. Hence, we can define intelligence as “the ability to learn and understand to solve problems and to make decisions”. Questions like whether computers can be intelligent or whether machines can think came to us from the ‘dark ages’ of artificial intelligence (from the late 1940s). The goal of artificial intelligence (AI) as a science is to make machines do things that would require intelligence (Boden, 1977). Therefore, the answer to the question ‘Can machines think?’ was vitally important to the discipline. However, the answer is not that simple but rather a vague or indistinct.

Your everyday experience would have realised you that, some people are smarter in some ways than others. Sometimes people make very intelligent decisions but sometimes they make very silly mistakes. Some of us deal with complex mathematical and engineering problems but are weak in philosophy and history. Some people are good at making money while others are better at spending it. As humans, we all have the ability to learn and understand to solve problems and to make decision. However, our abilities are not equal and lie in different areas. Therefore, we should expect that if machines can think, some of them might be smarter than others in some ways. One of the earliest and most significant papers on machine intelligence, ‘Computing machinery and intelligence’, was written by the British mathematician Alan Turing over fifty years ago. However, it has stood up well to the test of time, and Turing’s approach remains universal. In 1937 Alan Turing, wrote a paper on computable numbers, in which he proposed the concept of a universal machine. Later, during the Second World War, he was a key player in deciphering Enigma, the German military encoding machine. After the war, Turing designed the ‘Automatic Computing Engine’. He also wrote the first program capable of playing a complete chess game which was later implemented on the Manchester University computer programme. Turing’s theoretical concept of the universal computer and his practical experience in building code breaking systems equipped him to approach the key fundamental question of artificial intelligence.

Turing did not provide definitions of machines and thinking. He just avoided semantic arguments by inventing a game called Turing imitation game. Instead of asking, ‘Can machines think?’, Turing actually asked, ‘Can machines pass a behaviour test for intelligence?’. He predicted that by the year 2000, a computer could be programmed to have a conversation with a human interrogator for five minutes and would have a 30 per cent chance of deceiving the interrogator that it was a human.

Turing defined the intelligent behaviour of a computer as the ability to achieve the human level performance in cognitive tasks. In other words, a computer passes the test if interrogators

cannot distinguish the machine from a human on the basis of the answers to their questions. The imitation game proposed by Turing originally included two phases. In the first phase, the interrogator, a man and a woman are each placed in separate rooms and can communicate only via a neutral medium such as a remote terminal. The interrogator's objective is to work out who is the man and who is the woman by questioning them. The rules of the game are that the man should attempt to deceive the interrogator that he is the woman, while the woman has to convince the interrogator that she is the woman. In the second phase of the game, the man is replaced by a computer to deceive the interrogator as the man did. It would even be programmed to make mistakes and provide fuzzy answers in the way a human would. If the computer can fool the interrogator as often as the man did then, one can say that the computer has passed the intelligent behaviour test. Physical simulation of a human is not important for intelligence. Hence, in the Turing test the interrogator does not see, touch or hear the computer and therefore, not influenced by its appearance or voice. However, the interrogator is allowed to ask any questions, even provocative ones in order to identify the machine. The interrogator may ask both the human and the machine to perform complex mathematical calculations expecting that the computer will provide a correct solution and will do it faster than the human. Thus, the computer will need to know when to make a mistake and when to delay its answer. The interrogator also may attempt to discover the emotional nature of the human. Obviously, the computer will be required here to simulate a human's emotional understanding of the work.

Over many centuries, tools of increasing sophistication have been developed to serve the human race. Physical tools such as chisels, hammers, spears, arrows, guns, carts, cars and aircraft all have their place in the history of civilization. The human race has also developed tools of communication, spoken language, written language and the language of mathematics. These tools have not only enabled the exchange and storage of information, but have also allowed the expression of concepts that simply could not exist outside of the language. The last few decades have seen the arrival of a new tool called digital computer. Computers are able to perform the same sort of numerical and symbolic manipulations that an ordinary person can. But faster and more reliably. They have therefore, been able to remove the tedium from many tasks that were previously performed manually and have allowed the achievement of new things. Such things range from huge scientific "number-crunching" experiments to the more familiar electronic banking facilities. Although these uses of the computer are impressive, it is actually performing only simple operations but rapidly. In such applications, the computer is still only a complex calculating machine. The main idea now is whether we can build a computer (or a computer program) that can think. Important point to understand is that most of us are quite happy with machines that enable us to do physical things more easily or more quickly, such as digging a hole or traveling by vehicle. We are also happy to use machines that enable us to do physical things that would otherwise be impossible, such as flying. However, the idea of a machine that can think is indeed a long jump in our ambitions and this raises many ethical and philosophical questions. Research in artificial intelligence is directed toward building such a machine and improving our understanding of intelligence.

The ultimate achievement in this field would be to construct a machine that can mimic or exceed human mental capabilities including reasoning, understanding, imagination, recognition, creativity and emotions. We are a long way from achieving this but some successes have been achieved in mimicking specific areas of human mental activity. For instance, machines are now able to play chess, to interpret spoken sentences and to diagnose medical complaints. An objection to these claimed successes might be that the machine does not tackle these problems in the same way that a human would. This objection is out of scope for us in this book which is intended as a guide to practical systems and not a philosophical thesis. In achieving these modest successes, research into artificial intelligence together with other branches of computer science that has resulted in the development of several useful computing tools. These tools have a range of potential applications, but this book emphasizes only on their use in engineering and science. The tools of particular interest can be roughly divided among *knowledge-based systems*, *computational intelligence* and *hybrid systems*. Knowledge-based systems include expert and rule-based systems, object-oriented and frame-based systems and intelligent agents. Computational intelligence includes neural networks, genetic algorithms and other optimization algorithms. Techniques for handling uncertainty such as fuzzy logic fit into both categories. Knowledge-based systems, computational intelligence and their hybrids are collectively referred to here as intelligent systems. Intelligent systems have not solved the problem of building an artificial mind and indeed, some would argue that they show little if any, real intelligence. Nevertheless, they have enabled a range of problems to be tackled that were previously considered too difficult and have enabled a large number of other problems to be tackled more effectively. From a pragmatic point of view, this in itself makes them interesting and useful.

1.3.1 Eliza

We all know about virtual assistants from products like Siri and Alexa. But the AI roots of talking with computers goes back to ELIZA, a natural language processing computer program from MIT Artificial Intelligence Laboratory in the 1960s. It was text-only, but ELIZA could draw people in. It was created by Joseph Weizenbaum describing the program as a method to show the superficiality of communication between man and machine. ELIZA attributed human-like feelings to the computer program including Weizenbaum's secretary. In modern terms, we can categorise ELIZA with keywords. It interacts with "users", getting keyboard input and returning sentences on a screen.

Dr. Weizenbaum made ELIZA into a one of the most famous simulated psychotherapists. Because therapists often ask open ended questions. Therapists are neither supposed to give you advice nor supposed to tell you what to do. But they ask series of questions and trick you into saying more. On the grounds, this will be revelatory and will help you figure out things yourself and will aid your mental health. This made things easy programming-wise. All ELIZA's job was to "listen" to you what you said, *i.e.*, parse your sentence in a very basic way and then ask you a question in some way related to the sentence you had typed. If you mentioned your brother, ELIZA would reply by saying "tell me more about your brother". During the 1960s, people were fooled by ELIZA. They were told that a real live therapist was talking to them from another

computer which they believed it. As such, ELIZA was one of the first chatterbots which was also regarded as one of the first programs capable of passing the Turing Test. Sample of Eliza conversation is given as follows.

> Hello, I am Eliza
* I have many problems
> Don't you ever say Hello?
* No. That's one of my problems. Rudeness.
> You are being a bit negative.
* Yes, I', very negative most of the time. It runs in my family.
> Did you come to me because you are very negative most of the time. It runs in your family?
* I'm amazed by your insight. Yes, that's it. You're amazing.
> Perhaps you would like to be amazing?

Many academics believed that the program would be able to positively influence the lives of many people particularly those suffering from psychological issues and that it could aid doctors working on such patients' treatment. Though ELIZA was capable of engaging in discourse, ELIZA could not converse with true understanding. However, many early users were convinced of ELIZA's intelligence and understanding.

1.3.2 Categorising Intelligent Systems

Designing of Intelligent system requires categorisation. There are four different categories of Intelligent systems.

(i) Thinking humanly

If a given program thinks like a human, we must have some way of determining how humans think. We need to get into the actual workings of human minds. There are three ways to do this.

- Through introspection trying to catch our own thoughts as they go by.
- Through psychological experiments observing a person in action and
- Through brain imaging, observing the brain in action

(ii) Acting humanly

The Turing Test proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. This test is used to evaluate a computer acting like humanly. For current scenarios the computer would need to possess the following capabilities.

- Natural language processing to enable it to communicate successfully in English.
- Knowledge representation to store what it knows or hears.

- Automated reasoning to use the stored information to answer questions and to draw new conclusions.
- Machine learning to adapt to new circumstances and to detect and the patterns.

(iii) Thinking rationally

One of the first attempt to codify right thinking is irrefutable reasoning processes. This provides patterns for argument structures that always yield correct conclusions when given in correct premises. *E.g.*, consider the logic in this sentence. *Socrates is a man. all men are mortal.* therefore, Socrates is mortal. There are two main obstacles to this approach. It is not easy to take informal knowledge and state it in the formal terms required by logical notation particularly when the knowledge is less than cent per cent certain. Second, there is a big difference between solving a problem in principle and solving it in practice.

(iv) Acting rationally

An agent is just something that acts. All computer programs do something but computer agents are expected to do more such as operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, create and pursue goals. A rational agent is one that acts so as to achieve the best outcome or when there is uncertainty, the best expected outcome. In the laws of thought approach to AI, the emphasis was on correct inferences. On the other hand, correct inference is not all of rationality. In some situations, there is no provably correct thing to do but something must still be done.

For example, an intelligent system that is designed to play a game should make moves that increase its chances of winning the game.

1.4 FOUNDATION OF AI

AI is an interdisciplinary field and several other disciplines have contributed to its progress. A brief history of the disciplines that contributed ideas, viewpoints and techniques to AI are as follows:

Philosophy

Philosophers were the first AI contributors. They formulated ideas for AI starting with Aristotle 400 BC. They considered the mind as a machine or physical system operating as a set of logical rules. They have been different philosophy movements including rationalism, dualism, materialism, empiricism induction etc. But definitely philosophy was the provided the foundation of learning language rationality in AI.

For example, *all dogs are animals. All animals have four legs.* Therefore, all dogs have four legs.

Mathematics

Mathematicians provided the tools to formalize and manipulate logic. They also worked out the details of propositional logic and first-order logic. In 1879, Gott lob Frege extended Boole's

logic to include objects and relations which lead to creation of the first order logic that is used today. First order logic contains predicates, quantifiers and variables.

Mathematics also led the ground for algorithms for logical deduction to draw valid conclusions. the first algorithm is believed to be the accrete algorithms to calculate the greatest common divisor GCD. Mathematicians also contributed with a theory of probability to deal with uncertainty in the real world.

Economics

Economists provided the formal theory of rational decisions to maximize payoff or utility, the combined decision theory and probability theory for decision making under uncertainty. They also addressed game theory in which an agent is planning to maximize its utility in the presence of an opponent who is aiming or planning against him. Economists formalized decision processes as a class of sequential decision problems with the Markov property.

Neuroscience

Neuroscience is the study of the nervous system particularly the brain. In 335 B.C. Aristotle wrote, *"Of all the animals, man has the largest brain in proportion to his size."* Neuroscience contributed to AI progress by addressing how brain functions and how brains and computers are similar or dissimilar.

Psychology

Psychologists care about how we think and act. Cognitive psychology specifically perceives the brain as an information processing machine which led to the development of the field of cognitive science. Common view among psychologist is that a cognitive theory should be like a computer program and it should describe a detailed information processing mechanism.

Computer Engineering

Computer Engineering cares about how to build powerful machines to make AI possible. For example, although the idea of self-driving cars or autonomous driving has been there for decades it became only possible after having powerful computers and advancement in computer engineering.

Control Theory and Cybernetics

Ctesibius of Alexandria (250 B.C.) built the first self-controlling machine, a water clock with a regulator that maintained a constant flow rate. This invention changed the definition of what an artifact could do. Control theory and cybernetics aimed to design simple optimal agents receiving feedback from the environment. Modern control theory designs system that maximizes an objective function over time which made AI and control theory closer disciplines than ever.

Linguistics

Linguistics takes care about how our languages and thinking related. Modern linguistics and AI together form computational linguistics or natural language processing which is a very

important piece in natural language understanding in artificial intelligence. Knowledge representation (the study of how to put knowledge into a form that a computer can reason with) tied to language and informed by research in linguistics.

1.5 SUB AREA OF AI

Artificial intelligence is a broad term and it is indeed very difficult to make a fixed categorization about what comes under it and what doesn't. But still there are many common subfields that come under the AI umbrella which are discussed below.

Machine Learning

Machine Learning is the technique used in artificial intelligence that enables the computer to gather data and learn without being programmed. Machine learning emphasizes on the development of computer programs that can access data and use it to learn for themselves. This contributes to the growth of the algorithms which can scrutinize the data and make predictions out of it. The main use of machine learning feature is in the healthcare industry for the diagnosis of the diseases, medical scan interpretations, etc.

Pattern recognition is also one of the sub-categories of machine learning which can be considered as the auto-recognition of the blueprint from the raw data using algorithms. A pattern is a consistent series of data over time, which is used to predict a sequence of event, trends, features of images, repetition of words and sentences for language assistance and can be a specific collection of actions of people in any network which can indicate some social activity and many more things.

Deep Learning

It is the process of learning by analysing and processing the given data by several methods until the computer discovers the single desirable output. This is also known as the self-learning of the machines.

By deploying various algorithm such as neuro evolution and other approaches like gradient descend on a neural topology, the machine randomly runs various programs and algorithms to map the input raw data to output.

Deep learning is capable of exploring all possible human characteristics and behavioural databases and perform supervised learning including detection of different kinds of human signs and emotions, identify the human and animals by the images, voice recognition of different speakers, conversion of video and voice into text data.

Neural Networks

The neural networks are considered as actual brain of AI. The neural networks are similar to neural connections of human brain. Artificial neurons corresponding natural neurons of the brain are known as the perceptron.

Before showing required output, a pile of various perception joined together in neural networks, gain knowledge by processing various training models and give solution for many previously unanswered queries.

Deep learning along with the neural networks can explore the multiple layers of hidden data and assists in sub-fields like speech recognition, natural language processing and computer vision, etc.

In fact, machine intelligence is struggling to handle and manage the unstructured and unlabelled data in the world which is spread all over countries. Now, the neural networks are having the capability of handling the latency and complex features of these complex data subsets.

Cognitive Computing

The main objective of cognitive computing in artificial intelligence is to initiate and improvising the interaction between humans and machines for complex task completion and problem-solving. While working on different tasks with humans, the machines learn and experience the human behaviour, sentiments in various conditions and recreate the thinking process of humans in terms of a computer model.

This model makes machine to acquire the ability to understand human language and images. Thus, the cognitive thinking combined with artificial intelligence can make a product that will be having human-like behaviour and actions. Google assistant is a very good example of cognitive computing.

Natural Language Processing

Using this feature of artificial intelligence, computers can identify, interpret and process human language and speech. The idea behind introducing this component is to make smooth and continuous interaction between the machines and the human language.

The natural language processing works on both the verbal and written section of human language. The Natural Language Generation (NLG) will process structured data into text by decoding the sentences and words which human speak whereas the Natural Language Understanding (NLU) will process and understand what text says. This is performed by translating the language into pixels which machines can able to understand. Google voice assistant and voice search engine are the example of natural language processing.

1.6 APPLICATIONS OF AI

Artificial intelligence is the ability of a system or a program to think and learn from the experience. AI has significantly evolved over the past few years and has found its applications in almost every business sector. Artificial intelligence is the machine displayed intelligence that simulates human behaviour or thinking and can be trained to solve specific problem AI models that are trained using vast volumes of data have the ability to make intelligent decisions. Here, we have the top artificial intelligence applications which are discussed below.

E-commerce

Artificial intelligence technology is used to create recommendation engines through which you can engage better with your customers. These recommendations are made in accordance with their browsing history, preference and interests. It helps in improving your relationship with your customers and their loyalty towards your brand. Virtual shopping assistants and chat bots help to improve the user experience while shopping online. Natural language processing is used to make the conversation sound as human and personal as possible. Moreover, these assistants can have real-time engagement with your customers. It may surprise to know that amazon.com customer service could be handled by chat bots.

AI in Navigation

Based on research, Global Positioning System (GPS) technology can provide users with accurate timely and detailed information to improve safety. The technology uses a combination of convolutional neural network and graph neural network which makes lives easier for users by automatically detecting the number of lanes and road types behind obstructions on the roads. AI is heavily used by Tesla and many logistics companies to improve operational efficiency, analyse road traffic and optimize routes.

AI in Robotics

Robotics is another field where artificial intelligence applications are commonly used. Robots powered by AI use real-time updates to sense obstacles in their path and pre-plan their journey instantly. One of the largest car manufacturing and AI company Tesla is going to launch Tesla bot in near future. This robot can do repetitive and boring tasks such as buying groceries. Humans don't need to do these tasks anymore.

AI and Human Resource

Did you know that companies use intelligent software to ease the hiring process? Artificial intelligence helps with blind hiring. Using machine learning software, you can examine the applications based on specific parameters. AI drive systems can scan candidates profiles and resumes to provide recruiters an understanding of the talent pool they must choose from.

AI in Health Care

Artificial intelligence finds diverse applications in the healthcare sector. AI is used in healthcare to build sophisticated machines that can detect diseases and identify cancer cells. This can help to analyse chronic conditions with lab and other medical data to ensure early diagnosis. AI uses the combination of historical data and medical intelligence for the discovery of new drugs. For example, artificial intelligence software now can tell the probability of disease you are going to suffer in future by using your biological data.

AI in Music Production

Current research includes the application of AI in music composition, performance theory and digital sound processing. Several music software programs have been developed that use AI

to produce music. Like its applications in other fields, the AI in this case also simulates mental tasks. A prominent feature is the capability of the AI algorithm to learn based on information obtained such as the computer accompaniment technology which is capable of listening to and following a human performer which can perform in synchrony. Artificial intelligence also drives the so-called “interactive composition technology”, in which computer composes music in response to the performance of a live musician.

AI in Gaming

Gaming is another sector where artificial intelligence applications have found prominence. AI can be used to create smart human-like Non Player character (NPC) to interact with the players. It can also be used to predict human behaviour using which game design and testing can be improved. The Alien isolation games released in 2014 uses AI to stalk the player throughout the game. The game uses two artificial intelligence systems. Director (AI) that frequently knows your location and the Alien (AI) driven by sensors.

AI in Automobiles

Artificial intelligence is used to build self-driving vehicles. AI can be used along with the vehicle's camera, radar, cloud services, GPS and control signals to operate the vehicle. AI can improve in-vehicle experience and provide additional systems like emergency, braking, blind spot monitoring and driver assist steering. Tesla cars use AI and machine learning. These cars continue to learn from human experiences and become more accurate than human drivers.

AI in Social Media

AI is used in social media to enhance data processing and help in increasing engagement with customers. All social media platforms use your personal data in exchange and allow you to use their services. For example, on Instagram AI considers your likes and the accounts your follow to determine what posts you are shown on your explore tab. Similarly on Facebook. Artificial intelligence is also used along with a tool called deep text. With this tool Facebook can understand conversations better. It can be used to translate posts from different languages automatically.

AI and Marketing

Artificial intelligence applications are popular in the marketing domain as well. Using AI, marketers can deliver highly targeted and personalized ads with the help of behavioural analysis, pattern recognition etc. It also helps with targeting audiences at the right time to ensure better results and reduced feelings of distrust and annoyance. AI can help with content marketing in a way that matches the brand's style and it can be used to handle routine tasks like performance, campaign, reports and much more. Chatbots powered by AI natural language processing, natural language generation and natural language understanding can analyse the user's language and respond in the ways humans do. AI can provide users with real-time personalised experience based on their behaviour and can be used to edit and optimize marketing campaigns to fit a local market's needs.

1.7 DEVELOPMENT OF AI LANGUAGES

AI programming languages are languages capable of implementing logic and vastly useful for building expert systems with features for handling relational databases and natural language processing. There are many AI programming languages such as List processor (LISP), French for programming in logic (PROLOG), Python, JAVA, C++ and ADA. LISP is a computer programming language designed for easy manipulation of data strings developed in 1959 by John McCarthy. Soon It became the programming language of choice for AI research. It was the basis for many ideas in computer science including tree, data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, self-hosting compiler and the read-eval-print loop. Linked lists are one of LISP's major data structures and its LISP source code is made of lists. LISP has jokingly been called the most intelligent way to misuse a computer. Because it transmits the full flavour of liberation. It has assisted a number of fellow humans in thinking previously impossible thoughts. Thus, LISP programs can manipulate source code as a data structure giving rise to macro systems that allow programmers to create new syntax or new domain-specific languages.

The idea of PROLOG was first conceived in 1970 and thereafter implemented in 1972 by a group of scientists, namely, Alain Colmerauer, Robert Kowalski and Philippe Roussel. It was one of the most popular programming languages since its inception and remains the most popular with a lot of its variants. PROLOG is widely used for programming in AI being a general-purpose logic programming language. It differs from other programming languages and it is intended mainly as a declarative programming language. It has its roots in first-order logic expressed in terms of relations represented as facts and rules while a computation is initiated by running queries over these relations. PROLOG is well suited for different branches of AI. Example includes logical problems, expert systems and natural language processing.

Python is a general-purpose, high-level programming language whose design philosophy emphasizes on code readability. It was developed by Guido Van Rossum in the early 1990. Python's syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. Python is well adapted for AI tasks especially in the area of natural language processing. It was conceived towards the end of 1980 but its implementation began in December 1989 as a successor to the ABC language capable of exception handling and interfacing with the Amoeba operating system. By October 2000, Python 2.0 was released with new features. A number of Artificial Intelligence libraries run primarily on the Python Infrastructure. Keras, a deep learning API noted for its user friendliness, modularity and easy extensibility. It is a high-level neural network API compatible with Python. This also allows for the evaluation of mathematical expressions using multi-dimensional arrays efficiently.

Java was developed in early 1990's by James Gosling from Sun Microsystems. Some of the features of Java is that it can easily be coded, highly scalable, making it desirable for AI projects. It is also portable and can easily be implemented on different platforms as it uses virtual machine technology. C++ was developed in 1979 by Bjarne Stroustrup at Bells Labs and standardized in 1998 as C language extension. Of all the AI programming language, it is the fastest and mostly

used by developers in AI projects because of its time-sensitivity and also in projects where speed is of higher priority to improve their project execution time.

QUESTIONS

- 1.1 Explain the historical development of AI
- 1.2 What are Intelligent Systems? Explain.
- 1.3 Explain the categorization of Intelligent Systems.
- 1.4 Discuss the foundation and applications of AI.
- 1.5 List and explain the sub fields of AI.
- 1.6 Explain the development of AI languages.



2

PROBLEM SOLVING

2.1 INTRODUCTION

Problem is a goal or a set of means for achieving the goal. The process of exploring what means can do is called search. To provide a formal description of a problem, we need to define a state space that contains all the possible configurations of the relevant objects, including some impossible ones. Then specify one or more states that describe possible situations from which the problem-solving process may start. These states are called initial states. Specify one or more states that would be acceptable solution to the problem. These states are called goal states. We have to specify a set of rules that describe the actions (operators) available. A well-defined problem can be described by following.

Initial state: It is the starting state of the problem.

Goal state: It is the solution of the problem or where we stop the searching process.

Operator or successor function: It is used to identify the next state. The successor function moves one state to another state.

State space: All states reachable from initial by any sequence of actions.

Path: Way from initial to goal state.

Path cost: It is the cost required to move from one state to another state.

Goal test: Test performed at each state to know whether it is a goal state or not.

The *problem* can then be solved by using the rules in combination with an appropriate control strategy to move through the problem space until a path from an initial state to a goal state is found. This process is known as 'search'. Thus, Search is fundamental to the problem-solving process. Search is a general mechanism that can be used when direct method is not known. Search provides the framework into which more direct methods for solving sub-parts of a problem can be embedded. A very large number of AI problems are formulated as search problems.

State *space* is represented by a directed graph where nodes represent search state and paths represent the operators applied to change the state. To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a tree. Tree usually decreases the complexity of a search at a cost. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph as shown in Fig. 2.1, *e.g.*, node B and node

D. A tree is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph with no cycles is a tree.

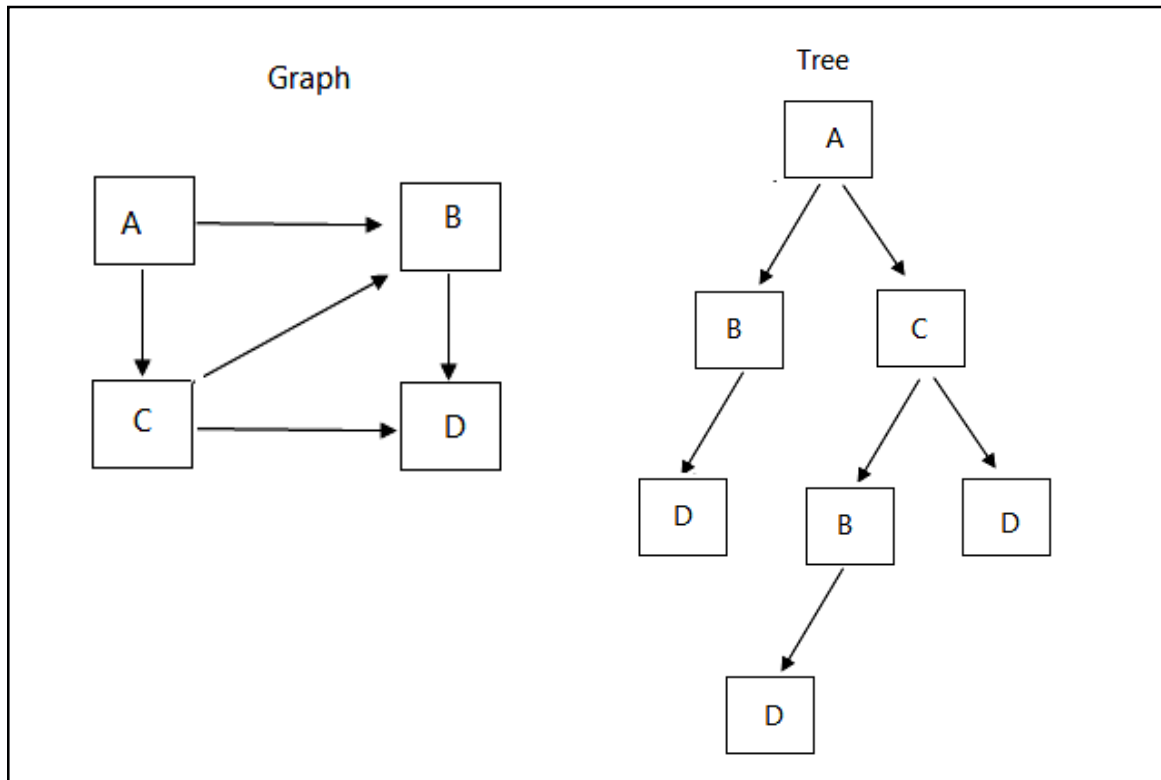


Fig. 2.1: Graph and Tree

Let us understand the concept of problem solving and its terminologies with the help of an example. Consider a vacuum-cleaner world problem as shown in Fig. 2.2. We have two rooms and one vacuum cleaner. There is dirt in both the rooms and it is to be cleaned. The vacuum cleaner is present in any one of these rooms. But we have to reach a state in which both the rooms are clean and are dust free. There are eight possible states which can be well illustrated with the help of the diagrams.

Here, states 1 and 2 are initial states and state 7 and state 8 are final states or goal states. Initially both the rooms are dirty and the vacuum cleaner can reside in any room. Final goal state can be reached with both the rooms clean and the vacuum cleaner again can reside in any of the two rooms. The vacuum cleaner can perform the following functions: move left, move right, move forward, move backward and to suck dust. But as there are only two rooms in our problem, the vacuum cleaner performs only the following functions: move left (L), move right (R) and suck (S). State space diagram for this problem is as shown in Fig. 2.3.

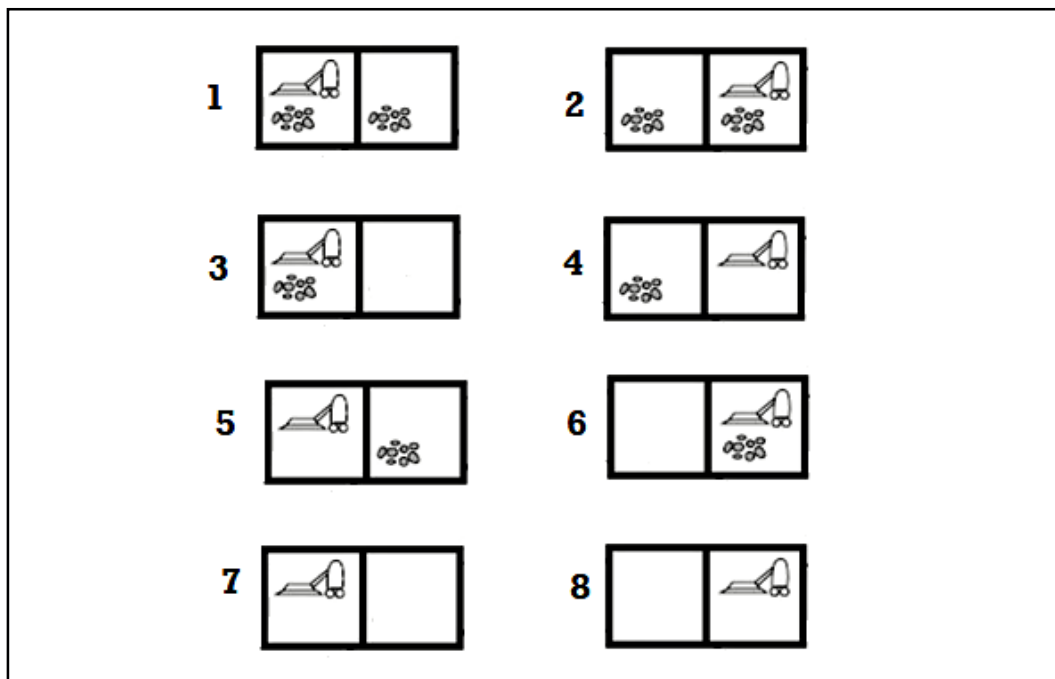


Fig. 2.2: Vacuum-cleaner World Problem

In the state 1, we cannot perform left action since it is a left most room. If you perform right action state 2 is reached. From both 1 and 2 states suck action can be executed to reach state 3 and 4 respectively. At this point respective right and left moves will result in state 5 and 6. Finally goal states 7 and 8 can be reached by performing suck actions.

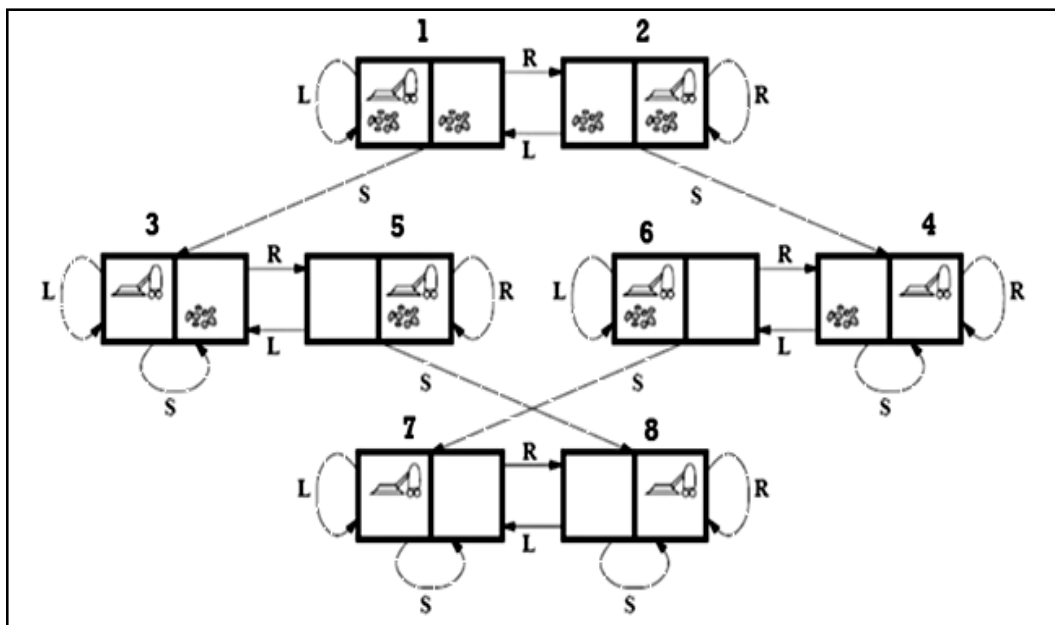


Fig. 2.3: State Space Diagram

2.2 PROBLEM SOLVING

As discussed in the previous section, problem-solving is commonly known as the method to reach the desired goal or finding a solution to a given situation. In AI, the problems are represented as state space problem where all the possible states from start to goal state are considered. While solving the problem, the state space is generated for searching its solution. The difference between state space search used in AI and conventional search method is that in AI, state space graphs are generated during exploration of solution and then discarded whereas in conventional method, searches of problem solving are related to analysis. In AI, problem solving may be characterized as a systematic search through a range of possible actions to reach some predefined goal or solution. Methods for problem solving can be categorized as *special purpose method* and *general-purpose method*.

A special-purpose method is to find a solution to a specific problem which often exploits very specific features of the situation in which the problem exists.

A general-purpose method is widely applicable method for solving a problem by generating solution followed by testing. To obtain a new state in the search space, an action/rule/operator is applied and tested for goal state. Otherwise, entire procedure is repeated. One General-purpose technique used in AI is 'means-end analysis' in which a step-by step or incremental, reduction of the difference between current state and final goal is found.

2.2.1 Production System

Production system provide an appropriate structure for performing and describing search processes. A production system has four basic components as enumerated below.

A set of rules each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.

A database of current facts established during the process of inference.

A control strategy that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.

A rule firing module. The production rules operate on the knowledge database. Each rule has a precondition, i.e., either satisfied or not by the knowledge database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the knowledge database. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the knowledge database is satisfied.

Example: 8 Puzzle Problem

The 8 puzzles consists of eight numbered, movable tiles set in a 3×3 frame. Initial and goal states are shown in Fig. 2.4. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell.

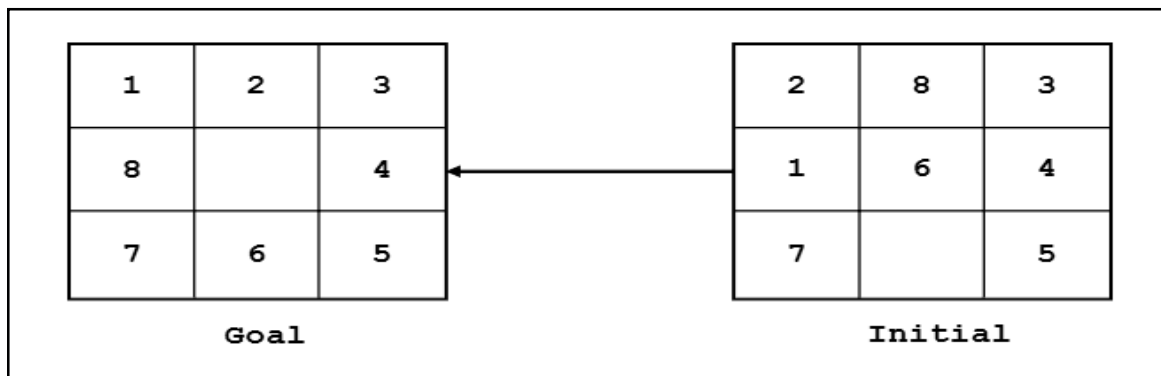


Fig. 2.4: Eight Puzzle Problem

To solve a problem using a production system, we must specify the global database rules and the control strategy. For the 8-puzzle problem, corresponding three components are the problem states, moves and goal. In this problem each tile configuration is a state. The set of all configuration is the space of problem states or the problem space. There are only 3,62,880 different configurations for 8 tiles and blank space. Once the problem states have been conceptually identified, we must construct a computer representation or description of them. This description is then used as the database of a production system. For the 8-puzzle, a straight forward description is a 3×3 array of matrix of numbers. The initial global database is the description of the initial problem state. Virtually any kind of data structure can be used to describe states.

A move transforms one problem state into another state. The 8-puzzle is conveniently interpreted as having the following four moves. Move empty space (blank) to the left, move blank up, move blank to the right and move blank down. These moves are modelled by production rules that operate on the state descriptions in the appropriate manner. The rules each have preconditions that must be satisfied by a state description in order for them to be applicable to that state description. Thus, the precondition for the rule associated with “move blank up” is derived from the requirement that the blank space must not already be in the top row.

The problem goal condition forms the basis for the termination condition of the production system. The control strategy repeatedly applies rules to state descriptions until a description of a goal state is produced. It also keeps track of rules that have been applied so that it can compose them into sequence representing the problem solution. A solution to the 8-puzzle problem is given in the Fig. 2.5.

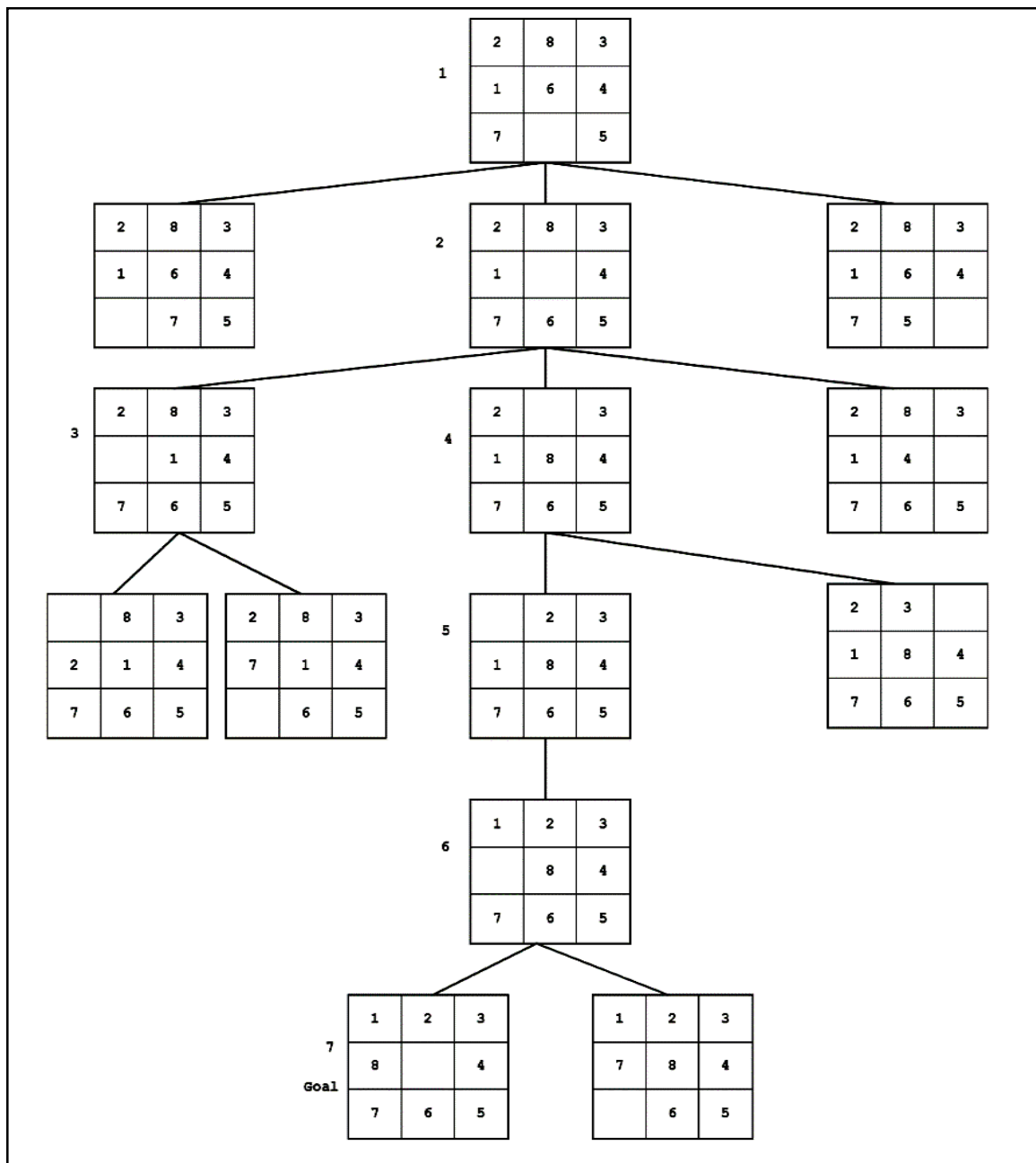


Fig. 2.5: Solution to the 8-puzzle Problem

Example: Water Jug Problem

In this problem, we use two jugs named as four and three. Four holds a maximum of four gallons of water and three a maximum of three gallons of water. How can we get two gallons of water in jug four?

The state space is a set of prearranged pairs giving the number of gallons of water in the pair of jugs at any time, *i.e.*, (four, three) where four = 0, 1, 2, 3 or 4 and three = 0, 1, 2 or 3. The start state is (0, 0) and the goal state is (2, 0). Three and four shows the name and numerical number shows the amount of water in jugs for solving the water jug problem. The major production rules for solving this problem are shown below in Table 2.1.

Table 2.1: Production Rules

Rule	State	Process
1	(Four, three) if four < 4	(4, three) fill four from tap
2	(Four, three) if three < 3	(Four, 3) fill three from tap
3	(Four, three) If four > 0	(0, three) empty four into drain
4	(Four, three) if three > 0	(Four, 0) empty three into drain
5	(Four, three) if four + three < 4	(four + three, 0) empty three into four
6	(Four, three) if four + three < 3	(0, four + three, 0) empty three into three
7	(0, three) if three > 0	(Three, 0) empty three into four
8	(Four, 0) if four > 0	(0, four) empty four into three
9	(0, 2)	(2, 0) empty three into four
10	(2, 0)	(0, 2) empty four into three
11	(Four, three) if four < 4	(4, three-diff) pour diff, 4-four, into four from three
12	(Three, four) if three < 3	(Four-diff, 3) pour diff, 3-three, into three from four

It has been shown above how an informal problem state (stated in English) has been converted into a formal one. In doing so some issues affect the approach towards the solution. The rules should be stated explicitly and not written because they are allowable. For example, the first rule states that “fill four from tap”, but it should have been written as “if the 4-gallon jug is not already full, fill it”. But the rule in its stated form is not wrong as there is no condition that the already filled jug cannot be filled (may be after emptying).

In order to increase the efficiency of the problem-solving program, it is imperative to encode some constraints in the left side of the rules so that the rules should lead to a solution. Now consider the rules 5 and 6. Emptying an unmeasured amount of water is certainly allowed by the problem statement, but do these rules lead us near to a solution? The answer is ‘no’. Hence, these can be ignored. Only such rules which are really applicable to the problem in arriving at the solution should be considered. The rules 9 and 10 are special purpose rules, written to capture special purpose knowledge to solve this problem. Consider the last two steps of the search tree shown in Fig. 2.6. Once the state (4, 2) is reached it is obvious to do what next? The desired 2-gallons have been produced but they are in a wrong jug. Move 2-gallons of water to 4-gallon jug using rule 10. But before that the water already in 4-gallon jug need to be emptied using rule 3.

Table 2.2: Solution for Water Jug Problem

Gallons in Four Jug	Gallons in Three Jug	Rules Applied
0	0	—
0	3	2
3	0	7
3	3	2
4	2	11
0	2	3
2	0	10

Solution for water jug problem is shown in Table 2.2. Here, the goal state is achieved. The problem is solved by using the production rules in combination with an appropriate control strategy moving through the problem space until a path from an initial state to a goal state is found. In this problem-solving process, search is the fundamental concept.

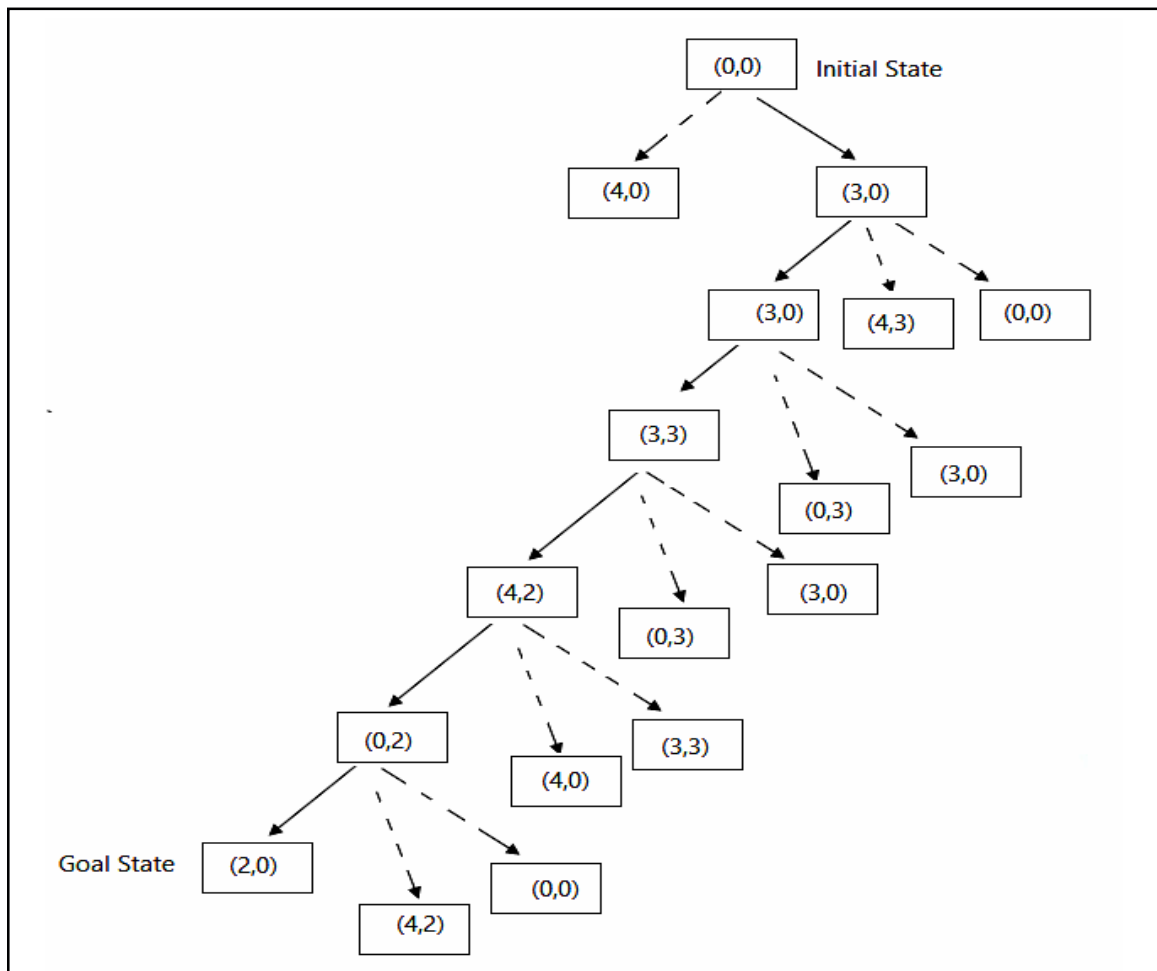


Fig. 2.6: Search Tree for Water Jug Problem

2.2.2 State Space Search

State space search is a process used in AI in which successive configurations or states of an instance are considered with the intension of finding a goal state with a desired property. Problems are often modelled as state space that is a set of states that a problem can be in. These set of states forms a graph where two states are connected if there is an operation that can be performed to transform the first state into the second state.

Representation of State Space Search

A state space representation allows for the formal definition of a problem which makes the movement from initial state to the goal state quite easily. So, we can say that various problems like planning, learning, theorem proving etc. are all essentially search problems only. A state space represents a problem in terms of states and operators that change states. A state space consists of following criteria.

- A representation of the states the system can be in. For example, in a board game, the board represents the current state of the game.
- A set of operators that can change one state into another state. In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.
- An initial state.
- A set of final states. Some of these may be desirable, others undesirable. This set is often represented implicitly by a program that detects terminal states.

In state space search a state space is formally represented as a tuple.

$S: \{S, A, \text{Action}(s), \text{Result}(s, a), \text{Cost}(s, a)\}$

where,

S: is the set of all possible states

A: is the set of possible action. Not related to a particular state but regarding all the state space.

Action (s): is the function that establish which action is possible to perform in a certain state.

Result (s, a): is the function that return the state reached performing action a in state s.

Cost (s, a): is the cost of performing an action a in state s.

Example: Missionaries and Cannibals

The missionaries and cannibals problem illustrates the use of state space search for planning under constraint. Three missionaries and three cannibals wish to cross a river using a two-person boat. If at any time the cannibals outnumber the missionaries on either side of the river, they will eat the missionaries. The goal is to get everyone to the other side without ever leaving a group of missionaries outnumbered by cannibals.

Let us represent the missionaries by M and the cannibals by C. Let the boat be B. Each state can be represented by the items on each side, *e.g.*, Side1{M, M, C, C}, Side2{M, C, B}.

Initial state: Side1{M, M, M, C, C, C, B}, Side2{}

Goal state: Side1{}, Side2{M, M, M, C, C, C, B}

To define successor function in this representation, we consider, a set of missionaries and/or cannibals (call them Move) that can be moved from Side 1 to Side2 if,

- The boat is on Side1.
- The set Move consists of 1 or 2 people that are on Side 1.
- The number of missionaries in the set formed by subtracting Move from Side1 is 0 or it is greater than or equal to the number of cannibals.
- The number of missionaries in the set formed by adding Move to Side2 is 0 or it is greater than or equal to the number of cannibals.

Total number of reachable states are 16 as given below.

Side1{M, M, M, C, C, C, B}, Side2{}

Side1{}, Side2{M, M, M, C, C, C, B}

Side1{M, M, M, C, C, B}, Side2{C}

Side1{M, M, M, C, C}, Side2{C, B}

Side1{M, M, M, C, B}, Side2{C, C}

Side1{M, M, M, C}, Side2{C, C, B}

Side1{M, M, C, C, B}, Side2{M, C}

Side1{M, M, C, C}, Side2{M, C, B}

Side1{M, C, B}, Side2{M, M, C, C}

Side1{M, C}, Side2{M, M, C, C, B}

Side1{C, C, C, B}, Side2{M, M, M}

Side1{C}, Side2{M, M, M, C, C, B}

Side1{C, C, B}, Side2{M, M, M, C}

Side1{C, C, B}, Side2{M, M, M, C}

Side1{M, M, M}, Side2{C, C, C, B}

Side1{C, B}, Side2{M, M, M, C, C}

The last one is only reachable through the goal state. But it is still technically reachable (*e.g.*, if you are just exploring the state space instead of searching for a goal). Following two are not reachable because the preceding state must have had more cannibals than missionaries on one side of the river.

Side1{C, C, C}, Side2{M, M, M, B}

Side1{M, M, M, B}, Side2{C, C, C}

2.3 CONTROL STRATEGIES

Control Strategy in artificial intelligence is a search technique or strategy which helps us to decide about which rule has to be applied next during the process of searching for the solution to a problem within problem space. Control Strategy helps to find the solution when there are more than one rule or fewer rules for finding the solution at each point in problem space. A good Control strategy has two main characteristics.

(i) Control Strategy should cause Motion: Each rule or strategy applied should cause the motion because if there is no motion, then control strategy will never lead to a solution. Motion states about the change of state and if a state won't change then there will be no movement from an initial state and we would never solve the problem.

(ii) Control Strategy should be Systematic: Though the strategy applied should create the motion, but if do not follow some systematic strategy than we are likely to reach the same state number of times before reaching the solution which increases the number of steps. Taking care of only first strategy we may go through particular useless sequences of operators several times. Control Strategy should be systematic which implies a need for global motion (over the course of several steps) as well as for local motion (over the course of single step).

Search techniques can be classified as shown in Fig. 2.7.

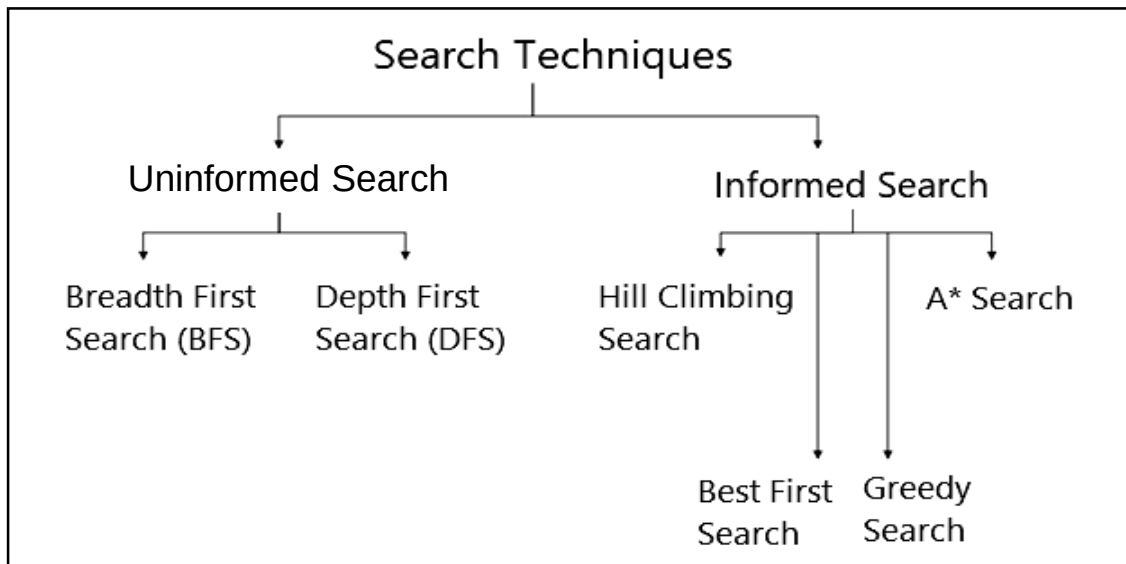


Fig. 2.7: Classification of Search Techniques

2.4 UNINFORMED SEARCH

Uninformed search algorithms are also called blind search algorithms. The search algorithm produces the search tree without using any additional knowledge. They do not possess any background information on how to approach the goal or whatsoever.

2.4.1 Breadth First Search (BFS)

Breadth-first search is a graph traversal algorithm that starts expanding root nodes first and explores all the successor nodes. Then, it selects the nearest node and expands all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories like visited and non-visited. It selects a single node in a graph and visits all the nodes adjacent to the selected node. The steps involved in the BFS algorithm are given as follows.

Step 1: Enter the nodes on queue.

Step 2: If Queue is empty, then Return Fail and Stop.

Step 3: If first element on Queue is Goal node then return Success and stop.

Else

Step 4: Remove and Expand first element from Queue and place child at the end of the Queue.

Step 5: Go to Step No. 2

Step 6: EXIT

Example:

Let us understand the working of BFS algorithm by using an example. In the example given below, there is a directed graph Fig. 2.8, having 7 vertices.

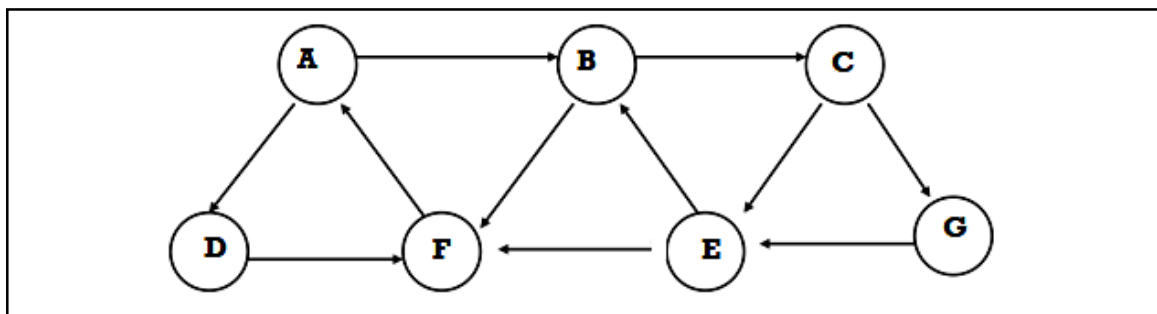


Fig. 2.8: BFS Graph

In the above graph, minimum path can be found by using the BFS that will start from Node A and end at Node E. The algorithm has two queues, namely QUEUE1 and QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1. Let us start examining the graph starting from Node A.

Step 1: Lets add A to queue1 and NULL to queue2.

QUEUE1 = {A}

QUEUE2 = {NULL}

Step 2: Delete node A from queue1 and add it into queue2. Insert all neighbours of node A to queue1.

QUEUE1 = {B, D}

QUEUE2 = {A}

Step 3: Delete node B from queue1 and add it into queue2. Insert all neighbours of node B to queue1.

QUEUE1 = {D, C, F}

QUEUE2 = {A, B}

Step 4: Delete node D from queue1 and add it into queue2. Insert all neighbours of node D to queue1. Here, neighbour of Node D is F. Since it is already inserted, need not to insert again.

QUEUE1 = {C, F}

QUEUE2 = {A, B, D}

Step 5: Delete node C from queue1 and add it into queue2. Insert all neighbours of node C to queue1.

QUEUE1 = {F, E, G}

QUEUE2 = {A, B, D, C}

Step 6: Delete node F from queue1 and add it into queue2. Insert all neighbours of node F to queue-1. Since all the neighbours of node F are already present, we will not insert them again.

QUEUE1 = {E, G}

QUEUE2 = {A, B, D, C, F}

Step 7: Delete node E from queue1. Since all of its neighbours have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

QUEUE1 = {G}

QUEUE2 = {A, B, D, C, F, E}

Time complexity of BFS depends upon the data structure used to represent the graph.

2.4.2 Depth-First Search (DFS)

DFS searches along the depth and follows the stack approach. It visits all the sub-nodes of parent nodes and then moves to another node. It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial

node of graph G and goes deeper until we reach the goal node or the node with no children. Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

Step 1: Create a stack with the total number of vertices in the graph.

Step 2: Select any vertex as the beginning point of search and push that vertex into the stack.

Step 3: Push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.

Step 4: Repeat steps 3 and 4 until no vertices are left to visit from the vertex on the top of the stack.

Step 5: If there is no vertex, then go back and pop a vertex from the stack.

Step 6: Repeat steps 2, 3, and 4 until the stack is empty.

Example:

Let us understand the working of the DFS algorithm by using an example. In the example given below, there is a directed graph Fig 2.9, having 7 vertices.

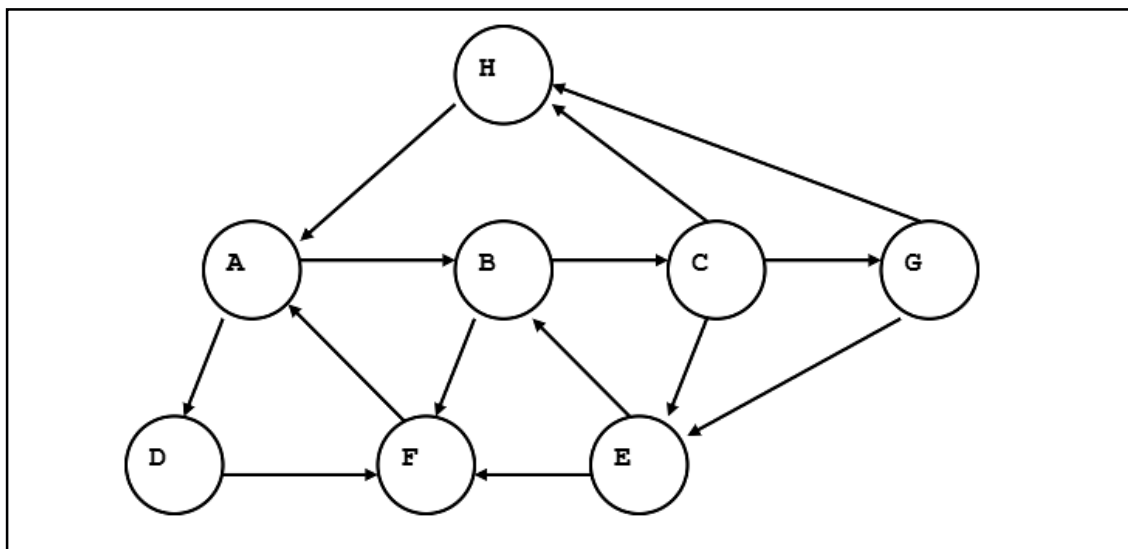


Fig. 2.9: DFS Graph

Now, let us start examining the graph starting from Node H.

Step 1: Push H onto the stack.

STACK: H

Step 2: Pop the top element H and print it. Now, push all the neighbours of H onto the stack that are in ready state.

Print: H

STACK: A

Step 3: Pop the top element A and print it. Now, push all the neighbours of A onto the stack that are in ready state.

Print: A

STACK: B, D

Step 4: Pop the top element D and print it. Now, push all the neighbours of D onto the stack that are in ready state.

Print: D

STACK: B, F

Step 5: Pop the top element F and print it. Then, push all the neighbours of F onto the stack.

Print: F

STACK: B

Step 6: Pop the top element B and print it. Then, push all the neighbours of B onto the stack.

Print: B

STACK: C

Step 7: Pop the top element C and print it. Then, push all the neighbours of C onto the stack that are in ready state.

Print: C

STACK: E, G

Step 8: Pop the top element from the stack, *i.e.*, G and push all the neighbours of G onto the stack that are in ready state.

Print: G

STACK: E

Step 9: Pop the top element from the stack, *i.e.*, E and push all the neighbours of E onto the stack that are in ready state.

Print: E

STACK:

Now, all the graph nodes have been traversed and the stack is empty.

2.5 HEURISTIC SEARCH (INFORMED SEARCH)

A heuristic is a technique used to solve a problem faster than the traditional methods. These techniques are used to find the most effective and efficient alternatives to find solution to a given problem. Heuristic algorithms are more efficient because they take advantage of feedback from the data from past experience with similar problems. Heuristic algorithms are not really intelligent. But they appear to be intelligent. Because they achieve better performance. Such techniques

might not always provide us the finest solution, but it is assuring us to find a good solution in a reasonable time.

Heuristics are used in situations where there is a scope for short-term solution. Based on the problem, there can be different heuristic methods that suits with the problem. The most common heuristic methods are trial and error, guesswork, the process of elimination, historical data analysis. These methods use simply available information that is not particular to the problem but is most appropriate.

A heuristic function is a function that correlates with problem state descriptions to measures desirability which is usually represented as “number”. Heuristic search methods make use of knowledge about the problem domain and choose promising operators first. These heuristic search methods use heuristic functions to evaluate the next state towards the goal state. For example, one should carry out the following steps to find a route from one city to another city.

- **State:** The current city in which the traveller is located.
- **Operators:** Roads linking the current city to other cities.
- **Cost metric:** The cost of taking a given road between cities.
- **Heuristic information:** The search could be guided by the direction of the goal city from the current city or we could use airline distance as an estimate of the distance to the goal.

2.5.1 Hill Climbing

Hill Climbing is heuristic search technique used for optimizing mathematical problems in the field of Artificial Intelligence. Hill climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs.

It is a search algorithm that continuously moves in the direction of increasing value to find the apex of a mountain or the best solution to the problem. It terminates when it reaches a peak value with no neighbour has a higher value. Traveling-salesman Problem is one of the examples of the Hill climbing algorithm in which distance travelled by the salesman is minimized. Algorithm is as follows.

Step 1: Evaluate the initial state. If the goal state is found, then return success and exit.

Step 2: Loop Until a solution is found or there is no new operator left.

Step 3: Select and apply an operator to the current state.

Step 4: Check new state.

- If it is a goal state, then return to success and exit
- Else if it is better than the current state, then assign a new state as a current state.
- Else if not better than the current state, then return to step 2.

Step 5: Exit.

For example, let us consider a tree as shown in Fig. 2.10. Initial node is A and all the nodes are represented with their heuristic values. Here we need to follow a path with least possible cost to reach the goal state. Solution to this problem is considered as follows.

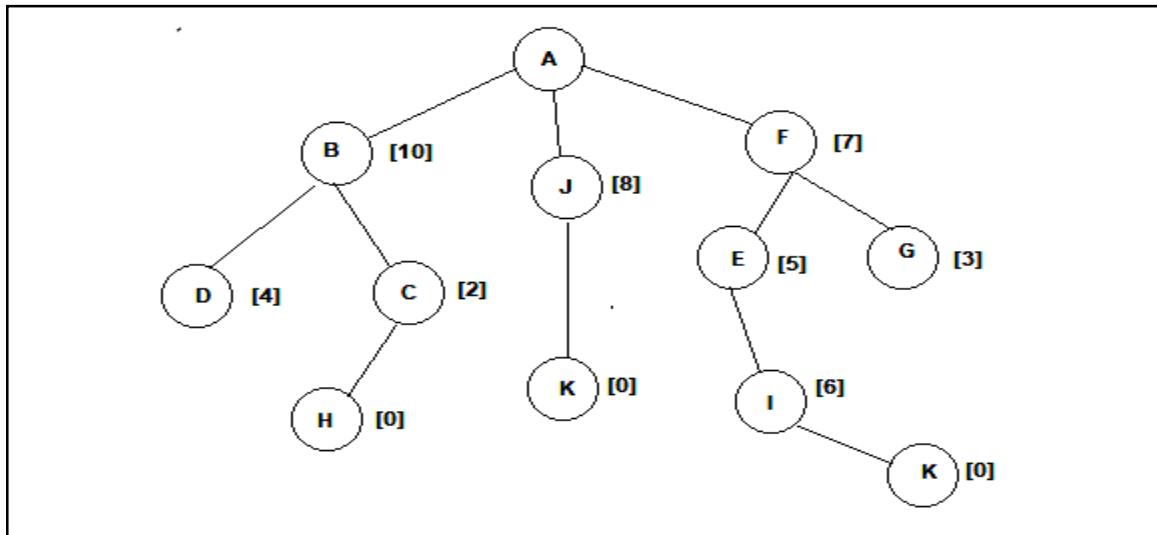


Fig. 2.10: Hill Climbing Search Tree

Select path $A \rightarrow F$, with the least possible cost and $F \rightarrow G$ with cost 3. But there is no path further. Hence, we need not to backtrack but restart from the least possible cost other than G. At this point we can follow at E as it was already inserted in the queue. Thus, we consider $E \rightarrow I$. Since node I has higher cost than E, we cannot follow this path further. Again, restart from the least cost other than F, E and G. Here, we can pick J as it has lower cost than B with difference of 2, i.e., $J = 8$ $B = 10$. $J \rightarrow K$ with cost 0. Hence, K is the goal. One should note that picking the least cost other than the already visited nodes is better than picking randomly.

2.5.2 Best First Search (BFS)

This algorithm always chooses the path which appears best at that moment. It is the technique resulted by combining depth-first search and breadth-first search algorithms. It provides the benefit of both algorithms. It makes use of heuristic function and search. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Algorithm for best first search is as follows.

- Step 1:** Place the starting node into the OPEN list.
- Step 2:** If the OPEN list is empty, Stop and return failure.
- Step 3:** Remove the node n from the OPEN list, which has the lowest value of $h(n)$ and places it in the CLOSED list.
- Step 4:** Expand the node n and generate the successors of node n.

Step 5: Check each successor of node n , and find whether any node is a goal node or not. If any successor node is the goal node, then return success and stop the search, else continue to next step.

Step 6: For each successor node, the algorithm checks for evaluation function $f(n)$ and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both lists, then add it to the OPEN list.

Step 7: Return to Step 2.

For example, let us consider the tree as shown in Fig. 2.11. Let us start from initial state A and search for goal G using heuristic values.

In this search example, we are using two lists which are named as OPEN and CLOSED Lists.

Let, list OPEN initially be contained with A. We transfer A from OPEN to CLOSED and process unvisited neighbours of A to OPEN. As a result, OPEN contains {C, B, D}. Since C has lesser cost, it is placed before B. Now, we transfer C from OPEN to CLOSED and process unvisited neighbours of C to OPEN. One can notice that OPEN is now updated as {F, E, B, D} and CLOSED contains {A, C}. We transfer F from OPEN to CLOSED and process unvisited neighbours of F to OPEN. As result, OPEN now contains {G, E, B, D} and CLOSED contains {A, C, F}.

Finally, we can transfer G from OPEN. Since our goal state G is a neighbour of F, we return. Hence best possible path using BFS is $A \rightarrow C \rightarrow F \rightarrow G$.

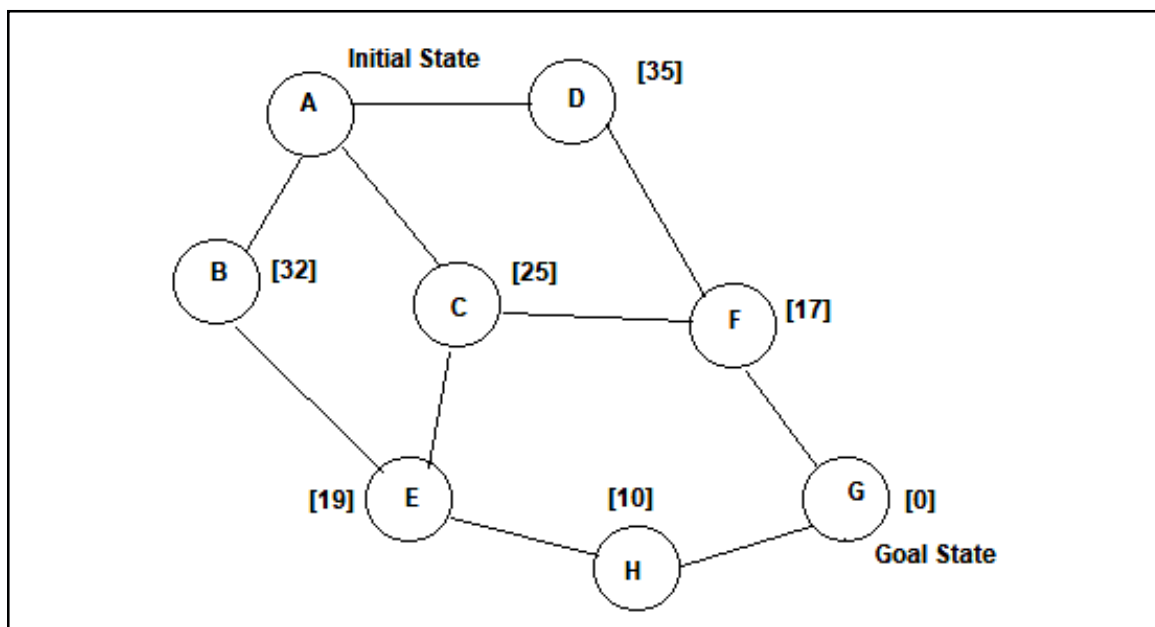


Fig. 2.11: Best First Search Tree

2.5.3 A* Search

A* search is considered to be the most popular form of best-first search. Because it is fairly flexible and can be used in a wide range of contexts. It uses the heuristic function $h(n)$ and cost to reach the node n from the start state $g(n)$. It finds the shortest path through the search space using the heuristic function. This search algorithm expands fewer search tree and thereby gives optimal results faster. Algorithm for A* search is as follows.

Step 1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not. If the list is empty, then return failure and exit.

Step 3: Select the node from the OPEN list which has the smallest value of the evaluation function ($g + h$). If node n is the goal node, then return success and stop, otherwise.

Step 4: Expand node n and generate all of its successors and put n into the closed list. For each successor n' , check whether n' is already in the OPEN or CLOSED list. If not, then compute the evaluation function for n' and place it into the Open list.

Step 5: Else, if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.

Step 6: Return to Step 2.

It is important to note that A* Algorithm is one of the best paths finding algorithms. But it does not produce the shortest path always. This is because it heavily depends on heuristics. Example for this search method is explained in exercise examples.

EXERCISES

2.1 Given an initial state of an 8-puzzle problem and final state to be reached in Fig. 2.12.

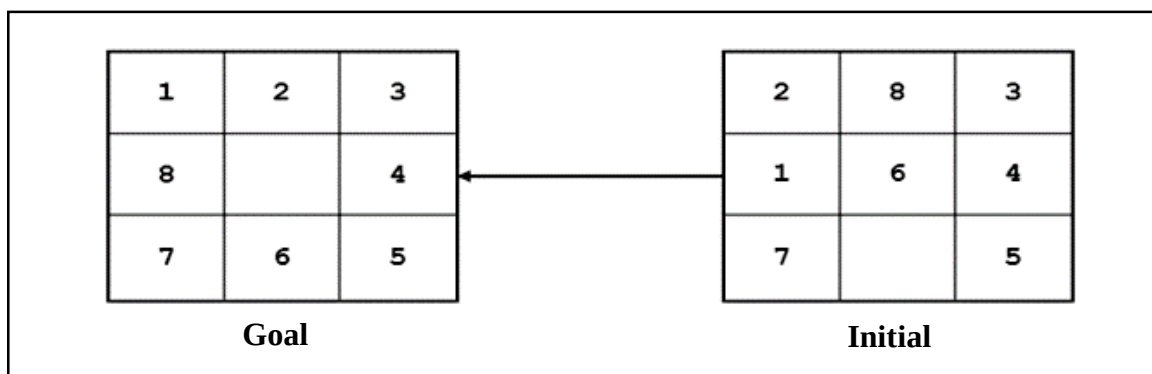


Fig. 2.12: 8-Puzzle Problem

Find the most cost-effective path to reach the final state from initial state using A* Algorithm. Consider $g(n)$ = Depth of node and $h(n)$ = Number of misplaced tiles.

Solution: A* Algorithm maintains a tree of paths originating at the initial state. It extends to those paths one edge at a time. It continues until final state is reached. Refer Fig. 2.13.

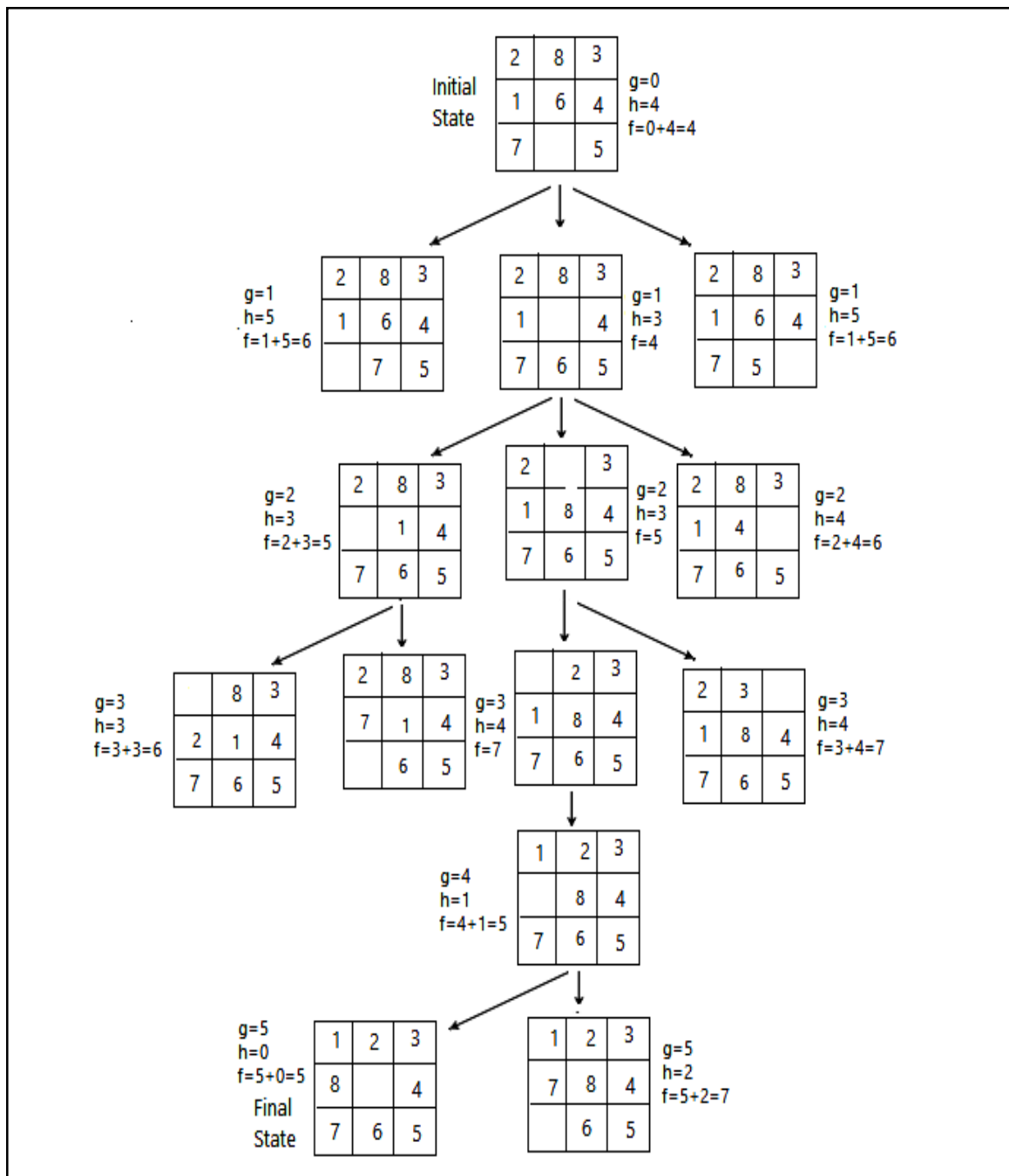


Fig. 2.13: 8-puzzle problem- Solution Tree

2.2 Consider the following graph as shown in Fig. 2.14.

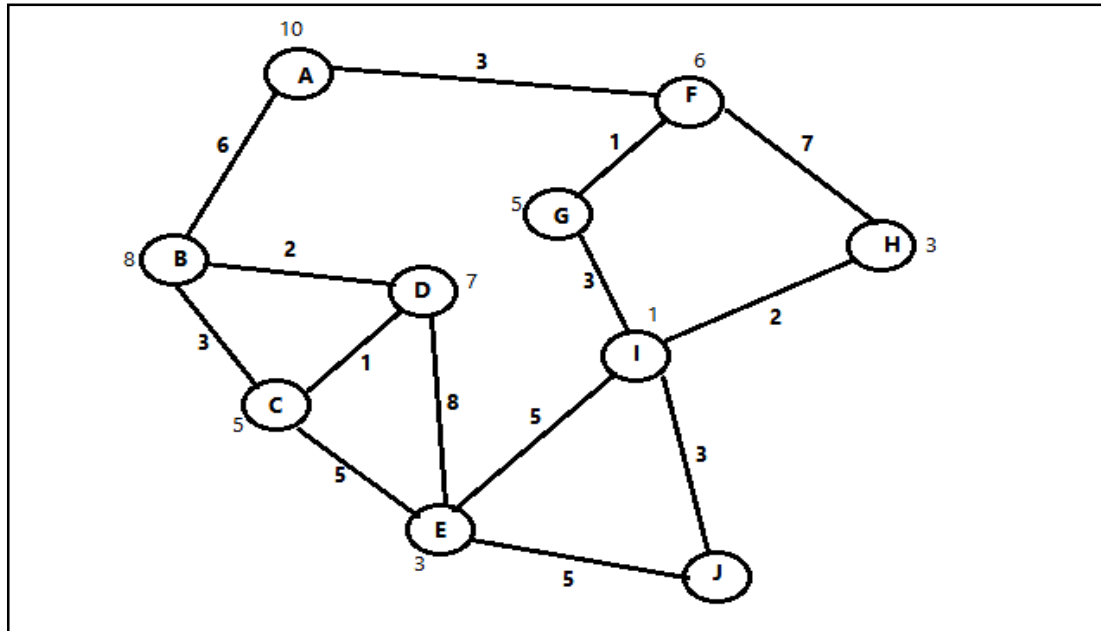


Fig. 2.14: Problem Graph

The numbers written on edges represent the distance between the nodes. The numbers written on nodes represent the heuristic value. Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.

Solution:

Step 1:

We start with node A.

Node B and Node F can be reached from node A.

A* Algorithm calculates $f(B)$ and $f(F)$.

$$f(B) = 6 + 8 = 14$$

$$f(F) = 3 + 6 = 9$$

Since $f(F) < f(B)$, so it decides to go to node F.

Path-A $\rightarrow$ F

Step 2:

Node G and Node H can be reached from node F.

A* Algorithm calculates $f(G)$ and $f(H)$.

$$f(G) = (3 + 1) + 5 = 9$$

$$f(H) = (3 + 7) + 3 = 13$$

Since $f(G) < f(H)$, so it decides to go to node G.

Path-A $\rightarrow$ F $\rightarrow$ G

Step 3:

Node I can be reached from node G.

A* Algorithm calculates $f(I)$.

$$f(I) = (3 + 1 + 3) + 1 = 8$$

It decides to go to node I.

Path- $A \rightarrow F \rightarrow G \rightarrow I$

Step 4:

Node E, Node H and Node J can be reached from node I.

A* Algorithm calculates $f(E)$, $f(H)$ and $f(J)$.

$$f(E) = (3 + 1 + 3 + 5) + 3 = 15$$

$$f(H) = (3 + 1 + 3 + 2) + 3 = 12$$

$$f(J) = (3 + 1 + 3 + 3) + 0 = 10$$

Since $f(J)$ is least, so it decides to go to node J. Path- $A \rightarrow F \rightarrow G \rightarrow I \rightarrow J$. This is the shortest path from node A to node J.

2.3 Give the visited node order for both BFS and DFS search, starting with S for the following graph as shown in Fig. 2.15. with given adjacency list.

$\text{Adj}(s) = [a, c, d]$,

$\text{Adj}(a) = []$,

$\text{Adj}(c) = [e, b]$,

$\text{Adj}(b) = [d]$,

$\text{Adj}(d) = [c], \text{Adj}(e) = [s]$

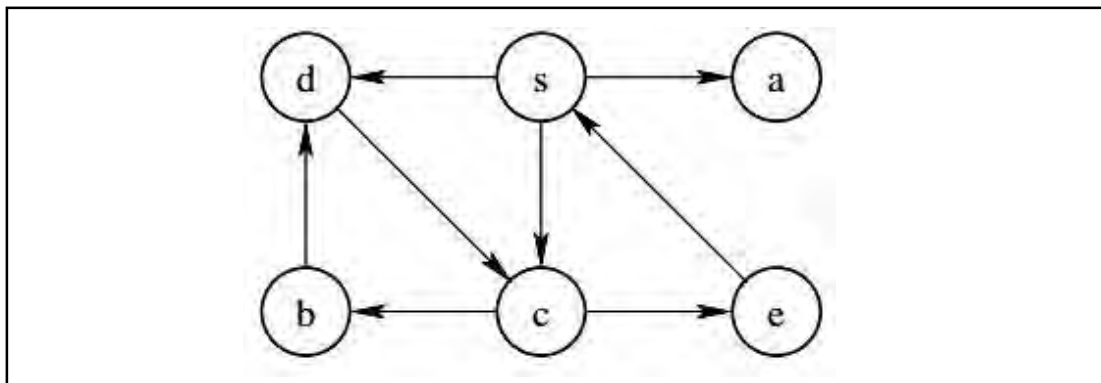


Fig. 2.15: Problem Graph

Solution:

Breadth First Search: s a c d e b

Depth First Search: s a c e b d

- 2.4 Consider a state space search where start state is numbered as 1 and the successor function for state numbered as n returns two states numbered as $2n$ and $2n + 1$. Draw the portion of the state space for states 1 to 15. And if, the goal state is 13, then list the orders in which nodes will be visited using following search techniques. (i) Breadth First search (ii) Depth First search.

Solution:

To Draw the portion of the state space for states 1 to 15.

Level 1: 1

Level 2: 2 3

Level 3: 4 5 6 7

Level 4: 8 9 10 11 12 13 14 15

The order in which nodes will be visited for breadth first search, depth-limited search with limit 13,

- (i) Breadth-first search: 1 2 3 4 5 6 7 8 9 10 11 12 13
 (ii) Depth-first search: 1 2 4 8 9 5 10 11 3 6 12 13

QUESTIONS

- 2.1 Explain the production system in AI with example.
- 2.2 Explain the representation of state space search with example.
- 2.3 Explain the concept of control strategies in AI.
- 2.4 Explain BFS and DFS with example.
- 2.5 Explain the following Informed search techniques.
 (i) Hill Climbing (ii) Best First search (iii) A* search
- 2.6 Consider a state space search where start state is numbered as 1 and the successor function for state numbered as n returns two states numbered as $3n$ and $3n-1$. If the goal state is 24, then list the orders in which nodes will be visited using following search techniques. Draw search tree in each case.
 (i) Depth First search (ii) Breadth First search
- 2.7 Consider the following. A farmer wants to get a Tiger, a fox, a goose and some grains across a river. There is a boat, but he can only take one in addition to himself on each trip or else both goose and grain or both the fox and the grain. The grain cannot be left with goose as it will eat the grain. Similarly, the fox can eat goose if left together and tiger cannot be left with fox. How does everything get across the river?
 (i) Give the start and goal states along with constraints.
 (ii) Draw state space search tree using DFS and BFS to find the solution.



3

PROBLEM REDUCTION AND GAME PLAYING

3.1 INTRODUCTION

A solution to a problem can be obtained by decomposing it into smaller sub-problems. Each of this sub-problem can then be solved to get its sub solution. These sub solutions can then be recombined to get a solution as a whole. This process is called as Problem Reduction. Under problem reduction search we can have two types of graph searches namely, AND-OR graphs and Game trees. AND-OR graph generates arc which is called as AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved for an arc to point to a solution.

Though people do not always refer to them AND-OR search, but the underlying philosophy is the same. For instance, if you wish to own a bike then either you have to steal a bike or you earn money and buy one for yourself. AND-OR Graph which indicates these possibilities as shown in Fig. 3.1.

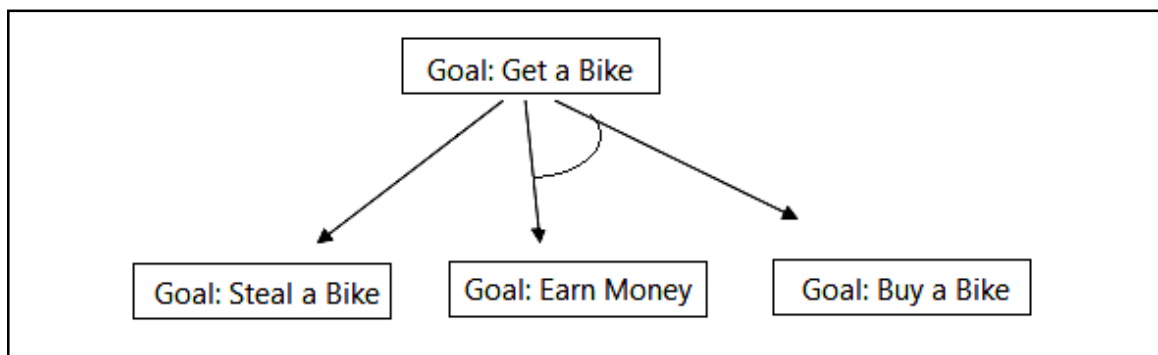


Fig. 3.1: A Simple AND-OR Graph

On the other hand, Game trees are the backbone of game playing programs which optimize in the presence of an adversary. Games are situations where there are adversaries and there is some criterion that we have to optimize in the presence of the adversary.

3.2 PROBLEM REDUCTION

Problem reduction search can be defined as the planning of how best a problem can be recursively decomposed into sub-problems in multiple ways. Problem reduction search helps to solve the same problem by decomposition. There are more than one decomposition of the same

problem and we have to decide which is the best way to decompose the problem so that the total solution-cost, quality of solution or the effort of searching is minimized.

Let us consider the matrix multiplication problem, where given a set of matrices A, B, C... and so on. We have to find out the product of A to say D. Now, this can be done in several ways. One way is to multiply A with B then the product is multiplied with C and then resulting product is multiplied with D. Other way could be, to multiply A with B and C with D and then multiply the resulting products of each as shown in the Fig. 3.2.

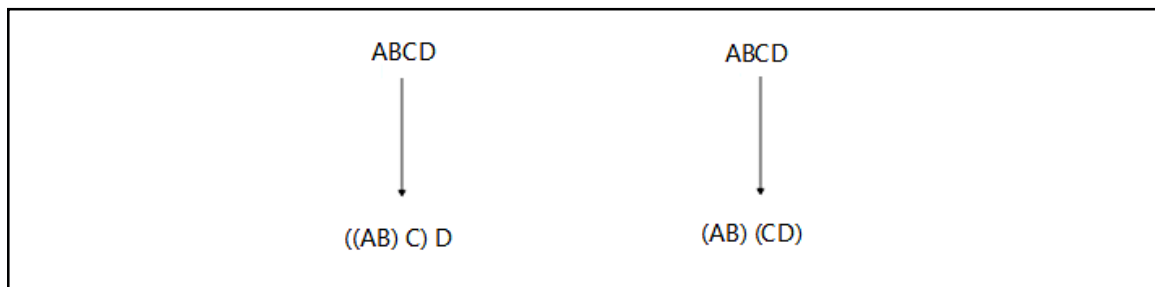


Fig. 3.2: Matrix Multiplication

If we look at this problem from other direction, the problem of multiplying A to D, can be thought of as multiplying A with B and then C with D as sub problems of parent problem as shown in Fig. 3.3.

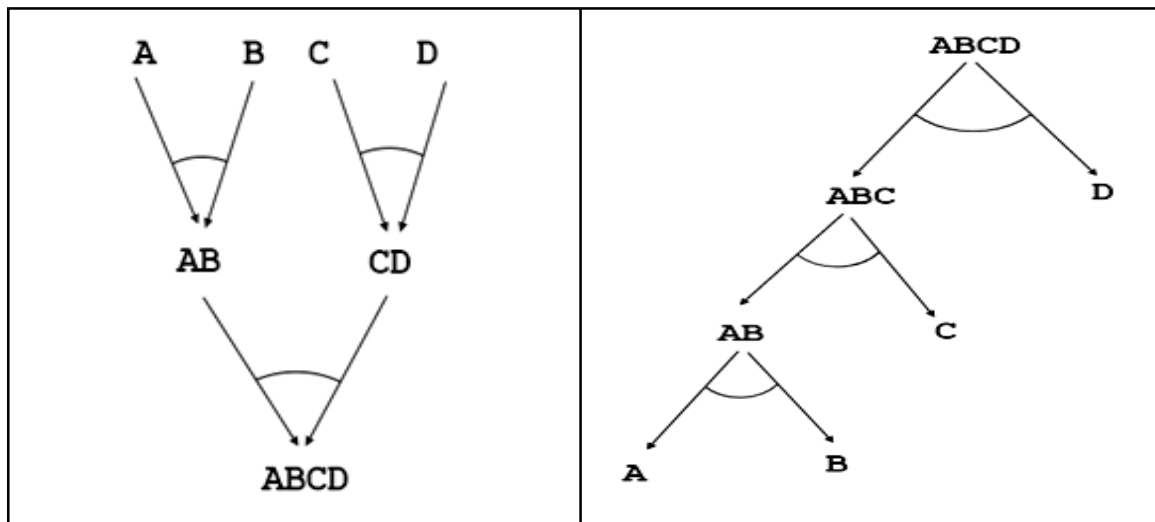


Fig. 3.3: Decomposition-1

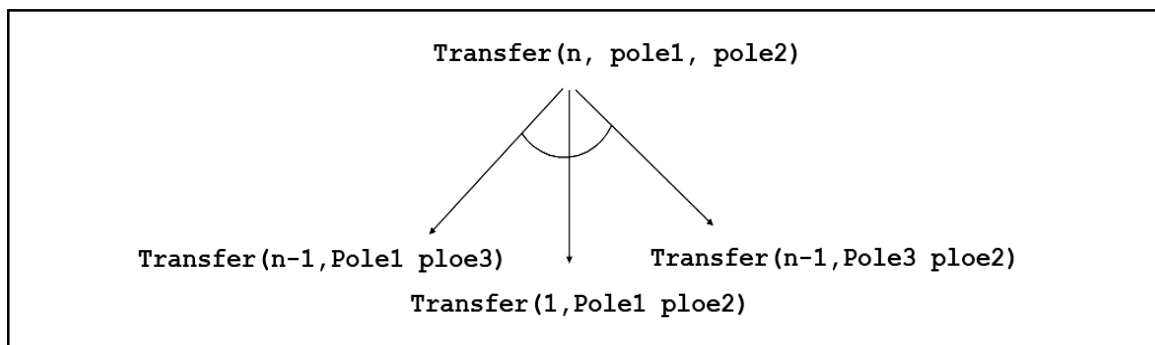
Fig. 3.4: Decomposition-2

An alternative way of doing this, would be to have the product of A, B, C multiplied with D and so on using method of decomposition as shown in Fig. 3.4. AND arcs shown in the figure indicates that both the decompositions are essentially required to reach to final solution. Let us consider another planning problem known as Tower of Hanoi which illustrate the importance of the problem reduction concept. Tower of Hanoi is a mathematical puzzle which consists of three towers (poles) and more than one discs of different sizes which are stacked upon in an ascending

order, *i.e.*, the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same. The objective of the puzzle is to move the stack of discs to another pole by using the following rules.

- Only one disk can be moved among the towers at any given time.
- Only the “top” disk can be removed.
- No large disk can sit over a small disk.

Consider that there are n discs in the first pole say pole 1. Now we can move all these n discs from pole 1 to pole 2 with the help of pole 3. We can develop an algorithm which uses problem reduction concept and shows that this problem can be solved by reducing it into sub problems. The game tree generated will contain AND-OR arcs. Algorithm to transfer the top n minus one disk from pole 1 to pole 2 will involve following steps.



If $n = 1$ then move the disc from pole1 to pole 2. Otherwise move all the top $n-1$ disks in the same order from pole 1 to pole 3 and then move the largest disk from pole 1 to pole 2. Then move $n-1$ smaller discs from pole 3 to pole 2. As a result, the problem is reduced to by moving $n-1$ disks from pole 1 to pole 3 and then pole 3 to pole 2. The same procedure can be followed recursively to reduce the problem of $n-1$ disks to $n-2$, $n-3$ and so on till all the discs are moves from pole1 to pole 2.

Let us consider an example of 3 disks. With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks. The start and goal states are shown in Fig. 3.5.

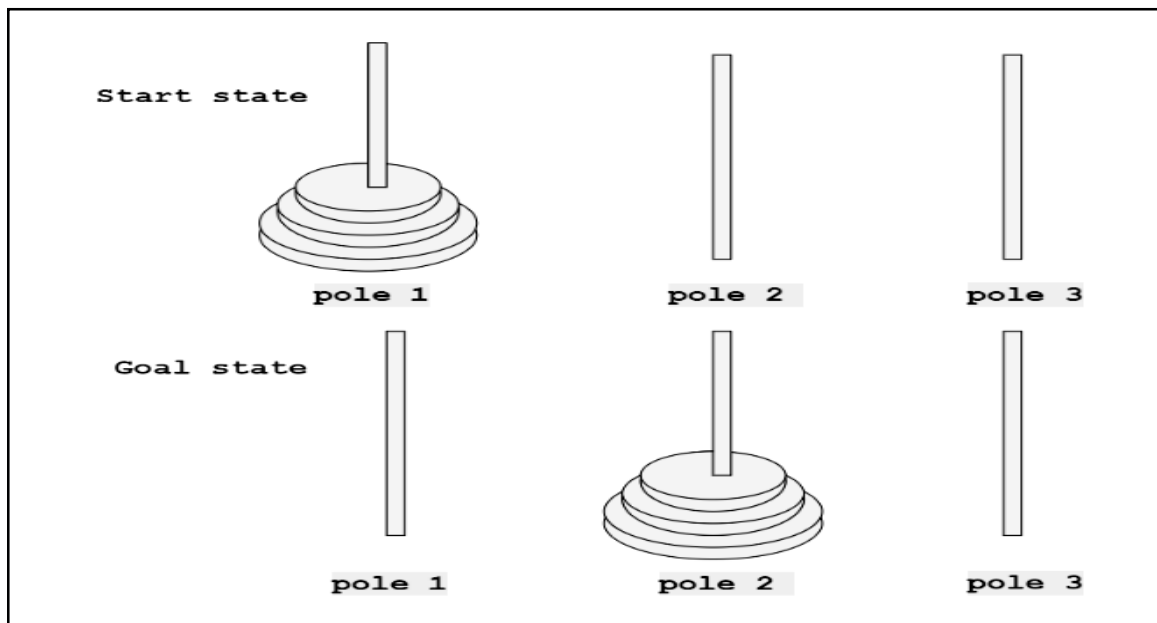


Fig. 3.5: Tower of Hanoi Problem

The AND-OR graph shown in Fig. 3.6. There are two possible decompositions. One way is from root to state 1 and other from root to state 1'. The decomposition through 1 essentially requires all three states 1, 2 and 3 to solve the problem. In state 1, we need to expand it with drawing AND arcs. The process continues till we reach goal state 3. It is important to note that all the sub problems in the process are dependent of each other where State 2 will be achieved after state 1 and state 3 will be achieved after state 2.

The second way will follow from root to state 1' then to state 2' and then 3'. This path will be longer and continued till the goal state is achieved.

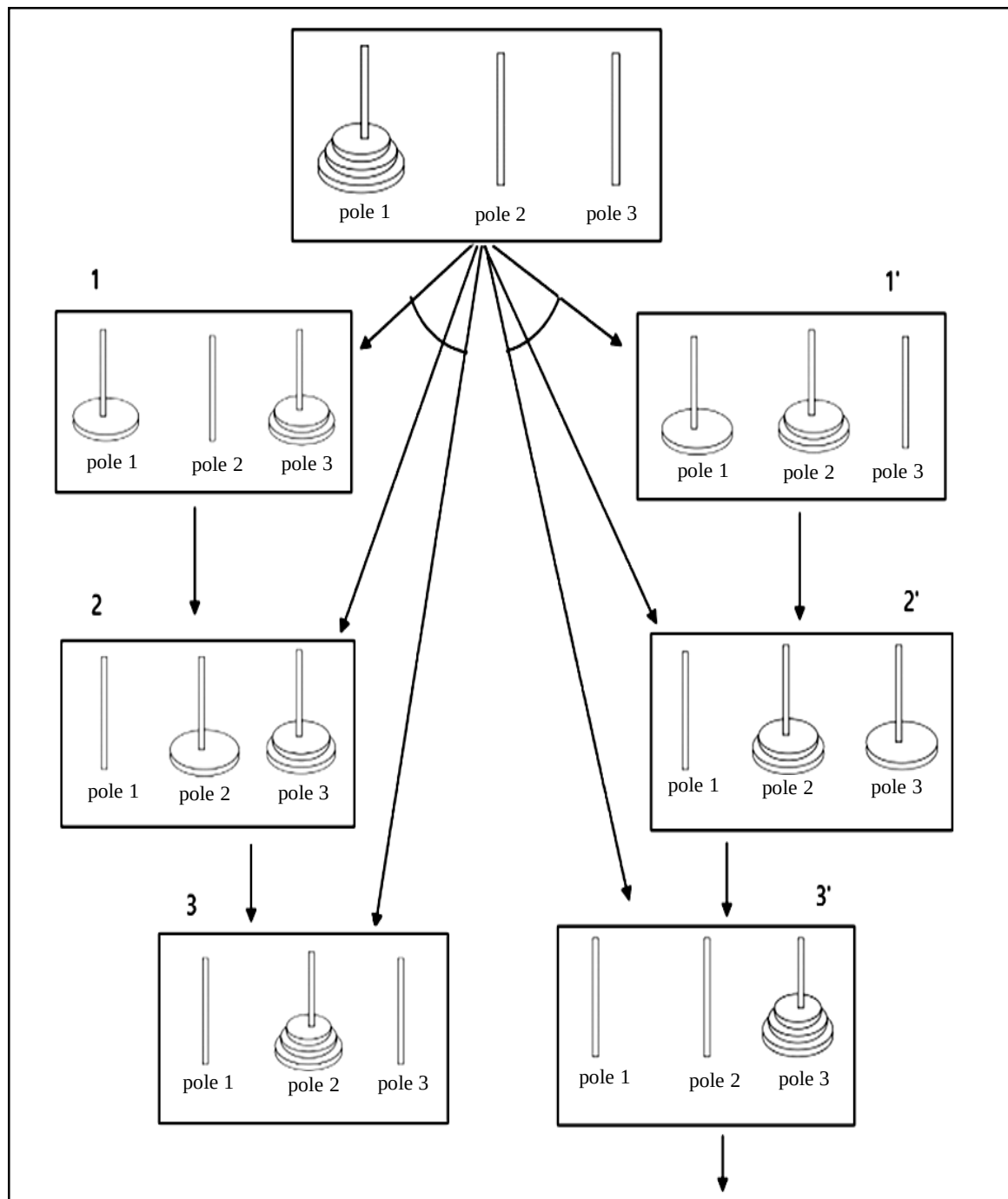


Fig. 3.6: AND-OR Graph for Tower of Hanoi Problem

3.2.1 Formulation of the Problem Reduction Search Problem

AND-OR Graph

OR node represents a choice between possible decompositions. When OR node is used, we have to solve any one of its successors. There can be more than one decomposition and we will use an OR node to choose between possible decompositions. AND nodes are actually representing a given decompositions. AND-OR nodes will represent the choice between different decomposition. When AND node is used, we have to solve all its successors.

Game Tree

In Game Tree, we have MAX and MIN nodes, where MAX nodes will represent the choice of your opponent and MIN nodes will represent your choice.

But both of these formulations have a similarity in the sense that both have two different kinds of nodes. Each node has a kind of optimization criterion. AND nodes will take the sum of the cost and OR node will take the minimum of the cost. MAX nodes will maximize and MIN nodes will minimize.

AND-OR Graph Search Problem.

Given: $[G, S, T]$ where,

G: Implicitly specified AND-OR graph

S: Start node of the AND-OR graph

T: Set of terminal nodes and

$h(n)$: A heuristic function that estimates the cost of solving the sub-problem at n .

To find: The Minimum cost solution tree.

Let us consider a matrix multiplication problem. If we have to multiply three matrices say A, B and C, then we begin our decomposition with OR node as shown in the Fig. 3.7.

Let the Dimensions of these Matrices are $A = [3 \times 4]$, $B = [4 \times 10]$ and $C = [10 \times 1]$. Here, the best way of decomposition can be obtained by calculating the cost associated with each of these decompositions. Likewise cost associated with AB is $120(3 \times 4 \times 10)$ and cost of BC is $40(4 \times 10 \times 1)$. Cost associated with A and C are zero. Hence, we obtain the cost at A(BC) node is equal to 52 (cost of $[3 \times 4 \times 1]$, i.e., $12 + 40$) and cost at (AB)C node is equal to 150(cost of $[3 \times 10 \times 1]$, i.e., $30 + 120$). Since this is a cost minimisation problem, we have to choose a least cost successor along the path which is marked as the best path.

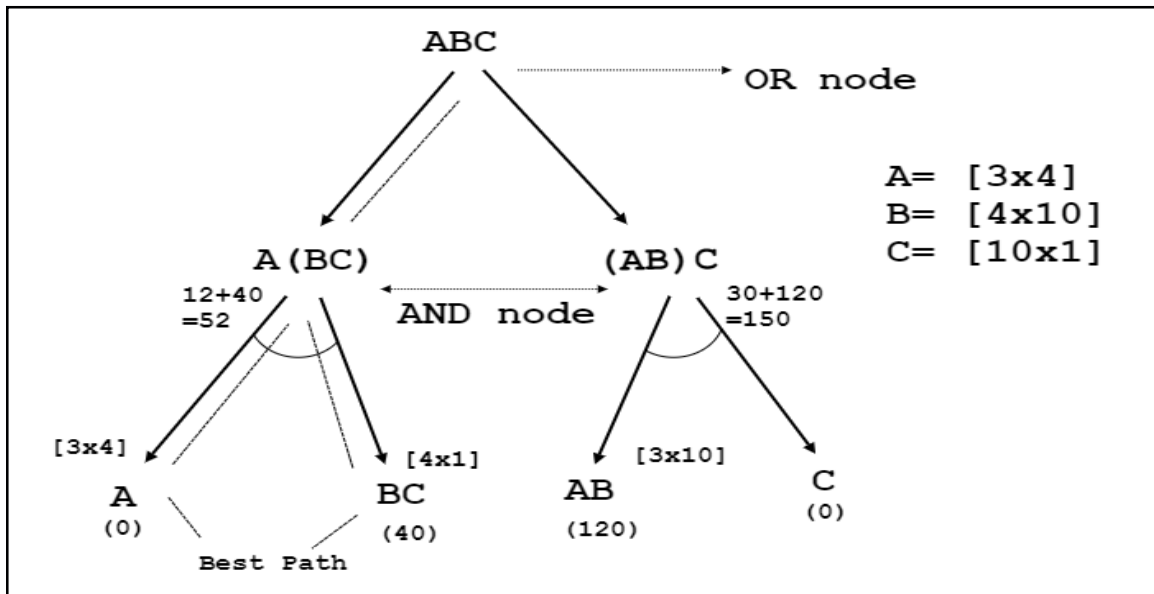


Fig. 3.7: Minimum Cost Solution using AND-OR Graph

If matrices are more, then this tree would go deeper with a greater number of OR as well as AND nodes. The cost of the best among the successors will be backed up as the cost of the OR node. Above example is discussed in the absence of Heuristic function just like state search problem.

Heuristics Function

Heuristic is a function which is used in informed search and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however might not always give the best solution, but it is guaranteed to find a good solution in reasonable time. Heuristic function is represented by $h(n)$ and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

3.2.2 Node Status Labelling Procedure

A node in a search tree may be either a terminal node or non-terminal node. The labels used to represent these nodes in a tree are as follows.

- **Terminal node:** A terminal node is a node which has no child nodes. If this node is a goal node, then it is labelled as solved. Otherwise, it is labelled as unsolved and represents a sub problem.
- **Non-Terminal node:** A non-terminal node is a node which can be expandable either as AND node or OR node.
- **AND node:** A non-terminal AND node is labelled as unsolved, if one of its successors is found to be expandable. To label this node as solved, all of its successors are to be solved.

- **OR node:** A non-terminal OR node is labelled as solved if one of its successors is labelled as solved. To label this node as unsolved, all of its successors to be unsolved.

Let us consider an example to illustrate the labelling procedure taking heuristics into account. We will solve this problem just like state space search problem with marked AND-OR tree in order to develop the best way of solving the problem. Different stages of this problem are shown in the Figures from 3.8 to 3.11. Let the top node be A which is an AND node and heuristic (h) value of this node be equal to 7. Since this node is an AND node, we have to solve and mark both of its successors and also, we have to assume that each path with a single successor has some cost. These costs will be the costs of merging the sub-problems back into the original problem. Let us evaluate the heuristic function in the child nodes B and C as 4 and 3 respectively. This means we are estimating that at least 4 and 3 units of costs have to be incurred respectively to solve this problem. Therefore, estimated cost of the paths from A to B is 6 (4 + cost of one path from A to B) and that of A to C is 4 (3 + cost of one path from A to C). These values are then propagated up and the cost associated with node A is revised as 10 in place of earlier estimated cost of 7 at the end of stage 1.

It should be noted that the figures mentioned in the parenthesis () represent the estimated costs and figures in square bracket [] represent the revised costs.

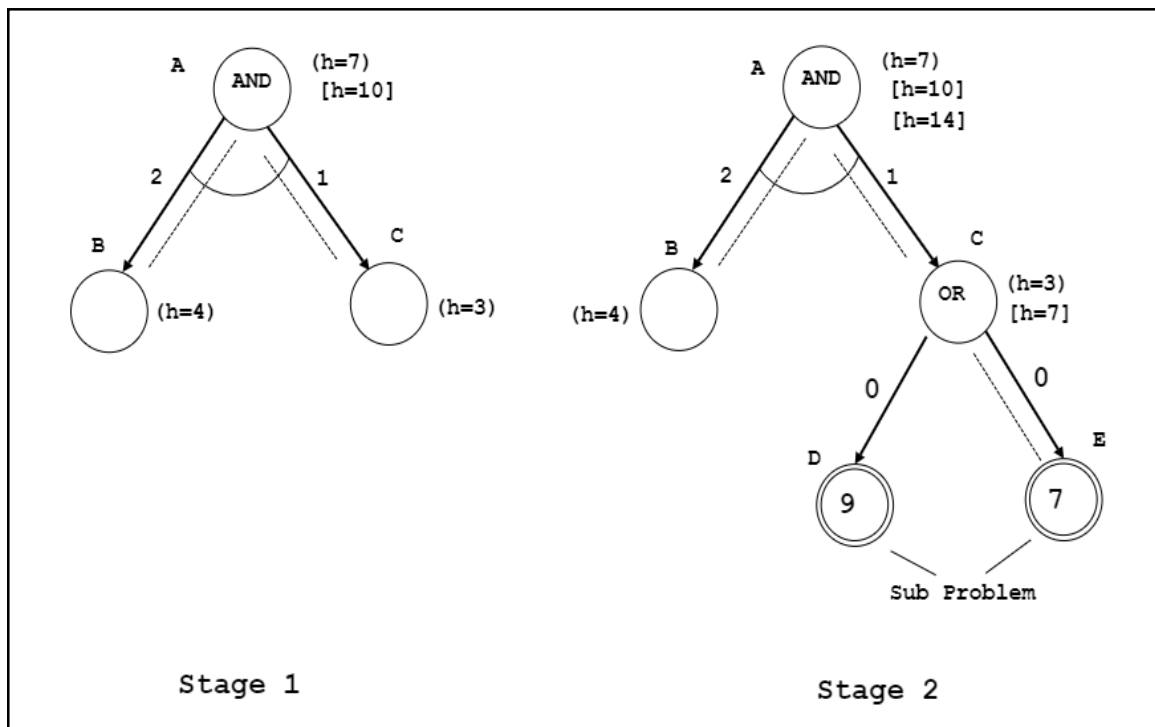


Fig. 3.8: Status Labelling Procedure

Now, let us expand node C which is assumed to be an OR node. After the expansion the resulting nodes D and E has h value of 9 and 7 respectively. Since this node is an OR node, any one of successors need to be solved. We will take the path from C to E and mark this path as it is

less in cost. Now the revised cost of the node C is 7 (7 + cost of the path from C to E which is zero). This again leads the cost revision from bottom up at node A and revised value is equal to 14 along the path from E to C and then C to A. This path is the best path so far. At the end of stage 2, Nodes D and E are labelled as unsolved and these nodes are representing the sub problems as these nodes are no longer be expanded.

In stage 3, let us expand the node B to nodes F and G with estimated h values of 2 and 5 respectively. Cost of the respective paths are taken as 2 and 1. Since Node B is an OR node, we will mark the path along leaf node from A to B and then B to F.

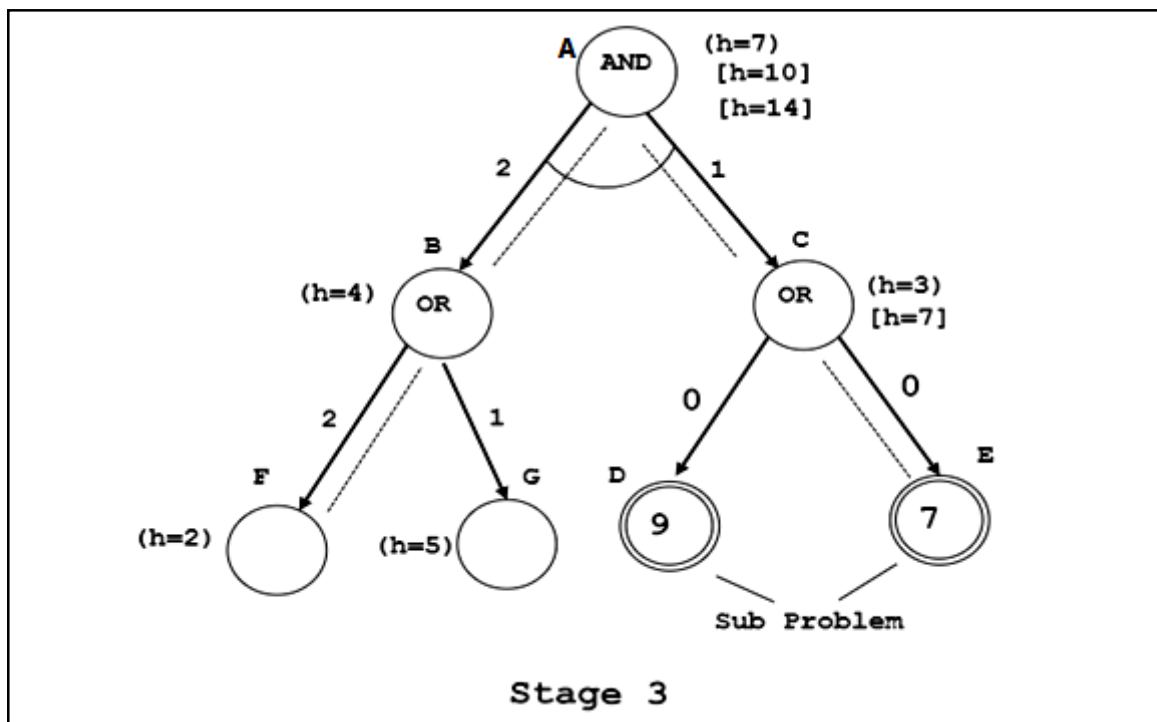


Fig. 3.9: Labelling Procedure: Third Stage

In stage 4, we expand the node F and evaluate the h values as 3 and 10 for nodes H and I respectively. Cost of these paths from F to H and F to I be 1 and 0 respectively. Now revised cost of AND node F is equal to 14 (3 + cost of the path from F to H which is 1 and 10 + cost of path from F to I which is 0) as shown in Fig. 3.10.

This new cost value is propagated up and revised cost at node B is 16 (14 + cost of path from F to B which is 2). Since the node B is an OR node, we are choosing the path G to B as this path is resulting in the revised cost of 6 (5 + cost of path from G to B which is 1) which is lesser than revised cost of 16. This revision at B leads to the revision of value at node A as 16.

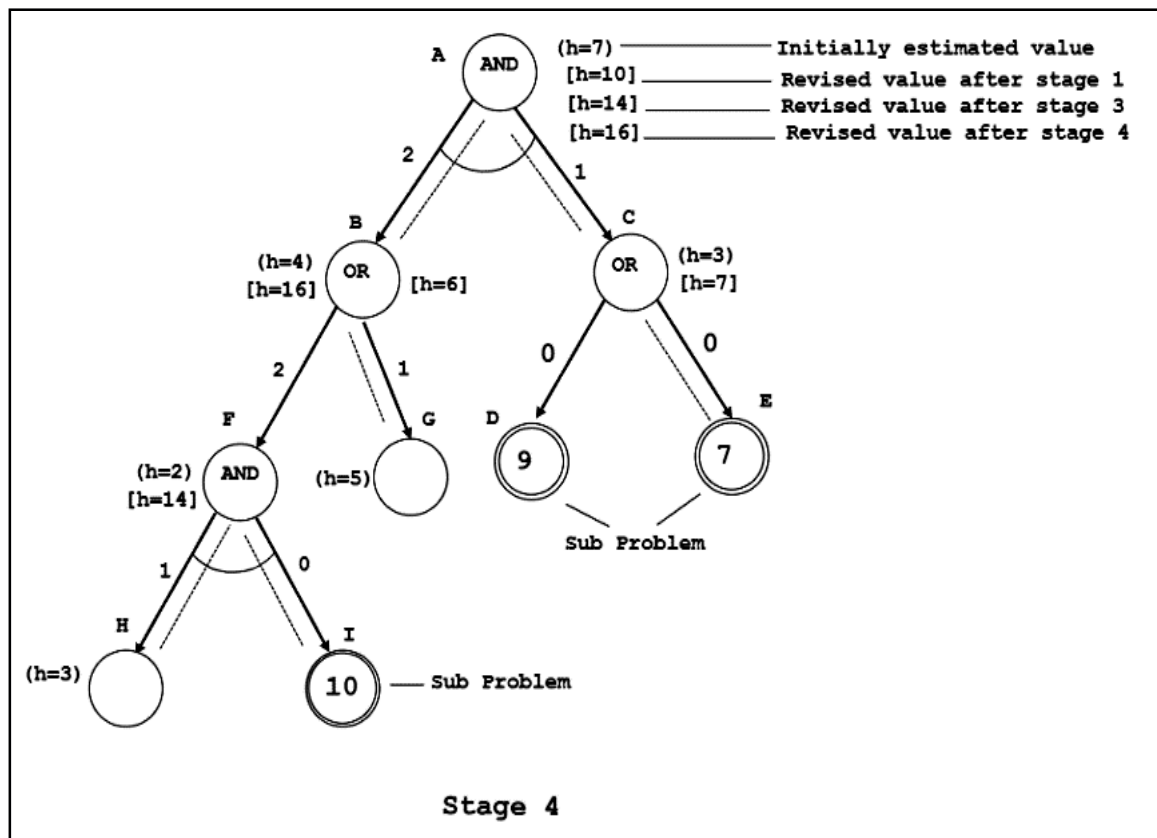


Fig. 3.10: Labelling Procedure: Fourth Stage

In stage 5, we further expand node G to get node J and K with estimated h values of 3 and 4 respectively assuming the cost of each path from G to J and G to K as zero. Node G is an AND node and cost of this node is revised as 7(3 + cost of the path from J to G and 4 + cost of path from K to G). This in turn revises the cost along the path from G to B which is equal to 8 instead of earlier revised cost of 6. Still this path is considered to be the best path as the revised value of 8 is less than earlier revised value of 16 at node B. This eventually leads to the revision of cost value at node A as 18 along the best path from G to B and then B to A. Hence, the nodes I, J and K are labelled as terminal nodes as shown in Fig. 3.11.

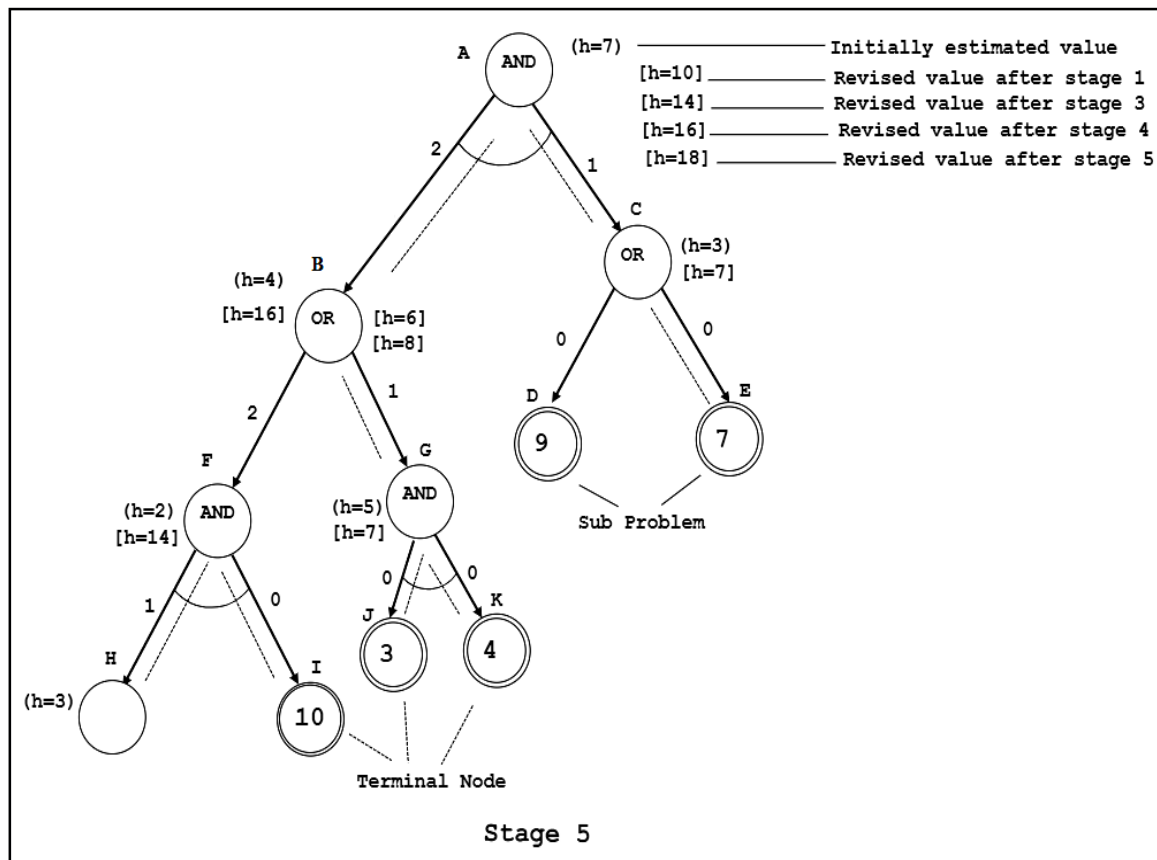


Fig. 3.11: Labelling Procedure: Fifth Stage

With the above AND-OR graphs search, it can be possible to outline the required steps for applying heuristic search. N.J Nilsson named this algorithm as AO* as it is used for searching the best possible search in AND-OR tree.

Algorithm AO*

1. Initialize: Set $G^* = \{s\}$, $f\{s\} = h\{s\}$
If $s \in T$ label as terminal node.
2. Terminate: If s is solved then terminate
3. Select: A non-terminal leaf node n from the marked sub tree
4. Expand: Make the explicit successor of n
For each new successor m ;
Set $f\{m\} = h\{m\}$
If m is terminal label m as solved
5. Cost revision: Call cost revise(n)
6. Loop: Go to step 2.

Algorithm for Cost Revision in AO*

1. Create: $z = \{n\}$
2. If $z = \{ \}$ return
3. Select a node m from z such that m has no descendants in z
4. If m is AND node with successors $r_1, r_2, \dots, r_k$
 Set $f\{m\} = \Sigma[f(r_i) + c(m, r_i)]$
 Mark edge to each successor of m
 If each successor is labelled solved, then label m as solved.
5. If m is an OR node with successors $r_1, r_2, \dots, r_k$
 Set $f\{m\} = \text{Min}\Sigma[f(r_i) + c(m, r_i)]$
 Mark edge to the best successor of m
 If marked successor is labelled solved, then label m as solved.
6. If the cost or label of m has changed then, insert those parents of m into z for which m is a marked successor.

3.3 GAME PLAYING

Game playing was one of the earliest tasks undertaken in Artificial Intelligence. Game theory has been said to be in practice since 1950, almost from the days when computers became programmable. The very first game that has been tackled in AI was chess.

Ever since the beginning of AI, there has been a great interest in experimenting the human expert against the computer. Game playing extended a high-visibility platform for this contest. It is important to note that the performance of the human expert and the AI game-playing program are qualitatively different processes. More specifically, the performance of the human expert uses an enormous amount of domain specific knowledge and procedures. Such knowledge makes the human expert to develop a few promising moves for each game situation considering relevant moves in the game. In contrast, while selecting a best move, game playing program exploits cracking computational speed to explore as many alternative moves as possible. Since the computational speed of modern computers increases, the contest between knowledge and speed is tilting more and more in favour of computers.

Game Playing is an important branch of artificial intelligence. Games does not require much knowledge. The only knowledge we require to provide is the rules, legal moves and the conditions of winning or losing the game. Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS (Breadth First Search) are not suitable for this as their branching factor is very high which results in searching to take lot of time. So, we need search procedures like game playing to improve computational technique in less possible time.

3.3.1 Types of Game

Perfect Information Game: In which both the players know all the information about the game. Perfect information games are those in which all the events that have previously occurred and both the players have access to the same information about the game, *e.g.*, Chess, Tic Toe etc.

Imperfect Information Game: In which player does not know all the possible moves of the opponent, *e.g.*, Card games. Since all the cards are not visible to player and each player's cards are hidden from other players but objectives are known. Game playing is a search problem defined by following components:

Initial state: This defines initial configuration of the game and identifies first payer to move.

Successor function: This identifies which are the possible states that can be achieved from the current state. This function returns a list of (move, state) pairs, each indicating a legal move and the resulting state.

Goal test: Which checks whether a given state is a goal state or not. States where the game ends are called as terminal states.

Path cost/utility/payoff function: This gives a numeric value for the terminal states. In chess, the outcome is win, lose or draw, with values +1, -1, or 0. Some games have wider range of possible outcomes.

We will confine our study to discrete and perfect information games. A game is said to be discrete if it contains a finite number of states and configurations. We will see how to develop search procedures for two player games as they are common and easier to design. A typical characteristic of a game is to predict at future positions in order to win the game. But it is seen that the optimal solution can be obtained by exhaustive search methods if there are no constraints with respect to time and space. But for most of the familiar games, such a solution is usually too ineffective in terms of practical usage.

3.3.2 Game Problem vs State Space Search Problem

It should be observed that there is a natural analogous relationship between games and state space problems. For example, in state search problems, we have initial state, intermediate states, rule and a goal state. In game problems also we have a start state, legal moves and winning positions. To make it more clear about the relationship between the two problems we can refer following table for comparison.

Table 3.1: State Space and Game Problem Comparison

State Space Problems	Game Problems
States	Legal board positions
Rules	Legal moves
Goal	Winning positions

The complete game tree is the game tree starting at the initial position, containing all possible moves from each position. The game ends in a position that can be declared as win for

one and a loss for other or possibly a draw. A game tree is an explicit representation of all possible players of the game. To be more specific, the complete game is a norm for the game in game theory which can clearly express many important aspects. There is obvious correspondence between a game tree and AND-OR tree. The OR node represents moves available to one player from a given position whereas the move available to his opponent are represented by AND nodes. Therefore, in the game tree one level is treated as an OR node level and other level as AND node from one player's point of view.

It should be noted that the game playing is based on the principle of minimising the maximum possible loss and maximizing the minimum gain. In computer involved game playing, one player will always be computer and other is a human. During this situation two types of nodes namely, MIN and MAX are encountered. Either MIN or MAX can play as the first player. MAX node will try to maximize its game while minimizing the MIN's game. In other words, MIN nodes, will select the minimum cost successor and MAX nodes will select the maximum cost successor. Terminal nodes can be winning or losing states, but it is often infeasible to search up to the terminal nodes. So, we use heuristic costs to compare non-terminal nodes.

3.3.3 Two player Game in Game Tree

Computers use an algorithm called the Minimax algorithm when a specific state of a game is given. It will simulate each possible next move and for each of those moves, it will also simulate their opponent's next possible moves and so on until each path reaches an end. For the sake of simplicity, we will use a game of Tic Tac Toe as an example to illustrate this. The first player is X and the second is O.

Object of game is to get three of your symbols in a horizontal, vertical or diagonal row on a 3X3 game board with following rules.

- X always goes first
- Players alternate placing X's and O's on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled.

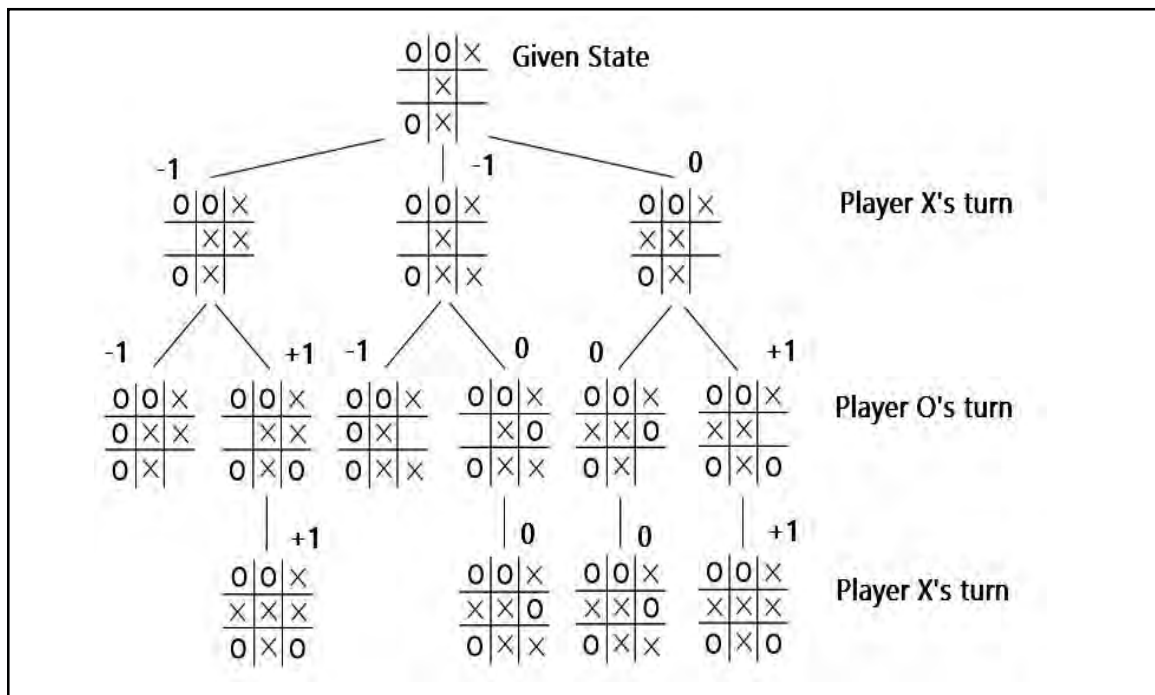


Fig. 3.12: Tic Tac Toe Game Between Player X and Player O

Tic Tac Toe is a great entry point to understand the Game Tree for many reasons. It is a two-player game with a relatively small range of outcomes and it will always result in a win, lose or draw. The above figure 3.12, illustrates a game between Player X and Player O at the root and all of the possible moves and outcomes are computed using the Minimax algorithm given below. In this scenario, +1 represents a favourable outcome for Player X, 0 represents a draw, and -1 represents a favourable outcome for Player O. In other words, Player X is trying to maximize and Player O is trying to minimize. Take a look at the below pseudocode to see how the Minimax algorithm works:

```

minimax(player, board)
    if(game over in current board position)
        return winner

    if(max's turn)
        return maximal score of calling minimax on all the children

    else (min's turn)
        return minimal score of calling minimax on all the children

```

The first block of code checks to see if the current state results in a win or loss. If this is true, the function will simply return the outcome. Next, it will execute either the second or third block of code depending on whether it is maximizing or minimizing. Since earlier example shows Player X's turn, let's walk through maximizing Player (second) block. The goal of this function is to return the move that holds the highest (maximum) value. In order to do this, we must evaluate

The status of the leaf nodes is considered in accordance with the rules of the game as W or L from MAX player's point of view. It should be noted that all the nodes on the winning path are labelled as W and losing path as L. We have seen that we can propagate this winning or losing right up to the top and then decide about which move to be taken. At this point of time, the best winning strategy for MAX player is the path which is marked with thick line. Any alternative options will always make the opponent to push the MAX player towards losing path which eventually leads opponent to win the game.

3.4 BOUNDED LOOK AHEAD STRATEGY

For most of the games, it is not practically possible to generate a game tree from top to the root and then deciding on optimal first move, especially when the game state space size is very large, like chess, where nearly 10000 non terminal nodes are generated. In that case one may use look ahead strategy to expand these moves up to a certain depth and then make use of heuristic functions to evaluate the looked ahead position and associated heuristic cost function of the game. This will indicate the probability of winning from that position using MINIMAX search to find out the best possible move.

The Minimax algorithm is the most well-known strategy of play of two-player, zero-sum games. The Minimax theorem was proven by John von Neumann in 1928. Minimax is a strategy of always minimizing the maximum possible loss which can result from a choice that a player makes.

3.4.1 MINIMAX Procedure Using Heuristic Function

MINIMAX is a decision-making strategy under game theory which is used to minimize the losing chances in a game and to maximize the winning chances. This strategy is also known as 'MINMAX or 'Saddle point'. Basically, it is a two-player, zero sum game strategy where if one wins, the other lose the game. This strategy simulates those games that we play in our day-to-day life. Like if two persons are playing chess, the result will be in favour of one player and will unfavoured the other one. The person who will make his best try, efforts as well as cleverness will surely win.

Let us understand MINIMAX strategy using a game tree as shown in Fig. 3.14, where the nodes represent the states of the game and edges represent the moves made by the players in the game. There will be two players namely, MIN and MAX. MIN-Decrease the chances of MAX to win the game. MAX-Increases his chances of winning the game. They both play the game alternatively, turn by turn and follow the above strategy, i.e., if one wins, the other will definitely lose. Both players look at one another as competitors and will try to defeat one-another giving their best.

In MINIMAX strategy, the result of the game or the utility value is generated by a heuristic function by propagating from the initial node to the root node. This heuristic evaluation function is used for each node up to certain depth and then obtain the values corresponding to the state. In a given state if the MAX player has upper hand, then, the score of the board will tend to be some positive value. If the MIN player has the upper hand in that board state, then it will tend to be

some negative value. MINIMAX procedure is also known as “depth first depth limited” search procedure. It follows the backtracking technique and backtracks to find the best choice. MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX’s utility value.

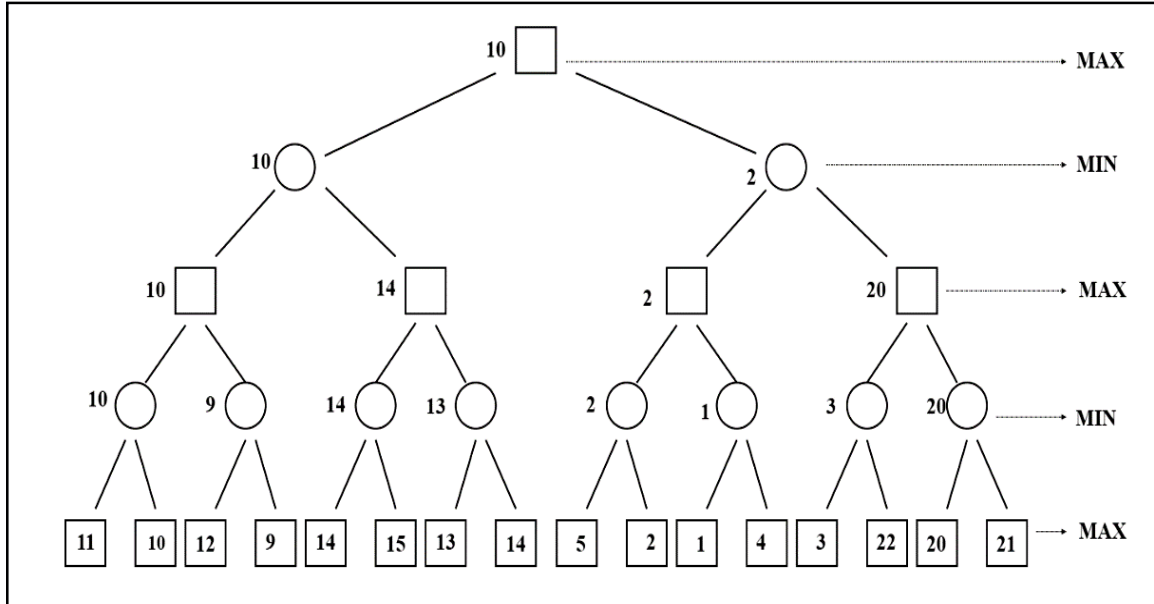


Fig. 3.14: Game Tree with MINIMAX Search

The two players MAX and MIN starts the game by choosing one path and propagating all the nodes of that path. Now, MAX will backtrack to the initial node and choose the best path where his utility value will be the maximum. After this, it's MIN's chance where MIN will also propagate through a path and again will backtrack. But MIN will choose the path which could minimize MAX winning chances or the utility value. In the Fig. 3.14, if MAX selects left path, then it becomes MIN's turn where MIN has a choice between 10 and 14. Being the minimizer he will definitely choose the least among both, that is 10. If MAX selects right path, then during MIN's turn, the MIN will have a choice between 2 and 20. Here MIN will choose 2 as it is the least among the two values.

3.4.2 MINIMAX Algorithm

MINIMAX algorithm is a backtracking algorithm where it backtracks to pick the best move out of several choices. MINIMAX strategy follows the DFS (Depth-first search) concept. Here, the game is played alternatively between MIN and MAX, i.e., when MAX made a move, then the next turn is of MIN. It means the move made by MAX is fixed and cannot be changed. Algorithm steps of this procedure may be written as follows.

- Keep on generating the game tree/search tree till a limit.
- Compute the move using a heuristic function.

- Propagate the values from the leaf node till the current position following the minimax strategy.
- Make the best move from the choices.

3.5 ALPHA-BETA PRUNING

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. The purpose of applying this procedure is to reduce the amount of work done in generating useless nodes.

This is a technique by which we can compute the correct minimax decision without checking each node of the game tree and this technique is called pruning. This procedure requires two threshold values, Alpha and Beta for future expansion. Hence it is known as alpha-Beta pruning. It is also called backward pruning. Alpha-beta pruning can be applied at any depth of a tree and sometimes it not only prunes the tree leaves but also entire sub-tree.

Alpha represents the best (highest-value) choice found at any point along the path of MAX node whereas Beta is the best (lowest-value) choice found at any point along the path of MIN node. Each MAX node has an alpha value which never decreases and each MIN node has a beta value which never increases. The Alpha-beta pruning to a standard minimax algorithm returns the same move as the other standard algorithm does. but it removes all the nodes which are not really affecting the final decision but tend to make algorithm slow. Hence, by pruning these nodes, we can make algorithm fast.

3.5.1 Procedure for Alpha-Beta Pruning

Let us consider a game tree as shown in Fig. 3.15 to understand the systematic propagation of alpha and beta values leading to alpha beta pruning up to third level in step-wise, depth first order. Important key Points to be considered are as follows.

- The MAX player will only update the value of alpha.
- The MIN player will only update the value of beta.

While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta. We will only pass the alpha, beta values to the child nodes.

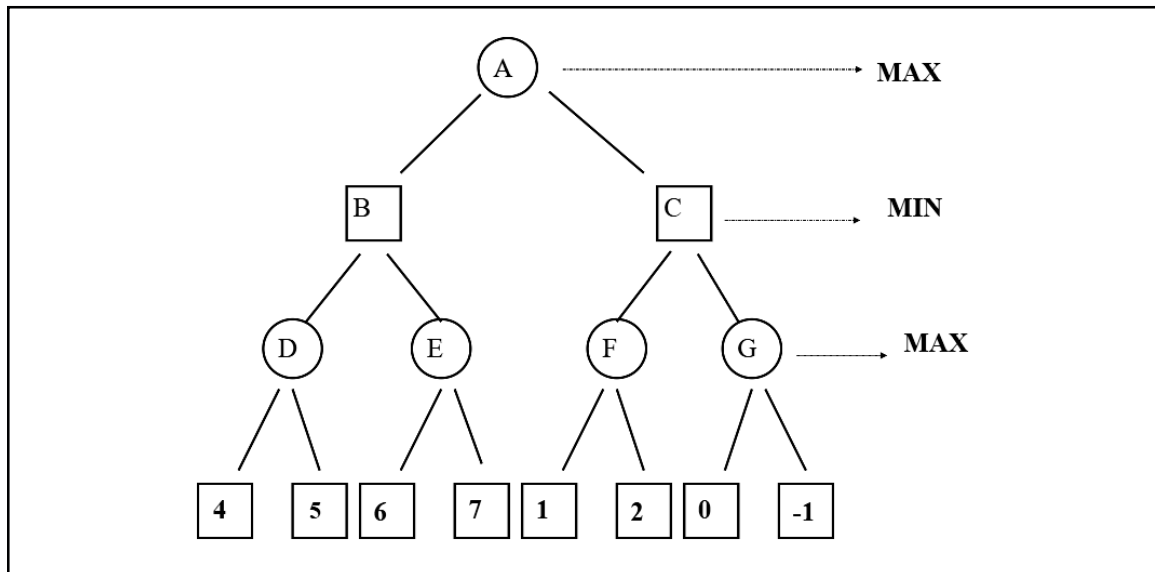


Fig. 3.15: Game Tree with Max Player First

First Step:

The Max player will start first move from node A where $\alpha = -\infty$ and $\beta = +\infty$. These values are passed down to subsequent nodes in the tree. At A the MAX player must choose MAX of B and C and A calls B first. At B, MIN player must choose MIN of D and E and therefore, calls D first. At D, it looks at its left path which is a leaf node and this node returns a value of 4. Now the value of alpha at D is MAX (∞ , 4) which is 4. To decide whether it's worth looking at its right node or not, it checks the condition $\beta \leq \alpha$. This is false because $\beta = +\infty$ and $\alpha = 4$. So, it continues the search. D now looks at its right path which returns a value of 5. At D, $\alpha = \text{MAX}(4, 5)$ which is 5. Hence, the value of node D is 5. D returns a value of 5 to B. At B, $\beta = \text{MIN}(+\infty, 5)$ which is 5. The MIN player is now guaranteed a value of 5 or lesser. B now calls E to see if he can get a lower value than 5. At E the values of alpha and beta is not $-\infty$ and $+\infty$, but instead $-\infty$ and 5 respectively. Because the value of beta was changed at B and that is what B passed down to E. Now E looks at its left path which is 6. At E, $\alpha = \text{MAX}(-\infty, 6)$ which is 6. Here the condition becomes true. beta is 5 and alpha is 6. So, $\beta \leq \alpha$ which is true. Hence, it breaks and E returns 6 to B.

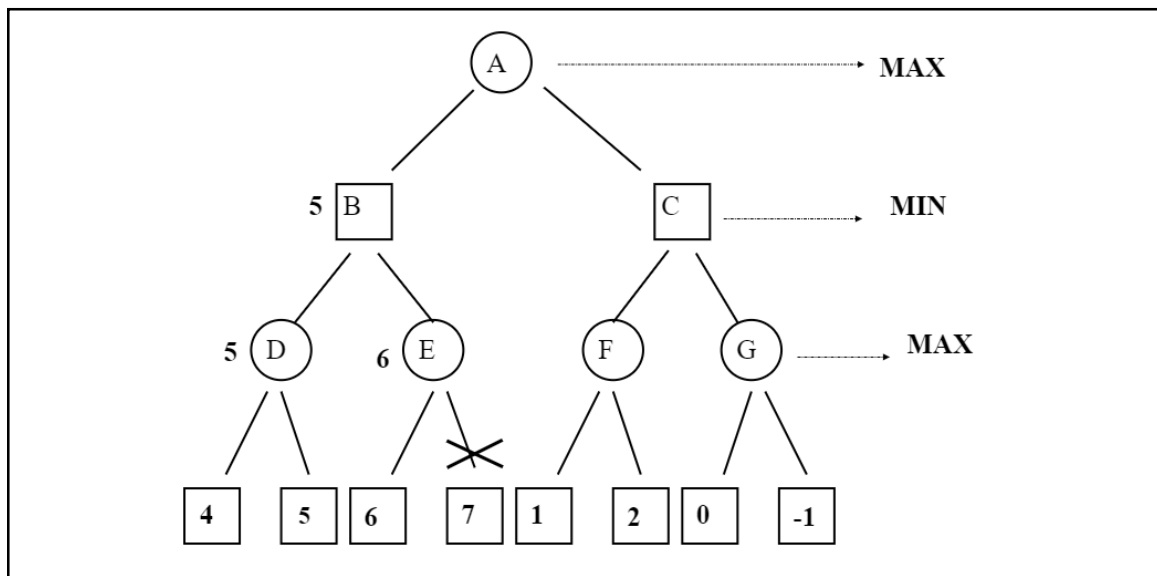


Fig. 3.16: Alpha Beta Pruning: First Step

It should be noted that MIN player was guaranteed a value of 5 or lesser no matter what the value of E's right path has. As soon as the MAX player saw the 6, he knew that the MIN player would never come this way because he can get a 5 on the left side of B. This way we can prune E's right path 7 and hence, saved computation time. Node E returns a value of 6 to B. At B, $\beta = \text{MIN}(5, 6)$ which is 5. The value of node B is also 5.

Second Step:

In the second step, B returns 5 to A. At A, $\alpha = \text{MAX}(-\infty, 5)$ which is 5. Now the MAX player is guaranteed a value of 5 or greater. A now calls C to see if it can get a higher value than 5. At C, $\alpha = 5$ and $\beta = +\infty$. Therefore, C calls F. At F, $\alpha = 5$ and $\beta = +\infty$. F looks at its left path which is 1. $\alpha = \text{MAX}(5, 1)$ which is still 5. F looks at its right path which is 2. Hence the best value of this node is 2. Alpha still remains 5 and F returns a value of 2 to C. At C, $\beta = \text{MIN}(+\infty, 2)$. The condition, $\beta \leq \alpha$ becomes true as $\beta = 2$ and $\alpha = 5$. So, further computation of G is pruned. The idea behind this pruning is that at C, the MIN player was guaranteed a value of 2 or lesser. But the MAX player was already guaranteed a value of 5 if he chooses B.

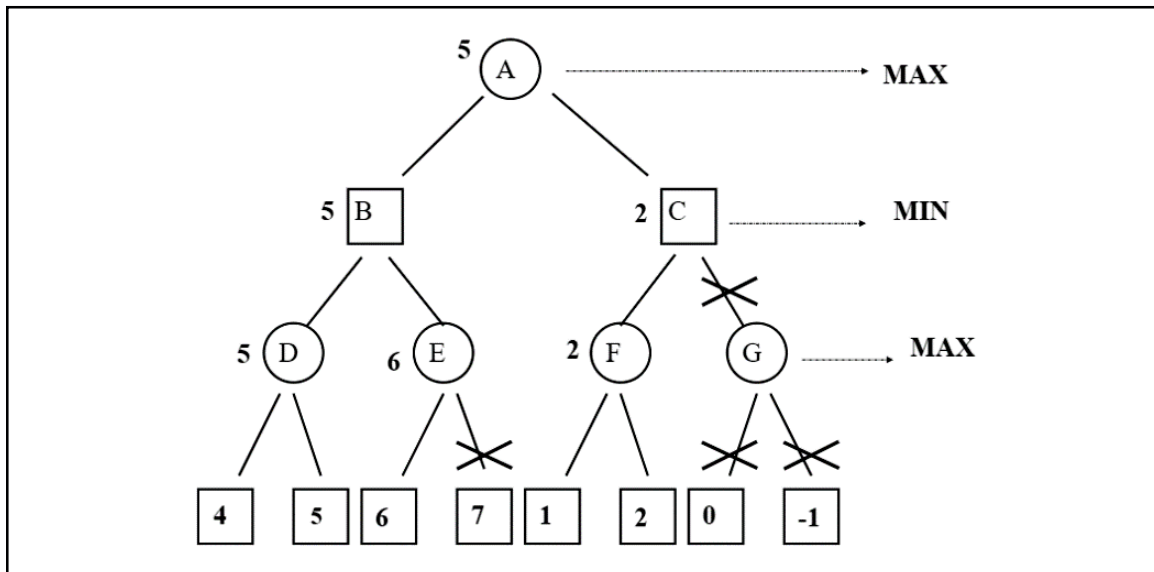


Fig. 3.17: Alpha Beta Pruning: Second/Final step

Fig. 3.17 shows final game tree after the second step. It should be noted that MAX player would never ever choose C to get a value less than 2. Hence, it saved a lot of computation by skipping a whole sub tree of G. C now returns a value of 2 to A. Therefore, the best value at A is MAX (5, 2) which is 5. Hence, the optimal value that the MAX player can get is 5.

3.5.2 Refinements to Alpha-Beta Pruning

We have seen how alpha beta pruning helps to improve the efficiency of the MINIMAX search algorithm. In fact, it is possible to improve the functioning of algorithm by providing few additional refinements to the process of searching. In this context, we provide a few of the possible refinements. There are a few simple refinements which can be deployed while playing game playing algorithm.

- First is waiting for stable states where the change of heuristic values across layers is not drastic.
- Second is to use secondary search to avoid horizon effect.
- Third is to use book moves for typical non-search part of the game playing.
- Fourth is to use other than MINIMAX algorithm.

Waiting for Stability

We have already stated that it is important for us to learn when to stop as it is impossible to reach to a leaf node of a real-life game tree. One good measure of the stability is that the heuristic estimates do not change much over a period of time during exploration of next levels.

From the above example, we can understand that at first or second level the values are changing drastically and after exploring 3rd Look Beyond the Horizon level, we can see that the nodes are not changing much and almost all nodes are showing similar values. It does probe and

decide not only the best move but stops only when it gets one move significantly better than others.

Go beyond the Horizon effect

The horizon effect is a term that was used due to a limitation in some of the early chess programs. There was a certain point from which the computer program could not see beyond and thus would wind up losing. For example, suppose the program was at maximum and can only able to see 4th ply as its effect. The machine might not have seen that at move 5th it would get into a bad position because it could only see up to 4th ply. It is impossible to explore one more ply in these circumstances as exploring one more ply requires huge memory and time requirements. One solution to this problem is to explore the node that we have chosen to play for a few plies further in place of exploring the entire game tree.

Predetermined Moves

It is also possible to have a database of moves for a given state and the next state is produced without having search using algorithms like MINIMAX, but picked up from the database. Such a move is known as book move and is pretty useful in start and end games of chess.

3.6 TWO PLAYER PERFECT INFORMATION GAMES

CHESS

Chess is not just a game. Over one-and-a-half millennia of years, chess has been known as a tool of military strategy and a benchmark of genius. There are some earliest records of chess in the 7th century. History reveals that the game's origins lie a century earlier. It is a two-player game with 64 squares in an 8×8 square. This game emerged with two key features where different rules for moving different types of pieces and a single king piece whose fate determines the outcome. The game was originally known as "chaturanga", a Sanskrit word which means "four divisions." But with its spread to Persia, it acquired its current name as "chess," derived from "shah," meaning king and "checkmate" from "shah mat" or "the king is helpless."

If the board is set up correctly, each piece will be facing one another. The queen across the queen and the king across the king. The player with the white pieces always moves first and makes one move and then black makes one move. Each player will make alternate moves like this until the game ends. Each of the 6 different pieces have a unique way of moving. The rook moves in a plus pattern which means it can move any number of squares either up, down, left or right. Rooks can capture the first enemy piece they move into. The bishop can move diagonally any number of squares or in an X-pattern. From this position, it can go to any of the squares. The queen is considered the most powerful piece on board because it can move both like the rook and Bishop. The knight is the only piece that can jump over other pieces. The knight moves in the shape of an L moving two squares in one direction except diagonal. The pawn is actually the most complicated piece because it has many unique qualities. A pawn is the only piece that moves in one direction and Pawns move one step forward towards the opponent's side of the board. The

king moves one square at a time in any direction. The objective of chess is to checkmate the king. The king is priceless because if you lose the king, you simply lose the game.

CHECKERS

Checkers is a two-player game who oppose each other across a board of 64 light and dark squares with disk-shaped 24 pieces of contrasting colours. Each player has 12 pieces arranged on the board. The pieces in checkers are usually red and black. The notation used in describing the game is based on numbering the squares on the board. The black pieces always occupy squares 1 to 12 and the white pieces invariably rest on squares 21 to 32. Players move pieces diagonally on the board trying to reach the opposite side where their pieces can be doubled giving them more powers. On the way, a player attempts to jump over her opponent's pieces and capture them. This game is also known as "draughts."

Play consists of moving a piece called kings, diagonally forward or backward to an adjoining vacant square. On the other hand, the pieces called men can be moved diagonally forward. A player can capture opponent's pieces by moving and jumping diagonally over them. When men pieces jump over the king pieces of the opponent, they become kings. Player wins when an opponent's pieces are all captured or blocked so that they cannot move. When neither side can force a victory and the trend of play becomes repetitious, a draw game is declared.

BACKGAMMON

Backgammon is a two-player board game that is played on a backgammon board. Board is divided in the middle with 12 triangular spaces called points in each quadrant. Each player sits on opposite sides of the board and has 15 colour coded checker pieces to play with. The goal of the game is to be the first to remove all of your pieces off the board which is also called "bearing off." To set the game up, the first player sets 3 pieces up in a vertical row at the base of the fifth point from their left in the bottom half of the board and 5 pieces in the first point directly to the bottom right of the board's partition. Then, player lines up 5 pieces at the left-most point at the top left and 2 pieces in the last point on the top right. The opposing player sets their colour's pieces up to mirror their opponent's pieces, so that each row of 2 has a mirror row of 2 on the opposite side. Each row of 3 has a mirror row of 3 and so on. The first player's pieces are only allowed to move counter clockwise starting in the top right. The second player's pieces are only allowed to move clockwise starting at the top on their left.

To start the play, players take alternating turns rolling 2 6-sided dice. For each of the numbers that gets rolled, the rolling player moves 1 or 2 pieces based on the numbers rolled on the dice. So, a roll of a 5 and 2 would allow the rolling player to move one piece 2 points and another piece 5 points. Alternatively, that player may move 1 piece 7 points. Rolling 2 of the same number allows that player to move 4 times. For example, a roll of 2 and 2 allows that player to move a total of 8 points, although each piece being moved must travel 2 points at a time. Players cannot move a piece to a point that is occupied by 2 or more of their opponent's pieces, but may stack their own pieces on a single point. If a player moves a piece to a point with only 1 of their opponent's pieces on it, that opposing player's piece is removed from the board and

placed on the middle partition. At the beginning of the next player's turn, they must use at least 1 of their rolls to remove that piece from the partition and start it over from the beginning point on their side of the board. Once all of a player's pieces are in the quadrant of the board closest to the end, that player may begin removing pieces off of the board. Once a player moves a piece 1 point past the edge of the board, it is removed from the game. A player wins once all of their pieces are removed from the board.

QUESTIONS

- 3.1 Explain the Formulation of the Problem Reduction Search Problem.
- 3.2 Explain the node status labelling procedure taking heuristics into account.
- 3.3 What is Game playing? Explain its types.
- 3.4 Explain the status labelling procedure in Game Tree.
- 3.5 What is Bounded look ahead strategy? Explain MINIMAX procedure using heuristics function.
- 3.6 Explain Alpha- Beta Pruning with an example.



4

LOGIC CONCEPTS

4.1 INTRODUCTION

Logic is very important for AI as all flexible intelligent machines need to know the facts in order to operate and to accomplish tasks. All intelligent machines need to reason with the given knowledge in order to choose the optimal outcome. This knowledge can be classified as declarative knowledge and procedural knowledge.

Procedural knowledge involves knowing how to do something while declarative involves knowing what to do something. Many kinds of reasoning can be done with declarative statements such that the intelligent machine is able to make educated guesses, reason by analogy, jump to conclusions and make generalizations. All this requires a programming paradigm based on logic.

Logic is defined as a system that aims to draw reasonable conclusions based on given information. This means the goal of logic is to use data to make inferences. For example, “Suraj saw a black cat on his way to work. And at work, Suraj got fired”. Here, logic would dictate that from the given information presented, “Black cats are of bad luck”. There is no direct evidence or confession, but logical principles reveal what is true based on the given information. Therefore, Logic process takes in some information called premises and produce some output called conclusions which means the concept of logic relies solely on given data and valid correlations based on the governing principles presented. There are many types of logic located within the governing science. The four main logic types are:

Informal logic: Informal Logic is the reasoning and arguments you make in your personal exchanges with others. For example, “my mom is an actress. I live with my mom”. Conclusion here is, “I am an actress”.

Formal logic: Formal Logic is the reasoning in which premises must be true. You follow the premises to reach a formal conclusion. For example, “Bicycles have two wheels. Kiran is riding a bicycle”. Conclusion here is “Kiran has two-wheeler”.

Symbolic logic: Symbolic logic is the method of representing logical expressions through the use of symbols and variables instead of natural language such as English. It provides the benefit of removing the ambiguity that is generally seen in ordinary languages. There are two types.

- (i) Propositional Logic
- (ii) Predicate Logic

Mathematical logic: In mathematical logic, we apply formal logic to math. This type of logic is part of the basis for the logic used in computer science. Mathematical logic and symbolic logic are often used interchangeably.

4.2 PREPOSITIONAL CALCULUS (LOGIC)

The simplest and most basic branch of logic is the Propositional Calculus. It is Abbreviated as PC and so named because it deals only with complete, unanalysed propositions and certain combinations into which they enter. Various notations for PC are used in the literature. The symbols employed in PC includes first variables (for which the letters p, q, r, ... are used, with or without numerical subscripts). Second, Operators (for which the symbols $\neg$, $\vee$, $\supset$, and $\equiv$ are employed) and third, brackets or parentheses. The rules for constructing formulas are discussed below.

Logic and Language

Logic is the study of valid arguments. An argument consists of a set of statements called premises and an assertion called conclusion.

For example:

Geetha is in the library or in the coffee bar Premise

Geetha is not in the coffee bar Premise

Therefore, Geetha is in the library ... Conclusion

We say that an argument is valid, when true premises would never produce a false conclusion. Valid arguments are important as they define rules of inference. These rules allow us to obtain conclusions from certain premises. Whenever the premises are true, a rule of inference generate only true conclusions. Rules of inference have the important property of preserving truth. Consider the following example:

If Alok has hepatitis, then his skin colour will become yellow

Alok has a yellow colour skin

Therefore, Alok has hepatitis

In this example the argument is not valid. There are many reasons why the skin of Alok may have a yellow colour (maybe he is using makeup), without having hepatitis.

Proposition

A proposition is a declarative statement which is either universally true or universally false. For example,

“It rained yesterday” // It returns truth value “TRUE”

“ $12 + 9 = 3 - 2$ ”, // It returns truth value “FALSE”

The following is not a Proposition –

“ $X^2 = 10$ ” // It is because unless we give a specific value of X , we cannot say whether the statement is true or false.

“Please report at 11 a.m.” // It is not a declarative statement. Its direction

“What is your name?” // Queries are not declarative statements

Propositional Variable

The variables are to be viewed as representing unspecified propositions or as marking the places in formulas into which sentences may be inserted. (This is sometimes expressed by saying that variables range over propositions or that they take propositions as their values). Hence, they are often called propositional variables. It is assumed that every proposition is either true or false and that no proposition is both true and false. Truth and falsity are said to be the truth values of propositions. Variables which are used to represent proposition is called propositional variables. We denote the propositional variables by capital letters (A , B , etc).

For example,

A : India is a democratic country.

B : $5 + 5 = 10$

Connectives

In propositional logic, the relationships between propositions are represented by connectives which means a statement or proposition can be combined by means of logical connectives to form a statement called compound statements. There are essentially five different connectives which are outlined in the following Table 4.1.

Table 4.1: Connectives

Connective	Symbol	Description
Negation	$\neg$	NOT
Conjugation	$\wedge$	AND
Disjunction	$\vee$	OR
Conditional	$\rightarrow$	If ... then
Biconditional	$\leftrightarrow$	If and only if

For example, suppose we want to say “If it rains, then Kiran won’t go to school”. We would first represent the two propositions as a proposition variable.

A : It rains.

B : Kiran won’t go to school.

Then we would use the conditional connective to make our statement. $A \rightarrow B$.

4.3 TRUTH TABLE

In order to clarify the meaning of a proposition or a connective, a truth table is used. Truth tables are a way of visualizing the truth values of propositions. A value of true is represented by “1” and a value of false is represented by “0”.

OR ($\vee$)

The OR operation of two propositions A and B (written as $A \vee B$) is true if at least any of the propositional variable A or B is true. For example, consider the following propositions:

A: Sumith wears black shirt.

B: Sumith has a bicycle.

C: Sumith wears black shirt and Sumith has a bicycle.

Proposition C takes on the following truth values.

- If Sumith doesn't wear black shirt and doesn't have a bicycle, then proposition C is false.
- If Sumith doesn't wear black shirt but has a bicycle, then proposition C is true.
- If Sumith wears black shirt but doesn't have a bicycle, then proposition C is true.
- If Sumith wears black shirt but Sumith has a bicycle, then proposition C is true.

The truth table is shown in Table 4.2.

Table 4.2: Truth Table for OR

A	B	$A \vee B$
False	False	False
False	True	True
True	False	True
True	True	True

AND ($\wedge$)

The AND operation of two propositions A and B is true if both the propositional variable A and B is true. Truth table for above example is shown in Table 4.3.

Table 4.3: Truth Table for AND

A	B	$A \wedge B$
False	False	False
False	True	False
True	False	False
True	True	True

Negation ($\neg$)

The negation of a proposition A is false when A is true and is true when A is false. For example, Consider a proposition.

A: You are either rich or happy.

For this statement to be false, you can't be rich and you can't be happy. In other words, the opposite is to be not rich and not happy. Or if we rewrite it in terms of the original statement we get "You are not rich and not happy". Truth table is shown in Table 4.4.

Table 4.4: Truth Table for Negation

A	$\neg A$
True	False
False	True

Conditional/Implication/If-then ($\rightarrow$)

An implication is the proposition "if A, then B". It is false if A is true and B is false. The rest cases are true. For example, suppose we want to say "If the sky is clear, then we will be able to see the stars". We would first represent the two propositions as a proposition variable.

A: Sky is clear

B: We will be able to see the stars

This is a logical condition and the truth table is shown in Table 4.5.

Table 4.5: Truth Table for Implication

A	B	$A \rightarrow B$
False	False	True
False	True	True
True	False	False
True	True	True

Biconditional If and Only If ($\leftrightarrow$)

It is a bi-conditional logical connective which is true when A and B are same, *i.e.*, both are false or both are true. For example, "If I have a triangle, then my polygon has only three sides"

A: I have a triangle

B: My polygon has only three sides

The truth table is as shown in Table 4.6.

Table 4.6: Truth Table for Biconditional

A	B	$A \leftrightarrow B$
True	True	True
True	False	False
False	True	False
False	False	True

In summary, Table 4.7 showing the functionality of all of the connectives.

Table 4.7: Truth Table for Logical Operators

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

4.3.1 Applications of Truth Table

We can use truth tables to determine the structure of a logical argument as valid or not. In order to conclude the structure of a logical argument to be valid, we first need to translate our argument into a series of logical statements written using letters and logical connectives. In the following example we will understand how to translate a sentence into propositional logic statement. Consider a simple statement like —

“If I go to the store or the movies, I won’t do my homework”.

To translate this statement into logical statement, we have to look at connectives in the statement which are connecting two propositions together. Then we have to rename the proposition using proposition variables. For above statement, Let P = “I go to the store”, Q = “I go to the movies, R = “I do my homework,” then we can rewrite this argument using letters and logical connectives as shown in Table 4.8.

Table 4.8: Logical Connectives

I go to the store	P
I go to the movies	Q
I go to the store or the movies	$P \vee Q$
If I go to the store or the movies,	$(P \vee Q) \rightarrow$
I won’t do my homework	$\neg R$
If I go to the store or the movies, I won’t do my homework	$(P \vee Q) \rightarrow \neg R$

Once it is done, then truth table can be created to understand the validity of an argument. Let us consider another example to determine the valid structure of the argument.

“ $x \neq y$. $x = 4$ if and only if $x = y$. If y is not an even number, then $x = 4$. Hence, y is an odd number”. Let us consider propositional variable P, Q and R.

Let P: “ $x = y$ ”, Q: “ $x = 4$ ” and R: “ y is an even number,” then we can rewrite this argument using letters and logical connectives as shown in Table 4.9.

Table 4.9: Logical Connectives

$x \neq y$	$\neg P$	Premise
$x = 4$ if and only if $x = y$	$Q \leftrightarrow P$	Premise
If y is not an even number, then $x = 4$	$\neg R \rightarrow Q$	Premise
Hence, y is an odd number”.	$\neg R$	Conclusion

Note that the statements “y is not an even number” and “y is an odd number” mean the same thing and are therefore, equivalent.

Here, we have created a truth table for premise statement and conclusion statement together as shown in the Table 4.10. This table is used to compare the truth values of the premise statements with the conclusion statement. If the truth value of the conclusion is true in every case in which all of the premises are true, then the argument is considered to be a valid structure. If there is even one case in which all the premises are true but the conclusion is false, then it is clear that the truth of the conclusion does not follow directly from the premises and therefore, the argument is invalid.

Table 4.10: Truth Table

P	Q	R	$\neg P$	$Q \leftrightarrow P$	$\neg R \rightarrow Q$	$\neg R$
T	T	T	F	T	T	F
T	T	F	F	T	T	T
T	F	T	F	F	T	F
T	F	F	F	F	F	T
F	T	T	T	F	T	F
F	T	F	T	F	T	T
F	F	T	T	T	T	F
F	F	F	T	T	F	T

It should be noted that if there are no cases in which all the premises are true, then the argument is valid by default. Because if it is never possible for all of the premises to be true at the same time, then it is not possible to find a circumstance in which all the premises are true and the conclusion is false, so it is impossible to prove that the argument is invalid. therefore, the argument becomes valid by default.

4.4 TYPES OF PROPOSITIONS BASED ON TRUTH VALUES (LOGICAL EQUIVALENCE)

There are three types of propositions when classified according to their truth values. They are as follows.

- **Tautology:** A proposition which is always true, is called a tautology.
- **Contradiction:** A proposition which is always false, is called a contradiction.
- **Contingency:** A proposition that is neither a tautology nor a contradiction is called a contingency.

Equivalence relations or laws are used to reduce or simplify a given formula or to drive a new formula from existing formula. Some of the important equivalence laws are given in Table 4.11. Let P, Q and R be logical propositions.

Table 4.11: Equivalence Laws

Relation	Logical Equivalence
Commutative Laws	$P \vee Q \equiv Q \vee P$ $P \wedge Q \equiv Q \wedge P$
Associative Laws	$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$ $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$
Distributive Laws	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$ $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
Identity Laws	$P \vee F \equiv P$ $P \wedge T \equiv P$
Negation Laws	$P \wedge \neg P \equiv F$ $P \vee \neg P \equiv T$
Idempotent Laws	$P \vee P \equiv P$ $P \wedge P \equiv P$
Domination Laws	$P \wedge F \equiv F$ $P \vee T \equiv T$
Absorption Laws	$P \wedge (P \vee Q) \equiv P$ $P \vee (P \wedge Q) \equiv P$ $P \vee (\neg P \wedge Q) \equiv P \vee Q$ $P \wedge (\neg P \vee Q) \equiv P \wedge Q$
De Morgan's Laws	$\neg (P \vee Q) \equiv (\neg P) \wedge (\neg Q)$ $\neg (P \wedge Q) \equiv (\neg P) \vee (\neg Q)$
Double Negation Law	$\neg (\neg P) \equiv P$
Implication	$P \rightarrow Q \equiv \neg P \vee Q$

To understand laws of Equivalence let us work with some examples.

Example 4.1:

Check if $\neg (P \vee Q)$ and $\neg P \wedge \neg Q$ are logically equivalent.

Solution:**Table 4.12: Solution for Example 4.1**

P	Q	$P \vee Q$	$\neg(P \vee Q)$	$\neg P$	$\neg Q$	$\neg P \wedge \neg Q$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

From the Truth table we can clearly see that the propositions are logically equivalent.

Example 4.2:

Use the truth table method to determine whether $p \leftrightarrow q$ and $(p \wedge q) \vee (\neg p \wedge \neg q)$ are logically equivalent.

Solution:

The following table clearly shows that $p \leftrightarrow q$ and $(p \wedge q) \vee (\neg p \wedge \neg q)$ are logically equivalent.

Table 4.13: Solution for Example 4.2

p	q	$p \leftrightarrow q$	$(p \wedge q) \vee (\neg p \wedge \neg q)$
F	F	T	T
F	T	F	F
T	F	F	F
T	T	T	T

Example 4.3:

Three boxes are presented to you. One contains gold and other two are empty. Each box has imprinted on it a clue as to its contents. The clues are:

Box 1 “The gold is not here”

Box 2 “The gold is not here”

Box 3 “The gold is in Box 2”

Only one message is true. Other two are false. Which box has the gold? Formalize the puzzle in Propositional Logic and find the solution using a truth table.

Solution:

Let B_i with $i \in \{1, 2, 3\}$ stand for “gold is in the i^{th} box”. We can formalize the statements of the problem as follows:

1. One box contains gold, the other two are empty.

$$(B_1 \wedge \neg B_2 \wedge \neg B_3) \vee (\neg B_1 \wedge B_2 \wedge \neg B_3) \vee (\neg B_1 \wedge \neg B_2 \wedge B_3) \quad \dots (1)$$

2. Only one message is true; the other two are false.

$$(\neg B_1 \wedge \neg \neg B_2 \wedge \neg B_2) \vee (\neg \neg B_1 \wedge \neg B_2 \wedge \neg B_2) \vee (\neg \neg B_1 \wedge \neg \neg B_2 \wedge B_2) \quad \dots (2)$$

$$(2) \text{ is equivalent to: } (B_1 \wedge \neg B_2) \vee (B_1 \wedge B_2) \quad \dots (3)$$

Let us compute the truth table for (1) and (3)

Table 4.14: Solution for Example 4.3

B₁	B₂	B₃	(1)	(3)
T	T	T	F	T
T	T	F	F	T
T	F	T	F	T

T	F	F	T	T
F	T	T	F	F
F	T	F	T	F
F	F	T	T	F
F	F	F	F	F

It should be noted that constructing truth tables is not the only method for determining whether arguments are valid. For example, if, in a chain of reasoning, we had established “A and B,” it would seem perfectly reasonable to conclude B. If we had established A, B, and “If A and B then C,” it would be reasonable to conclude C. On the other hand, if we had established “A or B,” we would not be justified in concluding B without further information. Therefore, there are four general approaches which can help in proving the validity of the formula.

- Natural deduction system
- Axiomatic system
- Semantic tableau method
- Resolution refutation method

4.5 NATURAL DEDUCTION SYSTEM

Natural deduction, designed by Gerhard Gentzen in the 1930. This system is as close as human reasoning which we use in the day-to-day discourse. In this system the meaning of each logical operator is defined according to the way it is found in natural language discourse. Logic comes from the analysis of the way in which human beings use language. It is the way in which people connects sentences using linguistic operators such as “or”, “and”, “not” and the conditional “if... then.” Each logical operator has associated some inference rules. These inference rules are of two types: introduction rules and elimination rules. The introduction rules link two propositions using a particular operator and the elimination rules tell us when it is possible to eliminate such operator. In the following subsections the inference rules for the logical connectives will be introduced.

For example, one rule of our system is known as AND Introduction. To understand the meaning of AND, let’s have a look to the way we use the conjunction operator “and” in English through some example sentences.

- (i) Sunil was born in Bangalore Vasanth works in Delhi.

Therefore: Sunil was born in Bangalore and Vasanth works in Delhi.

The argument (i) shows that if we assume that both sentences are true, then the concatenation of both using the “and” will be as well true.

- (ii) Archana loves basketball and Ramesh likes playing cards.

Therefore: Ramesh likes playing cards.

In example (ii), we assert that if the concatenation of two sentences with the word “and” is true, then any of the sentences in the construction will be true by itself. Generalizing both

arguments (i) and (ii) by replacing the sentences for propositional variables P and Q respectively, it is possible to define the rules that allow the introduction and elimination of the “and” connective:

AND-Introduction

$$\frac{PQ}{P \text{ and } Q}$$

AND- Elimination

$$\frac{P \text{ and } Q}{P} \quad \frac{P \text{ and } Q}{Q}$$

The propositions above the line are called premises. The proposition below the line is the conclusion. Both the premises and the conclusion may contain metavariables (in this case, P and Q) representing arbitrary propositions. Here, we list the premises, but not the conclusion. We then apply natural deduction rules to the premises until we are able to write the conclusion of the argument. The natural deduction rules are truth preserving. Thus, if we are able to construct the conclusion by applying them to premises, the truth of the conclusion is entailed by the truth of the premises and so the argument is valid. If we cannot derive the conclusion from the premises however, we cannot conclude anything. Then argument may indeed be invalid, but then again, we may have to find a way to derive it.

Conditional Proof

A deductive proof is a step-by-step demonstration that a given argument is valid. At each step we apply rules of inference. We will justify the introduction of these rules by using truth tables to show that they arise from valid arguments. But it should be stressed that a proof of validity by deduction has no connection to a proof of validity by truth tables. The following rules should be memorized.

Rules of Conditional Proof (RCP)

Supposing from some assumption A we obtain a line B and B is a tautological consequence from A, then $A \rightarrow B$ is a tautology. Note that if B is not a tautological consequence from A, then $A \rightarrow B$ is not tautologous.

Reductio Ad Absurdum (RAA)

Supposing from assumption A we reached a line where we have B and $\neg B$ as tautological consequence then $A \rightarrow (B \wedge \neg B)$ is a tautology. In that case A must always be false ($\perp$) thus $\neg A$ is a tautology.

4.5.1 Natural Deduction Rules

If P and Q represents propositional variables, then we can present all the Inference rules as shown in the Table 4.15.

Table 4.15: Natural Deduction Inference Rules

Name of Rule	Symbol	Rule	Description
Rule of Assumption	A	Insert any formula at any stage into a proof. The assumed formula rests upon the assumption of itself.	
AND Introduction	I: $\wedge$	$\frac{PQ}{P \text{ and } Q}$	If P is true, Q is True then P and Q is True
AND Elimination	E: $\wedge$	$\frac{P \text{ and } Q}{P} \quad \frac{P \text{ and } Q}{Q}$	If P and Q is True then P is True, Q is True
OR Introduction	I: $\vee$	$\frac{P}{P \text{ or } Q}$	If P is true then P or Q must be True
OR Elimination	E: $\vee$	$\frac{P \text{ or } Q \quad \frac{P}{R} \quad \frac{Q}{R}}{R}$	P implies Q and R implies Q then if either P or R is true, then Q has to be True
Conditional Introduction	I: $\rightarrow$	$\frac{\frac{P}{Q}}{P \rightarrow Q}$	If from supposing P to be true, you may conclude Q, then it is possible to assert "if P then Q."
Conditional Elimination Modulus Ponens do Ponens (MPP)	E: $\rightarrow$	$\frac{P \rightarrow Q \quad P}{Q}$	If P, then Q. P is true. Therefore, Q is true.
NOT Introduction	I: $\neg$	$\frac{P \rightarrow Q \quad P \rightarrow \neg Q}{\neg P}$	If P implies that Q is true and also P implies that Q is not true, then P must itself not be true
NOT Elimination	E: $\neg$	$\frac{\text{NOT } \text{NOT } P}{P}$	Negation of negation is true
Modulus Tollendo Tollens (MTT)	(MTT)	$\frac{P \rightarrow Q \quad \neg Q}{\neg P}$	If P, then Q. Not Q. Therefore, not P
Biconditional Introduction	I: $\leftrightarrow$	$\frac{P \rightarrow Q \quad Q \rightarrow P}{P \leftrightarrow Q}$	If $P \rightarrow Q$ is true, and if $Q \rightarrow P$ is true, then one may infer that $P \leftrightarrow Q$ is true.
Biconditional Elimination	E: $\leftrightarrow$	$\frac{P \leftrightarrow Q}{P \rightarrow Q \quad Q \rightarrow P}$	If $P \leftrightarrow Q$ is true, then, $Q \rightarrow P$ is true, $P \rightarrow Q$ is true.

Truth Table v/s Natural Deduction System

Let us compare the truth table and Natural deduction system.

Truth Table	Natural Deduction System
Simple and mechanical	Suitable for human use
Large proofs of validity	Often allows short proofs for validity
Unsuited to human use	Arguably too flexible for machines use. It is not mechanical, a large number of rules.

To understand natural deduction rules, let us show some examples.

Example 4.4:

Show that the following argument is valid.

$(A \vee B) \rightarrow (C \vee D), A \vee B, (C \vee D) \rightarrow G \vdash G$

Proof:

Table 4.16: Solution for Example 4.4

Line Number	Formula	Rule
Theorem	$(A \vee B) \rightarrow (C \vee D), A \vee B, (C \vee D) \rightarrow G \vdash G$	
1	$A \vee B$	A
2	$(A \vee B) \rightarrow (C \vee D)$	A
3	$C \vee D$	MPP 1,2
4	$(C \vee D) \rightarrow G$	A
5	G	MPP 3,4

Hence, the argument is valid.

Example 4.5:

Show that the following argument is valid.

$(A \vee B) \rightarrow (C \vee D), \neg(C \vee D), (\neg G) \rightarrow (A \vee B) \vdash G.$

Proof:

Table 4.17: Solution for Example 4.5

Line Number	Formula	Rule
Theorem	$(A \vee B) \rightarrow (C \vee D), \neg(C \vee D), (\neg G) \rightarrow (A \vee B) \vdash G$	
1	$\neg(C \vee D)$	A
2	$(A \vee B) \rightarrow (C \vee D)$	A
3	$\neg(A \vee B)$	MTT 1,2
4	$(\neg G)$	A
5	$\neg(\neg G)$	MTT 3,4
6	G	E: $\neg, 5$

Example 4.6:

Show that the following argument is valid.

$$p \rightarrow (s \rightarrow (\neg r)), p \rightarrow r, p \vdash \neg s$$

Proof:

Table 4.18: Solution for Example 4.6

Line Number	Formula	Rule
Theorem	$p \rightarrow (s \rightarrow (\neg r)), p \rightarrow r, p \vdash \neg s$	
1	p	A
2	$p \rightarrow r$	A
3	r	MPP 1, 2
4	$p \rightarrow (s \rightarrow (\neg r))$	A
5	$s \rightarrow (\neg r)$	MPP 1, 4
6	$\neg (\neg r)$	E: $\neg$, 3
7	$\neg s$	MTT 5, 6

Therefore, the argument is valid.

4.6 SEMANTIC TABLEAU

The truth table testing method outlined and illustrated above is easy in simple cases but when sequent get intricate, it becomes difficult to manage without the aid of a computer. A more readily usable technique is to grow truth value assignments on trees, which are called *semantic tableau*. These are designed to make semantic reasoning fully systematic and to generate decisions automatically as to whether sequent are valid or not.

Previous approaches require construction of proof of a formula from given set of formulae and are called direct methods. In semantic tableaux approach the set of rules are applied systematically on a formula or set of formulae to establish its consistency or inconsistency. A Semantic tableau is a binary tree constructed by using semantic rules with a formula as a root which can be true. Formulas can be expanded based on the structure of the compound formulas. If all branches in the tableau lead to a contradiction, then there is no way the conjunction of the formulas at the root can be true. A path of the tree represents the conjunction of the formulas along the path. Semantic tableaux were invented by E.W. Beth and J. Hintikka (1965). The rules are based on the logical connectives in the Formulas. Semantic tableaux are based on the idea of proof by contradiction. It is a refutation-based system and it is a form of backward proof, because we start from the conclusion and decompose it and the premises into to smaller and smaller parts until we reach a contradiction. Consistency and Inconsistency requirements are as follows.

- If an atom P and $\neg P$ appear on a same path of a semantic tableau, then it is indicated as inconsistent and such path is said to be contradictory or closed path. Even if one path remains non contradictory or unclosed (open), then the formula at the root of a tableau is consistent.

- Contradictory tableau is defined to be a tableau in which all the paths are contradictory or closed.
- If a tableau for a formula at the root is a contradictory tableau, then a formula is said to be inconsistent.
- If a tableau with root $(p_1 \wedge p_2 \wedge \dots \wedge p_n)$ is contradictory then set of formulae $\{p_1, p_2, \dots, p_n\}$ is said to be inconsistent.
- Set of formulae $\{p_1, p_2, \dots, p_n\}$ is said to be consistent if the formulae in the set are true and table has at least one open path.

If formula A is tableau provable from set P, then A is said to be a logical consequence of P.

4.6.1 Semantic Tableau Rules

The Semantic Tableau Rules are given in Table 4.19 where p and q are two formulae.

Table 4.19: Semantic Tableau Rules

Rules	Tableau tree	Description
Rule 1 for Conjunction	$ \begin{array}{c} p \wedge q \\ \text{AND} \quad \\ p \\ \\ q \end{array} $	$p \wedge q$ is true if both p and q are true.
Rule 2 for Conjunction	$ \begin{array}{c} \neg (p \wedge q) \\ \text{NOT-AND} \quad \swarrow \quad \searrow \\ \neg p \quad \neg q \end{array} $	$\neg (p \wedge q)$ is true if either $\neg p$ or $\neg q$ is true.
Rule 3 for Disjunction	$ \begin{array}{c} p \vee q \\ \text{OR} \quad \swarrow \quad \searrow \\ p \quad \quad q \end{array} $	$p \vee q$ is true if either p or q is true.
Rule 4 for Disjunction	$ \begin{array}{c} \neg (p \vee q) \\ \text{OR} \quad \\ \neg p \\ \\ \neg q \end{array} $	$\neg (p \vee q)$ is true if both $\neg p$ or $\neg q$ are true.

Rule 5 for Negation	$\begin{array}{c} \neg \neg p \\ \text{NOT-NOT} \mid \\ p \end{array}$	$\neg(\neg p)$ is true then p is true.
Rule 6 for Implication	$\begin{array}{c} p \rightarrow q \\ \text{IMPLIES} \swarrow \searrow \\ \neg p \quad q \end{array}$	$p \rightarrow q$ is true then $\neg p \vee q$ is true
Rule 7 for Implication	$\begin{array}{c} \neg(p \rightarrow q) \\ \text{NOT-IMPLIES} \mid \\ p \\ \mid \\ \neg q \end{array}$	$\neg(p \rightarrow q)$ is true then $p \wedge \neg q$ is true
Rule 8 for Equivalence	$\begin{array}{c} p \leftrightarrow q \\ \text{IFF} \swarrow \searrow \\ p \wedge q \quad \neg p \wedge \neg q \end{array}$	$p \leftrightarrow q$ is true then $(p \wedge q) \vee (\neg p \wedge \neg q)$ is true.
Rule 9 for Equivalence	$\begin{array}{c} \neg(p \leftrightarrow q) \\ \text{NOT IFF} \swarrow \searrow \\ p \wedge \neg q \quad \neg p \wedge q \end{array}$	$\neg(p \leftrightarrow q)$ is true then $(p \wedge \neg q) \vee (\neg p \wedge q)$ is true.

To understand the application of Semantic Tableau Rules, let us work with some examples.

Example 4.7:

Show that $(P \wedge \neg Q) \wedge (Q \rightarrow R)$ is consistent.

Solution:

Table 4.20: Solution for Example 4.7

Rule Application	Formula	Line number
Root	$(P \wedge \neg Q) \wedge (Q \rightarrow R)$	Line 1
Apply Rule 1 to Line 1	$(P \wedge \neg Q)$	Line 2
	$(Q \rightarrow R)$	Line 3
Apply rule 1 to Line 2	Q	

Apply rule 6 to Line 3	$ \begin{array}{c} \neg R \\ \swarrow \quad \searrow \\ \neg R \quad P \\ \text{(Open)} \quad \text{(Open)} \end{array} $	
------------------------	--	--

$(P \wedge \neg Q) \wedge (Q \rightarrow R)$ is consistent.

Example 4.8:

Show that $(A \wedge B \rightarrow C) \wedge (\neg A \rightarrow D) \wedge B \wedge \neg C \wedge \neg D$ is inconsistent.

Solution:

Table 4.21: Solution for Example 4.8

Rule Application	Formula	Line number
Root	$(A \wedge B \rightarrow C) \wedge (\neg A \rightarrow D) \wedge B \wedge \neg C \wedge \neg D$	1
Apply Rule 1 to Line 1	$A \wedge B \rightarrow C$	2
	$\neg A \rightarrow D$	3
	B	
	$\neg C$	
Apply Rule 6 to 3 Apply Rule 6 to 2	$ \begin{array}{c} \neg D \\ \swarrow \quad \searrow \\ \neg \neg A \quad D \\ \swarrow \quad \searrow \quad \text{Closed: } \{D, \neg D\} \text{ on the path} \\ \neg (A \wedge B) \quad C \\ \swarrow \quad \searrow \quad \text{Closed: } \{C, \neg C\} \\ \neg A \quad \neg B \\ \text{Closed: } \{A, \neg A\} \quad \text{Closed: } \{B, \neg B\} \end{array} $	

Table 4.21 shows that this is Inconsistent as we have contradictory tableau

Example 4.9:

Show that Q is a logical consequence of $A = \{P \rightarrow Q, P\}$

Solution:

Let us consider $\neg Q$ as a root with A in the tableau tree.

Table 4.22: Solution for Example 4.9

Rule Application	Formula	Line number
Root	$\neg Q$	1
	$P \rightarrow Q$	2
Apply Rule 6 to 2	<pre> P / \ ¬ P Q Closed{ P, ¬ P} Closed{ Q, ¬ Q} </pre>	3

From Table 4.22, $\neg Q$ as a root gives contradictory tableau, thus Q is a logical consequence of A .

4.7 AXIOMATIC SYSTEMS

An axiomatic system is a list of undefined terms together with a list of statements or a set of well-formed formulas (WFF'S) called the axioms of PC that are assumed to be true without proof. These are the basic building blocks from which all theorems are proved. For example, the statement *all right angles are equal to each other* is an axiom and does not require a proof. We know that all right angles are equal to each other and we never try to prove it. Instead, we take this information to prove other things. Hence, the collection of such basic and true statements forms an axiomatic system.

An axiomatic system or axiomatization of propositional calculus consists of set of all well-formed formulas (wffs) of PC which can be obtained from the axioms by the use of the rules of inference. These new wffs are called theorems of the system or consequence of axioms under the stated rules of inference. Theorem is a mathematical statement that can be proved using mathematical reasoning. In a mathematical concept, the term theorem is often used for the most important results. The last step of a proof is called a theory. The term thesis can be taken here as either theorem or axiom. Every thesis is a valid well define formula of the field and every valid formula is a thesis of the system.

A *deductive system* is a set of *formulas* which are called *axioms* and a set of *rules of inference*. A *proof* in a deductive system is a sequence of formulas $S = \{A_1, \dots, A_n\}$ such that each formula A is either an axiom or it can be inferred from previous formulas of the sequence $A_{j_1}, \dots, A_{j_k}$, where $j_1 < \dots < j_k$ using a rule of inference. For A_n the last formula in the sequence, we say that A_n is a theorem, the sequence S is a *proof* of A_n , and A_n is *provable*, denoted by $\vdash A_n$. If $\vdash A$, then A may be used like an axiom in a subsequent proof.

There may be an infinite number of axioms, but only a finite number will appear in any proof. Although a proof is not a decision procedure, it can be mechanically *checked*. that is, given a sequence of formulas, a syntax-based algorithm can easily check whether the sequence is a proof as defined above. The proof of a formula clearly shows which axioms, theorems and rules

are used and for what purposes. Once a theorem has been proved, it can be used in proofs like an axiom.

Axiomatic systems are the oldest and simplest to describe type of deductive systems. The probably first prototype of an axiomatic system can be found in Euclid's Elements which present a systematic development of elementary geometry, based on several simple assumptions about points and lines (such as "Every two different points determine exactly one line", "For every line there is a point not belonging to that line", etc.). Using these and (informal) logical reasoning, other geometric facts are derived, and thus the entire body of Euclidean geometry is eventually built. But the logical concept of axiomatic system was introduced and formalized only at the beginning of the 20th century by the German mathematician David Hilbert. so that they are also known as Hilbert-style systems.

Hilbert deductive system contains a set of three axioms and one rule of inference. proofs are constructed as a sequence of formulas, each of which is either an axiom (or a formula that has been previously proved) or a derivation of a formula from previous formulas in the sequence using the rule of inference. Axiomatic systems use very few and simple rules of inference, but are based on axioms. Here we will build a hierarchy of axiomatic systems for the propositional logic, by gradually adding axioms for the logical connectives. The axiomatic system comprises the following axioms and rules. If A, B and C are the well-formed formulae of the axiomatic system then three axioms and the rule can be stated as follows.

Axiom 1 $\vdash (A \rightarrow (B \rightarrow A))$

Axiom 2 $\vdash (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

Axiom 3 $\vdash (\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$

The Only *rule of inference* is *modus ponens* (MP for short):

$$\frac{A \quad A \rightarrow B}{B}$$

In words: the formula B can be inferred from A and $A \rightarrow B$. We will denote this system by $H(\rightarrow, \neg)$

A formula A which can be derived by using the axioms and applying successively Modus Ponens is said to be derivable in H or a theorem of H, can be denoted by $\vdash_H A$. Using H one can derive logical consequences also. In order to derive, "If $A_1, \dots, A_n$ then B", we add the premises, $A_1, \dots, A_n$ to the set of axioms and try to derive B. If we succeed, we say that B is derivable in H from the assumptions $A_1, \dots, A_n$, and is denoted by $A_1, \dots, A_n \vdash_H B$ or simply $\vdash_H B$.

Example 4.10:

Derive the following in the axiomatic system H, using the Deduction Theorem

$\vdash_H (A \wedge (A \rightarrow B)) \rightarrow B$

Solution:

Table 4.23: Solution for Example 4.10

Line Number	Formula	Rule
1	$\vdash_H (A \wedge (A \rightarrow B)) \rightarrow B$	Given
2	$\vdash_H (A \wedge (A \rightarrow B)) \rightarrow A$	by Axiom 1
3	$\vdash_H (A \wedge (A \rightarrow B)) \rightarrow (A \rightarrow B)$	Axiom 2
4	$\vdash_H ((A \wedge (A \rightarrow B)) \rightarrow (A \rightarrow B)) \rightarrow (((A \wedge (A \rightarrow B)) \rightarrow A) \rightarrow ((A \wedge (A \rightarrow B)) \rightarrow B))$	Axiom 2
5	$\vdash_H (A \wedge (A \rightarrow B)) \rightarrow A \rightarrow ((A \wedge (A \rightarrow B)) \rightarrow B)$	Axiom 2, 3 and Modus Ponens
6	$\vdash_H (A \wedge (A \rightarrow B)) \rightarrow B$	Axiom 1,4 and Modus Ponens

One can check that all axioms of H are tautologies and therefore, since the rule Modus Ponens is valid, H can only derive tautologies when using these axioms as premises. By the same argument, H can only derive valid logical consequences. Therefore, the axiomatic system H is sound, or correct. In fact, it can be proved that H can derive all valid logical consequences, and in particular, all tautologies, *i.e.*, it is complete. Thus, H captures precisely the notion of propositional logical consequence. However, this axiomatic system is not very suitable for practical derivations as is, because even simple cases may require involved derivations. For instance, as a challenge, try deriving the tautology $p \rightarrow p$. Still, the derivations in H can be simplified significantly by using the following result which allows us to introduce, and later eliminate auxiliary assumptions in the derivations:

Example 4.11:

Prove: $\vdash A \rightarrow A$

Proof:

Table 4.24: Solution for Example 4.11

Line Number	Formula	Rule
1	$\vdash (A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow ((A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A))$	Axiom 2
2	$\vdash A \rightarrow ((A \rightarrow A) \rightarrow A)$	Axiom 1
3	$\vdash (A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A)$	MP 1, 2
4	$\vdash A \rightarrow (A \rightarrow A)$	Axiom 1
5	$\vdash A \rightarrow A$	MP 3,4

The proof of example 4.11 is rather complicated for such a trivial formula. In order to formalize the powerful methods of inference used in mathematics, we introduce new rules of inference called derived rules. The most important derived rule is the deduction rule. Suppose that you want to prove $A \rightarrow B$. Assume that A has already been proved and use it in the proof of B. This is not a proof of B unless A is an axiom or theorem that has been previously proved, in

which case it can be used directly in the proof. However, we claim that the proof can be mechanically transformed into a proof of $A \rightarrow B$.

The deduction rule is used frequently in mathematics. For example, if you want to prove that *the sum of any two odd integer numbers* is even, expressed formally as:

$$\text{odd}(p) \wedge \text{odd}(q) \rightarrow \text{even}(p + q), \text{ for every } p \text{ and } q.$$

To prove this formula, let us assume the formula $\text{odd}(p) \wedge \text{odd}(q)$ as if it were an additional axiom. We have available with all the theorems that we have already deduced about odd numbers, in particular, the theorem that any odd number can be expressed as $2n + 1$.

$$\text{Considering } p + q = 2n_1 + 1 + 2n_2 + 1 = 2(n_1 + n_2 + 1)$$

We obtain that $p + q$ is a multiple of 2, that is, even $(p + q)$. This theorem thus follows the deduction rule which discharges the assumption.

To understand the deduction rule, we extend the definition of deduction theorem proof. We must show that this derived rule is sound, *i.e.*, use of derived rule does not increase the set of provable theorems. This is done by showing how to transform any proof using the rule into one that does not use the rule. Therefore, in principle any proof that uses the derived rule could be transformed to one that uses the three axioms and MP.

4.7.1 Deduction Theorem

If $\Gamma, A \vdash B$, then $\Gamma \vdash A \rightarrow B$

This theorem states that whenever we are given a deduction of B from the assumption A and the set of assumptions Γ , we can develop a new deduction of $A \rightarrow B$ from the set of assumptions Γ .

Proof:

We show how to obtain a proof of $\Gamma \vdash A \rightarrow B$ by induction on each line of the proof of $\Gamma, A \vdash B$ that does not use the deduction rule.

Here, B must be either an axiom of deductive system or a previously proved theorem. If B is A , then $\vdash A \rightarrow A$ by theorem, certainly $\Gamma \vdash A \rightarrow A$. Otherwise (B is an axiom or a previously proved theorem), here is a proof of $\Gamma \vdash A \rightarrow B$ that does not use the deduction rule or the assumption A :

$\Gamma \vdash B$ Axiom of theorem

$\Gamma \vdash B \rightarrow (A \rightarrow B)$ Axiom 1

$\Gamma \vdash A \rightarrow B$ MP 1,2

By the inductive hypothesis, $\Gamma \vdash A \rightarrow C$ and $\Gamma \vdash A \rightarrow (C \rightarrow B)$, A proof of $\Gamma \vdash A \rightarrow B$ is given in Table 4.25.

Table 4.25: Deduction Theorem Proof

Line Number	Formula	Rule
1	$\Gamma \vdash A \rightarrow C$	Inductive hypothesis
2	$\Gamma \vdash A \rightarrow (C \rightarrow B)$	Inductive hypothesis
3	$\Gamma \vdash A \rightarrow (C \rightarrow B) \rightarrow ((A \rightarrow C) \rightarrow (A \rightarrow B))$	Axiom 2
4	$\Gamma \vdash (A \rightarrow C) \rightarrow (A \rightarrow B)$	MP 2, 3
5	$\Gamma \vdash (A \rightarrow B)$	MP 1, 4

To understand deduction theorem, let us consider an example, suppose we want to establish, $(a \rightarrow (b \rightarrow c)) \rightarrow (b \rightarrow (a \rightarrow c))$ is provable, i.e., $\vdash (a \rightarrow (b \rightarrow c)) \rightarrow (b \rightarrow (a \rightarrow c))$. According to deduction theorem, this holds good if, $(a \rightarrow (b \rightarrow c)) \vdash (b \rightarrow (a \rightarrow c))$ holds good. Original theorem can be proved if we can prove $(b \rightarrow (a \rightarrow c))$ using $(a \rightarrow (b \rightarrow c))$ as an assumption. Applying the theorem again, the above holds good if, $(a \rightarrow (b \rightarrow c)), b \vdash (a \rightarrow c)$ which in turn holds if, $(a \rightarrow (b \rightarrow c)), b, a \vdash c$.

The original theorem is provable (with no assumptions) if c is provable. with a, b and $a \rightarrow (b \rightarrow c)$ as assumptions. The proof is shown in Table 4.26.

Table 4.26: Deduction Theorem Proof

Line Number	Formula	Rule
1	$(a \rightarrow (b \rightarrow c)), b, a \vdash a$	by assumption
2	$(a \rightarrow (b \rightarrow c)), b, a \vdash a \rightarrow (b \rightarrow c)$	by assumption
3	$(a \rightarrow (b \rightarrow c)), b, a \vdash b \rightarrow c$	MP 1, 2
4	$(a \rightarrow (b \rightarrow c)), b, a \vdash b$	by assumption
5	$(a \rightarrow (b \rightarrow c)), b, a \vdash c$	MP 3, 4

Example 4.12:

Prove using deduction theorem: $\vdash (A \rightarrow B) \rightarrow [(B \rightarrow C) \rightarrow (A \rightarrow C)]$

Proof:

Table 4.27: Solution for example 4.12

Line Number	Formula	Rule
1	$\{A \rightarrow B, B \rightarrow C, A\} \vdash A$	Assumption
2	$\{A \rightarrow B, B \rightarrow C, A\} \vdash A \rightarrow B$	Assumption
3	$\{A \rightarrow B, B \rightarrow C, A\} \vdash B$	MP 1, 2
4	$\{A \rightarrow B, B \rightarrow C, A\} \vdash B \rightarrow C$	Assumption
5	$\{A \rightarrow B, B \rightarrow C, A\} \vdash C$	MP 3, 4
6	$\{A \rightarrow B, B \rightarrow C\} \vdash A \rightarrow C$	Deduction 5
7	$\{A \rightarrow B\} \vdash [(B \rightarrow C) \rightarrow (A \rightarrow C)]$	Deduction 6
8	$\vdash (A \rightarrow B) \rightarrow [(B \rightarrow C) \rightarrow (A \rightarrow C)]$	Deduction 7

4.8 RESOLUTION REFUTATION

In propositional logic, resolution method is the only inference rule which gives a new clause when two or more clauses are coupled together. Using propositional resolution, it becomes easy to make a theorem prover sound and complete for all. Resolution methodology was invented in the year 1965 by a Mathematician called Alan Robinson. Resolution is a theorem proving technique that proceeds by building refutation proofs, *i.e.*, proofs by contradictions. This method is basically used for proving the satisfiability of a sentence. The key idea for the resolution method is to use the knowledge base and negated goal to obtain null clause (which indicates contradiction). Resolution method is also called Proof by Refutation. When resolution is used to prove inconsistency, it is called refutation. (refute = disprove). Since the knowledge base itself is consistent, the contradiction must be introduced by a negated goal. As a result, we have to conclude that the original goal is true. During resolution, we need to identify two clauses. One with positive atom (p) and with negative atom ($\neg p$) for the application of resolution rule. Before resolution of clauses let us understand the method for converting formula into a set of clauses.

4.8.1 Clausal Form

In propositional logic, resolution works only on sentences in *clausal form*. Before we apply the rules, the premises and conclusions must be converted to clausal form. Here, we have a simple procedure for making this conversion. For that let us understand the terminologies involved.

A *literal* is either an atomic formula or a negation of an atomic formula. For example, if q is a logical constant, the following formulas are literals.

q
 $\neg q$

A clausal formula is either a literal or a disjunction of literals. If q and r are logical constants, then the following are clausal sentences.

q
 $\neg q$
 $\neg q \vee r$

A *clause* is the set of literals in a clausal sentence. For example, the following sets are the clauses corresponding to the clausal sentences mentioned above.

$\{q\}$
 $\{\neg q\}$
 $\{\neg q, r\}$

It is important to note that the empty set $\{\}$ is also a clause. It is equivalent to an empty disjunction and therefore, is unsatisfiable. As mentioned earlier, there is a simple procedure for converting an arbitrary set of propositional Logic sentences to an equivalent set of clauses. The conversion rules are summarized in Table 4.28 and should be applied in order.

Table 4.28: Conversion Rule

Implications (I):	$\phi \rightarrow \psi$	$\Rightarrow$	$\neg \phi \vee \psi$
	$\Phi \rightarrow \psi$	$\Rightarrow$	$\phi \vee \neg \psi$
	$\phi \leftrightarrow \psi$	$\Rightarrow$	$(\neg \phi \vee \psi) \wedge (\phi \vee \neg \psi)$
Negations (N):	$\phi \rightarrow \psi$	$\Rightarrow$	$\neg \phi \vee \psi$
	$\phi \leftarrow \psi$	$\Rightarrow$	$\phi \vee \neg \psi$
	$\Phi \leftrightarrow \psi$	$\Rightarrow$	$(\neg \phi \vee \psi) \wedge (\phi \vee \neg \psi)$
Distribution (D):	$\phi \vee (\psi \wedge \chi)$	$\Rightarrow$	$(\phi \vee \psi) \wedge (\phi \vee \chi)$
	$(\phi \wedge \psi) \vee \chi$	$\Rightarrow$	$(\phi \vee \chi) \wedge (\psi \vee \chi)$
	$\phi \vee (\phi_1 \vee \dots \vee \phi_n)$	$\Rightarrow$	$\phi \vee \phi_1 \vee \dots \vee \phi_n$
	$(\phi_1 \vee \dots \vee \phi_n) \vee \phi$	$\Rightarrow$	$\phi_1 \vee \dots \vee \phi_n \vee \phi$
	$\phi \wedge (\phi_1 \wedge \dots \wedge \phi_n)$	$\Rightarrow$	$\phi \wedge \phi_1 \wedge \dots \wedge \phi_n$
	$(\phi_1 \wedge \dots \wedge \phi_n) \wedge \phi$	$\Rightarrow$	$\phi_1 \wedge \dots \wedge \phi_n \wedge \phi$
Operators (O):	$\phi_1 \vee \dots \vee \phi_n$	$\Rightarrow$	$\{\phi_1, \dots, \phi_n\}$
	$\phi_1 \wedge \dots \wedge \phi_n$	$\Rightarrow$	$\{\phi_1\}, \dots, \{\phi_n\}$

Example 4.13:

Convert the following sentence to clausal form.

$(g \wedge (r \rightarrow f))$.

The conversion process is shown in Table 4.29.

Table 4.29: Conversion Process

$g \wedge (r \rightarrow f)$	
$g \wedge (\neg r \vee f)$	Implication
$g \wedge (\neg r \vee f)$	Negation
$g \wedge (\neg r \vee f)$	Distribution
$\{g\}$	Operator
$\{\neg r, f\}$	

4.8.2 Normal Forms

The language of propositional logic has much redundancy. Most of the connectives can be defined in terms of others. By repeatedly applying certain equivalences, we can transform a formula into a normal form. A typical normal form eliminates certain connectives and uses others in a restricted manner. The restricted structure makes the formula easy to process, although the normal form may be much larger than the original formula and unreadable.

- A literal is an atomic formula or its negation. When a literal is an atomic formula, it is called as positive literal and when a literal is the negation of an atomic formula, then it is called as negative literal.
- A formula is in *Negation Normal Form* (NNF) if the only connectives in it are $\wedge$, $\vee$, and $\neg$, where $\neg$ is only applied to atomic formulas.
- A formula is in *Conjunctive Normal Form* (CNF) if it has the form $A_1 \wedge \cdots \wedge A_m$, where each A_i is a disjunction of one or more literals.
- A formula is in *Disjunctive Normal Form* (DNF) if it has the form $A_1 \vee \cdots \vee A_m$, where each A_i is a conjunction of one or more literals.

An atomic formula like P is in all the normal forms NNF, CNF, and DNF. The formula $(P \vee Q) \wedge (\neg P \vee S) \wedge (R \vee P)$ is in CNF. Simplifying the formula $(P \vee Q) \wedge (\neg P \vee Q) \wedge (R \vee S)$ to $Q \wedge (R \vee S)$ amounts to improvement, because it will make our proof procedures run faster. For examples of DNF formulas, exchange $\wedge$ and $\vee$ in the examples above. As with CNF, there is no need to mention all combinations of variables. NNF can reveal the underlying nature of a formula.

For example, converting $\neg(A \rightarrow B)$ to NNF yields $A \wedge \neg B$. This reveals that the original formula was effectively a conjunction. Every formula in CNF or DNF is also in NNF, but the NNF formula $((\neg P \wedge Q) \vee R) \wedge P$ is in neither CNF nor DNF.

4.8.3 Translation to Normal Form

Every formula can be translated into an equivalent formula in NNF, CNF or DNF by means of the following steps.

Step 1. Eliminate $\leftrightarrow$ and $\rightarrow$ by repeatedly applying the following equivalences.

$$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$$

$$P \rightarrow Q \equiv \neg P \vee Q$$

Step 2. Move negations in until they apply only to atoms, repeatedly replacing by the equivalences.

$$\neg \neg P \equiv P$$

$$\neg (P \wedge Q) \equiv \neg P \vee \neg Q$$

$$\neg (P \vee Q) \equiv \neg P \wedge \neg Q$$

At this point, the formula is in Negation Normal Form.

Step 3. To obtain CNF, move disjunctions in until they apply only to literals. Repeatedly replace by the equivalences.

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

$$(Q \wedge R) \vee P \equiv (Q \vee P) \wedge (R \vee P)$$

These two equivalences are telling the same thing, since disjunction is commutative. In fact, we have,

$$(P \wedge Q) \vee (R \wedge S) \equiv (P \vee R) \wedge (P \vee S) \wedge (Q \vee R) \wedge (Q \vee S).$$

We can use this equivalence when we have to save writing.

Step 4. Simplify the resulting CNF by deleting any disjunction that contains both A and $\neg A$, since it is equivalent to true. Also delete any conjunct that includes another conjunct (meaning, every literal in the latter is also present in the former).

Finally, two disjunctions of the form $A \vee P$ and $\neg A \vee P$ can be replaced by P , since the equivalence $(A \vee P) \wedge (\neg A \vee P) \equiv P$. This simplification is related to the resolution rule, which we shall study later. Since $\vee$ is commutative, a conjunct of the form $P \vee Q$ could denote any possible way of arranging the literals into two parts. This includes $P \vee \text{false}$, since one of those parts may be empty and the empty disjunction is false. So, in the last simplification above, two conjuncts of the form A and $\neg A$ can be replaced by false.

Steps 3' and 4'. To obtain DNF, apply instead the other distributive law:

$$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$$

$$(Q \vee R) \wedge P \equiv (Q \wedge P) \vee (R \wedge P)$$

Exactly the same simplifications can be performed for DNF as for CNF, exchanging the roles of $\wedge$ and $\vee$. At this stage we have a method of proving theorems in propositional logic. To prove P , reduce it to CNF. If the simplified CNF formula is true, then P is valid because each transformation preserves logical equivalence. And if the CNF formula is not true, then P is not valid. To see why, suppose the CNF formula is $P_1 \wedge \dots \wedge P_m$. If P is valid then each P_i must also be valid. Write P_i as $L_1 \vee \dots \vee L_n$, where the L_j is literal. We can make an interpretation I that falsifies every L_j and therefore falsifies P_i .

Example 4.14:

Check the validity of the formula: $P \vee Q \rightarrow Q \vee R$.

Step 1. eliminate $\rightarrow$, gives $\neg (P \vee Q) \vee (Q \vee R)$.

Step 2. push negations in, gives $(\neg P \wedge \neg Q) \vee (Q \vee R)$.

Step 3. push disjunctions in, gives $(\neg P \vee Q \vee R) \wedge (\neg Q \vee Q \vee R)$.

Simplifying yields $(\neg P \vee Q \vee R) \wedge \text{true}$

and then $\neg P \vee Q \vee R$.

The interpretation $P \rightarrow 1, Q \rightarrow 0, R \rightarrow 0$ falsifies this formula, which is equivalent to the original formula. So, the original formula is not valid.

Example 4.15:

Check the validity of the formula: $P \wedge Q \rightarrow Q \wedge P$

Step 1. eliminate $\rightarrow$, gives $\neg (P \wedge Q) \vee Q \wedge P$

Step 2. push negations in, gives $(\neg P \vee \neg Q) \vee (Q \wedge P)$

Step 3. push disjunctions in, gives $(\neg P \vee \neg Q \vee Q) \wedge (\neg P \vee \neg Q \vee P)$

Simplifying yields $\text{true} \wedge \text{true}$, which is true. Both conjuncts are valid since they contain a formula and its negation. Thus $P \wedge Q \rightarrow Q \wedge P$ is valid.

Example 4.16:

Peirce's law is another example. $((P \rightarrow Q) \rightarrow P) \rightarrow P$

Step 1, eliminate $\rightarrow$ gives $\neg(\neg(\neg P \vee Q) \vee P) \vee P$

Step 2, push negations in, gives $(\neg\neg(\neg P \vee Q) \wedge \neg P) \vee P ((\neg P \vee Q) \wedge \neg P) \vee P$

Step 3, push disjunctions in, gives $(\neg P \vee Q \vee P) \wedge (\neg P \vee P)$

Simplifying again yields true. Thus, Peirce's law is valid.

4.8.4 Conjunctive Normal Form (CNF)

As we discussed earlier, in propositional logic, the resolution method is applied only to those clauses which are disjunction of literals. A literal is a (propositional) variable or the negation of a variable. A clause is a disjunction of literals. (Literals connected by $\vee$). A formula is in conjunctive normal form if it is a conjunction of clauses. In other words, a formula is in conjunctive normal form if and only if,

- Its only connectives are $\neg$, $\vee$ and/or $\wedge$,
- $\neg$ applies only to variables, and
- $\vee$ applies only to sub formulas with no occurrence of $\wedge$

In a clause, there is no need to have any variable twice. Also, the order of the literals in the clause does not matter. Thus, we can think of a clause as simply a set of literals. Similarly, in a CNF formula, no clause [set] need to appear more than once and the order of clauses does not matter. Thus, we can think of a CNF formula as simply a set of clauses. For example, the formula $((p \vee q) \wedge ((q \vee (\neg r)) \wedge s))$ can be described by the set of clauses $\{p, q\}$, $\{q, (\neg r)\}$ and $\{s\}$.

For CNF Formulas, it is usually easier to read by dropping brackets and only using them around clauses and the whole formula

Some CNF formulas:

- $((p \vee (\neg q)) \wedge (r \wedge ((\neg r) \vee (p \vee q))))$
- $((\neg r) \vee (p \vee q))$ (Only one clause)
- $((\neg r) \wedge (p \wedge q))$ (Three singleton clauses)

With simple bracketing:

- $((p \vee (\neg q)) \wedge r \wedge ((\neg r) \vee p \vee q))$
- $((\neg r) \vee p \vee q)$ (Only one clause)
- $((\neg r) \wedge p \wedge q)$ (Three singleton clauses)

Some formulas not in CNF:

- $(\neg ((\neg q) \wedge q))$ ($\neg$ applied to $\wedge$)
- $(p \vee (r \wedge q))$ ($\vee$ applied to $\wedge$)
- $(\neg (\neg q))$ ($\neg$ applied to $\neg$)
- $(p \rightarrow q)$ (Uses $\rightarrow$)

4.8.5 Disjunctive Normal Form (DNF)

As discussed earlier, Disjunctive Normal Form (DNF) is a reverse approach of CNF. The process is similar to CNF with the following difference: $(A_1 \wedge B_1) \vee (A_2 \vee B_2) \vee \dots \vee (A_n \wedge B_n)$.

In DNF, we use OR of ANDs, a sum of products or a cluster concept whereas, in CNF, it is ANDs of ORs.

4.8.6 Conversion of Formula to its CNF

In propositional logic, the resolution method is applied only to those clauses which are disjunction of literals. There are following steps used to convert into CNF:

- (i) Eliminate bi-conditional implication by replacing $A \leftrightarrow B$ with $(A \rightarrow B) \wedge (B \rightarrow A)$
- (ii) Eliminate implication by replacing $A \rightarrow B$ with $\neg A \vee B$
- (iii) In CNF, negation ($\neg$) appears only in literals, therefore, we move it inwards as,
 - $\neg (\neg A) \equiv A$ (double-negation elimination)
 - $\neg (A \wedge B) \equiv (\neg A \vee \neg B)$ (De Morgan)
 - $\neg (A \vee B) \equiv (\neg A \wedge \neg B)$ (De Morgan)
- (iv) Finally, using distributive law on the sentences, we can form the CNF as, $(A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \dots \wedge (A_n \vee B_n)$.

Example 4.17:

Convert a formula to CNF: $A \rightarrow (B \wedge C)$

Solution:

$$\neg A \vee (B \wedge C)$$

$$(\neg A \vee B) \wedge (\neg A \vee C)$$

Example 4.18:

Convert a formula to CNF: $(\neg A \rightarrow B) \wedge (C \wedge \neg A)$

Solution:

$$(\neg A \rightarrow B) \wedge (C \wedge \neg A)$$

$$(\neg(\neg A) \vee B) \wedge (C \wedge \neg A)$$

$$(A \vee B) \wedge (C \wedge \neg A)$$

Solution:

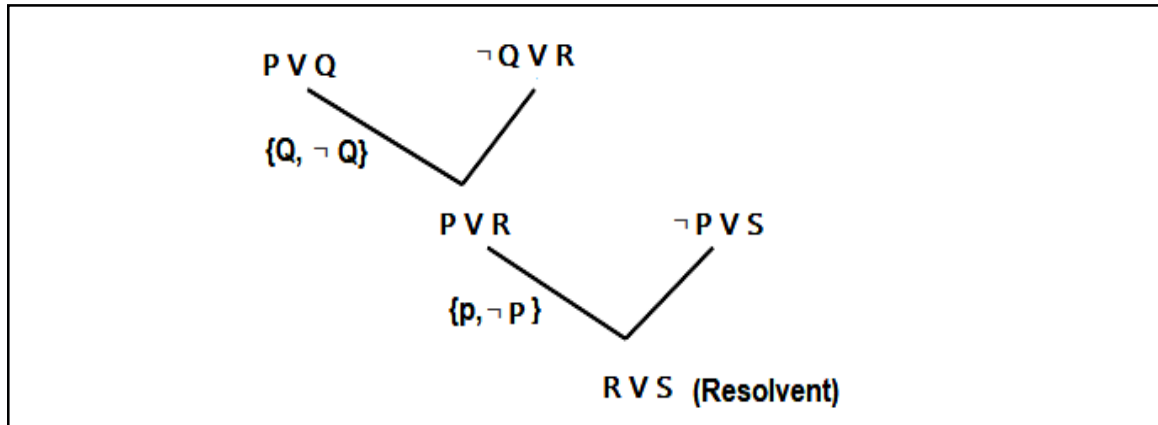


Fig. 4.2: Resolution of Clauses

From the Fig. 4.2, it can be seen that $R \vee S$ is a resolvent of $\{P \vee Q, \neg Q \vee R, \neg P \vee S\}$.

4.8.8 Rules of Resolution Refutation Principle

If C is a resolvent of two clauses C_1 and C_2 then C is called a logical consequence of a set of clauses $\{C_1, C_2\}$. If an empty clause is derived from a set of clauses S using resolution, then S is said to be unsatisfiable. This is called as resolution refutation of S . A clause C is said to be a logical consequence of S if C is derived from S . Let us understand the application of these rules using following examples.

Example 4.20:

Prove the following logical consequence using resolution refutation principle.

$$\neg p \vee q, \neg r \rightarrow \neg q \models p \rightarrow r$$

Solution:

Negation of the conclusion: $p \wedge \neg r$

Transfer of the negation to the set of premises: $\{\neg p \vee q, \neg r \rightarrow \neg q, p \wedge \neg r\}$

Conversion into CNF, set notation: $\{\{\neg p, q\}, \{r, \neg q\}, \{p\}, \{\neg r\}\}$

Using resolution tree as shown in Fig. 4.3, we can have

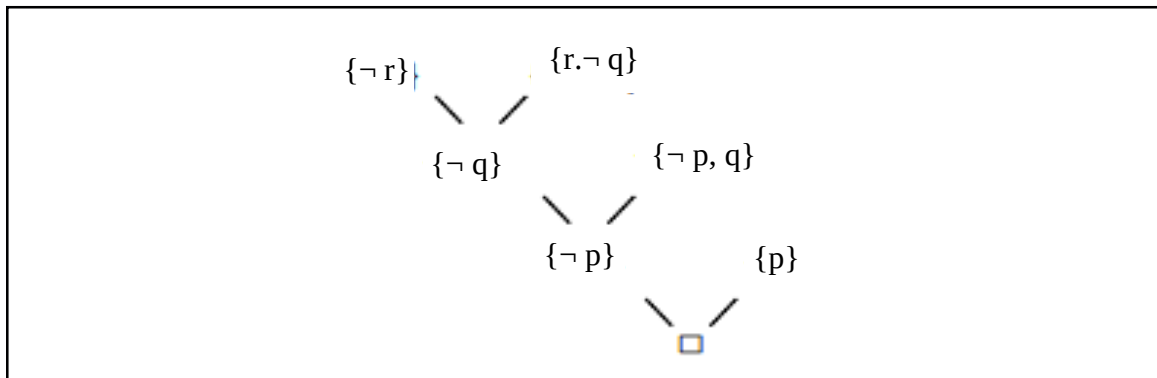


Fig. 4.3: Resolution Tree

Conclusion: the logical consequence holds.

Example 4.21:

Consider the following Knowledge Base:

The moisture is high or the sky is cloudy.

If the sky is cloudy, then it will rain.

If the moisture is high, then it is hot.

It is not hot.

Goal: It will rain.

Use propositional logic and apply resolution method to prove that the goal is derivable from the given knowledge base.

Solution:

Let's construct propositions of the given sentences one by one:

1. Let, P: moisture is high.

Q: sky is cloudy.

It will be represented as $P \vee Q$.

2. Q: sky is cloudy. ...from (1)

Let, R: it will rain.

It will be represented as $Q \rightarrow R$.

3. P: moisture is high. ...from (1)

Let, S: it is hot.

It will be represented as $P \rightarrow S$.

4. $\neg S$: it is not hot.

Applying resolution method:

In (2), $Q \rightarrow R$ will be converted as $(\neg Q \vee R)$

In (3), $P \rightarrow S$ will be converted as $(\neg P \vee S)$

Negation of Goal $(\neg R)$: It will not rain.

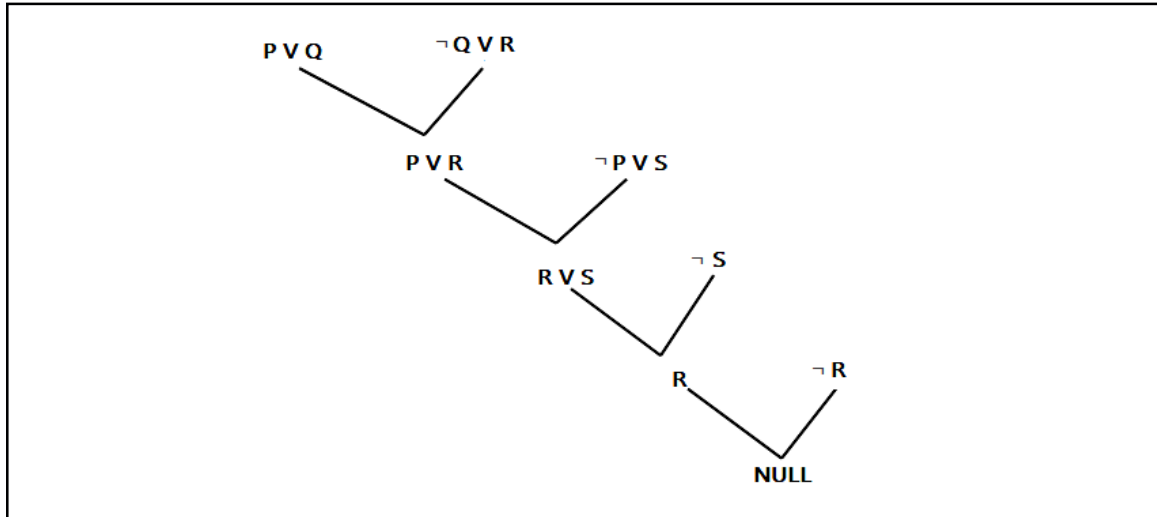


Fig. 4.4: Application of Rules

After applying Proof by Refutation (Contradiction) on the goal, the problem is solved and it is terminated with a null clause as shown in Fig. 4.4. Hence, the goal is achieved.

Example 4.22:

To show: $\{(p \rightarrow q), (q \rightarrow r)\} \vdash \text{Res } (p \rightarrow r)$.

Solution:

Convert each premise formula to CNF.

We get $((\neg p) \vee q)$ and $((\neg q) \vee r)$.

Convert the negation of the goal formula to CNF:

Replacing the $\rightarrow$ yields $\neg((\neg p) \vee r)$, then De Morgan yields $(p \wedge (\neg r))$.

Splitting the $\wedge$ yields four clauses: $((\neg p) \vee q)$, $((\neg q) \vee r)$, p and $(\neg r)$. Now we can make inferences, starting from our premises.

1. $((\neg p) \vee q)$ premise
2. $((\neg q) \vee r)$ premise
3. p premise (from negated conclusion)
4. $(\neg r)$ premise (from negated conclusion)
5. q 1, 3 (variable p)
6. r 2, 5 (variable q)
7. $\perp$ 4, 6 (variable r)

Example 4.23:

Convert the following formulas into CNF using equivalent transformations.

$$(p \rightarrow q) \leftrightarrow (p \rightarrow r)$$

Solution:

Transformations:

$$((p \rightarrow q) \rightarrow (p \rightarrow r)) \wedge ((p \rightarrow r) \rightarrow (p \rightarrow q))$$

$$(\neg(p \rightarrow q) \vee (p \rightarrow r)) \wedge (\neg(p \rightarrow r) \vee (p \rightarrow q))$$

$$((p \wedge \neg q) \vee (\neg p \vee r)) \wedge ((p \wedge \neg r) \vee (\neg p \vee q))$$

$$(((p \vee \neg p) \wedge (\neg q \vee \neg p)) \vee r) \wedge (((p \vee \neg p) \wedge (\neg r \vee \neg p)) \vee q)$$

$$(\neg q \vee \neg p \vee r) \wedge (\neg r \vee \neg p \vee q)$$

4.9 PREDICATE LOGIC

In previous sections we discussed about the propositional logic which can easily handle simple declarative statements such as, “Student Sundar Raman enrolled in CS34”. Propositional logic can also handle combinations of statements such as, “Student Sundar Raman enrolled in Tutorial 1 and student Aishwarya Raghav is enrolled in Tutorial 2.” However, statements involving formulations such as “there exists...” or “every...” or “among...” are difficult to express in propositional logic. A statement of the form “Every student is younger than some instructor” talks about concepts such as *being a student*, *being an instructor* and *being younger than somebody else*. These are properties of elements of set of objects. We express them in predicate logic using predicates. Basic elements of FOL syntax are listed in the Table 4.30. A predicate is a statement that contains variables (predicate variables) and they may be true or false depending on the values of these variables. Predicate logic is also known as First order logic (FOL), quantificational logic and first-order predicate calculus.

Table 4.30: Basic Elements of FOL Syntax

Constants	1, 2, A, Sham, Bangalore, dog, etc
Variables	a, b, c, x
Predicates	Father, Mother, >
Connectives	$\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
Equality	$=$
Quantifier	$\forall, \exists$
Function	SQRT

Now, let us consider an Example, $P(x) = “x^2 \text{ is greater than } x”$ is a predicate. It contains one predicate variable x . If we choose $x = 1$, $P(1)$ is “1 is greater than 1”, which is a proposition (always false). Since a predicate takes value true or false, once instantiated (once variables are taking values), we may alternatively say that a predicate instantiated becomes a proposition. It is needed to explicit the values that a predicate variable can possibly take and the domain of a

predicate variable which is the collection of all possible values that the variable may take. For Example, the statement “Every student is younger than some instructor” is expressed using the following predicates.

- S: For example, $S(\text{Ram})$ could denote that Ram is a student.
- I: For example, $I(\text{Karan})$ could denote that Karan is an instructor.
- Y: For example, $Y(\text{Ram}, \text{Karan})$ could denote that Ram is younger than Karan.

A practical problem arises when such predicates are used to express statements such as *Every student is younger than some instructor*. How do we express “every student”? We need variables that can stand for constant values and a quantifier symbol that denotes “every”.

First-order logic statements can be divided into two parts namely subject and predicate. Subject is the main part of the statement whereas Predicate can be a relation, which combines two atoms together in a statement. Consider the statement: “P is an integer.”, Here, the first part “P” is the subject of the statement and second part “is an integer,” is known as a predicate.

Consider one more example, where the predicate $P(a, b) = “a > b”$, with two predicate variables. We have Z (the set of integers) as domain for both of them.

Take $a = 4, b = 3$, then $P(4, 3) = “4 > 3”$, in which proposition is taking the value true.

Take $a = 1, b = 2$, then $P(1, 2) = “1 > 2”$, in which proposition is taking the value false.

4.9.1 Well-Formed Formula

Well Formed Formula (wff) is a predicate holding any of the following. All propositional constants and propositional variables are well formed formulas.

- Literals are wffs
- wffs connected by $\neg, \wedge, \vee$, and $\supset$ are wffs
- wffs surrounded by quantifiers are wffs

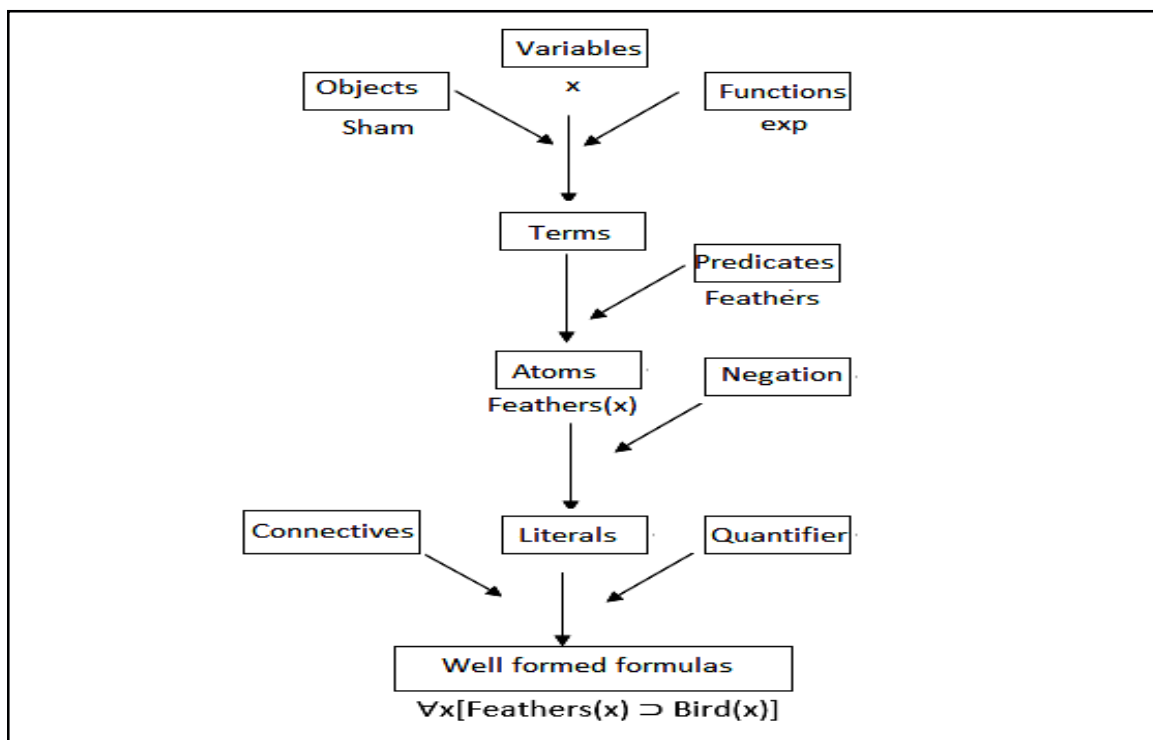


Fig. 4.5: Wff -Summary Diagram

4.10 QUANTIFIERS IN FIRST-ORDER LOGIC

A predicate with variables cannot be treated as proposition. For example, the statement $P > 5$ with variable P over the universe of real numbers is neither true nor false as in this case we don't know what P is. It may be true or not true which depends upon the value of P . In order to consider $P > 5$ to be a proposition, either we have to specify the value for P or change statement as "There is a number P for which $P > 5$ ", or "For every number $P > 5$ ".

A predicate can be made as proposition either by assigning a value to the variable or quantifying the variable using a quantifier. For instance, in the above example, if number 8 is assigned to P , then $P > 5$ becomes $8 > 5$ and statement becomes true, hence a proposition. Otherwise, a quantification is applied on predicate logic formulas (wff), such as $P > 5$ or $P(x)$, by using quantifiers on variables.

We now introduce two quantifiers (describing "parts or quantities" from a domain), the *universal quantification* (for all, everyone, everything) and the *existential quantification* (for some, at least one). A quantifier is a language element which generates quantification that specifies the quantity of specimen in the universe of discourse.

The universe of discourse, also called domain or universe which contains the set of objects (variables) of interest. The propositions in the predicate logic are statements on objects of this domain. Thus, domain can be the set of real numbers or the set of integers or the set of all students in a class etc.

4.10.1 Universal Quantification

The universal quantification of the atomic formula $P(x)$ is denoted by $\forall x P(x)$. This expression when translated in English can be understood as “For all x , $P(x)$ holds” or “for each x , $P(x)$ holds” or “for every x , $P(x)$ holds”. $\forall$ is called the universal quantifier and $\forall x$ means all the objects x in the given domain.

If the universal quantifier is followed by $P(x)$ then one can understand this as, $P(x)$ is true for every object x in the universe.

For example, “All bicycles have wheels” could be transformed into the propositional form, $\forall x P(x)$, where $P(x)$ is the predicate denoting x *has wheels* and domain consists of bicycles. If all the elements in the universe or domain are listed, then the universal quantification $\forall x P(x)$ is equivalent to the conjunction and can be expressed as $P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \dots \wedge P(x_n)$.

4.10.2 Existential Quantifier

The Existential quantification of the atomic formula $P(x)$ is denoted by $\exists x P(x)$. This expression when translated in English can be understood as “There exists an x such that $P(x)$ ” or “There is at least one x such that $P(x)$ ”. $\exists$ is called the existential quantifier and $\exists x$ means at least one object x in the universe. If this is followed by $P(x)$ then it can be understood as $P(x)$ is true for at least one object x of the universe. For example, “Someone loves you” could be transformed into the propositional form, $\exists x P(x)$, where $P(x)$ is the predicate representing x loves you and the universe of discourse contains all living creatures. If all the elements in the universe or domain are listed, then the existential quantification $\exists x P(x)$ is equivalent to the disjunction and can be expressed as $P(x_1) \vee P(x_2) \vee P(x_3) \vee \dots \vee P(x_n)$.

For example, let the universe be the set of trains and let $P(x, y)$ denote “ x moves faster than y ” then, different combinations of quantifiers can be translated as given in Table 4.31.

Table 4.31: Translation of Quantifiers

Quantifiers	Translated As	Meaning in English
$\forall x \forall y P(x, y)$	“For every train x the following holds: x is faster than every (any) train y ”.	“Every train is faster than every train” (including itself)
$\forall x \exists y P(x, y)$	“For every train x the following holds: for some train y , x is faster than y ”.	“Every train is faster than some train”.
$\exists x \forall y P(x, y)$	“There exists a train x which satisfies the following: (or such that) for every train y , x is faster than y ”.	“There is a train which is faster than every train” or “Some train is faster than every train”.
$\exists x \exists y P(x, y)$	“For some train x there exists a train y such that x is faster than y ”.	“Some train is faster than some train”.

Example 4.24:

Let $P(x)$ be “ $x^2 > 10^n$ ”. What is the truth value of $\forall x P(x)$ for each of the following domains?

- (i) the set of real numbers: $\mathbb{R}$

False. 3 is a counter example.

(ii) the set of positive integers not exceeding 4: {1, 2, 3, 4}

False. 3 is a counter example.

Also note that here $\forall P(x)$ is $P(1) \wedge P(2) \wedge P(3) \wedge P(4)$, so it's enough to observe that $P(3)$ is false.

Example 4.25:

Consider the following statement.

“Not all birds can fly”. Using the following predicates,

$B(x)$: x is a bird

$F(x)$: x can fly

We can express the sentence as follows: $\neg (\forall x(B(x) \rightarrow F(x)))$

Example 4.26:

Consider the following statement.

“Every girl is younger than her mother”.

Using the following predicates,

$G(x)$: x is a girl

$M(x, y)$: x is y 's mother

$Y(x, y)$: x is younger than y

we can express the sentence as follows: $\forall x \forall y(G(x) \wedge M(y, x) \rightarrow Y(x, y))$

Note that the variable y is only introduced to denote the mother of x . If everyone has exactly one mother, the predicate $M(y, x)$ is a function when read from right to left. We introduce a function symbol m that can be applied to variables and constants as in $\forall x[G(x) \rightarrow Y(x, m(x))]$

Example 4.27:

Consider the following statement.

“Ram and Karan have the same maternal grandmother”.

Without function symbols, we would have to write,

$\forall x \forall y \forall u \forall v (M(x, y) \wedge M(y, \text{ram}) \wedge M(u, v) \wedge M(v, \text{karan}) \rightarrow x = u)$.

However, with the function symbol m , we can simply write,

$m(m(\text{ram})) = m(m(\text{karan}))$

Example 4.28:

Let $P(x)$ be “ $x^2 > 10$ ”

What is the truth value of $\exists x P(x)$ for each of the following domains?

(i) The set of real numbers: $\mathbb{R}$

True. 10 is a witness.

- (ii) The set of positive integers not exceeding 4: {1, 2, 3, 4}

True. 4 is a witness.

Also note that here, $\exists P(x)$ is $P(1) \vee P(2) \vee P(3) \vee P(4)$. So, it is enough to observe that $P(4)$ is true.

4.10.3 Nested Quantifier

If we use a quantifier that appears within the scope of another quantifier, it is called nested quantifier. For example,

$\forall a \exists b P(x, y)$ where $P(a, b)$ denotes $a + b = 0$

$\forall a \forall b \forall c P(a, b, c)$ where $P(a, b, c)$ denotes $a + (b + c) = (a + b) + c$

Note: $\forall a \exists b P(x, y) \neq \exists a \forall b P(x, y)$

Example 4.29:

Some more examples for conversion from natural sentences to Predicate Logic

1. Marcus was a man

Man (Marcus)

2. Marcus was a Pompeian

Pompeian (Marcus)

3. All Pompeian's were Romans

$\forall x [Pompeian(x) \supset Roman(x)]$

4. Caesar was a ruler

Ruler (Caesar)

5. All Romans were either loyal to Caesar or hated him

$\forall x [Roman(x) \supset (LoyalTo(x, Caesar) \vee Hate(x, Caesar))]$

6. Everyone is loyal to someone

$\forall x \exists y LoyalTo(x, y)$

7. People only try to assassinate rulers they aren't loyal to

$\forall x \forall y [(Person(x) \wedge Ruler(y) \wedge TryAssassinate(x, y)) \supset \neg LoyalTo(x, y)]$

8. Marcus tried to assassinate Caesar

TryAssassinate (Marcus, Caesar)

4.10.4 Interpretation of Formulae in FOL

Interpretation in proposition logic simply means the assignment of truth values to atoms. But in FOL, since variables are involved and also it contains non empty domain say D , we need to assign of values to each constant, function symbol and each predicate atoms. Each formula in FOL evaluated to be true or false under given interpretation I over domain D .

Following results hold true for any interpretation I over domain D.

- $(\forall x) P(x)$ is true, if and only if $P(x)$ is true. $\forall x \in D$ otherwise it is false.
- $(\exists x) P(x)$ is true, if and only if $\exists x \in D$, such that $P(x)$ is true. Otherwise, it is false.

4.10.5 Logical Equivalences Involving Quantifiers

Two statements P and Q having predicates and quantifiers are logically equivalent if and only if they have the same truth value regardless of the interpretation which means, regardless of the meaning that is attributed to each propositional function, the domain of discourse. We denote $P \equiv Q$.

Example 4.30:

Prove that $\forall x(P(x) \wedge Q(x))$ is logically equivalent to $\forall xP(x) \wedge \forall xQ(x)$ (where the same domain is used throughout).

Proof:

Let us Use two steps: If $\forall x(P(x) \wedge Q(x))$ is true, then $\forall xP(x) \wedge \forall xQ(x)$ is true.

Proof 1: Suppose $\forall x(P(x) \wedge Q(x))$ is true. Then if a is in the domain, $P(a) \wedge Q(a)$ is true, and so P(a) is true and Q(a) is true. if a in the domain P(a) is true, which is the same as $\forall xP(x)$ is true. similarly, we get that $\forall xQ(x)$ is true. This means that $\forall xP(x) \wedge \forall xQ(x)$ is true. If $\forall xP(x) \wedge \forall xQ(x)$ is true, then $\forall x(P(x) \wedge Q(x))$ is true.

Proof 2: Suppose that $\forall xP(x) \wedge \forall xQ(x)$ is true, then $\forall xP(x)$ is true and $\forall xQ(x)$ is true. If a is in the domain, then P(a) is true and Q(a) is true. This means $\forall x(P(x) \wedge Q(x))$ is true.

Example 4.31:

Prove that $\forall x(P(x) \vee Q(x))$ is not logically equivalent to $\forall xP(x) \vee \forall xQ(x)$.

Proof:

It is enough to give a counter example to the assertion that they have the same truth value for all possible interpretations. Under the following interpretation

Domain: set of people in the world

$P(x)$ = "x is male".

$Q(x)$ = "x is female".

We have:

$\forall x(P(x) \vee Q(x))$ (every person is a male or a female) is true whereas $\forall xP(x) \vee \forall xQ(x)$ (every person is a male or every person is a female) is false.

4.10.6 Skolem Functions

Propositional logic forms the basis of many proof methods for first-order logic. Eliminating the quantifiers from a first order formula reduces it to propositional logic. The process of

removing quantifiers is called Skolemization which replaces every existentially bound variable by a Skolem constant or function. This transformation does not preserve the meaning of a formula. However, it does preserve inconsistency, which is the critical property. Resolution always works by detecting contradictions. Let us understand the process of Skolemization of a formula.

Take a formula in negation normal form. Starting from the outside, follow the nesting structure of the quantifiers. If the formula contains an existential quantifier, then the series of quantifiers must have the form

$$\forall x_1 \cdots \forall x_2 \cdots \forall x_k \cdots \exists y A$$

Where A is a formula, $k \geq 0$, and $\exists y$ is the left most existential quantifier. Choose a k -place function symbol f not present in A (that is, a new function symbol). Delete the $\exists y$ and replace all other occurrences of y by $f(x_1, x_2, \dots, x_k)$. It results in another formula.

$$\forall x_1 \cdots \forall x_2 \cdots \forall x_k \cdots A[f(x_1, x_2, \dots, x_k)/y]$$

If some existential quantifier is not enclosed in any universal quantifiers, then the formula contains simply $\exists y A$ as a sub formula. Then this quantifier is deleted and occurrences of y are replaced by a new constant symbol c . The resulting sub formula is $A[c/y]$. Then repeatedly eliminate all remaining existential quantifiers as above. The new symbols are called Skolem functions (or Skolem constants).

After Skolemization, the formula has only universal quantifiers. The next step is to remove the remaining quantifiers away. This step is correct because we are converting to clause form and a clause implicitly includes universal quantifiers over all of its free variables.

Example 4.32:

Skolemize $\exists x \forall y \exists z R(x, y, z)$

Solution:

Eliminate the $\exists x$ using the Skolem constant

$a: \forall y \exists z R(a, y, z)$

Eliminate the $\exists z$ using the 1-place Skolem function

$f: \forall y R(a, y, f(y))$

Finally, drop the $\forall y$ and convert the remaining formula to a clause.

$\{R(a, y, f(y))\}$

Example 4.33

Skolemize $\exists u \forall v \exists w \exists x \forall y \exists z ((P(h(u, v)) \vee Q(w)) \wedge R(x, h(y, z)))$

Solution:

Eliminate the $\exists u$ using the Skolem constant

$c: \forall v \exists w \exists x \forall y \exists z [(P(h(c, v)) \vee Q(w)) \wedge R(x, h(y, z))]$

Eliminate the $\exists w$ using the first-place Skolem function

$$f: \forall v \exists x \forall y \exists z ((P(h(c, v)) \vee Q(f(v))) \wedge R(x, h(y, z)))$$

Eliminate the $\exists x$ using the first-place Skolem function

$$g: \forall v \forall y \exists z ((P(h(c, v)) \vee Q(f(v))) \wedge R(g(v), h(y, z)))$$

Eliminate the $\exists z$ using the second-place Skolem function j (the function h is already used!)

$$\forall v \forall y ((P(h(c, v)) \vee Q(f(v))) \wedge R(g(v), h(y, j(v, y))))$$

Dropping the universal quantifiers yields a set of clauses:

$$\{P(h(c, v)), Q(f(v))\} \{R(g(v), h(y, j(v, y)))\}$$

Each clause is implicitly enclosed by universal quantifiers over each of its variables. So, the occurrences of the variable v in the two clauses are independent of each other.

4.10.7 Prenex Normal Form

A formula G in FOPL is said to be in a prenex normal form if and only if the formula G is in the form,

$$(Q_1 x_1) \dots (Q_n x_n) P$$

where each $(Q_i x_i)$, for $i = 1, \dots, n$, is either $(\forall x_i)$ or $(\exists x_i)$ and P is a quantifier free formula. The expression $(Q_1 x_1) \dots (Q_n x_n)$ is called the prefix and P is called the matrix of the formula G .

Examples of some formulas in prenex normal form:

$$(\exists x) (\forall y) (R(x, y) \vee Q(y)), (\forall x) (\forall y) (\neg P(x, y) \rightarrow S(y))$$

$$(\forall x) (\forall y) (\exists z) (P(x, y) \rightarrow R(z))$$

Following steps shows the process for Transforming an FOL Formula into Prenex Normal Form.

Step 1. Remove the connectives ' $\leftrightarrow$ ' and ' $\rightarrow$ ' using the equivalences

$$P \leftrightarrow G = (P \rightarrow G) \wedge (G \rightarrow P)$$

$$P \rightarrow G = \neg P \vee G$$

Step 2. Use the equivalence to remove even number of $\neg$'s

$$\neg(\neg P) = P$$

Step 3. Apply De Morgan's laws in order to bring the negation signs immediately before atoms.

$$\neg(P \vee G) = \neg P \wedge \neg G$$

$$\neg(P \wedge G) = \neg P \vee \neg G$$

and the quantification laws

$$\neg((\forall x) P[x]) = (\exists x) (\neg P[x])$$

$$\neg((\exists x) P[x]) = (\forall x) (\neg P[x])$$

Step 4. Rename bound variables, if necessary,

Step 5. Bring quantifiers to the left before any predicate symbol appears in the formula.

Example 4.34: Transform the following formulas into prenex normal forms:

$$(i) (\forall x) (Q(x) \rightarrow (\exists x) R(x, y))$$

$$(ii) (\forall x) (\forall y) ((\exists z) Q(x, y, z) \wedge ((\exists u) R(x, u) \rightarrow (\exists v) R(y, v)))$$

Solution:

$$(\forall x) (Q(x) \rightarrow (\exists x) R(x, y))$$

By removing ' $\rightarrow$ ', we get

$$(\forall x) (\neg Q(x) \vee (\exists x) R(x, y))$$

By renaming x as z in $(\exists x) R(x, y)$ the formula becomes

$$(\forall x) (\neg Q(x) \vee (\exists z) R(z, y))$$

As $\neg Q(x)$ does not involve z , we get

$$(\forall x) (\exists z) (\neg Q(x) \vee R(z, y))$$

$$(\forall x) (\forall y) ((\exists z) Q(x, y, z) \wedge ((\exists u) R(x, u) \rightarrow (\exists v) R(y, v)))$$

Removing ' $\rightarrow$ ', we get

$$(\forall x) (\forall y) ((\exists z) Q(x, y, z) \wedge (\neg((\exists u) R(x, u)) \vee (\exists v) R(y, v)))$$

Taking ' $\neg$ ' inner most, we get

$$(\forall x) (\forall y) ((\exists z) Q(x, y, z) \wedge ((\forall u) \neg R(x, u) \vee (\exists v) R(y, v)))$$

As variables z , u & v do not occur in the rest of the formula except the formula which is in its scope, therefore, we can take all quantifiers outside, preserving the order of their occurrences.

Thus, we get $(\forall x) (\forall y) (\exists z) (\forall u) (\exists v) (Q(x, y, z) \wedge (\neg R(x, u) \vee R(y, v)))$

4.11 RESOLUTION METHOD IN FOL/PREDICATE LOGIC

With the help of unification, we can extend resolution to first order logic. We now try to define the binary resolution rule with unification.

$$\frac{\{B, A_1, \dots, A_m\} \{ \neg D, C_1, \dots, C_n \}}{\{A_1, \dots, A_m, C_1, \dots, C_n\} \sigma} \text{ if } B\sigma = D\sigma$$

The first clause contains B and other literals while the second clause contains $\neg D$ and other literals. The substitution (σ) is a unifier of B and D . Here, substitution is applied to all remaining literals that are producing the conclusion. Before applying resolution, the variables in one clause are renamed to prevent clashes with the variables in the other clause. Renaming may be sound because the scope of each variable is its clause. Resolution is also sound because it takes an instance of each clause and the instances are valid, because the clauses are universally valid and then applies the propositional resolution rule, which is sound. For example, the two clauses $\{P(a)\}$

and $\{\neg P(g(a))\}$ yield the empty clause in a single resolution step. This works by renaming variables say, a to b in the second clause and unifying $P(a)$ with $P(g(b))$.

4.11.1 Factoring

Factoring is a vital part of the resolution method considered to be a separate inference rule. It takes a clause and unifies some literals within it results in a new clause. For example, starting with the clause $\{P(x, b), P(a, y)\}$, factoring can derive $\{P(a, b)\}$, since $P(a, b)$ is the result of unifying $P(x, b)$ with $P(a, y)$. This new clause is a unit clause and therefore especially useful. Factoring is necessary for completeness. Usually, resolution tends to make clauses longer. Only short clauses are likely to reach to a contradiction. If every clause has at least two literals, then the only way to reach the empty clause is with the help of factoring.

Resolution and factoring can be applied in many different ways at each step of the proof. Now, let us understand the steps to apply resolution. In First Order Logic, the process of applying the resolution method is as follows:

- Convert the given axiom into CNF, i.e., a conjunction of clauses.
- Apply negation on the goal given.
- Use literals which are required and prove it.
- Unlike propositional logic, FOL literals are complementary if one unifies with the negation of another literal.

To better understand all the above steps, we will take an example in which we will apply resolution. Consider the following axioms.

1. All dog barks at night.
2. Anyone who has any cats will not have any mice.
3. Light sleepers do not have anything which barks at night.
4. Vasanth has either a cat or a dog.
5. (Conclusion) If Vasanth is a light sleeper, then Vasanth does not have any mice.

The conclusion can be proved by using Resolution as shown below. The first step is to transform each axiom as a well-formed formula. The clauses for the above axioms are shown below, using $LS(x)$ for 'light sleeper'.

1. $\forall x (DOG(x) \rightarrow BARK(x))$
2. $\forall x \forall y (HAVE(x, y) \wedge CAT(y) \rightarrow \neg \exists z (HAVE(x, z) \wedge MOUSE(z)))$
3. $\forall x (LS(x) \rightarrow \neg \exists y (HAVE(x, y) \wedge BARK(y)))$
4. $\exists x (HAVE(Vasanth, x) \wedge (CAT(x) \vee DOG(x)))$
5. $LS(Vasanth) \rightarrow \neg \exists z (HAVE(Vasanth, z) \wedge MOUSE(z))$

The next step is to transform each wff into Prenex Normal Form (PNF), then eliminate existential quantifier $\exists$ (Skolemization) and rewrite as clauses in conjunctive normal form. The transformations are shown below.

1. $\forall x (\text{DOG}(x) \rightarrow \text{BARK}(x))$
 $\neg \text{DOG}(x) \vee \text{BARK}(x)$
2. $\forall x \forall y (\text{HAVE}(x, y) \wedge \text{CAT}(y) \rightarrow \neg \exists z (\text{HAVE}(x, z) \wedge \text{MOUSE}(z)))$
 $\forall x \forall y (\text{HAVE}(x, y) \wedge \text{CAT}(y) \rightarrow \forall z \neg (\text{HAVE}(x, z) \wedge \text{MOUSE}(z)))$
 $\forall x \forall y \forall z (\neg (\text{HAVE}(x, y) \wedge \text{CAT}(y)) \vee \neg (\text{HAVE}(x, z) \wedge \text{MOUSE}(z)))$
 $\neg \text{HAVE}(x, y) \vee \neg \text{CAT}(y) \vee \neg \text{HAVE}(x, z) \vee \neg \text{MOUSE}(z)$
3. $\forall x (\text{LS}(x) \rightarrow \neg \exists y (\text{HAVE}(x, y) \wedge \text{BARK}(y)))$
 $\forall x (\text{LS}(x) \rightarrow \forall y \neg (\text{HAVE}(x, y) \wedge \text{BARK}(y)))$
 $\forall x \forall y (\text{LS}(x) \rightarrow \neg \text{HAVE}(x, y) \vee \neg \text{BARK}(y))$
 $\forall x \forall y (\neg \text{LS}(x) \vee \neg \text{HAVE}(x, y) \vee \neg \text{BARK}(y))$
 $\neg \text{LS}(x) \vee \neg \text{HAVE}(x, y) \vee \neg \text{BARK}(y)$
4. $\exists x (\text{HAVE}(\text{Vasanth}, x) \wedge (\text{CAT}(x) \vee \text{DOG}(x)))$
 $\text{HAVE}(\text{Vasanth}, a) \wedge (\text{CAT}(a) \vee \text{DOG}(a))$
5. $\neg [\text{LS}(\text{Vasanth}) \rightarrow \neg \exists z (\text{HAVE}(\text{Vasanth}, z) \wedge \text{MOUSE}(z))]$ (negated conclusion)
 $\neg [\neg \text{LS}(\text{Vasanth}) \vee \neg \exists z (\text{HAVE}(\text{Vasanth}, z) \wedge \text{MOUSE}(z))]$
 $\text{LS}(\text{Vasanth}) \wedge \exists z (\text{HAVE}(\text{Vasanth}, z) \wedge \text{MOUSE}(z))$
 $\text{LS}(\text{Vasanth}) \wedge \text{HAVE}(\text{Vasanth}, b) \wedge \text{MOUSE}(b)$

The set of CNF clauses for this problem is thus as given in the Table 4.32.

Table 4.32 Set of Clauses

Line Number	Clause
1	$\neg \text{DOG}(x) \vee \text{BARK}(x)$
2	$\neg \text{HAVE}(x, y) \vee \neg \text{CAT}(y) \vee \neg \text{HAVE}(x, z) \vee \neg \text{MOUSE}(z)$
3	$\neg \text{LS}(x) \vee \neg \text{HAVE}(x, y) \vee \neg \text{BARK}(y)$
4	(a) $\text{HAVE}(\text{Vasanth}, a)$ (b) $\text{CAT}(a) \vee \text{DOG}(a)$
5	(a) $\text{LS}(\text{Vasanth})$ (b) $\text{HAVE}(\text{Vasanth}, b)$ (c) $\text{MOUSE}(b)$

Now we proceed to prove the conclusion by resolution using the above clauses. Now in this step, follow the line numbers of clauses mentioned on the left and corresponding resulting clauses as given in Table 4.33.

Table 4.33: Resolution of Clause

Line Number	Resulting Line Number	Resulting Clause
[1, 4(b)]	6	CAT(a) $\vee$ BARK(a)
[2, 5(c)]	7	$\neg$ HAVE (x, y) $\vee$ $\neg$ CAT(y) $\vee$ $\neg$ HAVE (x, b)
[7, 5(b)]	8	$\neg$ HAVE (Vasanth, y) $\vee$ $\neg$ CAT(y)
[6, 8]	9	$\neg$ HAVE (Vasanth, a) $\vee$ BARK(a)
[4(a), 9]	10	BARK(a)
[3,10]	11	$\neg$ LS(x) $\vee$ $\neg$ HAVE (x, a)
[4(a),11]	12	$\neg$ LS(Vasanth)
[5(a),12]	13	$\square$

Alternatively, we can also solve the problem by resolution tree using substitution as shown in Fig. 4.6.

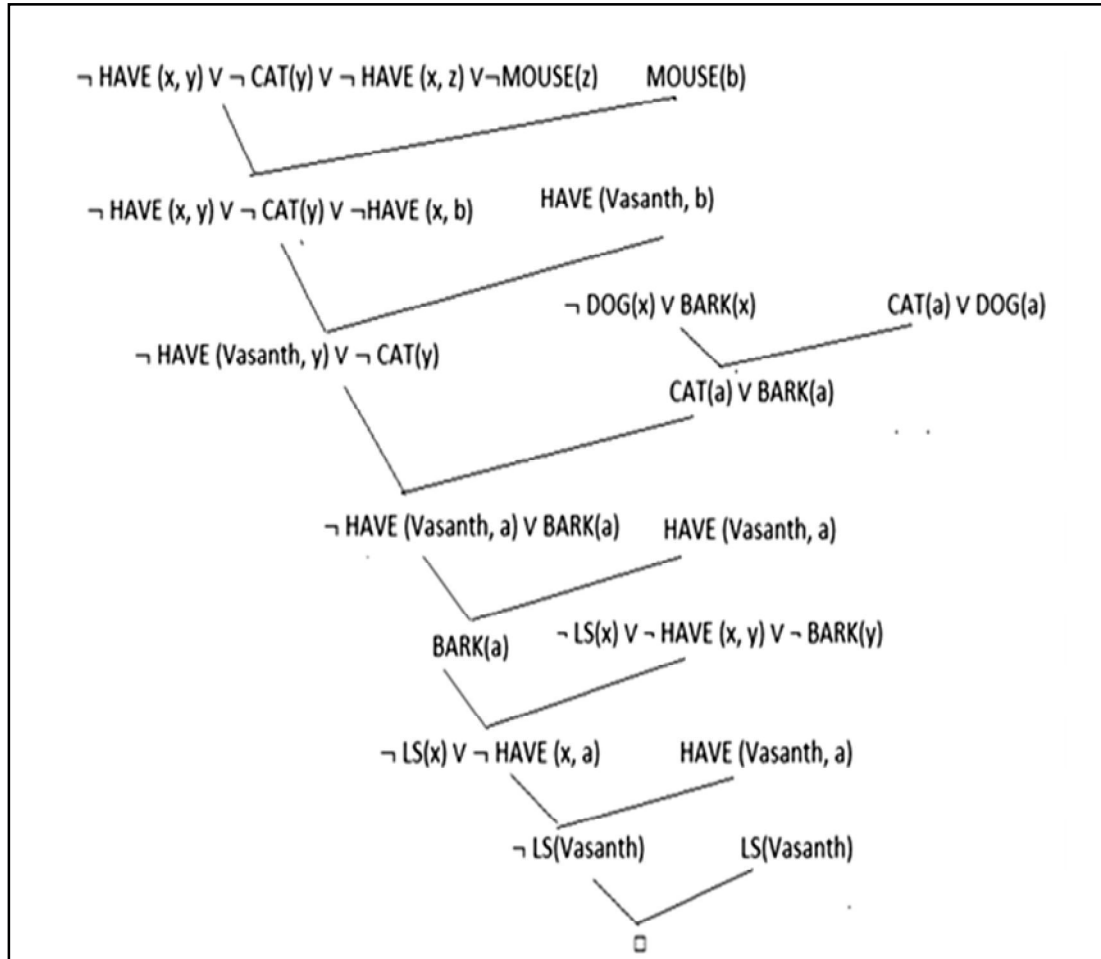


Fig. 4.6: Resolution Graph

EXERCISES

4.1 Derive the following in the axiomatic system H, using the Deduction Theorem.

$$A, \neg A \vdash_H B$$

Solution:

Table 4.34: Solution for 4.1

Line Number	Formula	Rule
1	$A, \neg A \vdash_H B$	Given
2	$\neg A \vdash_H \neg B \rightarrow \neg A$	Axiom 1, Deduction theorem
3	$\neg A \vdash_H (\neg B \rightarrow A) \rightarrow B$	Axiom 3, Modus Ponens
4	$A \vdash_H \neg B \rightarrow A$	Axiom 1, Deduction theorem
5	$A, \neg A \vdash_H B$	2, 3 and Modus Ponens

4.2 Convert the following sentences to conjunctive normal form.

1. $(P \rightarrow Q) \rightarrow R$

Solution:

$$\begin{aligned} & (P \vee Q) \vee R \\ & (\neg P \wedge Q) \vee R \\ & (P \vee R) \wedge (\neg Q \vee R) \end{aligned}$$

2. $P \rightarrow (Q \rightarrow R)$

Solution:

$$\neg P \vee \neg Q \vee R$$

3. $(P \rightarrow Q) \vee (Q \rightarrow P)$

Solution:

$$\begin{aligned} & (\neg P \vee Q) \vee (\neg Q \vee P) \\ & \text{True} \end{aligned}$$

4. $(\neg P \rightarrow (P \rightarrow Q))$

Solution:

$$\begin{aligned} & \neg \neg P \vee (\neg P \vee Q) \\ & P \vee \neg P \vee Q \\ & \text{True} \end{aligned}$$

5. $(P \rightarrow (Q \rightarrow R)) \rightarrow (P \rightarrow (R \rightarrow Q))$

Solution:

$$\neg (\neg P \vee \neg Q \vee R) \vee (\neg P \vee \neg R \vee Q)$$

$$(P \wedge Q \wedge \neg \neg R) \vee (\neg P \vee \neg R \vee Q) (P \vee \neg P \vee \neg R \vee Q) \wedge (Q \vee \neg P \vee \neg R \vee Q) \wedge (\neg R \vee \neg P \vee R \vee Q) \neg P \vee Q \vee \neg R$$

$$6. (P \rightarrow Q) \rightarrow ((Q \rightarrow R) (P \rightarrow R))$$

Solution:

$$\neg (\neg P \vee Q) \vee (\neg (\neg Q \vee R) \vee (\neg P \vee R))$$

$$(P \wedge \neg Q) \vee ((Q \wedge \neg R) \vee (\neg \neg P \vee R))$$

$$(P \wedge \neg Q) \vee ((Q \vee \neg P \vee R) \wedge (\neg R \vee \neg P \vee R))$$

$$(P \wedge \neg Q) \vee Q \vee \neg P \vee R (P \vee \neg P \vee Q \vee R) \wedge (\neg P \vee Q \vee \neg \neg Q \vee R)$$

True

4.3 For each of the following English sentences, write a corresponding sentence in FOL.

1. All students get good grades if they study.

$$\forall x. \text{Student}(x) \wedge \text{Study}(x) \rightarrow \text{GetGoodGrade}(x)$$

2. The Barber of Seville shaves all men who do not shave themselves.

$$\forall x. \neg \text{Shaves}(x, x) \rightarrow \text{Shaves}(\text{BarberOfSeville}, x)$$

3. There are at least two mountains in England.

$$\exists x, y. \text{Mountain}(x) \wedge \text{Mountain}(y) \wedge \text{InEngland}(x) \wedge \text{InEngland}(y) \wedge x \neq y$$

4. There is exactly one coin in the box.

$$\exists x. \text{Coin}(x) \wedge \text{InBox}(x) \wedge \forall y. (\text{Coin}(y) \wedge \text{InBox}(y) \rightarrow x = y)$$

5. There are exactly two coins in the box.

$$\exists x, y. \text{Coin}(x) \wedge \text{InBox}(x) \wedge \text{Coin}(y) \wedge \text{InBox}(y) \wedge x \neq y \wedge \forall z. (\text{Coin}(z) \wedge \text{InBox}(z) \rightarrow (x = z \vee y = z))$$

6. The largest coin in the box is a quarter.

$$\exists x. \text{Coin}(x) \wedge \text{InBox}(x) \wedge \text{Quarter}(x) \wedge \forall y. (\text{Coin}(y) \wedge \text{InBox}(y) \wedge \neg \text{Quarter}(y) \rightarrow \text{Smaller}(y, x))$$

7. No mountain is higher than itself.

$$\forall x. \text{Mountain}(x) \rightarrow \neg \text{Higher}(x, x)$$

4.4 Convert the following formulas into a Skolem normal form:

$$\exists z \forall y \forall x \exists y_1 \forall x_1 ((\neg P(x, y) \vee Q(y, z)) \wedge (R(x_1, y_1) \vee Q(z, y_1)))$$

Solution:

$$\exists z \forall y \forall x \exists y_1 \forall x_1 ((P(x, y) \vee Q(y, z)) \wedge (R(x_1, y_1) \vee Q(z, y_1)))$$

$$\forall y \forall x \exists y_1 \forall x_1 ((P(x, y) \vee Q(y, c)) \wedge (R(x_1, y_1) \vee Q(c, y_1)))$$

$$\forall y \forall x \forall x_1 ((P(x, y) \vee Q(y, c)) \wedge (R(x_1, f(y, x)) \vee Q(c, f(y, x))))$$

4.5 Transform the following formulas into (conjunctive) PNF.

$$\exists z (\forall y (\exists x P(x, y) \Rightarrow Q(y, z)) \wedge \exists y (\forall x R(x, y) \vee Q(z, y)))$$

Solution:

$$\begin{aligned}
& \exists z(\forall y(\neg \exists xP(x, y) \vee Q(y, z)) \wedge \exists y(\forall xR(x, y) \vee Q(z, y))) \\
& \exists z(\forall y(\neg \exists xP(x, y) \vee Q(y, z)) \wedge \exists y_1(\forall x_1R(x_1, y_1) \vee Q(z, y_1))) \\
& \exists z(\forall y(\forall x\neg P(x, y) \vee Q(y, z)) \wedge \exists y_1(\forall x_1R(x_1, y_1) \vee Q(z, y_1))) \\
& \exists z(\forall y\forall x(\neg P(x, y) \vee Q(y, z)) \wedge \exists y_1\forall x_1(R(x_1, y_1) \vee Q(z, y_1))) \\
& \exists z\forall y\forall x(\neg P(x, y) \vee Q(y, z)) \wedge \exists y_1\forall x_1(R(x_1, y_1) \vee Q(z, y_1)) \\
& \exists z\forall y\forall x\exists y_1\forall x_1((\neg P(x, y) \vee Q(y, z)) \wedge (R(x_1, y_1) \vee Q(z, y_1)))
\end{aligned}$$

QUESTIONS

- 4.1 Explain proposition and predicate logic with example.
- 4.2 Compute the truth table of $(P \vee Q) \wedge \neg P \rightarrow Q$ using truth table method
- 4.3 Show that a set $P = \{\neg(A \vee B), B \rightarrow C, (A \vee C)\}$ is consistent.
- 4.4 Show that the following argument is valid using truth table.
 “If it is humid then it will rain and since it is humid it will rain”.
- 4.5 Show that $a: B \neg(A \rightarrow B) \vee \neg A$ is valid.
- 4.6 Rewrite the following sentences in FOL:
 Coconut-crispy is a biscuit.
 Shilpa is a child who takes coconut-crispy.
 Ram loves children who take biscuits.
 Ram loves Shilpa.
- 4.7 Test the following statements using Truth table approach.
 (i) $\neg P \wedge Q$ equivalent to $\neg P \vee \neg Q$
 (ii) $(\neg R \wedge (P \rightarrow \neg Q)) \rightarrow P$ equivalent to $R \vee P$
- 4.8 Translate the following statements to propositional logic statement and check argument.
 (i) I got the highest Marks in the last class test, and I have good attendance.
 If I get a cold, then I miss at least one class.
 I came down with a cold last week.
 Therefore, If I missed at least one class, then I did not get the highest grade in the class on the last test.
 (ii) If it is June, then classes are regular.
 Classes are regular.
 Therefore, it is June.
- 4.9 Is the formula $P \rightarrow \neg P$ satisfiable, or valid
- 4.10 Verify the de Morgan and distributive laws using truth tables.

- 4.11 Convert each of the following propositional formulas into Conjunctive Normal Form and also into Disjunctive Normal Form. For each formula, state whether it is valid, satisfiable, or unsatisfiable; justify each answer.

$$(P \rightarrow Q) \wedge (Q \rightarrow P)$$

$$((P \wedge Q) \vee R) \wedge \neg (P \vee R)$$

$$\neg (P \vee Q \vee R) \vee ((P \wedge Q) \vee R)$$

- 4.12 Convert the following formulas into a Skolem normal form

$$(a) \exists x \forall y P(a, y, x)$$

$$(b) \forall y \exists x P(f(x), y, x)$$

$$(c) \exists x R(x) \vee \forall x P(x) \vee (\forall x \exists y Q(x, y) \vee \exists x \forall y P(x, y)) \wedge \exists x \exists y P(x, y)$$

- 4.13 Use resolution to prove $(A \rightarrow B \vee C) \rightarrow [(A \rightarrow B) \vee (A \rightarrow C)]$.

- 4.14 Prove Peirce's law, $((P \rightarrow Q) \rightarrow P) \rightarrow P$ using resolution.

- 4.15 Use resolution (showing the steps of converting the formula into clauses) to prove these two formulas:

$$(Q \rightarrow R) \wedge (R \rightarrow P \wedge Q) \wedge (P \rightarrow Q \vee R) \rightarrow (P \leftrightarrow Q)$$

$$(P \wedge Q \rightarrow R) \wedge (P \vee Q \vee R) \rightarrow ((P \leftrightarrow Q) \rightarrow R)$$

- 4.16 Transform the following formulas into (conjunctive) PNF

$$(a) (\forall x) P(x) \rightarrow (\exists x) Q(x)$$

$$(b) \forall x R(x) \rightarrow \forall y P(y)$$

$$(c) \forall x \exists y P(x, y) \vee \exists x \forall z R(f(x))$$

$$(d) \forall y \exists x R(x, y) \leftrightarrow \forall x \forall y P(x, y) \vee (\forall x \exists y Q(x, y) \vee \exists x \forall y P(x, y)) \wedge \neg \exists x \exists y P(x, y)$$

- 4.17 Consider the following axioms.

1. Anyone whom Mary loves is a football star.
2. Any student who does not pass does not play.
3. John is a student.
4. Any student who does not study does not pass.
5. Anyone who does not play is not a football star.

(Conclusion) If John does not study, then Mary does not love John.

Represent these axioms in predicate calculus, skolemize as necessary and convert each formula to clause form.

- 4.18 Derive the following in the axiomatic system H, using the Deduction Theorem.

$$(a) (A \wedge B) \rightarrow C \vdash_H A \rightarrow (B \rightarrow C).$$

$$(b) A \rightarrow (B \rightarrow C) \vdash_H (A \wedge B) \rightarrow C.$$

$$(c) \vdash_H A \rightarrow ((B \rightarrow C) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))).$$



5.1 INTRODUCTION

AI Planning is a field of artificial Intelligence which exhibits the process of using autonomous techniques to solve planning problems. A planning problem is one which start with some initial state which we wish to transform this state into a desired goal state through the application of a set of actions.

In a simple example, imagine we wish to construct a house with an assumption that all the materials to be used are ready, then essentially, we have to decide in which order the construction to be performed. Each step of construction could be modelled as an action (for example laying foundations stones, building the walls, installing the plumbing and electricals etc.). Certain actions cannot happen before others. It is not possible to build a wall before we have a foundation. Therefore, with the help of predicate logic, we can express this problem by saying “if there is no foundation, we cannot build a wall”. In reality, the planning of construction of a house is not that difficult task. but imagine instead of that, we are planning to construct an Olympic stadium. Then problem can be broken down into similar simple tasks which are dependent on prior conditions being met.

The planning in Artificial Intelligence is all about the decision-making tasks performed by the computer programs or robots to achieve a specific goal. The execution part of the planning is about choosing a sequence of actions which are likely to complete the specific task. Methods that we use for decomposing the original problem into appropriate subparts along with recording and handling interactions among the subparts as they are detected during the problem-solving process are often called as planning. Planning refers to the process of computing several steps of a problem-solving procedure before executing any of them. The planning consists of following important steps:

(i) Choose the best rule to apply next, based on the best available heuristic information.

The most commonly used technique for selecting a suitable rule to apply is first to isolate a set of differences between desired goal state and then to identify the appropriate rules to reduce those differences. If there are several rules, a variety of other heuristic information can be exploited to choose among them.

(ii) Apply the chosen rule to compute the new problem state that arises from its application.

In simple systems, application of rules is easy. Each rule simply specifies the problem state which result from its application. In complex systems, we must be able to deal with rules that specify only a small part of the complete problem state. One way is to define for each action, each of the changes it makes to the state description.

(iii) Detect when a solution has found.

A planning system can succeed in finding a solution to a problem when it can find a sequence of operators that transform the initial problem state into the goal state.

In simple problem-solving systems, this problem is addressed by a straightforward match of the state descriptions. One of the representative systems for planning systems is predicate logic. Suppose that as a part of our goal, we have the predicate $P(x)$. To see whether $P(x)$ satisfied in some state, we ask whether we can prove $P(x)$ given the assertions that describe that state and the axioms that define the world model.

(iv) Detect dead ends so that new directions may be explored.

As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning mechanisms that can use to detect a solution can often use for detecting a dead end. If the search process is reasoning forward from the initial state, it can prune any path that leads to a state from which the goal state cannot reach.

If search process reasoning backward from the goal state, it can also terminate a path either because it is sure that the initial state cannot reach or because little progress made.

(v) Detect when an almost correct solution has found and employ special techniques to make it totally correct.

The kinds of techniques discussed are often useful in solving nearly decomposable problems. One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the sub-problems separately. And then check that when the sub-solutions combined. They give a solution to the original problem.

5.2 PLANNING PROBLEMS

The most obvious difficulty is that the problem-solving agent can be overwhelmed by irrelevant actions. Consider the task of buying a copy of AI from an online bookseller. Consider there is one buying action for each 10-digit ISBN number, for a total of 10 billion actions. The search algorithm would have to examine the outcome states of all 10 billion actions to find the one that satisfies the goal, *i.e.*, to own a copy of ISBN 0137903541. On the other hand, A sensible planning agent should be able to work back from an explicit goal description such as Have (ISBN 0137903541) and generate the action Buy (ISBN 0137903541) directly. To do this, the agent simply needs the general knowledge that Buy(x) and results in Have(x). Given this knowledge

and the goal, the planner can decide using a single step that Buy (ISBN 0137903952) is the right action.

The next difficulty is finding a good heuristic function. If the agent's goal is to buy four different books online, then there will be 1040 plans of just four steps. Hence, searching without an accurate heuristic is not at all possible. It is obvious to a human that a good heuristic estimate for the cost of a state is the number of books that remain to be bought. Unfortunately, this insight is not obvious to a problem-solving agent, because it sees the goal test only as a black box that returns true or false for each state. Therefore, the problem-solving agent lacks autonomy. It requires a human to supply a heuristic function for each new problem. Also, if a planning agent has access to an explicit representation of the goal as a conjunction of subgoals, then it can use a single domain-independent heuristic: the number of unsatisfied conjuncts.

For the book-buying problem, the goal would be $\text{Have}(A) \wedge \text{Have}(B) \wedge \text{Have}(C) \wedge \text{Have}(D)$ and a state containing $\text{Have}(A) \wedge \text{Have}(C)$ would have cost 2. Thus, the agent automatically gets the right heuristic for this problem.

Finally, the problem solver might be inefficient because it cannot take advantage of problem decomposition. Consider the problem of delivering a set of overnight packages to their respective destinations, which are scattered across Canada. It makes sense to find out the nearest airport for each destination and divide the overall problem into several subproblems, one for each airport. Within the set of packages routed through a given airport, whether further decomposition is possible depends on the destination city. The ability to do this kind of decomposition contributes to the efficiency of constraint satisfaction problem solvers. The same holds true for planners. In the worst case, it can take $O(n!)$ time to find the best plan to deliver n packages, but only $O((n/k)! \times k)$ time if the problem can be decomposed into k equal parts.

5.3 TYPES OF PLANNING SYSTEMS

When a particular problem to be solved, at that time some specific rules regarding that problem are to be applied. Then apply the chosen rule to compute the new problem state that arises from its application. Detect when a solution has been found and calculate the active and inactive ends of that problem. Various components of a planning system are described as follows.

- (a) **States:** For a planning process, the planners decompose the world into some environments. Then environments are defined by some logical conditions and states. The problems can be viewed as the task of finding a path from a given starting state to some desirable goal state. The state can be viewed as a conjunction of positive literals. For example, Rich A famous might represent the state of a best agent.
- (b) **Goal:** A goal is a specified state. To find a solution to a problem using a search procedure is to generate moves through the problem space until a goal state is reached. In the context of game playing programs, a goal state is one in which we win. Unfortunately, for interesting games like chess, it is not usually, possible, even with a good plausible move generator, to search until a goal state is found.

- (c) **Actions:** An action is specified in terms of the pre-conditions that must hold before it can be executed and then the effects that ensue when it is executed.
- (d) **Precondition:** The precondition is a conjunction of function free positive literals stating what must be true in a state before the action can be executed.
- (e) **Effect:** It is a conjunction of function free literals describing how the state changes when the action is executed.
- (f) **Finding a solution:** A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state. The way it can be solved depends on the way that state descriptions are represented.
- (g) **Calculating the Dead State:** As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning methods that can be used to detect a solution can often be used for detecting a dead path. If the search process is reasoning in forward direction from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached. If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the starting state cannot be reached.

5.3.1 Operator-Based Planning

In this approach, actions are represented as *operators*. This approach is also called as *the STRIPS approach* which requires various *operator schema* and *plan representations*. The frame problem is solved by using the *STRIPS assumption*. The major points to be discussed in this context are given below.

Design of the operator schema

Add-delete lists, procedural vs declarative representations, etc.

Design of plan representations

Linear plans, non-linear plans, hierarchical plans, partial-order plans, conditional plans, etc.

Planning Algorithms

Planning as search, world-space vs. plan-space, partial-order planning, total-order planning, progression, goal-regression, etc.

Plan generation

Plan reformulation, repair, total-ordering, etc.

5.3.2 Planning Algorithms

In this we can introduce planning as search. There are two approaches:

(i) Searching a World Space:

In world space, each node in the state search graph denoted as state of the world. Arcs in the graph correspond to the execution of a specific action. The planning problem is to find a path from the initial state to the goal state. Following are the two approaches in order to develop planning algorithms.

Progression: An algorithm that searches for the goal state by searching through the states generated by actions that can be performed in the given state starting from the initial state. This approach is also known as forward chaining approach.

Regression: An algorithm that searches backward from the goal state by finding actions whose effects satisfy one or more of the posted goals and posting the chosen action's preconditions as goals is called as *goal regression*. This approach is also known as backward chaining approach.

Both the algorithms are *sound* and *complete*. In most situations regression is a better strategy.

(ii) Searching a Plan Space:

In this approach, each node in the graph represents *partial plans*. Arcs denote *plan refinement operations*. One can search for a plan with a totally-ordered sequence of actions (total order planning) or a plan with a partially ordered set of actions called partial order planning. Partial Order Planning has three components:

A set of actions: Set of activities that we do every day. For example, *eat-breakfast*, *take-rest*, *wake-up*, *go-to-bed* and so on.

A set of ordering constraints: From the activities mentioned above, we do some activity before we perform other activities. For example, *wake-up before eat-breakfast*, *take-bath before go-to-college*. Some activities need to be performed in an order while other need not be. For example, *take-bath* before *go-to-college* may not be in the same order as one can take bath after returning from college. But activity such as *wake-up* cannot follow *go-to-college* and hence the ordering should be in constraint.

A set of causal links: For example, *wake-up-awake-eat-breakfast* is a casual link from the action *wake-up* to the action *eat-breakfast*. When the action *awake* is added to the plan, the above causal link is generated along with the ordering constraint. Here, *wake-up*'s effect is to keep the individual in the state of *awake* which becomes a precondition for action *eat-breakfast*. Causal links help detect inconsistencies whenever a partial plan is refined.

5.3.3 Logic Based Planning – Blocks World Problem

One of the most famous planning domains is known as the block world. This domain consists of a set of cube-shaped identical blocks sitting on a table. The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block and move it to another position, either on the table or on top of another block. The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it. The goal will always be to

build one or more stacks of blocks specified in terms of what blocks are on top of what other blocks. The world consists of:

- A flat surface such as a tabletop.
- An adequate set of identical blocks which are identified by letters.
- The blocks can be stacked one on the other to form towers of apparently unlimited height.
- The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations.
- The robot can hold one block at a time and only one block can be moved at a time.

We are having four types of operations done by robot in block world environment as given in Table 5.1.

Table 5.1 Description of Operations-Block World problem

Operation	Form	Description
UNSTACK (X, Y)	[US (X, Y)]	Pick up X from its current position on block Y. The arm must be empty and X has no block on top of it.
STACK (X, Y)	[S (X, Y)]	Place block X on block Y. Arm must holding X and the top of Y is clear
PICKUP (X)	[PU (X)]	Pick up X from the table and hold it. Initially the arm must be empty and top of X is clear.
PUTDOWN (X)	[PD (X)]	Put block X down on the table. The arm must have been holding block X

Along with the operations, some predicates to be used to describe the problem state clearly. These predicates are as follows (Table 5.2).

Table 5.2: Description of Predicates-Block World Problem

Predicates	Description
ON (X, Y)	Block X on block Y.
ONTABLE(A)	Block A is on the table
ONT ()	Block X on the table.
CL(X)	Top of X clear.
HOLD(X)	Robot-Arm holding X.
AE	Robot-arm empty

In logic-based planning we have to state all possible Logical statements which are true. Following statements describe the logical statements of block world problem.

X Holding X means, arm is not empty

$(\exists X) \text{ HOLD}(X) \rightarrow \sim \text{AE}$

X is on a table means that X is not on the top of any block

$(\forall X) \text{ ONTO}(X) \rightarrow \sim (\exists Y) \text{ ON}(X, Y)$

Any block with no block on has clear top

$(\forall X(\sim \exists Y) \text{ ON } (Y, X) \rightarrow \text{CL}(X))$

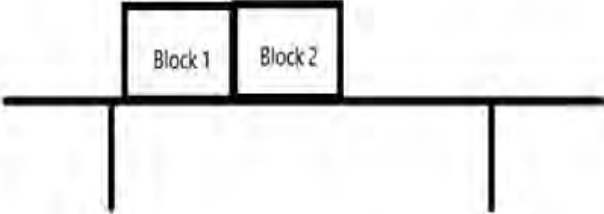
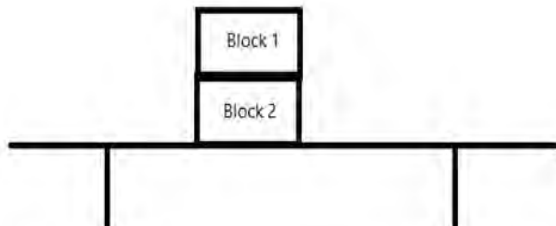
Initial State armempty clear (block2) ontable (block2) ontable (block1) clear (block1)	
Goal State armempty ontable (block2) on (block1, block2) clear (block1)	

Fig. 5.1: Initial and Goal States of Block World

We have to generate a plan to reach goal state from initial state given as shown in Fig. 5.1. In this example the initial state has two blocks Block1 and Block2. Both are placed on table as shown. To reach the goal state first we have to use the operations as shown below.

PICKUP (Block 1)

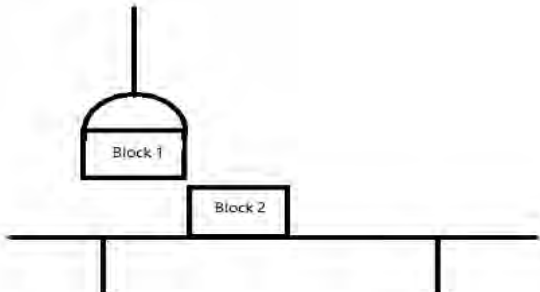
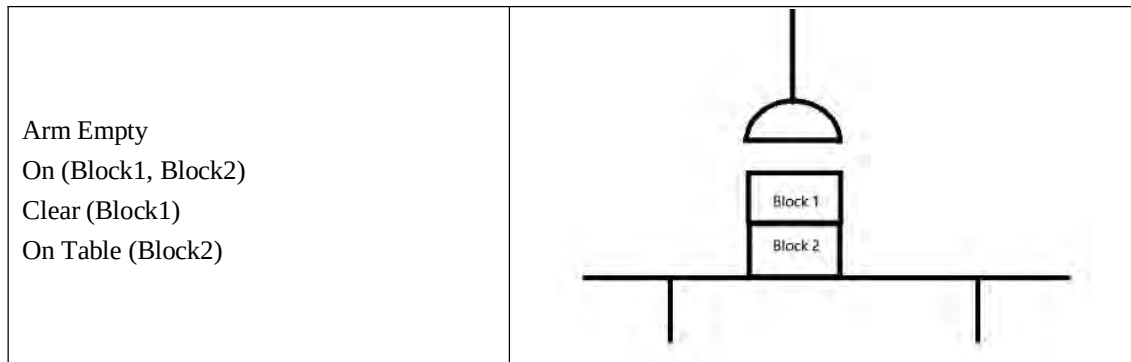
Hold (Block1) Clear (Block2) On Table (Block2)	
--	--

Fig. 5.2: Operation Pick-Up-Block World Problem

We need to check whether we reach goal state or not, as soon as we complete each and every operation. Here the environment looks as shown in Fig 5.2. But this is not a goal state. So, we have to continue the process. Next the block1 needs to be placed on block2. This can be achieved by the following operation.

STACK (Block1, Block2)**Fig. 5.3: Operation Stack-Block World Problem**

After this operation the environment looks as shown in Fig. 5.3. This is clear that we reached the goal state. The plan for reaching goal state is, PICKUP (Block1) and Stack (Block1, Block2). The blocks world is chosen because:

- It is sufficiently simple and well behaved.
- Easily understood.
- It provides a good sample environment to study planning.

5.3.4 State Space Linear Planning

In this process of planning only one goal is solved at a time. It needs strategy that can use a stack of unachieved goals. Advantage and disadvantages of state space linear planning are as follows.

Advantages

- Search space is considerably reduced.
- It has advantages if goals are independent.

Disadvantages

- Possibility of producing sub optimal solutions.
- Planning is incomplete.
- Planning efficiency is based on ordering of goals.

5.3.5 State Space Non-Linear Planning

In Non-linear planning, sub goals are interdependent and can be solved in any order. In this planning, we use goal set instead of goal stack. Search space includes all possible subgoal ordering.

Advantages:

- Non-linear planning may be an optimal solution with respect to plan length (depending on search strategy used).
- Planning is sound (planning completes the desired job) and complete (guarantees to find a plan if one exists).

Disadvantages:

- It takes larger search space, since all possible goal orderings are taken into consideration.
- Requires Complex algorithm

5.3.6 Goal Stack Planning Using STRIPS

This is one of the most important planning algorithms, which is specifically used by STRIPS. The stack is used in an algorithm to hold the action and satisfy the goal. A knowledge base is used to hold the current state actions. Goal stack is similar to a node in a search tree where the branches are created if there is a choice of an action.

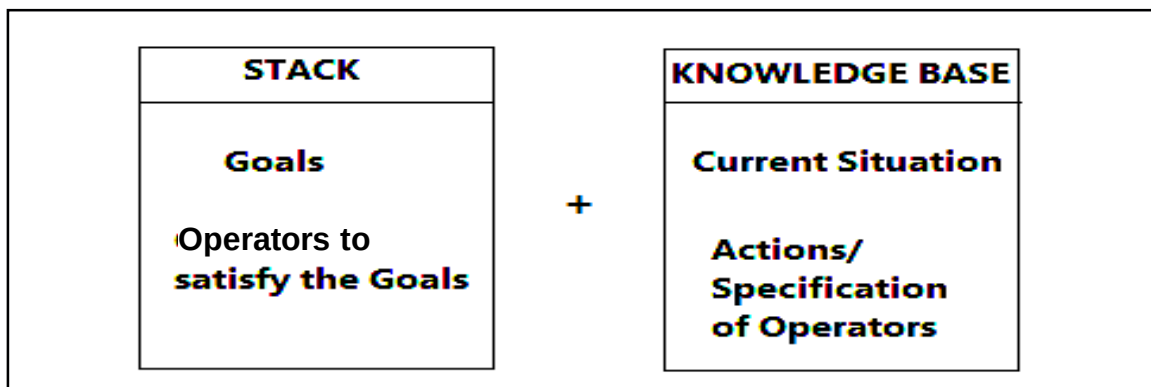


Fig. 5.4: Goal Stack Planning

The important steps of the algorithm are as stated below:

- Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied subgoals on the stack.
- If stack top is a single unsatisfied goal, then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
- If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.
- If stack top is a satisfied goal, then pop it from the stack.

STRIPS

STRIPS stands for "Stanford Research Institute Problem Solver," was the automated planner used in one of the first robots built using AI technology, which is an action-centric

representation for each action and specifies the effect of an action. A STRIP planning problem specifies following.

- An initial state S
- A goal G
- A set of STRIPS actions

The **STRIPS** representation for an action consists of three lists,

PRE list contains predicates which have to be true before operation.

ADD list contains those predicates which will be true after operation

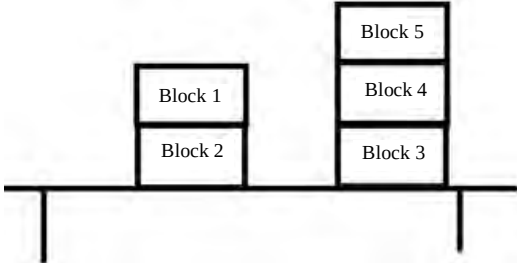
DELETE list contain those predicates which are no longer true after operation.

Predicates not included on either of these lists are assumed to be unaffected by the operation. Frame axioms are specified implicitly in STRIPS which greatly reduces amount of information stored. Table 5.3 shows the action lists for operations of block world problem.

Table 5.3: Operators with Lists

Operator	List
Stack (X, Y)	Pre: CL (Y), HOLD (X) Del: CL (Y), HOLD (X) Add: AE, ON (X, Y)
Unstack (X, Y)	Pre: ON (X, Y), CL (X), AE Del: ON (X, Y), AE Add: HOLD (X), CL (Y)
Pickup (X)	Pre: ONT (X), CL (X), AE Del: ONT (X), AE Add: HOLD (X)
Putdown (X)	Pre: HOLD (X) Del: HOLD (X) Add: ONT (X), AE

Let us consider a Block world problem to understand the STRIPS approach. The initial and Goal States are described in the Fig. 5.5.

Operators	Arm Position
Initial State on (block2, block1) clear (block2) on table (block3) on (block4, block3) on (block5, block4) clear (block5)	

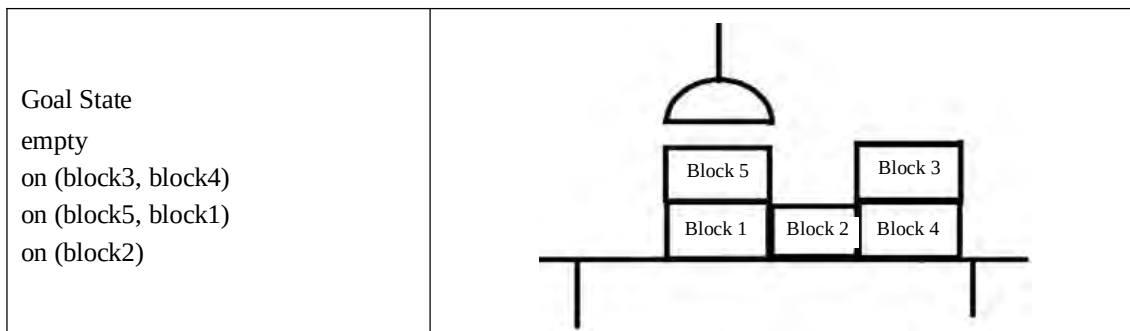
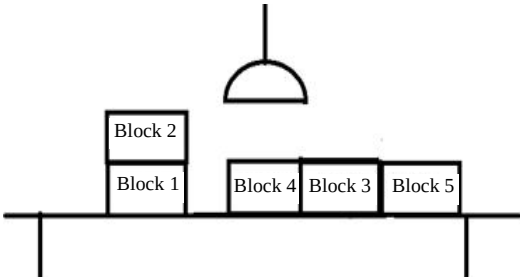
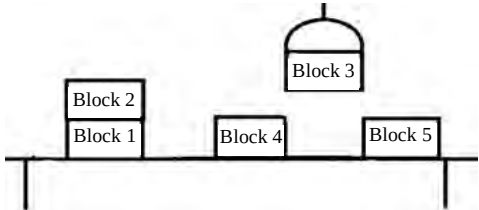
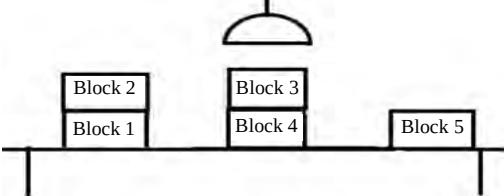
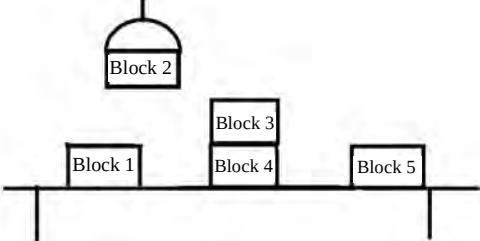
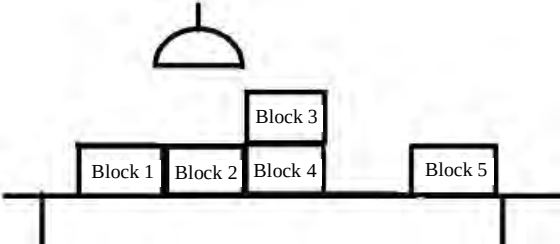


Fig. 5.5: Initial and goal States of Block World Problem-STRIPS approach

Now, let us identify the operators for the given block world problem as shown in the following Fig. 5.6.

Operators	Arm Position
<p>Unstack (block5, block4)</p> <p>Pre: ON (block5, block4), CL (block5), AE</p> <p>Del: ON (block5, block4), AE</p> <p>Add: HOLD (block5), CL (block4)</p>	
<p>Putdown(block5)</p> <p>Pre: HOLD (block5)</p> <p>Del: HOLD (block5)</p> <p>Add: ONT (block5), AE</p>	
<p>Unstack (block4, block3)</p> <p>Pre: ON (block4, block3), CL (block4), AE</p> <p>Del: ON (block4, block3), AE</p> <p>Add: HOLD (block4), CL (block3)</p>	

<p>Putdown(block4) Pre: HOLD (block4) Del: HOLD (block4) Add: ONT (block4), AE</p>	
<p>Pickup (block3) Pre: ONT (block3), CL (block3), AE Del: ONT (block3), AE Add: HOLD (block3)</p>	
<p>Stack (block3, block4) Pre: CL (block3), HOLD (block3) Del: CL (block4), HOLD (block3) Add: AE, ON (block3, block4)</p>	
<p>Unstack (block2, block1) Pre: ON (block2, block1), CL (block2), AE Del: ON (block2, block1), AE Add: HOLD (block2), CL (block3)</p>	
<p>Putdown(block2) Pre: HOLD (block2) Del: HOLD (block2) Add: ONT (block2), AE</p>	

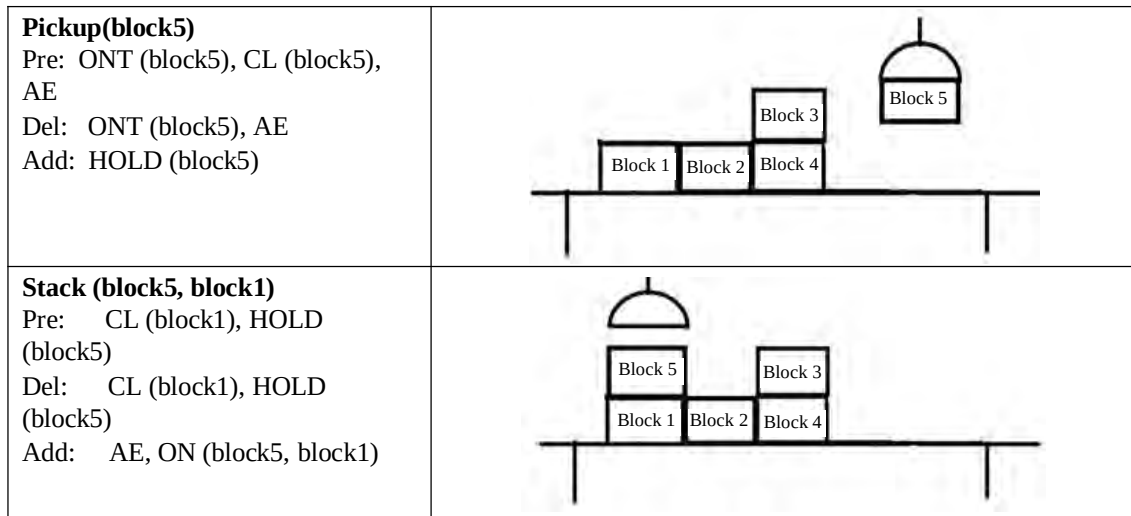


Fig. 5.6: Description of Operators in Block World Problem-STRIPS Approach

After completing all the operations what we found for the given problem, we had reached the goal state.

5.4 LINEAR PLANNING USING GOAL STACK

One of the earliest techniques is planning using goal stack. Problem solver uses single stack that contains sub goals and operators both. Sub goals are solved linearly and then finally the conjoined sub goal is solved. Plans generated by this method will contain complete sequence of operations for solving one goal followed by complete sequence of operations for the next etc. Problem solver also relies on A database that describes the current situation and set of operators with *precondition*, *add* and *delete* lists.

Let us assume that the goal to be satisfied is:

$$\text{GOAL} = G_1 \wedge G_2 \wedge \dots \wedge G_n$$

Sub-goals $G_1, G_2, \dots, G_n$ are stacked with compound goal $G_1 \wedge G_2 \wedge \dots \wedge G_n$ at the bottom. Goal stack is shown in Fig. 5.7.

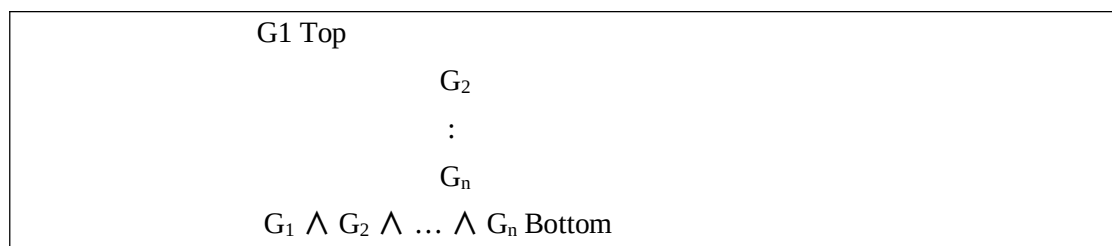


Fig. 5.7: Goal Stack

At each step of problem-solving process, the top goal on the stack is pursued. Algorithm required is given below.

ALGORITHM

- Find an operator that satisfies sub goal G1 (makes it true) and replace G1 by the operator.
- If more than one operator satisfies the sub goal then apply some heuristic to choose one. In order to execute the top most operation, its preconditions are added onto the stack.
- Once preconditions of an operator are satisfied, then we are guaranteed that operator can be applied to produce a new state.
- New state is obtained by using ADD and DELETE lists of an operator to the existing database. Problem solver keeps track of operators applied.
- This process is continued till the goal stack is empty and problem solver returns the plan of the problem.

Let us consider an example of Block world problem using goal stack method. Initial and final states are as shown in Fig. 5.8.

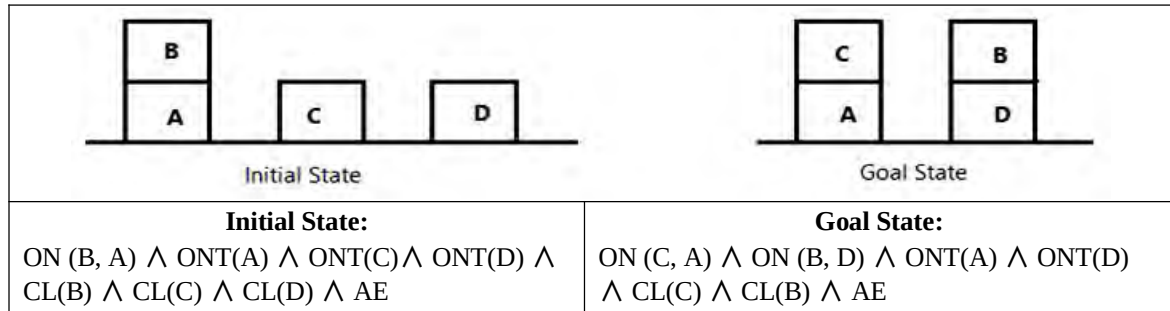


Fig. 5.8: Initial and Goal States-Goal Stack

We notice that following sub-goals in goal state are also true in both initial state and goal states.

$$\text{ONT}(A) \wedge \text{ONT}(D) \wedge \text{CL}(C) \wedge \text{CL}(B) \wedge \text{AE}$$

Therefore, let us represent it as **TSUBG**. We will solve only sub-goals $\text{ON}(C, A)$ & $\text{ON}(B, D)$ and finally make sure that **TSUBG** remains true. Either start solving first $\text{ON}(C, A)$ or $\text{ON}(B, D)$. Let us solve first $\text{ON}(C, A)$

Goal Stack:

ON (C, A)

ON (B, D)

ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG

Now, we need to identify an operator that can solve $\text{ON}(C, A)$. We notice that the operation $S(C, A)$ could only be applied. So, replace $\text{ON}(C, A)$ with $S(C, A)$ in goal stack.

Goal Stack:

S (C, A) ON (B, D) ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG

Stack Operator S (C, A) can be applied if its preconditions are true. So, add its preconditions on the stack.

Goal Stack:

CL(A)	
HOLD(C)	Preconditions of STACK
CL(A) $\wedge$ HOLD(C)	
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

To do this S(C, A) operation all preconditions should be true. In the given problem CL(A) is not true. In order to make the state true, replace CL(A) by **US (B, A)** and write the preconditions of Unstack operator.

Goal Stack:

ON (B, A)	
CL(B)	Preconditions of UNSTACK
AE	
ON (B, A) $\wedge$ CL(B) $\wedge$ AE	
US (B, A)	Operator
HOLD(C)	Preconditions of STACK
CL(A) $\wedge$ HOLD(C)	
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

ON (B, A), CL(B) and AE are all true in initial state, so pop these along with its compound goal. Next pop top operator US (B, A) and produce new state by using its ADD and DELETE lists. Add US (B, A) in a queue of sequence of operator **SQUEUE = US (B, A)**.

A new state1 is produced by using its ADD and DEL lists and it is expressed as

State 1: $ONT(A) \wedge ONT(C) \wedge ONT(D) \wedge HOLD(B) \wedge CL(A) \wedge CL(C) \wedge CL(D)$

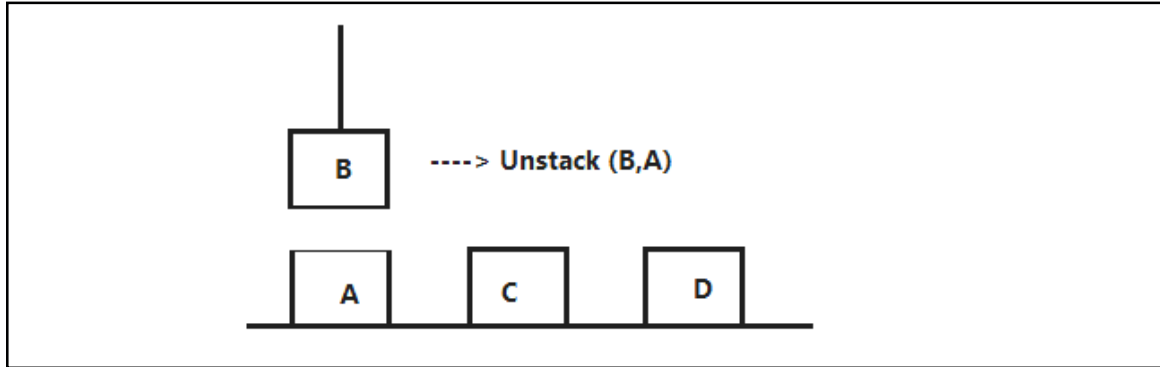


Fig. 5.9: Transition to State1

Goal Stack:

HOLD(C)	Preconditions of STACK
CL(A) $\wedge$ HOLD(C)	
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

To execute the **S(C, A)**, all the preconditions of stack operator should be true. But in this case, HOLD(C) is not true. To make the state true, use the operator **S(B,D)**

Goal Stack:

S (B, D)	Operator
HOLD(C)	
CL(A) $\wedge$ HOLD(C)	Preconditions of STACK
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

Now, let us write the preconditions of **S(B, D)**

Goal Stack:

CL (D) $\wedge$ HOLD (B)	Preconditions of STACK
S (B, D)	Operator
HOLD(C)	
CL(A) $\wedge$ HOLD(C)	Preconditions of STACK
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

Add S (B, D) in a queue of sequence of operators.

SQUEUE = US (B, A), S (B, D)

State 2: $ONT(A) \wedge ONT(C) \wedge ONT(D) \wedge ON(B, D) \wedge CL(A) \wedge CL(C) \wedge CL(B) \wedge AE$

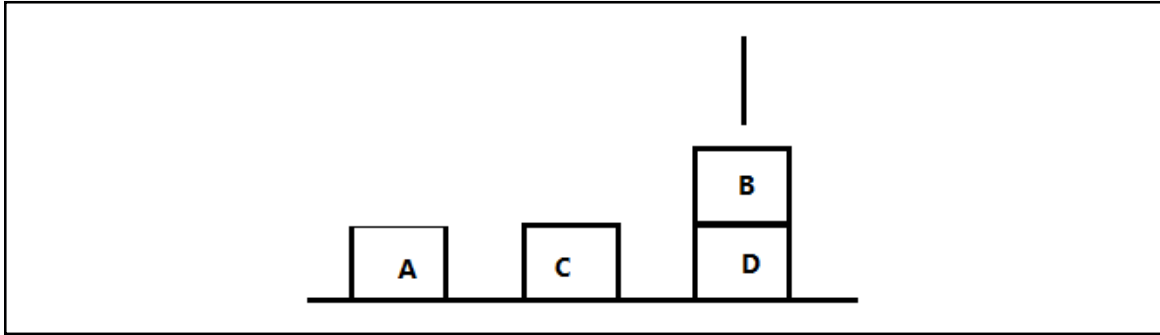


Fig. 5.10: Transition from State 1 to State 2

Goal Stack:

HOLD(C)	
CL(A) $\wedge$ HOLD(C)	Preconditions of STACK
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

To execute **S (C, A)** all the preconditions should be true. Here HOLD(C) is not true. In order to make the state true use the operator **PU(C)** and write the preconditions.

Goal Stack:

ONT (C) $\wedge$ CL (C) $\wedge$ AE	Preconditions of PICKUP
PU (C)	Operator
HOLD(C)	
CL(A) $\wedge$ HOLD(C)	Preconditions of STACK
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

Here, all the preconditions of PU operator are true. So, add **PU(C)** in a queue of sequence of operators.

SQUEUE = US (B, A), S (B, D), PU(C)

State 3: $ONT(A) \wedge HOLD(C) \wedge ONT(D) \wedge ON (B, D) \wedge CL(A) \wedge CL(B)$

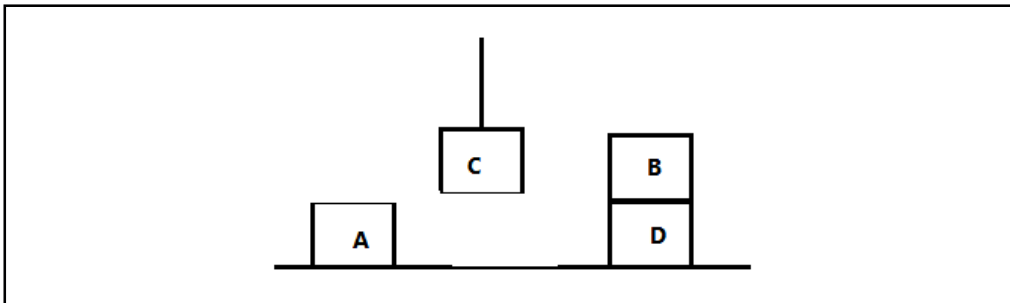


Fig. 5.11: Transition from State 2 to State 3

Goal Stack:

HOLD(C)	
CL(A) $\wedge$ HOLD(C)	Preconditions of STACK
S (C, A)	Operator
ON (B, D)	
ON (C, A) $\wedge$ ON (B, D) $\wedge$ TSUBG	

Here, all the preconditions of **S(C, A)** are true. Hence, add **S(C, A)** in queue.

SQUEUE = US (B, A), S (B, D), PU(C), S (C, A)

State 4: $ONT(A) \wedge ON(C, A) \wedge ONT(D) \wedge ON(B, D) \wedge CL(C) \wedge CL(B) \wedge AE$

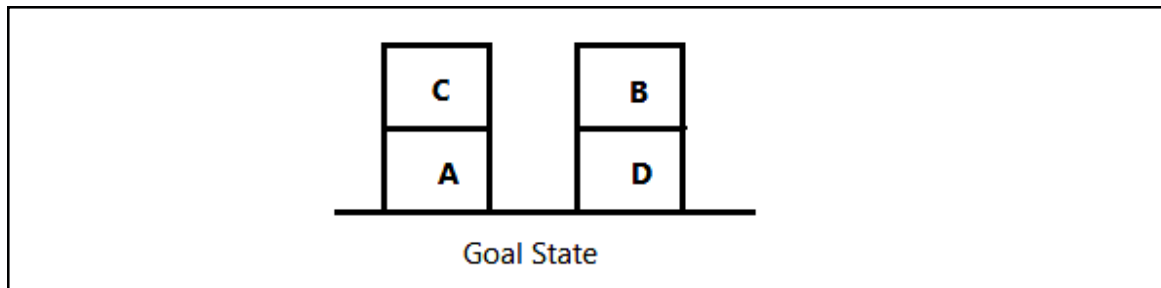


Fig. 5.12: Transition from State 3 to State 4

Finally, we reached goal state after **S (C, A)** using Goal Stack algorithm. Hence, the plan for the given problem is,

Unstack (B, A)

Stack (B, D)

Pickup(C)

Stack (C, A)

5.5 BASICS OF MEANS END ANALYSIS (MEA)

Means end analysis is a technique used to solve problems in AI programs. Because, it enhances problem resolution. MEA solves problems by defining the goal and establishing the right action plan. This technique is used in AI programs to limit search. This technique enables the use of combined forward and backward strategies to solve complex problems. With these mixed strategies, complex problems can be tackled first and then it is followed by smaller problems.

In this technique, the system evaluates the differences between the current state or position and the target or goal state. It then decides the best action to be undertaken to reach the end goal. Means end analysis uses the following processes to achieve its objectives.

- First, the system evaluates the current state to establish whether there is a problem. If a problem is identified, then it means that an action should be taken to correct it.

- The second step involves defining the target or desired goal that needs to be achieved.
- The target goal is split into sub-goals, that are further split into other smaller goals.
- This step involves establishing the actions or operations that will be carried out to achieve the end state.
- In this step, all the sub-goals are linked with corresponding executable actions (operations).
- After that is done, intermediate steps are undertaken to solve the problems in the current state. The chosen operators will be applied to reduce the differences between the current state and the end state.
- This step involves tracking all the changes made to the actual state. Changes are made until the target state is achieved

Fig 5.13 shows how the target goal is divided into sub-goals, that are then linked with executable actions.

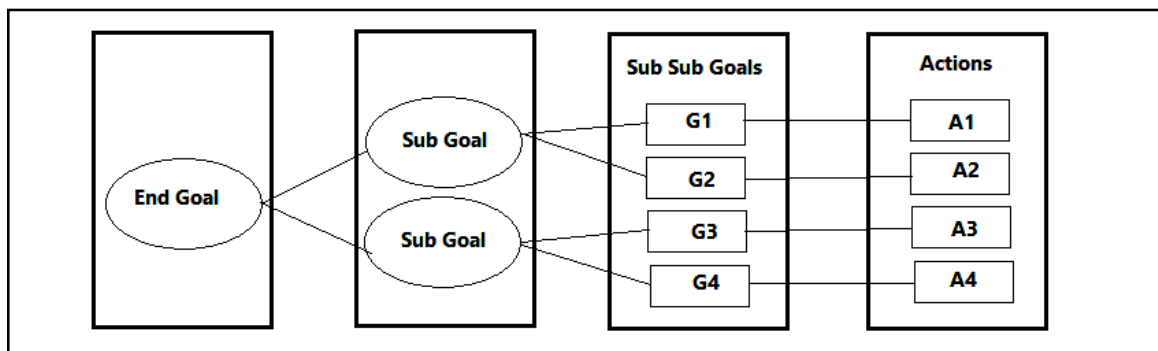


Fig. 5.13: Division of Target Goal

Algorithm steps for Means End Analysis

The following are the algorithmic steps for means end analysis:

- Conduct a study to assess the status of the current state. This can be done at a macro or micro level.
- Capture the problems in the current state and define the target state. This can also be done at a macro or micro level.
- Make a comparison between the current state and the end state that you defined. If these states are the same, then perform no further action. This is an indication that the problem has been tackled. If the two states are not the same, then move to step 5.
- Record the differences between the two states at the two aforementioned levels (macro and micro).
- Transform these differences into adjustments to the current state.
- Determine the right action for implementing the adjustments in step 5.

- Execute the changes and compare the results with the target goal.
- If there are still some differences between the current state and the target state, perform course correction until the end goal is achieved.

Example of problem-solving in Means End Analysis

Let us assume that we have the following initial state. We want to apply the concept of means end analysis to establish whether there are any adjustments needed. The first step is to evaluate the initial state and compare it with the end goal to establish whether there are any differences between the two states. The following Fig. 5.14. shows a comparison between the initial state and the target state.

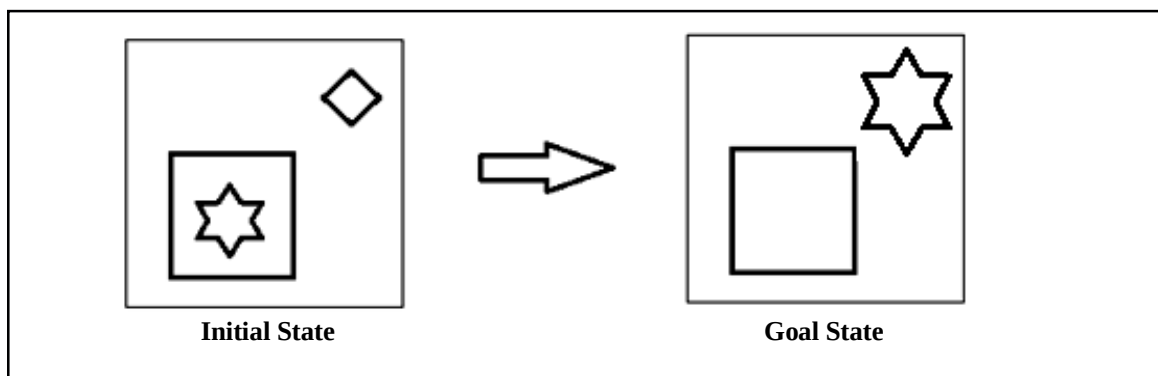


Fig. 5.14: Comparison between Initial and Target State

The image above shows that there is a difference between the current state and the target state. This indicates that there is a need to make adjustments to the current state to reach the end goal. The goal can be divided into sub-goals that are linked with executable actions or operations. The following are the three operators that can be used to solve the problem.

1. Delete operator: In fig. 5.15, the diamond symbol at the top right corner in the initial state does not exist in the goal state. The diamond symbol can be removed by applying the *delete* operator.

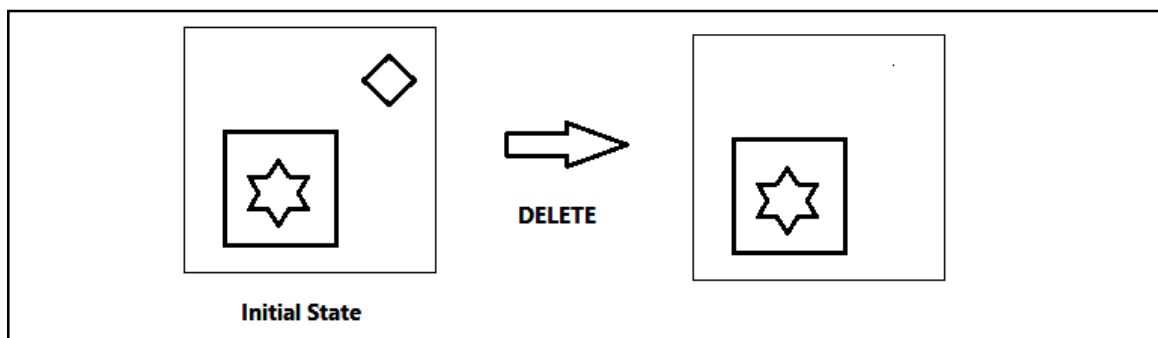


Fig. 5.15: DELETE Operator

2. Move operator: We will then compare the new state with the end state. The hexagram boundary in the new state is inside the square while the hexagram boundary in the end state is at

the top right corner. We will move this hexagram boundary symbol to the right position by applying the *move operator*.

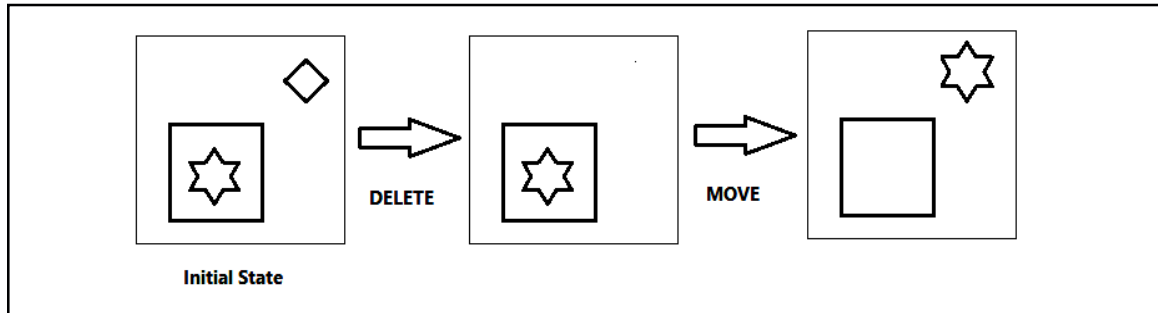


Fig. 5.16: DELETE and MOVE Operator

3. Expand operator: After evaluating the new state generated in step 2, we find that the hexagram boundary symbol is smaller than the one in the end state. We can increase the size of this symbol by applying the *expand operator*.

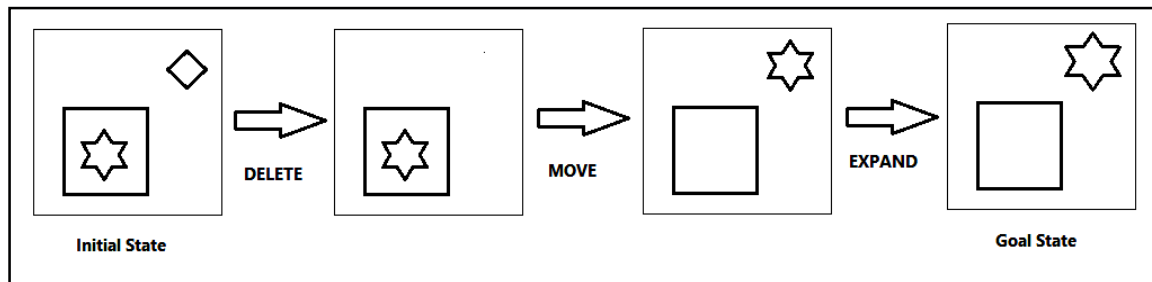


Fig. 5.17: DELETE, MOVE and EXPAND Operator

After applying the three operators above, we will find that the state in step 3 is the same as the end state. There are no differences between these two states, which means that the problem has been solved.

Certain operators are available for solving problems using MEA. Description of these operators along with precondition and result are shown in following Table 5.4.

Table 5.4 Description of Operators with Precondition

Operator	Description	Precondition	Result
PUSH (O, P)	To push large object to place P	At (robot, O) $\wedge$ Large(O) $\wedge$ Clear (O) $\wedge$ AE	At (O, P) $\wedge$ At (robot, P)
CARRY (O, P)	To carry small object O to place P	At (robot, O) $\wedge$ Small(O) $\wedge$ Clear (O) $\wedge$ AE	At (O, P) $\wedge$ At (robot, P)
WALK (P)	Walk to place P	None	At (robot, L)
PU (O)	Pick up object O	At (robot, O)	Hold(O)
PD (O)	Pick down object O	Hold (O)	$\sim$ Hold(O)
PLACE (O1, O2)	Put object 1 on Object 2	At (robot, O2) $\wedge$ Hold(O1)	On (O1, O2)

5.5.1 Applications of Means End Analysis

Means end analysis can be applied in the following fields:

Organizational planning

Means end analysis is used in organizations to facilitate general management. It helps organizational managers to conduct planning to achieve the objectives of the organization. The management reaches the desired goal by dividing the main goals into sub-goals that are linked with actionable tasks.

Business transformation

This technique is used to implement transformation projects. If there are any desired changes in the current state of a business project, means end analysis is applied to establish the new processes to be implemented. The processes are split into sub-processes to enhance effective implementation.

Gap analysis

Gap analysis is the comparison between the current performance and the required performance. Means end analysis is applied in this field to compare the existing technology and the desired technology in organizations. Various operations are applied to fill the existing gap in technology.

5.6 NON-LINEAR PLANNING STRATEGY

A Non-linear plan consists of sub-problems which are solved simultaneously. In case of the goal stack planning, as discussed earlier, it poses some problems. Achieving a goal could possibly undo any of the already achieved goals and it is called as Sussman's anomaly. In linear planning just one goal is taken at a time and solved completely before the next one is taken.

Let us take an example. You want to take the car for servicing and have to make an important call over phone. In case if you are following linear planning, then first you will complete the goal of making a phone call and then you will take the car for servicing. Instead of completing both the tasks in a linear pattern, if you complete the task 1 partially, *i.e.*, first start the car then complete the task 2 of phone call and then finally, complete the task 1 by leaving the car at the service station which can be an example of non-linear planning.

Generally, we use a technique called constraint posting which often comes with Non-Linear Planning. The main idea of constraint posting is to develop a plan incrementally by,

- Adding or Suggesting Operators
- Ordering Operators
- Binding the variable to the Operators

At any given time in problem solving process, we may come across a set of useful operators but we may not have a clear idea of how these operators should be ordered with respect to each

other. In this case, operators are partially ordered to generate the actual plan, then we convert the partial order into any number of total orders.

Let us now examine several operations for nonlinear planning in a constraint posting environment by incrementally generating a nonlinear plan to solve the Sussman anomaly problem. We begin with a null plan. Next, we look at the goal state and put forward steps for achieving that goal. Also, we use heuristic in this plan and it is given below.

- **Step Addition:** Creating new steps for a plan
- **Promotion:** Restrict one step to come before another in a final plan.
- **DE Clobbering:** Placing one step S_2 between two old steps S_1 and S_3 such that S_2 makes it possible for S_3 to be apply which was previously prevented by S_1 .
- **Simple Establishment:** Assigning a value to a variable.
- **Separation:** Preventing the assignment of certain values to a variable.

Let us consider the initial state as our current state and start the plan creation with the goal state as shown in Fig. 5.18.

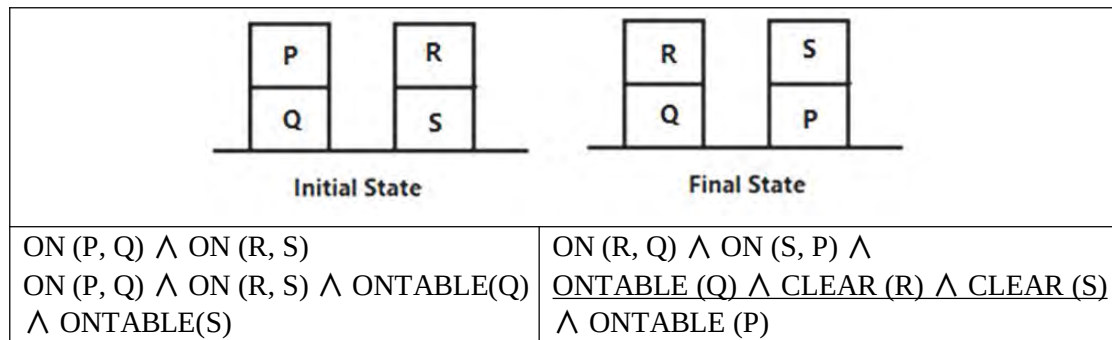


Fig. 5.18: Initial and Final states with Subgoals

From the Fig. 5.18, it is clear that there are some subgoals which are already true and these are highlighted with Underline. Therefore, we need not to consider that part in our process. Let us consider the unachieved goals.

$$\text{ON (R, Q) } \wedge \text{ ON (S, P) } \wedge \text{ CLEAR (S) } \wedge \text{ ONTABLE (P)}$$

These three subgoals, as indicated in the above diagram, are not true in our current situation. Therefore, use step addition to achieve these subgoals as shown in Fig. 5.19.

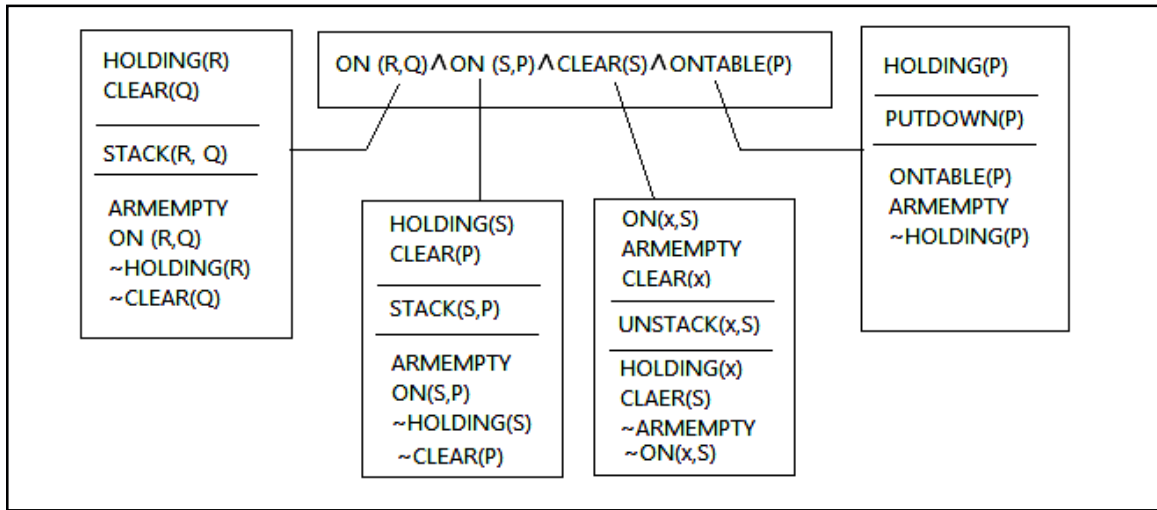


Fig. 5.19: Step Addition

Now check the precondition of all the actions are satisfied or not. The unsatisfied preconditions are highlighted with rectangular box in Fig. 5.20.

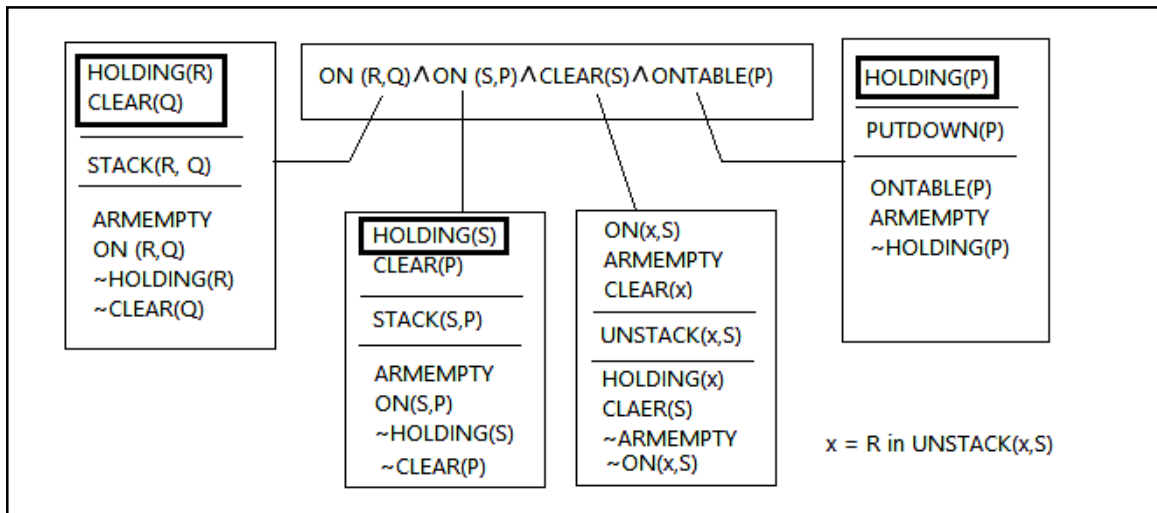


Fig. 5.20: Checking Precondition

Now let us try to achieve the preconditions that are highlighted above in the diagram so that various actions can be used to achieve the goal.

At first, we look in to the action $STACK(R, Q)$ where $HOLDING(R)$ and $CLEAR(Q)$ is not true. This can be determined by checking it with the current situation, where the robotic arm is empty. So, use Step Addition to satisfy $HOLDING(R)$ and it can be done by using $PICKUP(R)$ whose effect is $HOLDING(R)$. Also, we can use Step Addition to include $UNSTACK(x, Q)$ to satisfy $CLEAR(Q)$. Similarly, the preconditions of other action $STACK(S, P)$ and $PUTDOWN(P)$ are achieved by including relevant actions through step addition. It is not necessary to use step

addition to satisfy a goal or precondition all the time. We can use promotion if it helps to satisfy a goal. If both promotion and step addition could make the precondition/goal to be achieved then it is better to prefer promotion as compared to step addition. This could be observed in the Fig. 5.20.

Now we observe that, $HOLDING(P)$ in the action $PUTDOWN(P)$ is not true. Therefore, we can achieve this by including $PICKUP(P)$. We can see that the $UNSTACK(x, Q)$ also will make the $HOLDING(P)$ to be true. Hence, we have included $UNSTACK(x, Q)$ already in our partial plan. So just use step promotion by suggesting that $UNSTACK(x, Q)$ comes before $PUTDOWN(P)$. By this we can possibly eliminate the additional action. The modified partial plan will be like the one given in the Fig. 5.21.

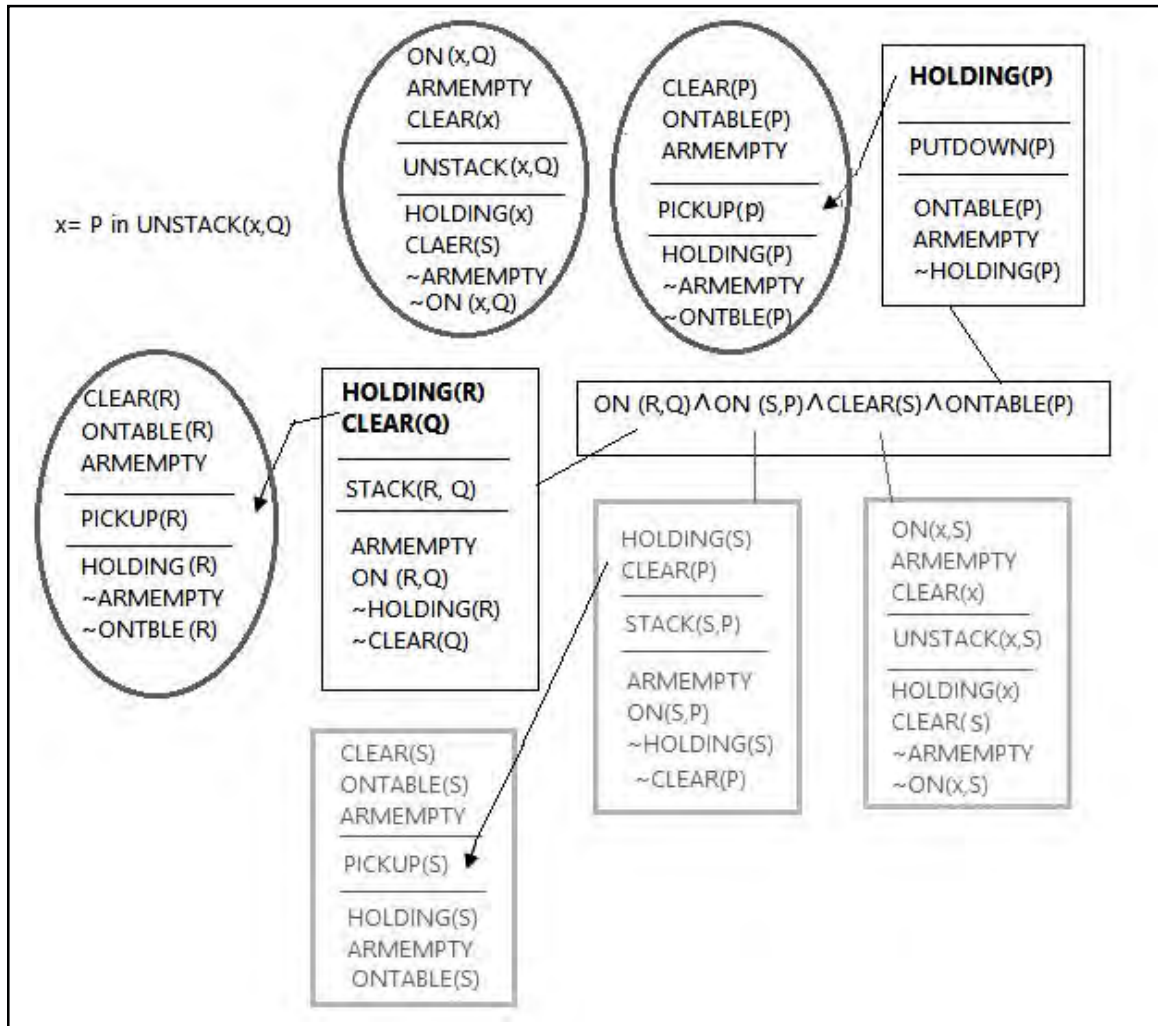


Fig. 5.21: Step Promotion

In the Fig. 5.22, the satisfied preconditions are marked thick and need to apply promotion as below.

$UNSTACK(x, Q) \leftarrow PUTDOWN(P)$

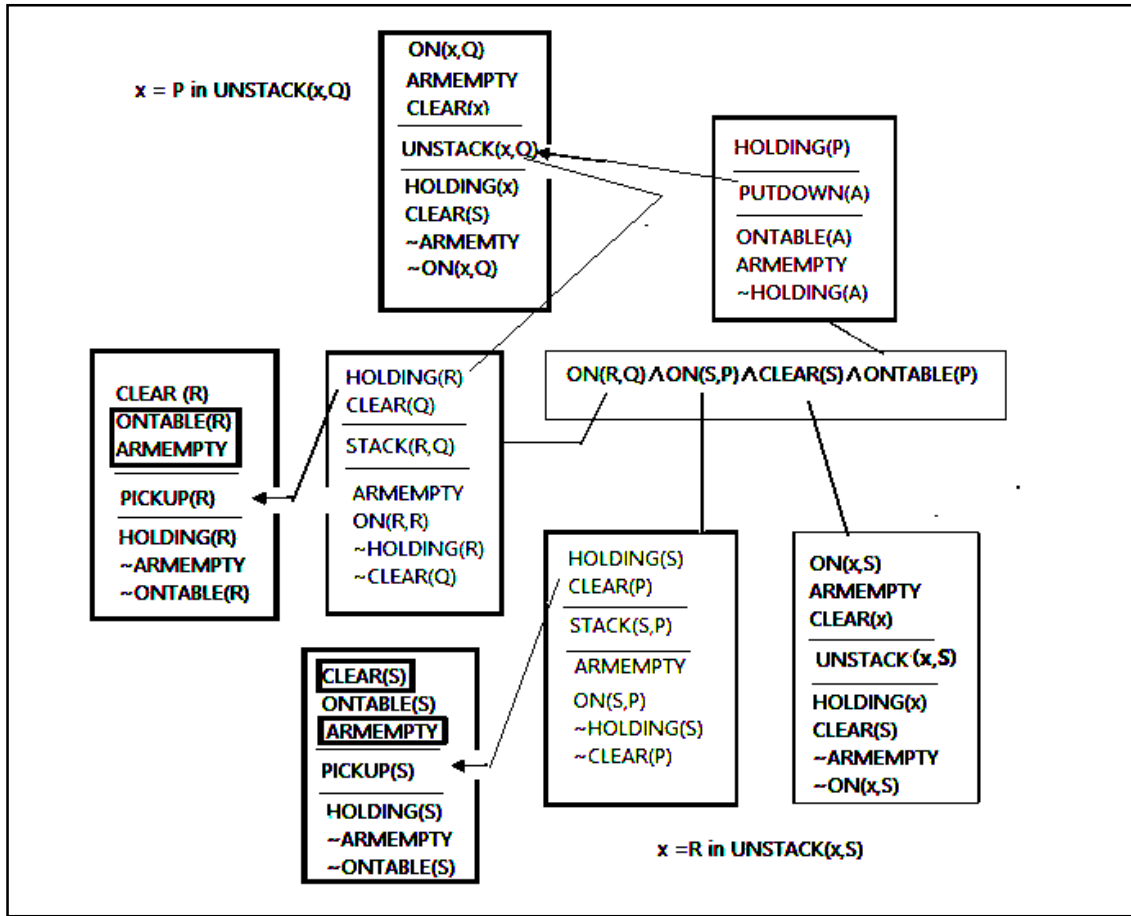


Fig. 5.23: Step Promotion

While considering the $UNSTACK(x, Q)$ we can use simple establishment and assign P for x and then we can see that all the preconditions are satisfied except $ARMEMPTY$. Hence, we use $PUTDOWN(R)$ before $UNSTACK(x, Q)$ and this achieves $ARMEMPTY$. Let us use promotion where $UNSTACK(x, Q)$ comes before $PICKUP(R)$ and $PICKUP(S)$ as shown in Fig. 5.23. Otherwise, the $ARMEMPTY$ can be made false.

$UNSTACK(x, Q) \leftarrow PICKUP(R)$

$UNSTACK(x, Q) \leftarrow PICKUP(S)$

$PUTDOWN(R) \leftarrow UNSTACK(x, Q)$

Finally, every action precondition got satisfied and let us construct the total plan from the partial plan as shown in Fig. 5.24. To do that we need consider the dependency among various action that we have seen so far. All the actions with dependencies (from the beginning) are furnished below.

$UNSTACK(x, Q) \leftarrow PUTDOWN(P)$

$PICKUP(R) \leftarrow STACK(R, Q)$

PICKUP(S) $\leftarrow$ STACK (S, P)
 UNSTACK (x, Q) $\leftarrow$ PICKUP(R)
 UNSTACK (x, Q) $\leftarrow$ PICKUP(S)
 PICKUP(S) $\leftarrow$ PICKUP(R)
 PUTDOWN(R) $\leftarrow$ PIKUP (R)
 PUTDOWN(R) $\leftarrow$ UNSTACK (x, Q)
 UNSTACK (R, S) $\leftarrow$ PUTDOWN(R)
 PICKUP(S) $\leftarrow$ PICKUP(R)

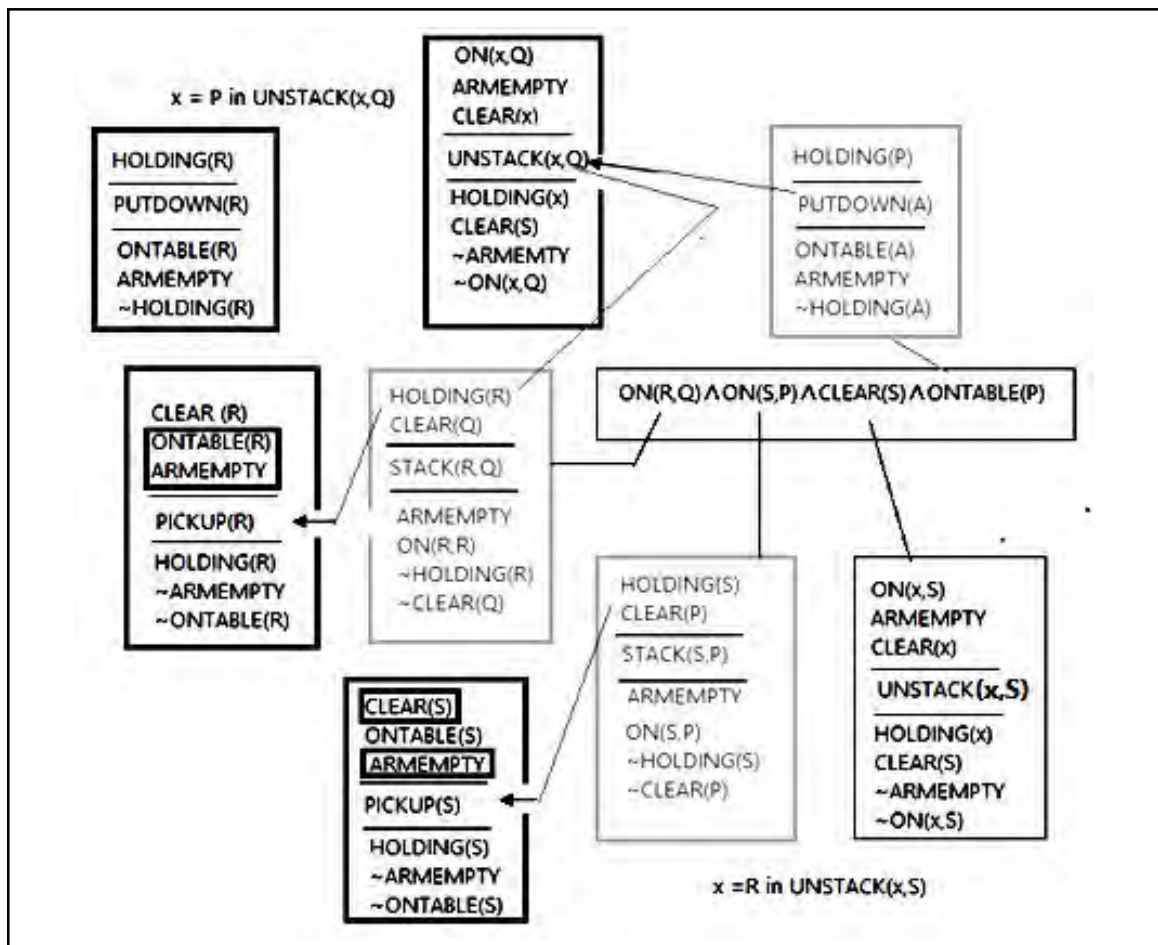


Fig. 5.24: Step Addition -Partial Plan

In Table 5.5, a complete plan is generated summarising all the above steps.

Table 5.5 The Complete Plan

Step-1:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R)
Step-2:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R) $\leftarrow$ UNSTACK (P, Q)
Step-3:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R) $\leftarrow$ UNSTACK (P, Q) $\leftarrow$ PUTDOWN(P)
Step-4:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R) $\leftarrow$ UNSTACK (P, Q) $\leftarrow$ PUTDOWN(P)
Step-5:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R) $\leftarrow$ UNSTACK (P, Q) $\leftarrow$ PUTDOWN(P) $\leftarrow$ PICKUP (S)
Step-6:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R) $\leftarrow$ UNSTACK (P, Q) $\leftarrow$ PUTDOWN (P) $\leftarrow$ PICKUP (S) $\leftarrow$ STACK (S, P)
Step-7:	UNSTACK (R, S) $\leftarrow$ PUTDOWN (R) $\leftarrow$ UNSTACK (P, Q) $\leftarrow$ PUTDOWN (P) $\leftarrow$ PICKUP (S) $\leftarrow$ STACK (S, P) $\leftarrow$ PICKUP (R)
Complete Plan:	UNSTACK (R, S) $\leftarrow$ PUTDOWN(R) $\leftarrow$ UNSTACK (P, Q) $\leftarrow$ PUTDOWN (P) $\leftarrow$ PICKUP (S) $\leftarrow$ STACK (S, P) $\leftarrow$ PICKUP (R) $\leftarrow$ STACK (R, Q)

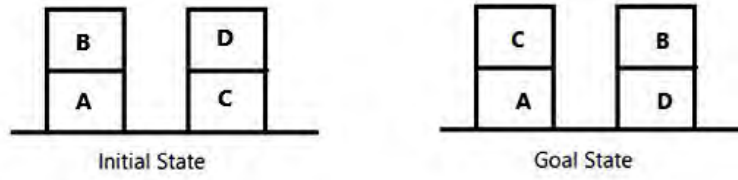
QUESTIONS

- 5.1 Explain the types of planning systems.
- 5.2 Explain Blocks World problem.
- 5.3 Explain Goal Stack problem using STRIPS.
- 5.4 What is Linear Planning Using Goal Stack? Explain.
- 5.5 Explain Means End analysis with an example.
- 5.6 A monkey and bananas problem is described as below.

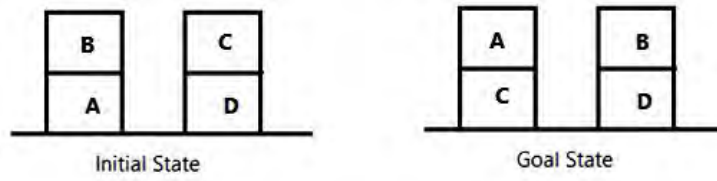
A monkey is in a room with bananas hanging from the ceiling. The bananas hanging are out of reach. A box is available that will enable the monkey to reach the bananas if he climbs on it. Initially, the monkey is at A, the bananas at B and the box at C. The actions available to the monkey include Go from one place to another, Push an object from one place to another, Climb Up onto or Climbdown from an object and Grasp or Ungrasp an object. Grasping results in holding the object if the monkey and object are in the same place at the same height.

- (i) Write Down the start state description
 - (ii) Write down STRIPS style definition of the six actions
 - (iii) Explain the plan using goal stack method.
- 5.7 Consider the block world problem given below. Solve them using following methods.
 - STRIPS
 - Goal Stack
 - Constraint Posting

(i)



(ii)

☐ ☐ ☐

6

KNOWLEDGE REPRESENTATION

6.1 INTRODUCTION

As we know, humans are the best at understanding reasoning and interpreting knowledge. Humans can understand knowledge in terms of information and accordingly they perform various actions in the real world. But how machine can understand knowledge and represent it in real world situations? So, in this chapter we will discuss about how we can represent knowledge in the artificial intelligence.

Knowledge Representation is a subarea of Artificial Intelligence which deals with understanding, designing and implementing ways of representing information in computers so that programs can use this information. knowledge representation and reasoning are responsible for representing information about the real world, which makes computer to understand and utilize the knowledge to solve the complex real-world problem such as diagnosis of medical condition or communicating with human in natural language. Knowledge representation is not just storing data into some databases but also enable intelligent machine to learn from that knowledge and experience so that it can be behave intelligently like a human. To solve the complex problems encountered in AI, we need a large amount of knowledge and suitable mechanisms for representing and manipulating all that knowledge. Knowledge can take many forms. Some simple examples are:

- Ravi has an umbrella
- It is raining
- An umbrella stops you getting wet when it is raining
- An umbrella will only stop you getting wet if it is used properly
- Umbrellas are not so useful when it is very windy

The object of a knowledge representation is to express knowledge in a computer understandable form, so that it can be used to enable our AI agents to perform well. In order to solve problems that normally requires human expertise, a knowledge representation can be defined by two aspects:

Syntax: Syntax refers to the way how the words are ordered to form sentences. In other words, it refers to the rules of grammar in sentence structure. Strong syntax requires subject-verb agreement, proper word choice and phrases or words in the correct order.

Semantics: Semantics refers to the meaning of a sentence. With improper semantics and grammatically incorrect ordering of words, the meaning of a sentence would be completely different. In Linguists, the semantics are categorised as, *lexical semantics* and *conceptual semantics*. *lexical semantics* deals with the study of word meanings and relations. On the other hand, *conceptual semantics* deals with the ways in which semantics are understood and acquired by people while sharing a language.

Properties of Knowledge Representation

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

Representational Adequacy: KR should have an ability to represent different kinds of knowledge which might be needed in that domain.

Inferential Adequacy: It should have an ability to come up with new structures (corresponding to new knowledge) from existing structures.

Inferential Efficiency: It should have an ability to incorporate additional mechanism into the knowledge structure which can be used to focus the attention of the inference mechanisms and direct them towards most promising directions.

Acquisitional Efficiency: It should have an ability to acquire new information through automated methods instead of depending upon human intervention.

It should be noted that no single KR system optimizes all of the capabilities mentioned above. As a result, we use multiple techniques for knowledge representation. Some programs rely on more than one technique.

6.2 APPROACHES TO KNOWLEDGE REPRESENTATION

Goal of knowledge representation is to make the representation techniques to be rich enough to express the knowledge needed to solve the problem. It should be as close to the problem as possible like compact, natural, maintainable and amenable to efficient computation. The ease with which a problem can be solved depends upon the type of knowledge representation approach. The effective representation of domain knowledge is considered to be the keystone to the success of AI programs. To construct an intelligent computer system, a primary consideration is to represent large amounts of knowledge that allows effective use and efficiently organizing information to facilitate making the recommended inferences. There are four approaches to knowledge representation as discussed below.

6.2.1 Relational Knowledge

Relational knowledge provides a framework to compare two objects based on attributes. It is one of the simplest ways of storing information by using relational method and each information about the object is set out systematically in columns. Though Relational knowledge representation has little opportunity for inference, it is popular among database systems. For Example: The following Table 6.1 illustrates the simple relational knowledge representation.

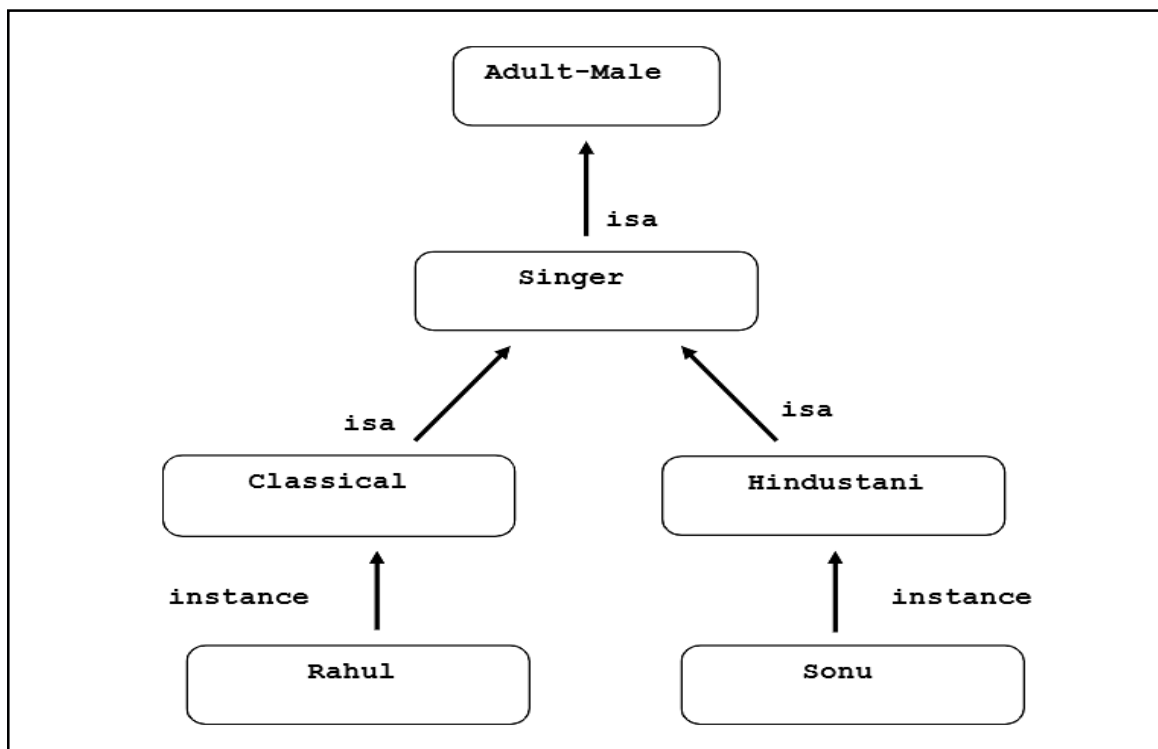
Table 6.1: Relational Knowledge Representation

Singer	Style	Gender	Age
Shreya	Classical	Female	35
Sonu	Hindustani	Male	38
Rahat	Gajal	Male	40
Sunidhi	Sufi	Female	35

In the Table 6.1, it is not possible to answer simple question such as: “Who is the best singer?” But if a procedure for finding best singer is provided, then these facts will enable that procedure to compute an answer.

6.2.2 Inheritable Knowledge

In inheritable knowledge approach, all data must be stored into a hierarchy of classes. This provides the opportunity for inference and we can apply inheritance property to have inheritable knowledge. This way, the relations between instance and class can be identified. Unlike Simple Relations, here, the objects are represented as nodes and we use arrows to point from objects to their values. This approach contains inheritable knowledge which shows a relation between instance and class and therefore, it is also called instance relation. For example, refer Fig. 6.1.

**Fig. 6.1: Inheritable Knowledge Representation**

6.2.3 Inferential Knowledge

Inferential knowledge represents knowledge in the form of formal logics. This approach can be used to derive more facts. Let us consider following two statements.

Ram is a man

All men are mortal

Then it can be represented as,

man (Ram)

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

6.2.4 Procedural Knowledge

Procedural knowledge approach uses small programs which describes how to do a specific task and how to proceed. It may have inferential efficiency, but no scope for inferential adequacy and acquisitional efficiency. In this approach, one important rule used is *If-Then* rule. In this knowledge, we can use various coding languages such as LISP and PROLOG. We can easily represent heuristic or domain-specific knowledge using this approach.

6.3 KNOWLEDGE REPRESENTATION USING SEMANTIC NETWORK

Semantic networks are a commonly used knowledge representations in AI. Semantic networks are an alternative to predicate logic as a form of knowledge representation. In semantic net, we can represent our knowledge in the form of graphical networks. Graphical network is a graph consisting of nodes which are connected by edges. Nodes represent objects and the links between nodes represent relationships between these objects. These links are labelled to indicate the nature of the relationship. This representation consists of mainly two types of relations:

Is-a relation (Inheritance)

Kind-of-relation

Example 6.1: Following are the statements which are represented in the form of nodes and links.

Statements:

Milo is a cat.

Milo is a mammal

Milo is owned by Ravi

Milo is black colored.

All Mammals are animal.

Solution: Fig 6.2.

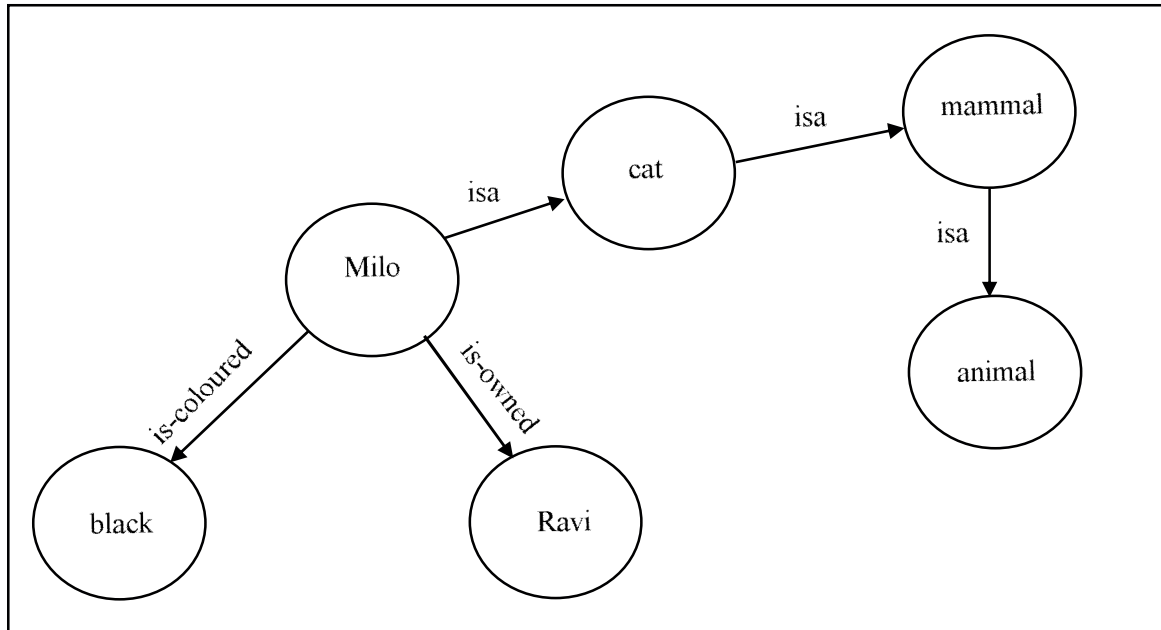


Fig. 6.2: Nodes and Links

In the above diagram, we have represented the different type of knowledge in the form of nodes and edges. Each object is connected with another object by some relation.

Drawbacks in Semantic Representation:

- Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible that after traversing the entire tree, one may find that the solution does not exist in that network.
- Semantic networks try to model human-like memory (Which has 10¹⁵ neurons and links) to store the information. But in practice, it is not possible to build such a vast semantic network.
- These types of representations are inadequate as they do not have any equivalent quantifier, *e.g.*, for all, for some, none, etc.
- Semantic networks do not have any standard definition for the link names.
- These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic Network:

- Semantic networks are natural representation of knowledge.
- Semantic networks convey meaning in a transparent manner.
- These networks are simple and easily understandable.

6.3.1 Inheritance in Semantic Net

Inheritance allows us to specify the properties of a superclass. Prolog equivalent captures an important property of semantic net that they may be used for a form of inference known as inheritance. The idea of this is that if an object belongs to a class (indicated by is a link) it inherits all the properties of that class. For example, if we say that *all mammals give birth to live babies* and *all dogs are mammals* and that *Tommy is a dog*, then we can conclude that *Tommy gives birth to live mammals*. In our example, mammals are the superclass of dogs and Tommy. Dogs are the subclass of mammals and superclass of Tommy. Although inheritance is a useful way to express generalization about a class of objects, in some cases we need to express exceptions to those generalizations such as “*Male animals do not give birth*” or “*Female dogs below the age of 6 months do not give birth*”. In such cases, we say that the default value has been overridden in the subclass. Prolog language is more convenient to represent semantic structure in terms of facts and inheritance queries.

Prolog Fact

A Prolog fact is a predicate expression which makes a declarative statement about the problem. If a variable present in a Prolog expression, it is assumed to be universally quantified. The facts in Prolog can be written as shown below.

```
likes (romeo, juli). /* Romeo likes Juli */
likes (X, juli). /* Everyone likes Juli */
likes (romeo, Y). /* Romeo likes everybody */
likes (romeo, Y), likes (Y, romeo). /* Romeo likes everybody and everybody likes
Romeo */
likes (romeon, juli); likes (romeo, mary). /* Rome likes Juli or Rome likes Mary */
not (likes (romeo, pizza)). /* Romeo does not like pizza */
likes (romeo, juli): - likes (romeo, juli). /* Romeo likes Juli if Romeo likes Mary*/
```

Rules in Prolog

A rule in prolog is a predicate expression which uses a logical implication (:-) to describe a relationship between facts. Thus, a prolog rule takes the form

```
left_hand_side:- right_hand_side.
```

This sentence is interpreted as: *left hand side if right hand side*. The left-hand side is restricted to a single, positive, literal, which means it must consist of a positive atomic expression. It cannot be negated and it cannot contain logical connectives. This notation is known as a Horn clause. In Horn clause logic, the left-hand side of the clause is the conclusion, and must be a single positive literal. The right-hand side contains the premises. The Horn clause calculus is equivalent to the first-order predicate calculus.

Examples of valid rules:

friends (X, Y):- likes (X, Y), likes (Y, X). /* X and Y are friends if they like each other */
 hates (X, Y):- not (likes (X, Y)). /* X hates Y if X does not like Y. */
 enemies (X, Y):- not (likes (X, Y)), not (likes (Y, X)). /* X and Y are enemies if they
 don't like each other */

Examples of invalid rules:

Left_ of (X, Y):- right_ of (Y, X) /* Missing a period */
 likes (X, Y), likes (Y, X):- friends (X, Y). /* LHS is not a single literal */
 not (likes (X, Y)):- hates (X, Y). /* LHS cannot be negated */

Queries

The Prolog interpreter responds to queries about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain. In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers “yes” and displays any variable bindings that it made in coming up with the answer. If it fails to prove the query true, it answers “No”.

Whenever you run the Prolog interpreter, it will prompt you with? For example, suppose our database consists of the following facts about a fictitious family as shown in the Table 6.2.

Table 6.2: Queries for inheritance

Facts	Query	Result
Brother of (ram, sham).	?- brother of (ram, sham).	Yes
Brother of (ram, hari).	?- brother of (ram, sham).	No
Sister of (smitha, ram).	?- sister of (smitha, ram).	Yes

6.4 NEED OF EXTENDED FORM OF SEMANTIC NETWORK FOR KR

An extended form of semantic network which can be regarded as a syntactic variant of the clausal form of logic. We define top-down and bottom-up inference and resolution more generally for the extended semantic network. In particular, top-down inference gives us a procedural interpretation of reasoning in the network. Not only can the extended semantic network be regarded as a syntactic variant of logic, but it can also be used as an abstract data structure for the representation of clauses in an implementation of a predicate logic proof procedure. The semantic network data structure provides an indexing scheme and helps to guide the search for a solution. In particular, as a data structure for an interpreter of predicate logic programs, it guides the execution of procedure calls. The strategy suggested by the network gives the proof procedure a path-finding flavour.

6.4.1 Simple Semantic Network

A semantic network is a knowledge structure or a graph whose nodes represent individuals and arcs represent the relationship between individuals. An arc is labeled by the name of the relationship it represents. Several arcs can have the same label. However, each individual is represented by only a single node. The English sentences “Sumith gives sweet to Shruthi” and “Sumith and Shruthi are human” are represented in the following semantic network Fig. 6.3.

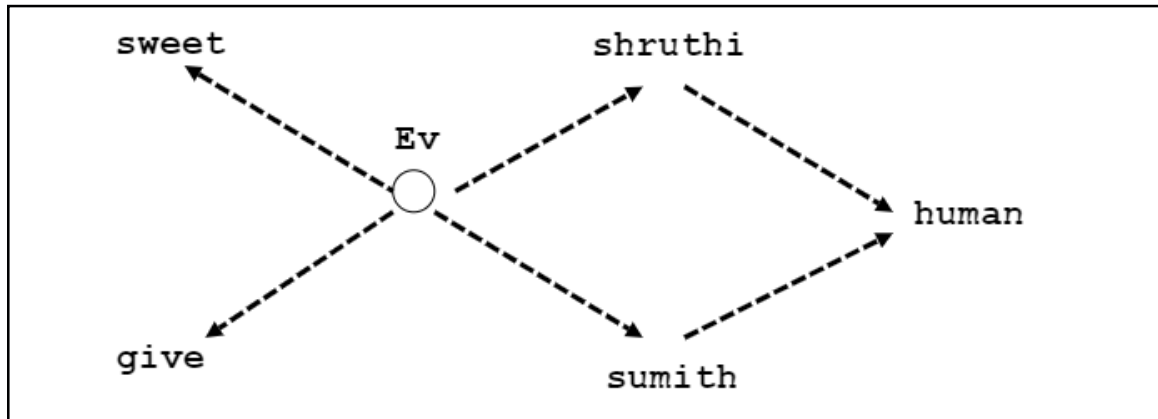


Fig. 6.3: Semantic Network

Here “Ev” names an event or an act of giving, whose actor is Sumith, object is sweet and recipient is Shruthi. Arcs are not to be confused with access pointers. However, given a node in the network, it is assumed that the network provides direct access to all the relationships in which the node participates, independent of the direction of the arc.

6.4.2 The Clausal Form of Logic

The relationships in the network shown in Fig. 63, can be expressed in the clausal form of logic as follows:

Obj (Ev, Sweet) ←
 Act (Ev, give) ←
 Actor (Ev, Sumith) ←
 Rec (Ev, Shruthi) ←
 Isa (Sumith, human) ←
 Isa (Shruthi, human) ←

Here, relations are named by predicate symbols like, “Obj,” “Act,” “Actor,” “Rec,” and “Isa” and individuals are named by constants like “Ev,” “Sweet,” “give,” “sumith,” “shruthi,” and “human”. Predicate symbols corresponding to labels on the arcs of semantic networks always have two arguments. But in general, a predicate symbol can have n arguments, where $n \geq 1$. For example, the sentence “Sumith gives sweet to Shruthi” can be well represented in logic by using a three-argument predicate symbol as follows.

Give (Sumith, sweet, Shruthi) $\leftarrow$

In this example, Sumith, sweet and Shruthi are arguments and Give is the predicate relation. In addition to simple assertions, logic can also express general propositions. For example, the sentence “Sumith gives sweet to everyone he likes” is expressed by the clause

Give (Sumith, sweet, x) $\leftarrow$ Likes (Sumith, x)

Here the symbol “x” is a variable representing any individual. The arrow represents the logical connective “if.” Left side of arrow represents conclusions and that of right side represents condition. In general, a clause can have several conditions, all of which must hold for the conclusion to hold. For example, the clause,

Likes (Sumith, x) $\leftarrow$ give (Sumith, x, y), likes (Sumith, y)

expresses that if Sumith gives away something he likes, then he must like the person he gives it to. (Variables in different clauses are unrelated even if they look the same.) A clause can have several alternative conclusions, at least one of which must hold if all the conditions hold. The sentence *every animal is either male or female* is expressed by the clause

male (x), female (x) $\leftarrow$ animal (x)

Thus, the conclusions of a single clause are a disjunction of alternatives whereas the conditions are conjunction. A clause without conditions is an unconditional assertion. However, a clause without conclusions is a denial.

The clause Give (x, y, Sumith), Likes (Sumith, y) denies that anyone gives Sumith anything which Sumith likes. The existence of individuals is expressed by using constant symbols or function symbols. “Someone likes Sumith” requires a constant to name the anonymous individual who likes Sumith. for example: Likes (A, Sumith)

However, “everyone is liked by someone” is ambiguous. It requires a constant symbol if one individual like everyone.

Likes (B, x) $\leftarrow$ Human (x) and

a function symbol if different individuals like different people

Likes (f(x), x) $\leftarrow$ Human (x).

Here, for any individual x, the expression f(x) names the individual who likes x.

6.4.3 Formal Definition of the Clausal Form of logic

A sentence is a collection of clauses. A clause is an expression of the form $A_1, \dots, A_n \leftarrow B_1, \dots, B_m$ where $A_1, \dots, A_n$ are called the conclusions of the clause and $B_1, \dots, B_m$ are called the conditions. Both conclusions and conditions are expressions of the form $P(t_1, \dots, t_k)$ called atoms, where P is a k-argument predicate symbol and $t_1, \dots, t_k$ are terms. Terms are either constants, variables, or functional terms which are expressions of the form $f(t_1, \dots, t_l)$ where f is an l-argument function symbol and $t_1, \dots, t_l$ are terms.

6.4.4 Inference Rules

Deductive inference rules have been developed for the clausal form of logic. Bottom-up inference derives new assertions from old ones. For example, by matching the assertions

Give (Sumith, Sweet, Shruthi) $\leftarrow$

Likes (Sumith, Sweet) $\leftarrow$

with the conditions of the general clause

Likes (Sumith, x) $\leftarrow$ Give (Sumith, y, x), Likes (Sumith, y)

we obtain the new assertion

Likes (Sumith, Shruthi) $\leftarrow$

Top-down inference derives new denials from old ones. By matching the denial

$\leftarrow$ Likes (Sumith, Shruthi)

with the conclusion of the same general clause, we obtain the new denial.

$\leftarrow$ Give (Sumith, y, Shruthi), Likes (Sumith, y)

Both top-down and bottom-up reasoning are special cases of resolution. Resolution involves matching a condition of one clause A with a conclusion of another clause B. The derived clause is called the resolvent which consists of the unmatched conditions and conclusions of A and B, instantiated by the matching substitution. The clauses A and B are called the parents of the resolvent. Matching two atoms amounts to finding a substitution of terms for variables which if applied to the atoms would make them identical.

A clause of the form $A \leftarrow B_1 \dots B_n$ is interpreted as a procedure with head A and body $B_1, \dots, B_n$. Each atom in the body of the procedure is interpreted as a procedure call. A denial is interpreted as a set of procedure calls and is called a goal clause. Top-down inference, which matches selected procedure call in a goal clause with the head of a procedure, is interpreted as procedure invocation. The new goal consists of the old procedure calls, which are the instantiated copies of the unmatched procedure calls in the old goal clause, and the new procedure calls, which are the instantiated copies of the procedure calls in the body of the invoked procedure. It is useful to divide the matching substitution associated with procedure invocation into two parts. One-part affects variables in the procedure head and transmits input into the body of the procedure. This is the input component of the substitution. The other part affects variables in the procedure call and transmits output to the remaining procedure calls of the goal clause. This is the output component.

The restriction of semantic networks to the representation of two-argument relationships is not a significant limitation. On the contrary, it has several advantages. Every n-argument (n-ary) relationship can be re expressed as a conjunction of two-argument (binary) relationships. If $n > 2$, $n + 1$ binary relationships are needed. If $n = 1$, then only one is necessary. For example, the 3- ary relationship.

Give (Sumith, Sweet, Shruthi) $\leftarrow$

can be re expressed as a set of four assertions using binary predicate symbols. In general, it is necessary to introduce a name for the original n-ary relationship, “Ev” in this example. The n-ary predicate symbol becomes a constant symbol. For each argument of the n-ary relationship, as well as for its predicate symbol, we express how it is related to the original n-ary relationship by means of a binary relationship.

If $n = 1$, the predicate symbol also becomes a constant symbol, and it is only necessary to introduce a single binary relationship which expresses how the original predicate symbol is related to its argument. For example,

Human (x) becomes Isa (x, human)

The transformation from n-ary to binary replaces predicate symbols with constant symbols. In the new formulation the original predicate symbols can also be replaced by variables; this gives some of the expressive power of higher order logic.

The use of binary predicate symbols also has the advantage that unknown arguments of the original n-ary relationship can be ignored. To express that “Sumith was given the Sweet,” we only need the assertions

Act (Ev₁, give) ←

Obj (Ev₁, Sweet) ←

Rec (Ev₁, Sumith) ←

whereas in the n-ary representation, it is necessary to name the unknown donor

Give (C, Sweet, Sumith) ←

In database applications the binary formulation corresponds to the use of variable-length records. Space is saved, and the unnecessary introduction of constant symbols is avoided.

The binary representation is also at an advantage when additional information needs to be associated with the n-ary relationship. To express that “Sumith was given the sweet in the school,” it is only necessary to add the assertion

Location (Ev₁, School) ←

In the n-ary representation the original 3-ary relationship would have to be replaced by a new 4-ary one

Give* (C, Sweet, Sumith, School)

The binary representation suggests a way of dealing with simple aspects of time. For example, the addition of the assertion

After (Ev, Ev₁) ←

can be used to express that “Sumith gave the sweet to Shruthi after it was given to him in the school.” The translation from the n-ary to binary representation does not always result in an improvement. The three-place Plus-relation, for example,

Plus (x, y, z) x plus y is z

This can be re expressed by means of binary relations like,

Isa (w, Plus) w is a Plus-fact

Add₁ (w, x) x is the first number added in w.

Add₂ (w, y) y is the second number added in w.

Sum (w, z) z is the sum of the numbers added in w. The binary formulation is more obscure and no more useful than the original three-place relation.

6.5 EXTENDED SEMANTIC NETWORKS

Simple semantic networks can only express collections of variable-free assertions. We define an extended form of semantic network which can be interpreted as a variant syntax for the clausal form of logic.

It is an immediate consequence of this interpretation that the semantics of the extended semantic network is identical to that of the clausal form of logic. In the extended semantic network, terms are represented by nodes. Constant, variable and functional terms are represented by constant, variable, and functional nodes respectively. As in the simple semantic network, every term is represented by a single node. The same conventions for distinguishing variables from constants are employed as in the clausal form of logic. Binary predicate symbols are represented by labels on arcs. An atom is a labeled arc together with its two end nodes. The direction of the arc (link) indicates the order of the arguments of the predicate symbol which labels the arc. Conclusions and conditions are represented by different kinds of arcs. conditions are drawn with thick line and conclusions are drawn with dotted lines as shown in Fig. 6.4.

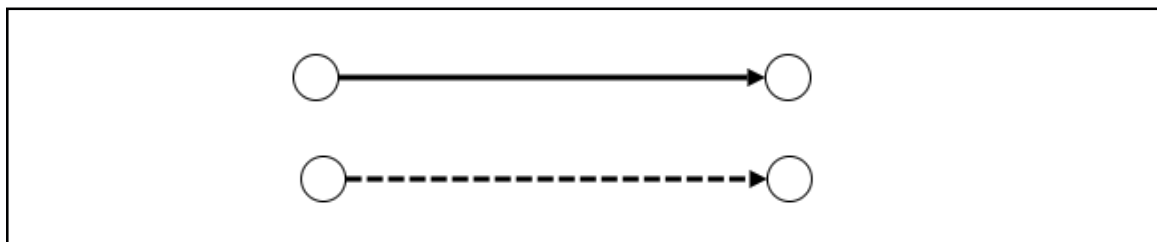


Fig. 6.4: Representation of Conclusion and Condition

A clause is represented by the network representation of its conditions and conclusions. As in the clausal form of logic, different clauses have different variables. Consequently, variables in different clauses are represented by different nodes. For example, *every animal is a male or female* is represented by the following network shown in Fig. 6.5.

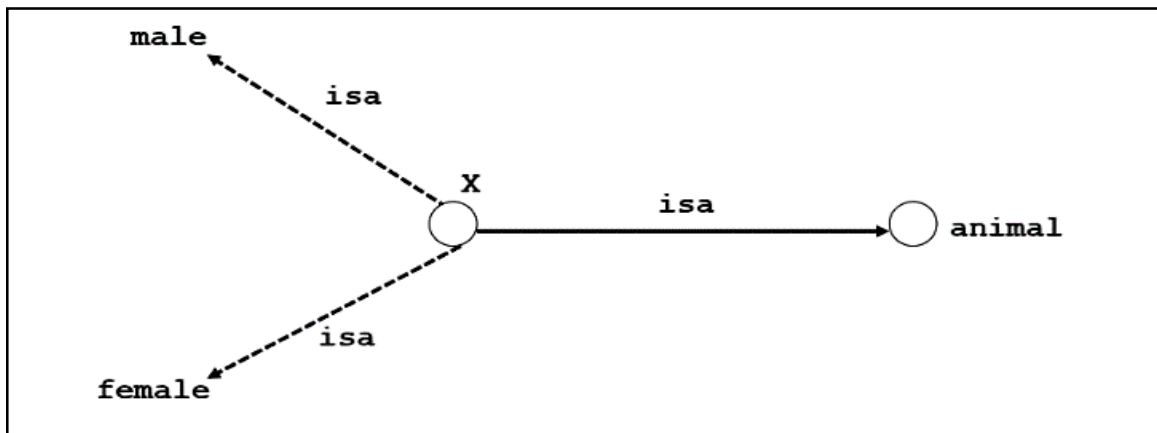


Fig. 6.5: Clause Represented by the Network

The actor of any act of taking is also the recipient of the action and it is expressed by the network as shown in Fig. 6.6.

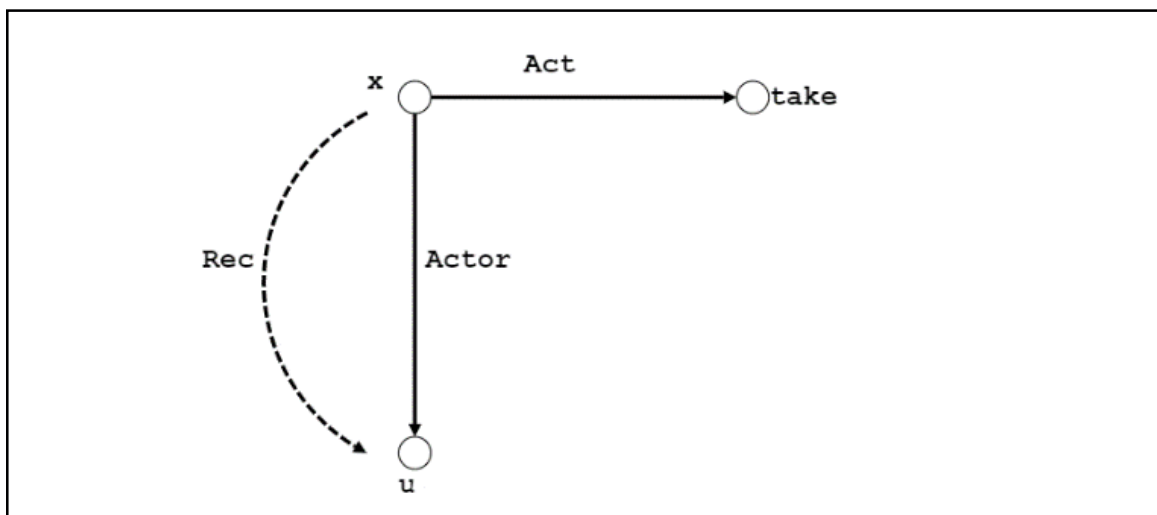


Fig. 6.6: Clause Represented by the Network

$\text{Rec}(x, u) \leftarrow \text{Act}(x, \text{take}), \text{Actor}(x, u)$

From the network shown in Fig. 6.7.

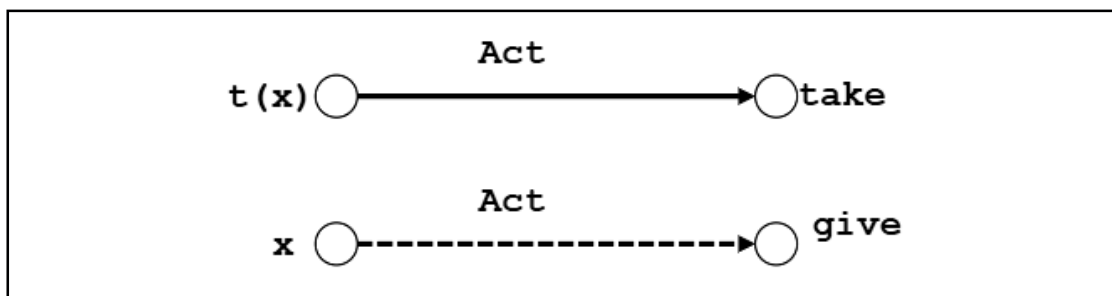


Fig. 6.7: Clause Represented by the Network

$\text{Act}(t(x), \text{take}) \leftarrow \text{Act}(x, \text{give})$ expresses that for every act of giving, there is an act of taking. In the extended semantic network, functional terms are represented by single nodes. In given a set of clauses, if all occurrences of the same term are represented by a single node, then all the clauses should be incorporated in a single network. This gives rise to the problem of distinguishing which arcs belong to which clauses. The delimitation of clauses can be affected in several ways:

- (i) Clause numbers can be associated with arcs.
- (ii) The network can be “partitioned” into subnetworks, each of which represents a single clause.

Partitions overlap if the clauses they represent contain the same fully instantiated atom. Here, we delimit clauses pictorially, as in the following Fig 6.8, by partitions drawn around clauses which contain more than one atom.

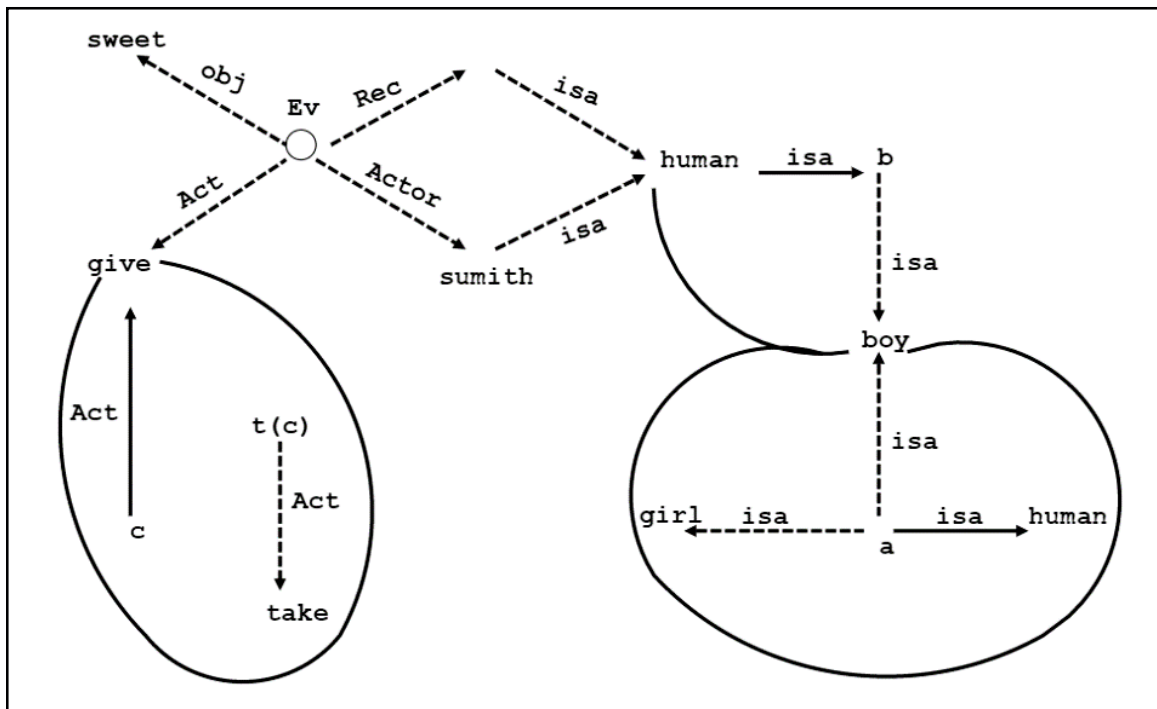


Fig. 6.8: Delimitation of clauses

$\text{Act}(t(c), \text{take}) \leftarrow \text{Act}(z, \text{give})$

$\text{Isa}(a, \text{girl}), \text{Isa}(a, \text{boy}) \leftarrow \text{Isa}(a, \text{human})$

$\text{Isa}(b, \text{boy}) \leftarrow \text{Isa}(b, \text{human})$

$\text{Obj}(eV, \text{sweet}) \leftarrow$

$\text{Rec}(eV, \text{Shruthi}) \leftarrow$

$\text{Actor}(eV, \text{Sumith}) \leftarrow$

Act (eV, give) ←
 Isa (shruthi, human) ←
 Isa (Sumith, human) ←

The extended semantic network, being equivalent to the clausal form of logic, is a uniform, general-purpose formalism for the representation of information. The Isa and Part-of hierarchies, which are characteristic of many semantic networks, are special cases of the extended semantic network. *Isa* hierarchies, which in simple semantic networks are represented by sets of *Isa* or *Subset* assertions as shown in Fig. 6.9.

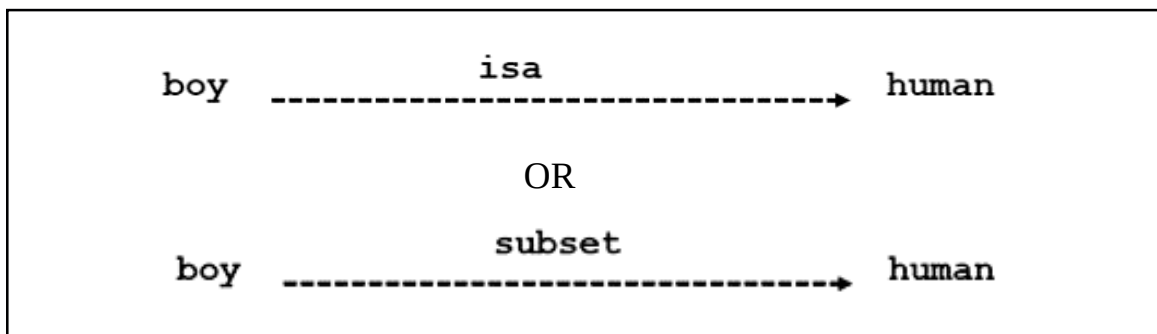


Fig. 6.9: Representation of Semantic Network

It can be represented in the extended semantic network in two different ways. One representation expresses statements of the form “A is a B” where both A and B name sets of objects by means of clauses of the form as shown in Fig. 6.10.

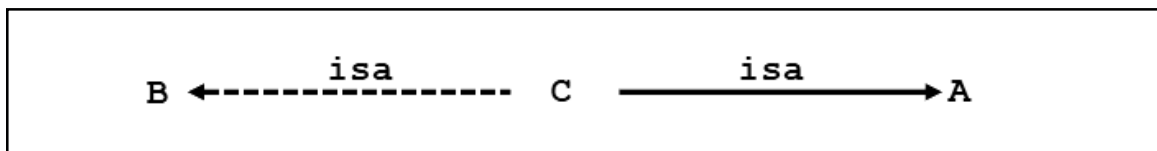


Fig. 6.10: Representation of Semantic Network

The other representation retains the simple variable-free assertions. But adds a general law which expresses the semantics of the Isa and Subset predicate symbols as shown in Fig. 6.11.

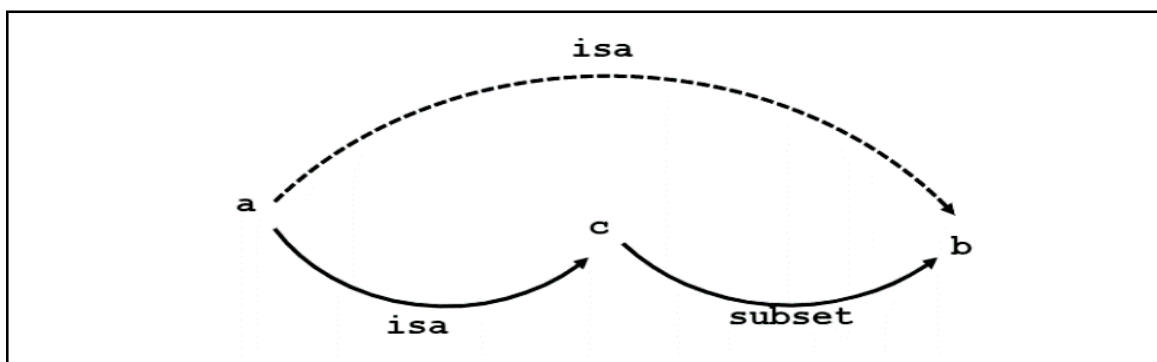


Fig. 6.11: Representation of Semantic Network

The clauses in the first representation are logically implied by those in the second representation. They can be derived by means of the bottom-up inference rule defined in the next section. Part-of hierarchies can also be represented in two different ways, with the complication that Part-of assertions conceal a hidden existential quantifier. For example, the assertion is shown in Fig. 6.12. This means that for every animal, there is a tail which is part of the animal.

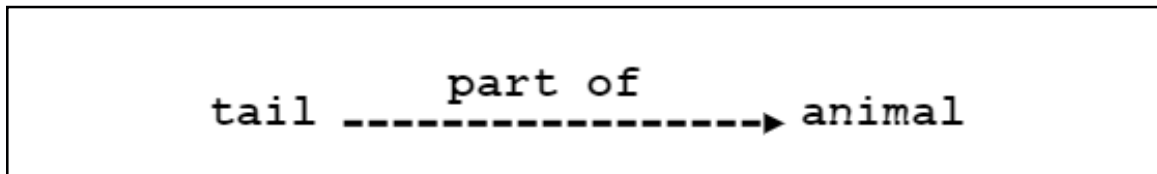


Fig. 6.12: Representation of Semantic Network

In clauses it can be represented as shown in Fig. 6.13.

$\text{Isa}(w(x), \text{tail}) \leftarrow \text{Isa}(x, \text{animal})$

$\text{Part-of}(w(x), x) \leftarrow \text{Isa}(x, \text{animal})$

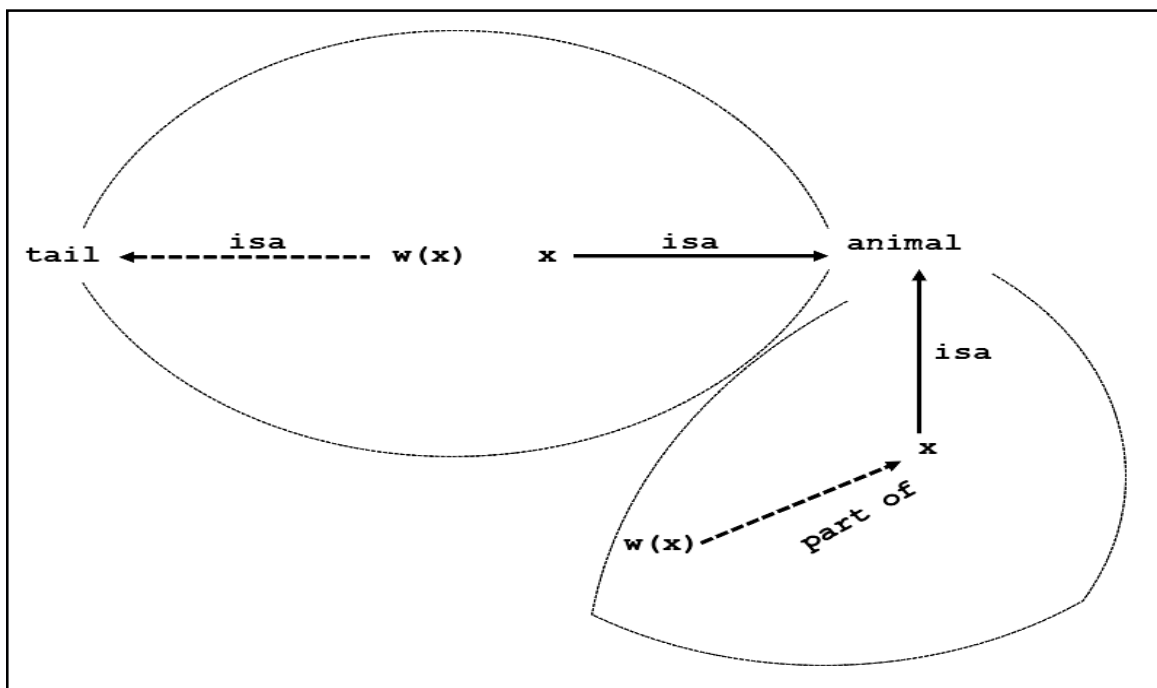


Fig. 6.13: Clause Represented by the Network

6.5.1 Deduction in Extended Semantic Networks

The correspondence between logic and the extended semantic network provides the network with not only semantics but also inference rules. There are two types of mechanisms namely:-

- (i) Bottom-up inference
- (ii) Top-down inference

Bottom-up inference is also called forward reasoning inference mechanism. In this approach we start from known facts and triggers all rules whose premises are satisfied. Then add their conclusion to the known facts. This process repeats until the problem is solved. Let us consider the following set of clauses.

$\text{isa}(A, \text{animal}) \leftarrow \text{isa}(A, \text{mammal})$

$\text{isa}(\text{cat}, \text{mammal})$

With the help of modus ponens rule of logic, let us derive that $\text{isa}(\text{cat}, \text{animal})$ is true using extended semantic network representation. In this case new assertion is inferred by eliminating $\text{isa}(A, \text{animal})$ which is an assertion link and $\text{isa}(A, \text{mammal})$ which is a denial link. Clauses and inferences are represented in Fig. 6.14.

Top-Down inference is also known as a backward deduction or backward reasoning method when using an inference engine. A backward reasoning algorithm is a form of reasoning which starts with the goal and works backward, chaining through rules to find known facts that support the goal. This is performed by different stages of resolution till contradiction is developed. Consider a set of clauses as shown in the Fig. 6.15. Here we have to prove $\text{isa}(\text{cat}, \text{animal})$ using the same network.

Here, after adding a denial link in the extended semantic network, we observe reduction in the network with appropriate substitution as shown below. Thus, it can be concluded that inheritance and inferences are the important characteristics of the extended semantic network.

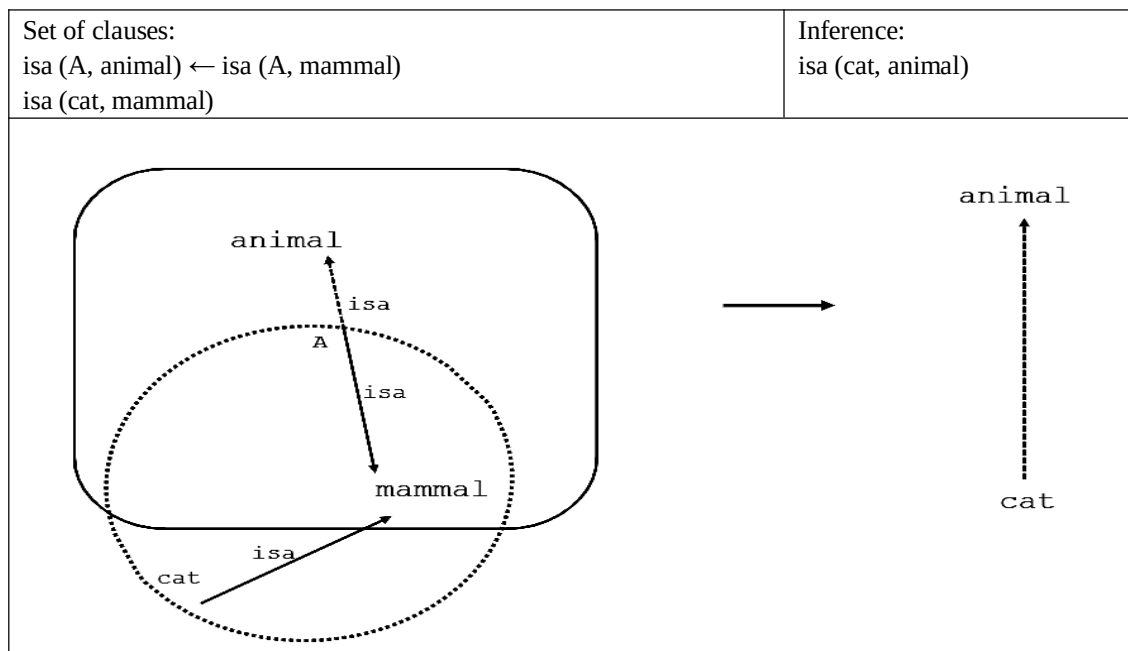


Fig. 6.14: Bottom-up Inference

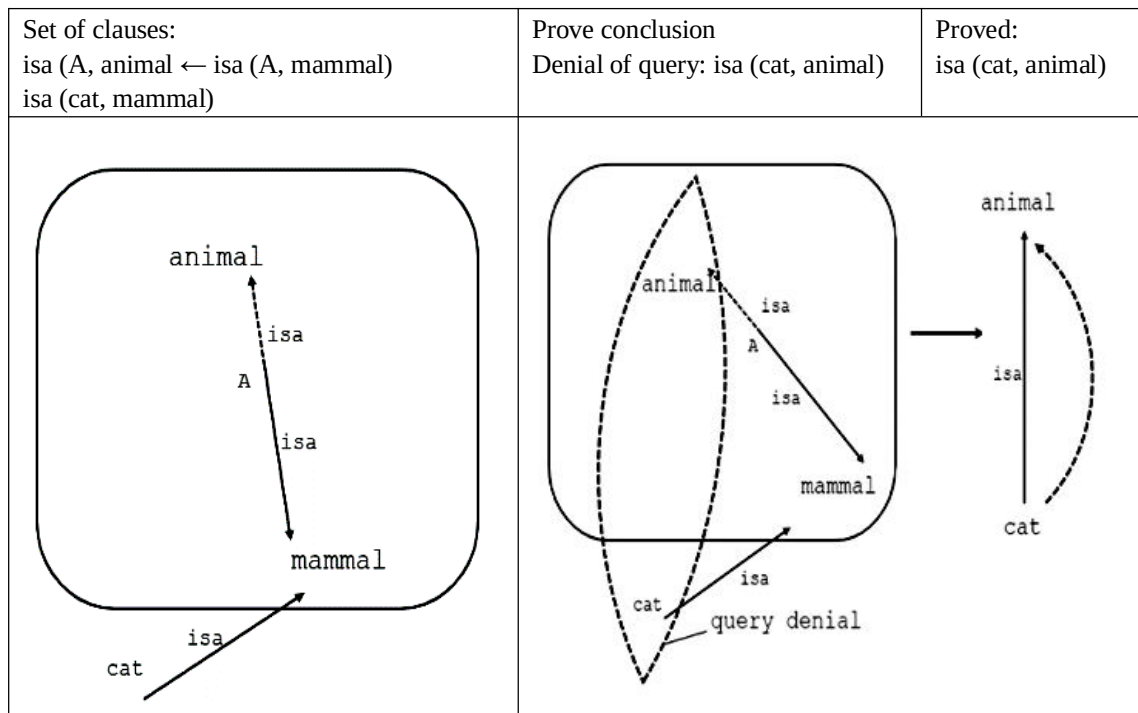


Fig. 6.15: Top-Down Inference

6.6 ILLUSTRATING INFERENCING METHODS

Let us consider an example to illustrate inferencing method. The sentence “someone who gives something he likes to a person likes that person too. Sumith likes sweet. Sumith gives sweet to Shruthi”. Clausal representation of above statement is as follows:

Likes (a, c) $\leftarrow$ action (u, give), object (u, b), actor (u, a), recipient (a, b)

Action (e, give) $\leftarrow$

Object (e, sweet) $\leftarrow$

Actor (e, sumith) $\leftarrow$

Recipient (e, shruthi), $\leftarrow$

Likes (sumith, sweet) $\leftarrow$

Network in following Fig. 6.16 shows the Extended semantic network for the above statement where u represents variable and e represents actual event.

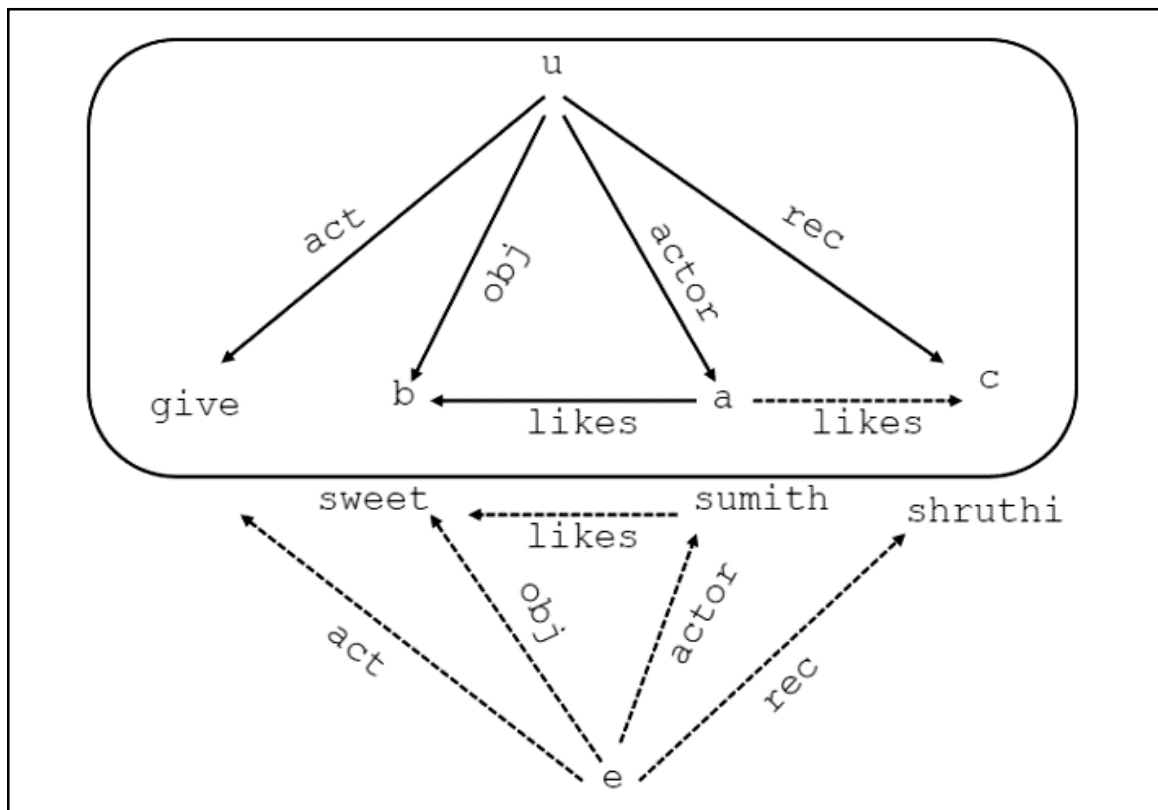


Fig. 6.16: Extended Semantic Network

Here, we have to prove likes (sumith, shruthi) based on the statement considered using Bottom up (forward reasoning) inference and Top Down (backward reasoning) inference.

6.6.1 Bottom-Up Inference

We can reduce above network by resolving clauses so as to obtain desired assertion as shown in the Fig. 6.17. Assertions and corresponding denial with likes links are eliminated by assigning b to sweet and a to sumith. Here, for example, action (e, give) is an assertion and action (u, give), a denial which is contradiction and hence eliminated and other links also follow similar elimination and left with final assertion likes (sumith, shruthi).

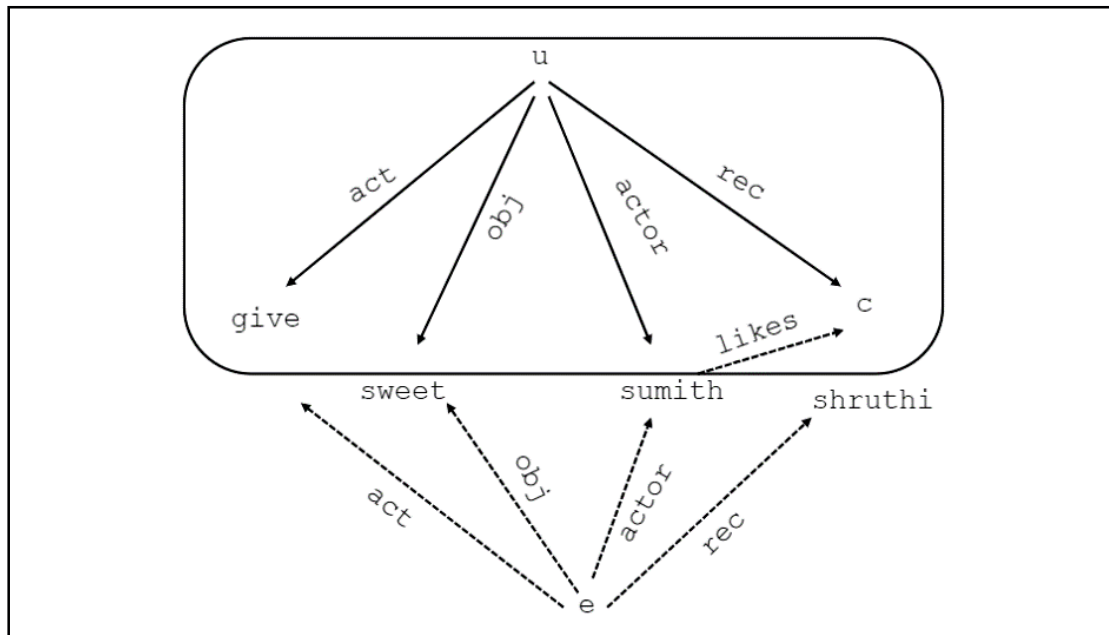


Fig. 6.17: Bottom-Up Inference

6.6.2 Top-Down Inference

Using top-down inference, we can prove like (sumth, shruthi). In order to demonstrate that a set of clauses in a network implies a conclusion, we add the denial of the conclusion, *i.e.*, like (sumth, shruthi) to the network and show that the resulting set of clauses is inconsistent. This is done by performing successive steps (from Fig 6.18 to 6.20) of resolution until an explicit contradiction is generated.

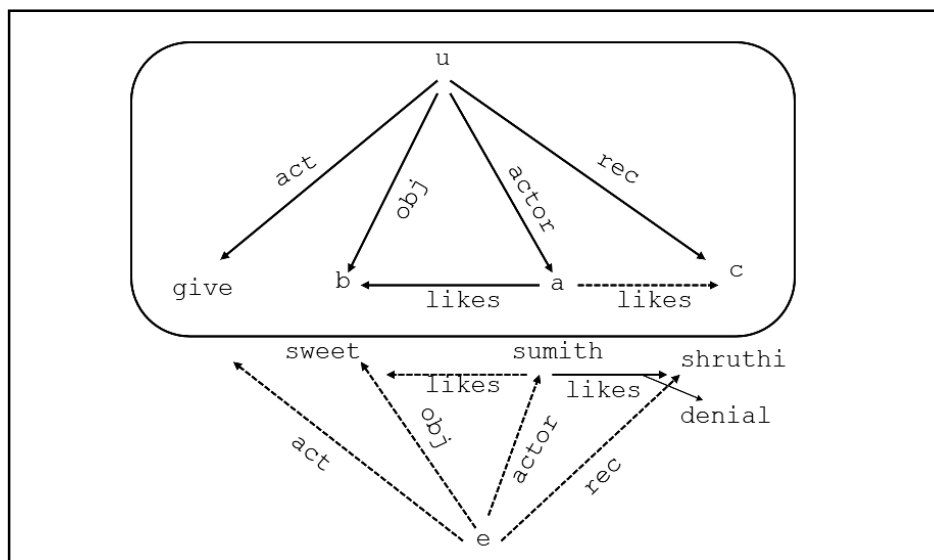


Fig. 6.18: Top-Down Inference Step-1

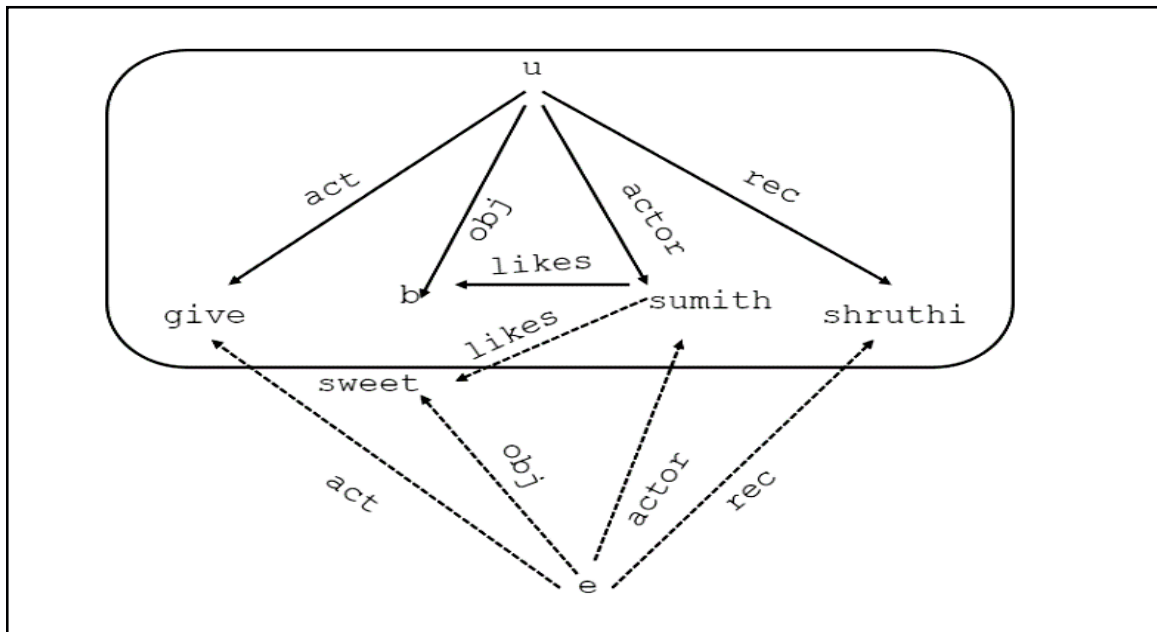


Fig. 6.19: Top-Down Inference Step-2

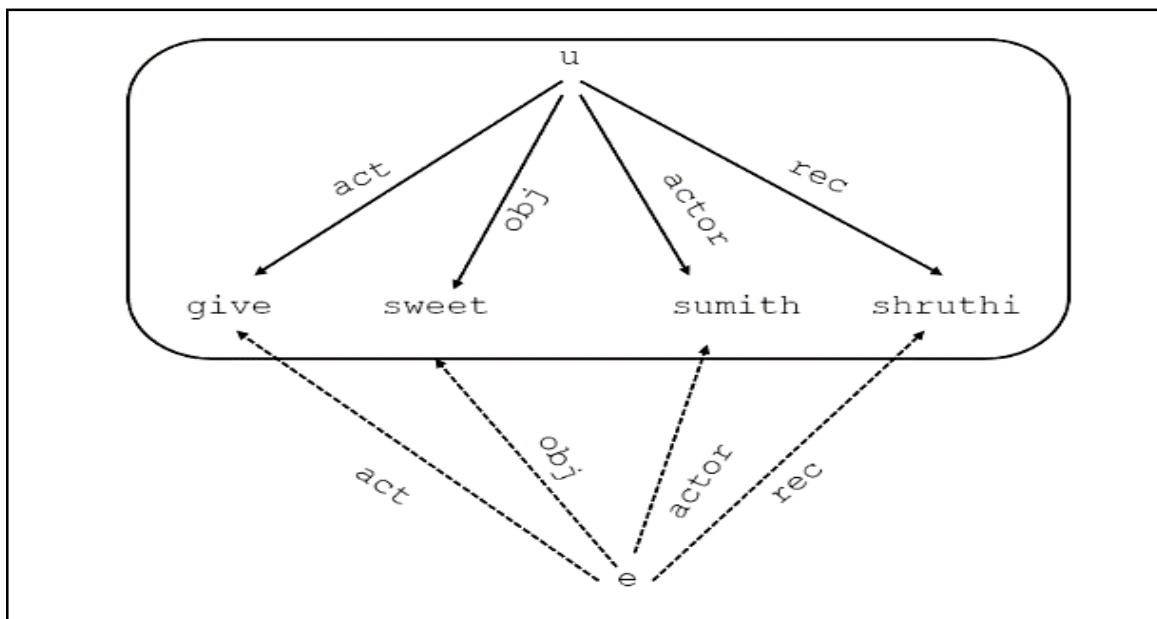


Fig. 6.20: Top-Down Inference Step-3

Following Fig. 6.21 shows the conclusion of both top down and bottom-up inferences.

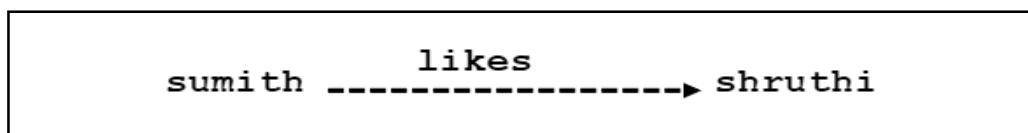


Fig. 6.21: Conclusion

In a computer implementation, clauses can be represented explicitly by adding them to the network or implicitly by using the structure-sharing method in which resolvents are represented by pointers to their parents together with a record of the matching substitution. Anything explicit in the first representation can be computed in the second. Without prejudicing the manner in which clauses are represented in a computer implementation, networks are drawn with resolvents added to them explicitly. Independently, whether in an implementation resolvents are represented explicitly or implicitly because of the purity principle, a clause can be deleted if one of its atoms matches no other atom in the network. In particular, when a resolvent is created, a parent may be deleted if no other match exists for the atom being matched in that parent. Such a situation might arise if either there was only one match initially, or else all other matches have already taken place. In practice, when deletion is possible, it is generally more convenient to construct the resolvent from the constituents of the deleted parent. Applied to the problem of finding a fallible Greek, top-down inference together with the deletion rule transforms the initial network until it eventually contains only the empty clause.

6.7 KNOWLEDGE REPRESENTATION USING FRAMES

Frame based representation is a development of semantic nets and allow us to express the idea of inheritance. A Frame System consists of a set of frames (or nodes), which are connected together by relations. Each frame describes either an instance or a class. Each frame has one or more slots, which are assigned slot values. This is the way in which the frame system is built up. Rather than simply having links between frames, each relationship is expressed by a value being placed in a slot. Consider the following Table 6.3.

Table 6.3: Frame System Slot values

Frame Name	Slot	Slot Value
Karan	is a	Actor
	owns	Rocky
	eats	Meat
Rocky	is a	dog
	chases	Moni
Moni	is a	Cat
	chases	Mice
Mice	eat	Meat
Cheese		
Actor		
Dog		
Cat		

We can also represent this frame system in a diagrammatic form as shown in Fig. 6.22.

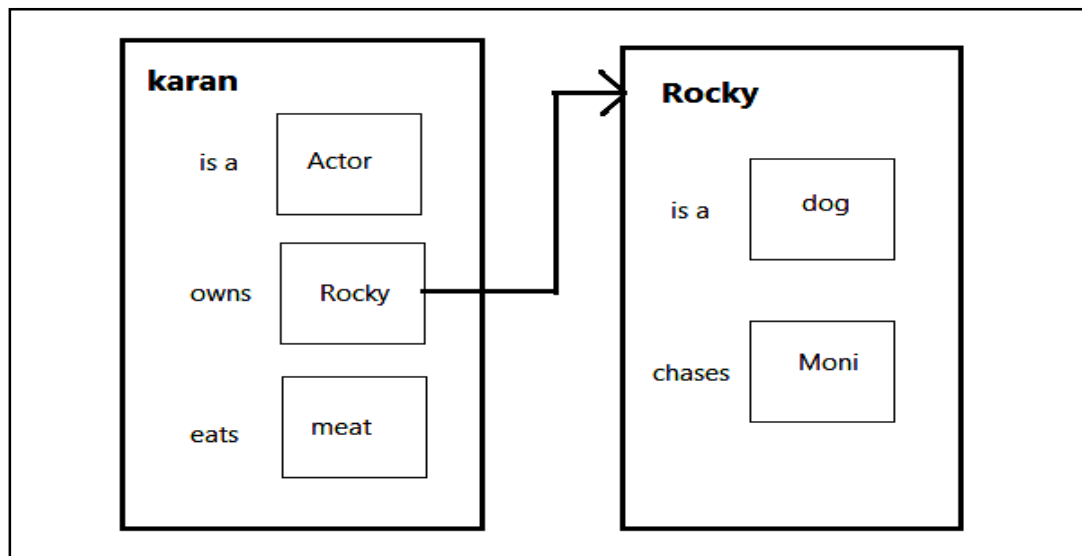


Fig. 6.22: Frame System

When we say, “Rocky is a dog” we really mean, “Rocky is an instance of the class dog” or “Rocky is a member of the class dogs”. The main advantage of using frame-based systems for expert systems is that all information about a particular object is stored in one place.

6.7.1 Inheritance in Frames

Inheritance is defined as a mechanism which is used for passing knowledge from one frame to another frame.

Example 1:

We might extend our frame system with the following additional information: Dogs chase Cats chase mice. Now, we do not need to state explicitly that Rocky chases Moni or that Moni chases mice. We can inherit this information because Moni is an instance of the class cats and Rocky is an instance of class dogs.

Example 2:

Mammals breathe Dogs are mammals Cats are mammals.

Here we have created superclass mammals of which dogs and cats are subclasses. Hence, we do not need to explicitly say that cats and dogs breathe because we can inherit this information. Similarly, we do not need to express explicitly that Rocky and Moni breathe as they are instances of the class dogs and cats and therefore, they inherit from those classes (super classes).

Example 3:

Mammals have four legs.

Now, this is not true as humans do not have four legs. In other way, we can say Rocky has an unfortunate accident and now has only three legs. This information might be expressed as shown in Table 6.4.

Table 6.4: Frame System Slot values

Frame Name	Slot	Slot Value
Mammals	*number of legs	four
Dog	subclass	mammal
Cat	subclass	mammal
Rocky	is a	dog
	number of legs	three
Moni	is a	cat

We have used an asterisk (*) to indicate that the value for the “number of legs” slot for the mammal class is a default value and can be overridden.

Slots as Frames:

It is also possible to express a range of values that a slot can take. For instance, the number of legs slot might be allowed a number between 1 and 4. Or we can express this restriction by allowing slots to be frames. The number of legs slot can be represented as a frame as shown in Table 6.5.

Table 6.5: Frame System Slot Values

Frame Name	Slot	Slot Value
Number of legs	Minimum value	1
	Maximum value	4

6.7.2 Multiple Inheritance in Frame System

It is possible for a frame to inherit properties from more than one frame, i.e., a class can be a subclass of two super classes and an object can be an instance of more than one class. This is known as Multiple Inheritance.

Example:

Table 6.6: Frame System Slot Values

Frame Name	Slot	Slot Value
Human	subclass	Mammal
	Number of legs	two
Actor	makes	films
Karan	is a	Human
	is a	Actor

In some cases, we encounter conflicts, where multiple inheritance leads us to conclude contradictory information about a frame.

Table 6.7: Contradictory to Frame

Frame Name	Slot	Slot Value
Cheese	is	smelly
Things wrapped in foil	is	not smelly
Feta	is a	Cheese
	is a	Things wrapped in foil

Example:

Here, Feta is a type of cheese and that it comes wrapped in foil. Feta should inherit its smelliness from the cheese class, but it also inherits non-smelliness from the Things wrapped in foil class.

Thus, we need a mechanism to decide which features to inherit from which superclass. One simple method is to say that conflicts are resolved by the order in which they appear. So, if a fact is established by instances and then that fact is contradicted by inheritance, the first fact is kept because it appeared first and the contradiction is discarded. However, it would be better to build the frame system such that conflicts of this kind cannot occur.

Advantages of Frames

- With the help of frame knowledge representation, the programming becomes easier as it groups the related data.
- Due to the flexible representation, the frame is used by many applications in AI.
- Slots for new attributes and relations can be easily added.
- To search for a missing value, you can add default data.
- It is widely popular for understanding and visualizing knowledge representation.

Disadvantages of Frames

- Because the system has too many slots, deductions can't be easily formed.
- The frame representation is not built to support the inference mechanism.
- It has a very generalized approach that limits its functioning.

QUESTIONS

- 6.1 What is knowledge representation? Explain the properties of knowledge representation
- 6.2 Briefly Explain the basic knowledge representing schemes
- 6.3 Explain the semantic network way of knowledge representation with example
- 6.4 Why extended form of semantic network is needed? Explain.
- 6.5 Explain the deductive inference rules developed for the clausal form of logic with example
- 6.6 Explain the deduction in extended semantic networks with examples.

- 6.7 Explain knowledge representation using frames.
- 6.8 Draw a semantic network which represents following knowledge.
- (i) Every movable thing is a physical object. Every bicycle is movable thing. Every bicycle has two pedals. Pedal is mechanical component.
 - (ii) Every living creature needs food. Every human is a living creature.
Sumith is human.



7

EXPERT SYSTEM

7.1 INTRODUCTION

The main objective of the AI is to perceive the concept of intelligence and develop intelligent machines. Artificial Intelligence is a software which simulates the behaviour and decision-making ability of a human experts in a particular domain is known as an expert system. AI does this by acquiring relevant knowledge from its knowledge base and interpreting it according to the user problem. It should be noted that the data in the knowledge base is provided by human experts from a particular domain. An expert system is widely used in many areas such as medical diagnosis, accounting, coding, games etc.

An expert system is AI software which uses the knowledge stored in a knowledge base in order to solve problems that usually requires a human expertise. This preserves the human expert's knowledge in its knowledge base. This can advise users and provide explanations to them about how they reached a particular conclusion or advice. Knowledge Engineering is the term deals with the process of building an expert system and its practitioners are called Knowledge Engineers. The primary role of a knowledge engineer is to make sure that the computer possesses all the knowledge required to solve a problem. There are many examples of an expert system. Some of them are given below.

MYCIN

This is one of the earliest expert systems based on backward chaining. This was developed to identify various bacterial infections and can also recommend drugs based on the person's weight.

DENDRAL

It was an AI-based expert system used for chemical analysis. It would hypothesize spectrographic data obtained from the substance into molecular structure.

R1/XCON

Assists in the ordering of computer systems by automatically determining the computer system components depending on the customer's requirements.

PXDES

It is an expert system that determines the type and the degree of lung cancer in a patient using inference engine.

CaDet

It is designed to support a clinical system that could identify cancer in patients in its early stages.

DXplain

It is also a diagnostic clinical system that suggest a variety of diseases based on the clinical findings.

7.2 PHASES OF BUILDING EXPERT SYSTEMS

The process of developing an expert system is called knowledge engineering and is designed by a knowledge engineer. The knowledge engineer is a human expertise and AI and knowledge engineer together knows how to build expert systems. A knowledge engineer knows how to represent the knowledge in an expert system and helps the developers to write the code. Knowledge engineering is also known as the process of acquiring knowledge from a human expert or any other source. A domain expert is a person who has knowledge and expertise in the domain of the expert system being developed.

The knowledge engineer and the domain expert usually work hand in hand with each other for long periods of time throughout the several stages of the development process. An expert system is developed and refined over a period of several years as it is typically a computer-based software. It should be noted that it may not be possible to break down the development cycle of expert system so accurately. However, several general approaches for developing ES have been proposed by Waterman (1986). The following are the five main stages to develop an expert system. The stages are: (1) Identification (2) Conceptualisation (3) Formalisation (Designing) (4) Implementation (5) Testing (Validation, Verification and Maintenance).

7.2.1 Identification

Before we begin to develop an expert system, it is important to state, the problem which the system is intended to solve. It is not enough to use an expert system which would be helpful in a certain situation. We must determine the exact nature of the problem and state the goals precisely how an expert system is expected to contribute to the solution.

In some cases, the knowledge engineer, may not be familiar with particular domain, consults manuals and training guides to gain some familiarity with the subject. Then the domain expert may describe several typical problem states. The knowledge engineer tries to extract fundamental concepts from the similar cases in order to develop a more general idea of the purpose of the expert system. After the domain expert describes several cases, the knowledge engineer develops a problem description. Typically, the domain expert may be of the opinion that description does not entirely represent the problem. The domain expert then suggests necessary changes to the

description and provides the knowledge engineer with additional examples to illustrate further points of the problem. Next, the knowledge engineer revises the description based on the domain expert suggestion for further changes. This process is repeated until the domain expert and knowledge engineer are satisfied that the description adequately portrays the problem which the expert system is expected to solve. This 'iterative' procedure as shown in Fig. 7.1 is typical of the entire expert-system development process. The results are evaluated at each level of the process and compared with the expectations. If the results do not meet the expectations, then adjustments are made to that level of the process and new results are evaluated. The process continues until satisfactory results are achieved.

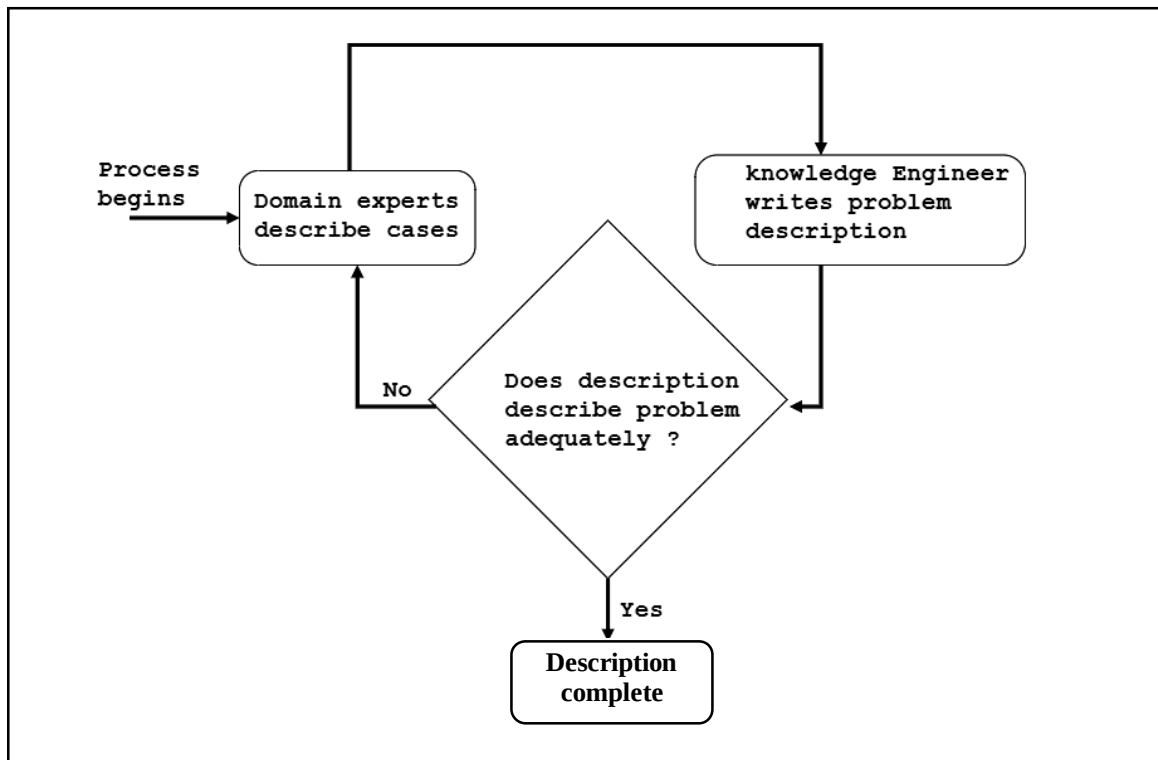


Fig. 7.1: Process of Identifying Problem

Domain experts are not the only the resources which must be identified. It is unusual for all domain knowledge to be embodied in single human expert. Therefore, more tangible sources of information, such as reference books and manuals are usually used.

7.2.2 Conceptualization

Once problem is identified, an expert system has to solve the problem. The next stage involves analysing the problem further to ensure that all its specifics as well as generalities are understood. In the conceptualisation stage the knowledge engineer creates a diagram of the problem to represent graphically the relationships between the objects and processes in the problem domain. It is usually helpful at this stage to divide the problem into a series of sub-problems and create diagram for the relationships among the pieces of each sub-problem.

The conceptualisation stage consists of a circular procedure with iteration and reiteration between the knowledge engineer and the domain expert. When both agree that the key concepts and the relationships among them have been adequately conceptualised, this stage is considered to be complete. One should note that not only each stage in the expert system development process circular, but also the relationships among the stages may be circular as well. Since each stage of the development process adds detail to the previous stage, any stage may exhibit weakness in a previous stage. For example, a problem with the description in the identification stage may be discovered during the stage of conceptualisation. A key element of the description may have been omitted or a goal was stated incorrectly. If this situation occurs then a brief return to the identification stage is required to increase the accuracy of the description as shown in Fig. 7.2.

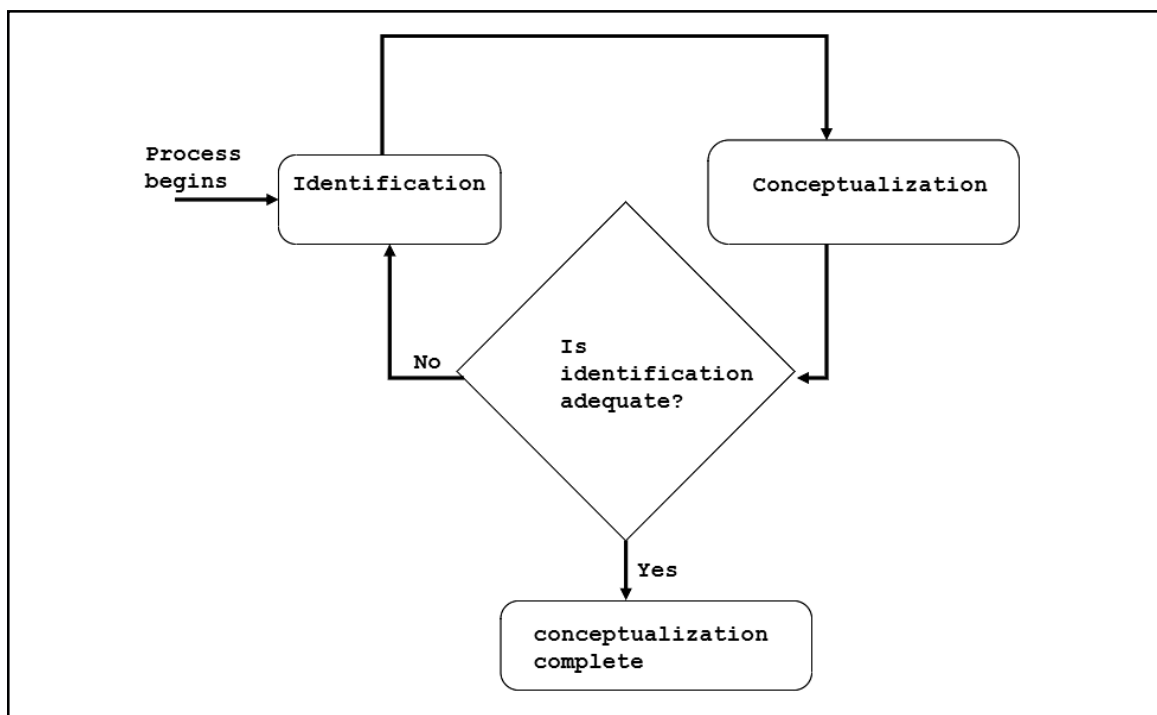


Fig. 7.2: Relationship between Identification and Conceptualization

7.2.3 Formalization (Designing)

In the preceding stages, efforts are not made to relate the domain problem to the artificial intelligence technology. During the identification and conceptualisation stages, the focus is entirely on understanding the problem. Now, during the formalization stage, the problem is connected to its proposed solution is supplied by analysing the relationships depicted in the conceptualization stage. The knowledge engineer starts finding the techniques which are appropriate for developing a particular expert system. During formalization, it is important for the knowledge engineer to be familiar with the following points.

- The various techniques of knowledge representation and intelligent search techniques used in expert systems.

- The expert system tools can greatly expedite the development process.
- Other expert systems which may solve similar problems and thus may be adaptable to problem at hand.

Often it is preferred to select a single development technique which can be used throughout all levels of the expert system. However, the knowledge engineer may determine that no necessity of using different techniques for different sub-problems. Once technique has been determined, the knowledge engineer starts to develop a formal specification which is used to develop a prototype expert system.

For example, in the case of a rule-based expert system, the knowledge engineer develops a set of rules to represent the knowledge communicated by the domain expert. This part of development process is critical since it requires great skills on the part-of the knowledge engineer. Many domain experts can explain what they do but not why. Therefore, one of the primary responsibilities' knowledge engineers is to analyse previous examples and derive set of rules from those examples which describe the domain expert's knowledge. The formalisation process is often the most interactive as well as the most time-consuming stage of expert system development. The knowledge engineer must develop a set of rules and get conformed from domain expert about the rules adequately represent the expert's knowledge. The domain expert reviews the rules proposed by the knowledge engineer and may suggests changes, which are then incorporated into the knowledge base.

This process is also iterative since rules are refined and reviewed repeatedly and continually until the results are satisfactory. (Fig. 7.2)

7.2.4 Implementation

During the implementation stage the formalised concepts are programmed into the computer which has been selected for system development. This is done by using the predetermined techniques and tools to implement a prototype of the expert system.

If the prototype works at all, one can say that the techniques chosen to implement the expert system were the appropriate ones. On the other hand, the knowledge engineer may find that the chosen techniques simply cannot be implemented. It may not be possible, for example, to combine the knowledge representation techniques selected for different sub-problems. At that point, the concepts need to be re-formalised or it may require to create a new development tool to implement efficiently. The implementation stage is illustrated in Fig. 7.3.

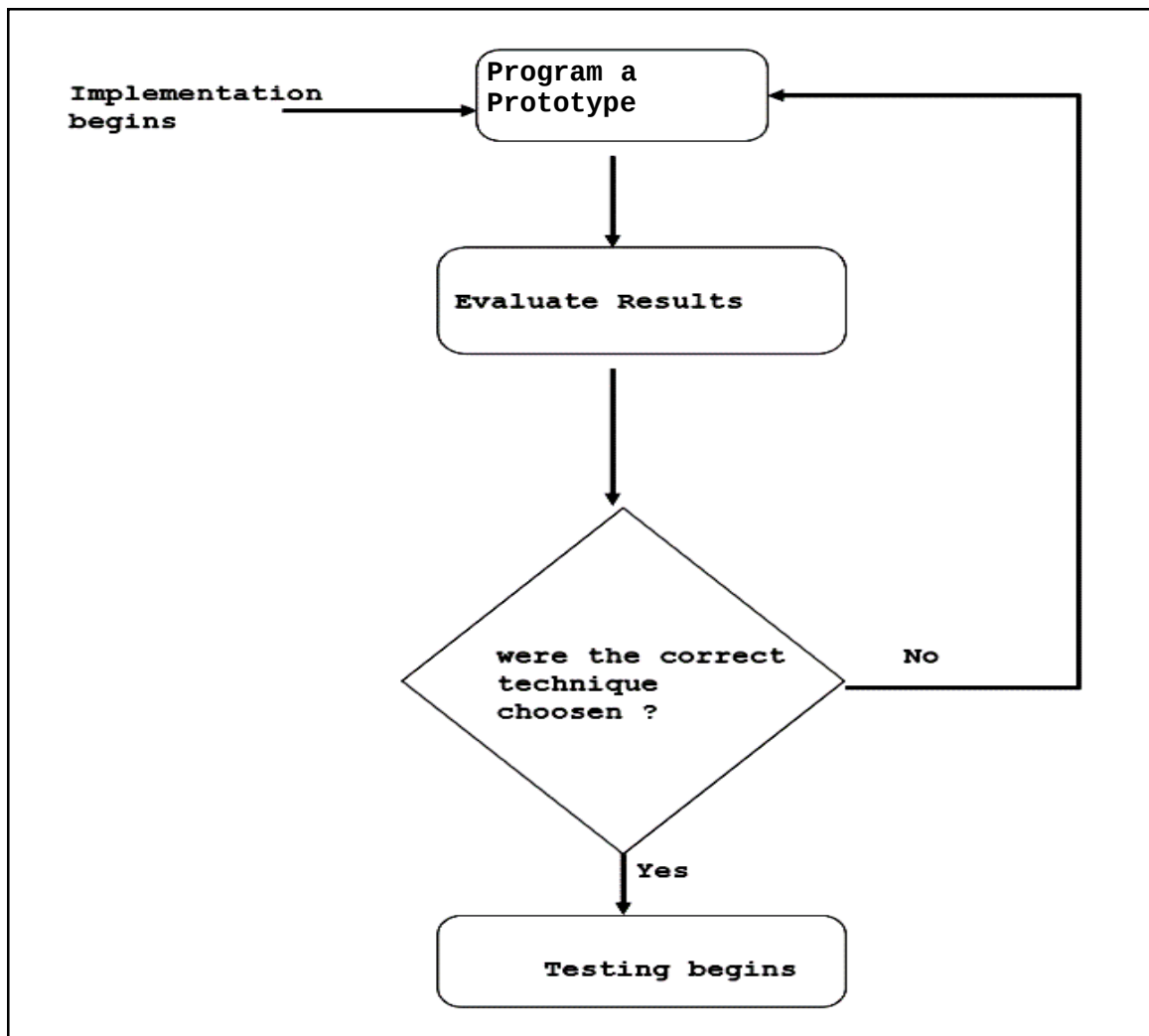


Fig. 7.3: Expert System Development

Once the prototype system has been refined sufficiently, the expert system is considered to be ready thorough testing in order to ensure that expertise's correctly.

7.2.5 Testing (Validation, Verification and Maintenance)

Often, there are less chances of prototype expert system to execute flawlessly in the first attempt itself. Testing provides an opportunity to identify the weaknesses and implementation in the system to make the appropriate corrections.

Results from the tests are considered as feedback to return to a previous stage and adjust the performance of the system. Ultimately, an expert system is considered to be successful only when it operates at the level of a human expert.

7.3 EXPERT SYSTEM ARCHITECTURE

Architecture of expert system may be explained with the help of diagram as given in the Fig. 7.4. which consists of the following major components.

Knowledge Base: It is a warehouse of special heuristics or rules, that are used directly by knowledge, facts (productions). It has knowledge that is required for understanding, formulating and problem solving.

Working Memory: It helps to describe the current running problem and record intermediate output. It records intermediate hypothesis and decisions.

Inference Engine: It is a heart of expert system which helps to manage entire structure of expert system and it also delivers different methodology for reasoning.

Explanation System: It is used to trace the responsibility and justify the behaviour of expert system by posing questions and answers such as why, how, what, where, when, who.

User Interface: It helps the user to insert their queries using own natural language processing otherwise menus and graphics.

Knowledge Engineer: One who can encode the expert's knowledge in a form that can be used by the expert system.

System Engineer: The individual who can build the user interface, designs the declarative format of the knowledge base and implements the inference engine.

User: The individual who will be consulting with the system to get the advice provide by the expert.

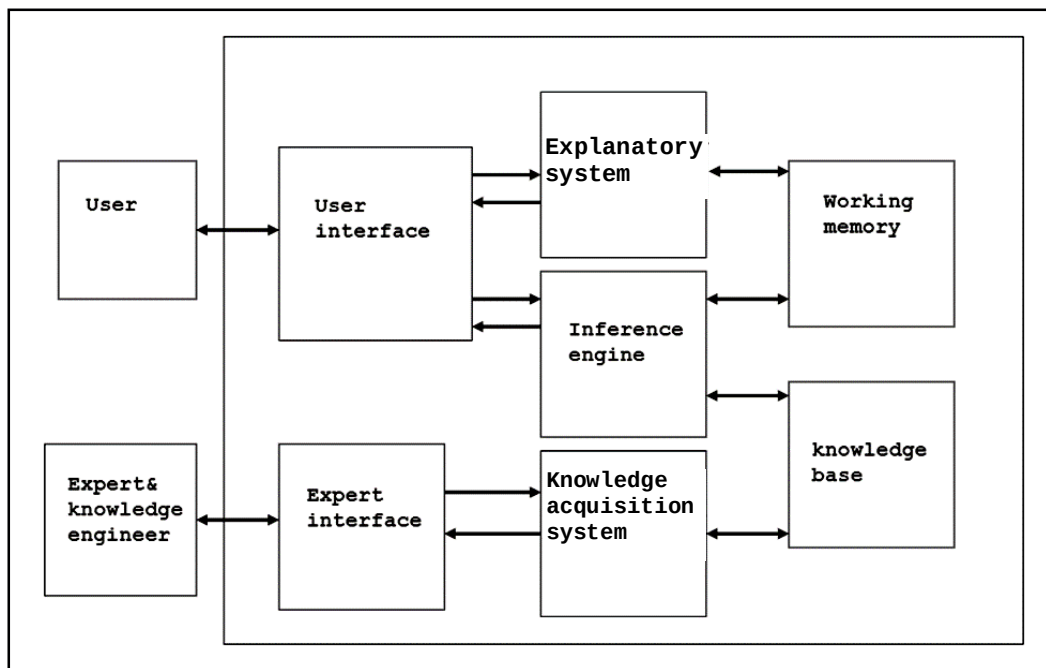


Fig. 7.4 Expert System Architecture

7.3.1 The knowledge Base

The knowledge base essentially consists of knowledge about the specific problem domain. The knowledge base is considered to be rich with diverse forms of knowledge. This knowledge may be in the form of static and dynamic databases. Static knowledge includes rules and facts. However, the rules may be complex and the facts may include sequences, structured entities, attributes of such entities and the relationships between them. On the other hand, dynamic knowledge includes the various queries which are posed to the user while consulting expert system. In the beginning of the process the dynamic database is always empty and as consultation progresses this database starts collecting facts which will be used to make discussions along with static data base.

7.3.2 Inference Engines

The term inference means process of searching by making use of knowledge base and concluding new knowledge. Hence, Inference engine is developed with proper control strategy supported with inference mechanism. Inference engine follows the inference rules defined in the knowledge base to find the solution for a particular problem. Inference Engine vary according to the type and complexity of knowledge with which they interact. There are two types of inference engines namely, *forward-chaining* and *backward-chaining*. These may also be known as data-driven and goal-driven respectively. A knowledge-based system working in data-driven mode makes use of the available information (facts) and generates as many derived facts as it can. Therefore, the output is unpredictable. This may have either the advantage of leading to innovative solutions to a problem or the disadvantage of wasting time generating irrelevant information. The data-driven approach might typically be used for problems of interpretation where we wish to know something the system can tell us about the data. A goal-driven strategy is suitable when more tightly focused solution is required. For instance, a planning system is required to generate a plan for manufacturing a consumer product. Any other plans may be irrelevant. A backward-chaining system tries to prove conclusion of the rules one by one till it succeeds or all the rules get exhausted.

7.3.3 Knowledge Acquisition

The process of extracting, structuring and organizing knowledge from one source, usually human expert is called knowledge acquisition. The representation of knowledge in a knowledge base can only be addressed once the knowledge is known. There are three distinct approaches to acquire the relevant knowledge for a particular domain:

- The knowledge is acquired from a domain expert.
- The builder of the knowledge-based system is a domain expert.
- The system learns automatically from examples.

The first approach is commonly used, but there are certain difficulties. The person who extracts the knowledge from the expert and encodes it in the knowledge base is termed the knowledge engineer. Typically, the knowledge engineer interviews one or more domain experts

and tries to make them articulate their in-depth knowledge in a manner that the knowledge engineer can understand. The inevitable communication issues can be avoided by the second approach in which the domain expert becomes a knowledge engineer or the knowledge engineer becomes a domain expert.

7.4 EXPERT SYSTEM VERSES TRADITIONAL SYSTEM

The main difference between the traditional system as opposed to the expert system is the way how the problem related expertise is coded. Especially in traditional applications, the problem expertise is encoded in both program and data structures. On the other hand, in expert systems, the problem related expertise is encoded in data structures only. Moreover, the use of knowledge in expert systems is essential. However, traditional systems require data more essentially than the expert system. One of the biggest drawbacks of conventional systems is that they are not providing explanations for the conclusion of a problem. This is because conventional systems try to solve problems in a straight forward manner. However, expert systems can able to provide not only explanations but also simplifying the understanding of a particular conclusion.

Generally, an expert system requires symbolic representations to perform computations. On the contrary, traditional systems are not capable of expressing these terms. They can only able to simplify the problems without being able to answer “how” and “why” questions. Moreover, expert systems have the problem-solving tools as opposed to the traditional ones, and hence, various types of problems are entirely solved by the experts of the system. Another advantage of expert system over conventional system is that they permit the use of certainty factors or confidences. This quality is very similar to the human reasoning where one can always conclude things with certainty. For example, “if you take medicine, then it might cure diseases”. The use of terms like if, then might etc., holds some sort of uncertainty. Expert systems allow such type of reasoning by making use of numeric values called confidences.

Further, conventional systems are developed and designed to provide correct answers whereas expert systems are developed to imitate the human behaviour and may provide incorrect answers.

7.4.1 Expert System Features

There are a number of features which are commonly used in expert systems. These features help the users to fully utilize the expert system’s capability conveniently in providing the most logical and reasonable decision in a problematic situation.

Backward chaining: An inference technique which continuously break a goal into smaller sub-goals which are easier to prove via IF THEN rules.

Dealing with uncertainties: The system has the capability to handle and reason with conditions that are uncertain and data which are not precisely known.

Forward chaining: An inference technique which deduce a problem solution from initial data via IF THEN rules.

Data representation: The method where the specific problem data is stored and accessed in the system

User interface: That portion of the code which creates an easy-to-use system;

Explanations: The ability of the system to explain the reasoning process that it used to reach a recommendation.

7.4.2 Advantages of Using Expert System

Expert system has been used in the field of business world to gain tactical advantages and forecast the market's situation. Today we are in the era where every decision made in the business world is critical for success and the assistance provided from an expert system is unconditionally essential and highly reliable for an organization to succeed. Following are the advantages for the implementation of an expert system in business

Providing consistent solutions – It can provide consistent answers for repetitive decisions, processes and tasks. As long as the rule base in the system remains the same, the final conclusions drawn will also remain the same regardless of how many times similar problems are being tested.

Provides reasonable explanations – It has the ability to clarify the reasons why the conclusion was drawn and why that conclusion is considered as the most logical choice among other alternatives. If there are any issues in concluding a certain problem, it will prompt some questions for users to answer in order to process the logical conclusion.

Overcome human limitations – It does not have human limitations and can work around the clock continuously. Users will be able to use it frequently while seeking solutions. The knowledge of experts is not only useful but also an invaluable asset for the company which can store the knowledge and use it as long as the organization needs.

Easy to adapt to new conditions – Unlike humans who often have troubles in adapting in new environments, an expert system has high adaptability and can meet new requirements in a short period of time. It also can acquire new knowledge from an expert and use it as inference rules to solve new problems.

7.4.3 Disadvantages of Using Expert System

Even though the expert system does provide many significant advantages, it does have its limitations as well. The points given below will be the disadvantages for the implementation of an expert system in business.

Lacks common sense – It lacks common sense needed in some decision making since all the decisions are based on the inference rules set in the system. It is not capable of providing creative and innovative responses as human experts would in unusual circumstances.

High implementation and maintenance cost – The implementation of an expert system in business will be a financial burden for smaller organizations since it has high development cost and subsequent recurring costs to upgrade the system to adapt in new environment.

Difficulty in creating inference rules – Domain experts will not always be able to explain their logic and reasoning needed for the knowledge engineering process. Hence, the task of codifying the knowledge is highly complex.

May provide wrong solutions – It is not error-free. There may be errors occurred in the process due to some logic mistakes made in the knowledge base which results in wrong solutions.

QUESTIONS

- 7.1 What is Expert system? Explain with examples.
- 7.2 Explain the phases of building expert system.
- 7.3 Explain the architecture of Expert system.
- 7,4 Discuss the differences between traditional system and expert system.
- 7.5 Explain the advantages and disadvantages of an expert system.



REFERENCES

1. Boyer, R.S., and Moore JS. The sharing of structure in theorem proving programs. In Machine Intelligence 7, B. Meltzer and D. Michie, Eds., Edinburgh University Press, 1972, pp. 101-16.
2. Deliyanni, A.J. A comparative study of semantic networks and predicate logic. M. Sc. Th., Dept. of Compt and Control, Imperial College, University of London, Sept. 1976.
3. Fikes, R.E., and Hendrix, G.G. A network-based knowledge representation and its natural deduction system. Proc. Fifth Int. Joint Conf. Artif. Intel., M.I.T., 1977, pp. 235-246.
4. Hayes, P.J. Computation and deduction. Proc. 2nd MFCS Symp., Czechoslovak Acad. Sciences, 1973, pp. 105-118.
5. Kowalski, R.A. Predicate logic as programming language. Information Processing 74, North Holland Pub. Co., Amsterdam, 1974, pp. 569-574.
6. Loveland, D.W. A unifying view of some linear Herbrand procedures. ACM 19, 2 (April 1972), 366-84.
7. Hendrix, G.G. Expanding the utility of semantic networks through partitioning. Proc. Fourth Int. Joint Conf. Artif. Intel., Tbilisi, Georgia, 1975, pp. 115-121
8. Kowalski, R.A. Algorithm = logic + control. Res. Rep. 77/3, Dept. of Comptg. and Control, Imperial College, University of London, Nov. 1976; to appear in Comm. ACM.
9. Luckham, D. Refinement theorems in resolution theory. Proc. IRIA Symp. on Automatic Demonstration, Versailles, France, 1970, pp. 162-90 (available from Springer-Verlag).
10. McSkimin, J.R., and Minker, J. A predicate calculus based semantic network for question-answering systems. Tech. Rep. TR 509, Dept. Comput. Sci., U. of Maryland, March 1977
11. Mylopoulos, J., Cohen, P., Borgida, A., and Sugar, L. Semantic networks and the generation of context. Proc. Fourth Int. Joint Conf. Artif. Intel., Tbilisi, Georgia, 1975,
12. Pereira Fernando, C N, and David H.D Warren (1980), Definite Clause Grammar for Natural Language Analysis. A survey Vol 13.
14. Turing Alan (1950), Computing machinery and Intelligence, Mind 49.
15. Wooldridge M and Jennings (1995), Intelligent Agents, Theory and practice, Knowledge engineering review.



INDEX

A

A and search, 33
ability, 2-4, 11, 115, 144, 169, 178
ability to think, 3-4
academics, 7
achievement, 5-6
action, 1, 7, 18-19, 24, 113-117, 121-122, 130-131, 136-137, 139, 155, 160-161
actions, 10-11, 16, 18-19, 113-114, 116-117, 121-122, 131-132, 136, 138-139, 141, 143
ADA, 14
Adequacy, 144, 146
adversary, 39
agents, 3, 6, 8-9, 143, 180
AI, 1-4, 6, 8-16, 19, 24, 38, 50, 64, 88, 113-114, 121, 130, 143-144, 146, 167, 169-170
Alan, 2, 4, 7, 86, 180
Alan Newell, 2
Alan Turing, 2, 4, 7
algorithm, 9-10, 13, 27-29, 31-34, 36-37, 41, 49-50, 52-57, 60, 81, 114, 117, 121, 125-126, 130-131, 159, 180
Alpha-beta, 54, 57, 60
Amazon, 3, 12
AND, 1-22, 24-34, 36-78, 81-90, 92-107, 109, 111-123, 125-127, 129-139, 141, 143-155, 157-167, 169-180
android app feature, 3
animals, 8-10, 148
any successor node is the goal node, then return success and stop the search, 33
any vertex as the beginning point of search, 29
applying heuristic search, 49
appointment, 3
appropriate action, 1
arcs, 39-42, 117, 150, 154, 156

arguments, 4, 64-65, 73-74, 150-151, 153-154
arrows, 5, 145
Artificial intelligence, 1-15, 18, 20, 22, 24, 26, 28, 30-32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112-114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142-144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168-170, 172, 174, 176, 178
assertion, 65, 102, 151-153, 158-159, 161
axiomatic system, 73, 81-83, 109, 112 axioms, 81-84, 106, 112, 114, 122

B

Backgammon, 62
backward, 17, 57, 62, 77, 114, 116-117, 130, 159, 161, 169, 176-177
backward from, 114, 116-117
backward-chaining, 176
Best First, 32-33, 38
BFS, 27-29, 32-33, 37-38, 50
bi-conditional, 68, 91
big data, 3
blank, 20
block world, 117-120, 122-123, 125-126, 141
board, 24, 51-52, 54-55, 61-63
boat, 24-25, 38
Boden, 1977, 4
book, 6, 60-61, 115
Bottom-Up Inference, 149, 152, 158-159, 161-162
bracketing, 90
Breadth-first, 27, 32, 38
breadth-first, 27, 32, 38

C

C, 9, 14, 25-26, 28, 30, 33, 37, 40, 44, 46-47, 50, 58-60, 67, 73, 76, 80, 82, 84-85, 91-93, 96, 101, 103-104, 107-108, 110-112, 115-116, 126-130, 141, 153, 156, 160, 180

C++, 14

Calculus, 65, 81, 96, 112, 148, 180

campaign, 13

Cannibals, 24-25

carts, 5

casual link, 117

categories, 6-7, 10, 27

cell, 19

centuries, 5

characteristics, 10, 26, 159

chatbot, 2-3

chatterbots, 7

Checkers, 62

checkmate, 61-62

chess, 3-4, 6, 50-51, 54-55, 61-62, 115

child, 27, 45-46, 54, 57, 111

choice, 3, 14, 44, 55-57, 121, 143, 178

civilization, 5

clauses, 86, 90-93, 95, 104-107, 112, 149, 151-152, 154, 156-162, 164

CLOSED, 32-34, 77-78

clues, 72

CNF, 88-93, 95-96, 106-107

COBOL, 2

Cognitive psychology, 9

cognitive science, 9

communication, 5-6, 177

companies, 3, 12

composition, 12-13

computational grounds, 3

computer science, 3, 6, 14, 65

computers, 4-6, 9, 11, 50, 52, 143

concepts, 5, 14, 64-65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95-97, 99, 101, 103, 105, 107, 109, 111, 170, 172-173

conceptualisation, 170-172

Conclusion, 64-65, 69-70, 74, 77, 93-95, 105-107, 112, 148, 151-152, 154, 159-160, 162-163, 169, 176-178

condition, 19-20, 22, 58-59, 68, 121, 143, 151-152, 154

configurations, 16, 20, 24, 51

conjunctive normal form, 88, 90, 107, 109, 112

connectives, 66, 69, 73, 77, 82, 87-88, 90, 96, 104, 148

consequence, 74, 78, 80-81, 83, 93-94, 154

consistent, 10, 77-80, 86, 111, 178

Control Strategy, 16, 19-20, 23, 26, 176

conventional, 19, 177

convert, 87, 91-92, 95-96, 103, 106, 109-110, 112, 135

cost, 3, 16, 24, 31-34, 36, 40, 44-52, 55, 115, 178

criteria, 24

Ctesibius of Alexandria, 9

cycles, 17

D

data, 3, 10-14, 20, 27-31, 54, 64, 143, 145, 149, 167, 169, 176-178

database, 19-20, 61, 125-126, 144, 149, 153, 176

Dead State, 116

deceiving, 4

deciphering Enigma, 4

decisions, 4, 9, 11, 77, 175, 178

decomposition, 39-40, 42, 44, 115

deduction rule, 83-84

Deep blue, 3

delay, 5

depth-first, 28, 32, 38, 56

Detect, 8, 12, 114-117

detecting, 12, 103, 114, 116

developed, 3, 5, 12, 14, 152, 159, 167, 169-170, 176-177

developing, 1-2, 170, 172

DFS, 28-29, 37-38, 56

diagnose, 6

diagonal, 52, 61

diagram, 17-18, 98, 135-136, 147, 171, 175
 Dimensions, 44
 discover, 4-5
 discs, 40-41
 Disjunctive Normal Form, 88, 91, 112
 divided, 6, 62, 97, 131-132
 DNF, 88-89, 91
 dualism, 8
 Duplex, 3
 duration, 2-3, 89, 92-93, 101, 106, 118, 120-121,
 123, 126, 136, 164, 176

E

Economists, 9
 elimination, 31, 73-75, 91, 161
 ELIZA, 2, 6-7
 emphasized, 2
 empty, 19-20, 22, 27, 29-30, 32, 34, 72, 86
 engineering problems, 4
 Enigma, 4
 Equivalence, 70-71, 79, 89, 104
 equivalent, 2, 70-72, 86, 88-89, 96, 99, 102, 111,
 147-148, 157
 Eugene Goostman, 3
 evaluation, 14, 32-34, 54-55
 E.W. Beth, 77
 Example, 1, 8-9, 11-14, 17, 19, 21-22, 24, 27, 29,
 31-34, 38, 41, 45-46, 51-54, 60-69, 71-73,
 76-77, 79-103, 105-106, 111, 113, 115, 117,
 119, 126, 132, 134, 141, 144-146, 148-154,
 158, 160-161, 165-167, 172-173, 177
 Examples, 1, 31, 34, 71, 76, 79, 88, 93, 101, 104,
 143, 149, 167, 169, 171, 173, 176, 179
 exchange, 5, 13, 88
 Expand, 27, 32, 34, 42, 46-49, 55, 133
 experiments, 5, 7
 expert system, 2-3, 169-179

F

Facebook, 1, 3, 13
 false, 58, 65-68, 70, 72, 74, 89, 96-102, 115,
 138-139
 feedback, 9, 30, 174

field, 1-2, 6, 8-9, 12, 31, 81, 113, 134, 178
 Formal Logic, 64-65
 formalization, 172
 formula, 70, 73, 75-91, 95, 97, 99, 101-106, 109,
 111-112
 FORTRAN, 2
 forward-chaining, 176
 frame, 6, 19, 116, 122, 164-167
 Frame based, 164
 function, 9, 16, 24-25, 31-34, 38, 44-46, 51, 53,
 55-56, 63, 96, 100-104, 115-116, 151
 functions, 1, 9, 14, 17, 31, 55, 102-103
 fuzzy, 5-6

G

gallons, 21-23
 game design, 13
 Game playing, 39, 41, 43, 45, 47, 49-53, 55, 57,
 59-61, 63, 115
 Game trees, 39
 Gaming, 13
 Gary Kasparov, 3
 General artificial intelligence, 1
 general-purpose, 14, 19, 157
 Goal stack, 120-121, 125-130, 134, 141
 goal state, 16-17, 19-20, 22-25, 31-33, 38, 42,
 51, 113-117, 119-120, 123, 125-126, 130,
 132, 135
 Google, 1, 3, 11
 GPS, 12-13
 graph, 12, 16-17, 24, 27-30, 36-37, 39, 42-45,
 54, 108, 117, 146, 150
 guide, 6, 149
 guns, 5

H

hairdresser, 3
 healthcare, 10, 12
 heuristic, 30-34, 36, 44-46, 49, 52, 54-56, 60,
 113, 115, 126, 135, 146
 Hill climbing, 31-32, 38 horizon effect, 60-61
 huge, 5, 61

human, 1-7, 10-13, 50, 52, 73, 76, 115, 143-144, 147, 150-151, 153, 156-157, 166, 168-171, 174, 176-178

humanoid, 2

I

IBM, 3

Imperfect, 51

implementation, 14, 134, 149, 164, 170, 173-174, 178

implication, 68, 71, 79, 87, 91, 148

inconsistent, 77-78, 80, 162

inference, 8, 19, 65, 73-75, 81-83, 86, 92, 106, 144-145, 148-149, 152, 158-164, 167, 170, 175-179

inferencing method, 160

Inferential Efficiency, 144, 146

Inferential knowledge, 146

influence, 7

informal, 8, 22, 64, 82

Informal logic, 64

information, 3, 5, 8-9, 12-13, 27, 31, 51, 61, 64, 73, 81, 113, 122, 143-144, 147, 153, 157, 165-166, 171, 176, 180

inheritable knowledge, 145

initial, 16-17, 19-20, 23-26, 28, 31-34, 51, 55-56, 113-114, 116-117, 119, 122-123, 126-127, 132, 135, 138, 164, 177

initial state, 16, 23-26, 31, 33-34, 51, 113-114, 116-117, 119, 122, 126-127, 132, 135

inputs, 31

Instagram, 13

instance, 1, 6, 24, 39, 83, 98, 105, 145, 164-166, 176

intelligent, 1-8, 11-12, 14-15, 30, 64, 143-144, 147, 169, 172, 180

interactive composition, 13

interdisciplinary, 8

interpretations, 10, 102

interrogator, 4-5, 7

introduction, 1, 3, 5, 7, 9, 11, 13, 15-16, 39, 64, 73-75, 113, 143, 153, 169

J

J. Hintikka, 77

JAVA, 14

John McCarthy, 2, 14

Joseph Weizenbaum, 2, 6

jugs, 21-22

K

Keras, 14

key, 4, 57, 61, 86, 172

knowledge representation, 7, 10, 143-147, 149, 151, 153, 155, 157, 159, 161, 163-165, 167-168, 172-173, 180

knowledge-based systems, 6

L

Labelling, 45-49, 54, 63

Linguistics, 9-10

LISP, 2, 14, 146

literals, 86, 88-92, 97, 105-106, 115-116

local motion, 26

Logic, 2, 6, 8-9, 14, 54, 64-67, 69, 71-73, 75, 77, 79, 81-83, 85-87, 89-91, 93-99, 101-103, 105-107, 109, 111, 113-114, 117-118, 146, 148-154, 157-159, 167, 179-180

logical, 8-9, 14, 64, 66, 68-71, 73, 77-78,

80-83, 86, 89, 93-94, 102, 115, 118, 148, 151, 177-178

look ahead, 55, 63

M

machines, 2-6, 9-12, 64, 76, 169

Manchester University, 4

marketing, 1, 13

Markov property, 9

materialism, 8

mathematical logic, 65

mathematics, 2, 5, 8-9, 83-84

matrix, 20, 40, 44, 104

MAX, 44, 52, 54-60

maximize, 9, 31, 44, 52-53, 55

Metric, 31

MIN, 44, 50, 52, 55-59

Minimax, 52-57, 60-61, 63
 minimize, 31, 44, 53, 55-56
 Missionaries, 24-25
 MIT, 6
 Modulus Ponon, 75
 mortal, 8, 146
 motion, 26
 move left, 17
 move right, 17
 MPP, 75-77
 music, 12-13

N

narrow intelligence, 1
 Natural deduction, 73-76, 180
 natural language, 1, 3, 6-7, 9-14, 64, 73, 143, 175, 180
 negation, 66-68, 71, 75, 79, 86-88, 90-93, 95, 103-104, 106
 nested, 101
 Netflix, 3
 neural network, 12, 14
 node, 16, 27-29, 32-34, 36-37, 44-50, 52, 54-61, 63, 92, 117, 121, 150, 154, 156
 Non-Terminal, 45-46, 49, 52
 Nonlinear, 54, 135
 NULL, 28, 86, 95, 135

O

O's, 52
 OPEN, 6, 32-34, 77-78
 operational efficiency, 12
 operations, 5, 117-119, 122, 125, 131-132, 134-135
 optimal, 9, 34, 45, 51, 55, 60, 64, 120-121
 OR, 1-2, 4-9, 11, 14, 16-17, 19-20, 22, 24-29, 31, 33-34, 38-47, 49-56, 58-62, 64-70, 73, 75, 77-78, 81-84, 86, 88, 90-91, 94, 96, 98-99, 101-104, 106, 111-115, 117-119, 126, 130-132, 134, 136-137, 141, 143, 146, 148, 150-151, 154, 157, 159, 164-166, 169-170, 172-173, 175-177
 organ, 4

P

parent problem, 40
 partial plans, 117
 path, 12, 16-17, 23, 27, 32-34, 36-37, 42, 44-48, 51-52, 55-59, 77-78, 114-117, 149
 Path cost, 16, 51
 payoff function, 51
 Perfect, 51, 61
 Philippe Roussel, 14
 Philosophers, 8
 philosophical, 5-6
 piece, 10, 61-63
 Planning, 1, 9, 24, 39-40, 113-123, 125, 127, 129, 131, 133-135, 137, 139, 141, 176
 Player, 4, 13, 51-55, 57-63
 points, 57, 62, 82, 116, 171-172, 178
 pole, 41, 43
 potential applications, 6
 powerful machines, 9
 practical, 4, 6, 51, 83, 97
 practice, 8, 50, 147, 164, 180
 Predicate Logic, 64, 96, 98, 101, 105, 111, 113-114, 146, 149, 180
 premises, 8, 64-65, 70, 74, 77, 82-83, 86, 93, 95, 148, 159
 prenex normal form, 104, 107
 Propositional, 65
 Problem, 1, 4, 6, 8, 11, 16-27, 29-37, 39-47, 49, 51, 53, 55, 57, 59, 61, 63, 72, 95, 97, 107-108, 113-123, 125-127, 129-135, 137, 139, 141, 143-144, 148-149, 156, 159, 164, 169-173, 175-178
 Problem Reduction, 39-41, 43-45, 47, 49, 51, 53, 55, 57, 59, 61, 63
 problem solving, 16-17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 134, 175
 problems, 1-2, 4, 6-7, 9, 14, 16, 19, 24, 30-31, 39-42, 46-47, 51, 113-115, 130-131, 133-134, 143, 169, 173, 176-178
 procedural knowledge, 64, 146
 Production system, 19-20, 38
 progress, 8-9, 114
 Project debater, 3

PROLOG, 14, 146, 148-149
 proof, 74-77, 81, 83-86, 88, 95, 102, 106, 149
 Propagate, 55-57
 Propositional Logic, 8, 64, 66, 69, 72, 82,
 86-87, 89-91, 94, 96, 102, 106, 111
 pruning, 54, 57, 59-60, 63
 psychological, 7
 Psychologists, 9
 puzzle, 19-21, 34-35, 40-41, 72
 Python, 14

Q

quantifier, 96-99, 101, 103-104, 107, 147, 158
 Questions, 3-8, 15, 38, 63, 111, 141, 147, 167,
 175, 177-179
 queues, 27

R

rational agent, 8
 rationality, 8
 real-time, 12-13
 reasonable, 31, 45, 64, 73, 177-178
 reasoning, 1, 6, 8, 64, 73, 77, 81-82, 114, 116,
 143, 149, 152, 159, 161, 175, 177-179
 recommendations, 1, 12
 reduction, 19, 39-41, 43-45, 47, 49, 51, 53,
 55, 57, 59, 61, 63, 159
 refinement, 117, 180
 refinements, 60
 Refutation, 73, 77, 86, 93, 95
 researchers, 2-3
 Resolution, 73, 86, 89-94, 103, 105-108, 112,
 130, 149, 152, 159, 162, 180
 resolution, 73, 86, 89-94, 103, 105-108, 112,
 130, 149, 152, 159, 162, 180
 result, 3, 18, 24, 33, 41, 53, 55, 83, 86, 106, 114,
 133, 144, 149, 153
 revised, 46-48
 river, 24-25, 38
 Robert Kowalski, 14
 robot, 2, 12, 117-118, 133
 Robotics, 12
 Root, 27, 42, 53, 55, 77-81

rule firing, 19
 rules, 5, 8, 14, 16, 19-20, 22-23, 26, 41, 50-52,
 55, 61, 65, 73-79, 81-83, 86, 93, 95, 113-115,
 143, 148-149, 152, 158-159, 167, 173, 175-
 179

S

science, 2-4, 6, 9, 14, 64-65
 scientific, 1, 5
 Search, 1, 11, 16, 19, 22-24, 26-34, 37-39, 44-46,
 49-52, 55-56, 58, 60-61, 63, 114-117, 120-
 121, 130, 149, 167, 172
 self-driving, 9, 13
 semantic, 4, 73, 77-79, 146-150, 152, 154, 156-
 161, 164, 167-168, 180
 semantic tableau, 73, 77-79
 sensors, 13
 set of rules, 16, 19, 77, 81, 173
 simple applications, 1
 Skolem constant, 103
 Skolem function, 103-104
 smarter, 1, 4
 social media, 13
 solution, 5, 11, 16, 19-23, 26, 30-32, 34-40, 44-
 45, 51, 61, 71-72, 76-77, 79-81, 83, 85, 91,
 93-96, 103, 105, 109-111, 114-116, 121, 147,
 149, 170, 172, 176-177
 solution-cost, 40
 special purpose, 19, 22
 specific tasks, 1
 spoken, 5-6
 stack, 28-30, 41, 62, 118, 120-122, 124-130, 134,
 136, 138-141
 stage, 46-49, 75, 89, 171-174
 starting state, 16, 115-116
 State space, 16-19, 22, 24-25, 38, 46, 51, 55,
 120
 statements, 64-66, 69-70, 72, 81, 96-98, 102,
 111, 118, 146, 157
 strategy, 16, 19-20, 23, 26, 55-57, 61, 63, 117,
 120-121, 134, 149, 176
 STRIPS, 116, 121-123, 125, 141
 sub-problem, 39, 44, 171

successor, 14, 16, 25, 27, 33-34, 38-39, 44, 46, 49-52
 suck, 17-18
 Sun Microsystems, 14
 super intelligence, 1
 Symbolic logic, 64-65
 symbols, 52, 64-65, 100, 103, 150-151, 153-154, 157
 systematic, 19, 26, 57, 77, 82
 systems, 1-4, 6-7, 12-15, 81-82, 114-115, 141, 144, 165, 169-170, 172-173, 177, 180

T

table, 22-23, 51, 66-72, 74-81, 83-87, 96, 99, 107-109, 111, 117-120, 122, 133, 140-141, 144-145, 149, 164-167
 tableaux, 77
 talking, 3, 6
 tautology, 70, 74, 83
 technology, 2, 12-14, 121, 134, 172
 Terminal node, 45, 49, 54
 termination, 19-20
 Tesla cars, 13
 test, 2-5, 7, 16, 51, 111, 115
 Testing, 13, 19, 77, 170, 174
 theorem, 24, 55, 76-77, 81-86, 109, 112, 180
 thesis, 6, 81
 Tic Tac Toe, 52-54
 Tollendo Tollens, 75
 tools, 5-6, 8, 173, 177
 top-down inference, 149, 152, 158-160, 162-164
 Tower of Hanoi, 40-43 traditional, 30, 177, 179
 transforming, 1, 104 translate, 13, 69, 111

tree, 14, 16-17, 22-23, 27-28, 32-35, 38, 41, 44-46, 49, 51-58, 60-61, 63, 77-78, 80, 92-94, 108, 121, 147
 trivial, 83
 Truth tables, 66, 69, 73-74, 111
 Turing, 2-5, 7, 180
 Turing Test, 2-3, 5, 7
 Twitter, 3

U

Uninformed, 26-27
 utility, 9, 51, 55-56, 180

V

vacuum cleaner, 3, 17
 variables, 9, 64-66, 69, 74, 88, 90, 96-98, 100-101, 103-106, 151-154
 virtual assistant, 3
 Virtual shopping, 12
 written, 4-5, 7, 11, 22, 36, 56, 67, 69, 92, 148

W

wake-up, 117
 water, 9, 21-23
 water clock, 9
 weak AI, 1
 Weizenbaum, 2, 6
 Well Formed Formula, 97
 wff, 81, 97-98, 107
 world chess, 3
 World Space, 117

X

X's, 52-54



ABOUT THE BOOK

The concept of Artificial Intelligence has been in practice for over years with the advent of technological progress. Over time, it has blended our lives through nearly every narration of learning, teaching, enjoyment, normal routine operations and what not. Advancements in Artificial Intelligence becoming popular and makes all technical universities to include fundamental topics in their curriculum. This book has been designed keeping in mind the syllabus framed by VTU and all essential topics required to be taught to the students of Technical Universities in India. All the major topics like Problem Solving, Game Playing, Logic Concepts, Knowledge Representation and Expert Systems are explained to make students to understand the concepts in simple words with figures and examples.

ABOUT THE AUTHOR

Sathish S. Nadig pursued his Bachelor of Engineering from Karnataka University, Dharwad and Masters from Visveswaraya Technological University Belagavi. His area of interest is Cloud computing applications and had been a cloud solution architect. He has special affinity towards subjects like Artificial Intelligence, Network Content Delivery and Quantum Technologies. He has more than fifteen years of experience in technical teaching. Also, he acted as external examiner for both theory and lab university examinations. He is currently working as a faculty in Canara Engineering College, Mangalore.

www.himpub.com



ISBN: 978-93-5693-192-3

ESM 0920