GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

ARTIFICIAL INTELLIGENCE

(ELECTIVE I)

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON ARTIFICIAL INTELLIGENCE

Class & Sem. :III B.Tech – II Semester Year : 2018-19

Branch : CSE Credits : 3

==

1. Brief History and Scope of the Subject

The seeds of modern AI were planted by classical philosophers who attempted to describe the process of human thinking as the mechanical manipulation of symbols. This work culminated in the invention of the programmable digital computer in the 1940s, a machine based on the abstract essence of mathematical reasoning. This device and the ideas behind it inspired a handful of scientists to begin seriously discussing the possibility of building an electronic brain.

The field of AI research was founded at a workshop held on the campus of Dartmouth College during the summer of 1956. Those who attended would become the leaders of AI research for decades. Many of them predicted that a machine as intelligent as a human being would exist in no more than a generation and they were given millions of dollars to make this vision come true.

In the 1940s and 50s, a handful of scientists from a variety of fields (mathematics, psychology, engineering, economics and political science) began to discuss the possibility of creating an artificial brain. The field of artificial intelligence research was founded as an academic discipline in 1956.

2. Pre-Requisites

- Mathematical Logic
- Formal Reasoning

3. Course Objectives:

- To familiarize the concepts of AI for representation of knowledge and problem solving

4. Course Outcomes:

At the end of the course, the students will be able to

- **CO1:** Analyze different problem solving and game playing techniques.
- **CO2:** Compare different approaches to represent knowledge.
- **CO3:** Analyze expert systems and their applications.
- **CO4:** Apply probability theory for real world problems.

5. Program Outcomes:

Graduates of the Computer Science and Engineering Program will have an ability to

a. apply knowledge of computing, mathematics, science and engineering fundamentals to solve complex engineering problems.

b. formulate and analyze a problem, and define the computing requirements appropriate to its solution using basic principles of mathematics, science and computer engineering.

c. design, implement, and evaluate a computer based system, process, component, or software to meet the desired needs.

d. design and conduct experiments, perform analysis and interpretation of data and provide valid conclusions.

e. use current techniques, skills, and tools necessary for computing practice.

f. understand legal, health, security and social issues in Professional Engineering practice.

g. understand the impact of professional engineering solutions on environmental context and the need for sustainable development.

h. understand the professional and ethical responsibilities of an engineer.

i. function effectively as an individual, and as a team member/ leader in accomplishing a common goal.

j. communicate effectively, make effective presentations and write and comprehend technical reports and publications.

k. learn and adopt new technologies, and use them effectively towards continued professional development throughout the life.

l. understand engineering and management principles and their application to manage projects in the software industry.

6. Mapping of Course Outcomes with Program Outcomes:

	a	b	c	d	e	f	g	h	i	j	k	l
CO1	H	H		M								
CO2				H								M
CO3		H		H	M							M
CO4	H				M							M

7. Prescribed Text Books

1.Elaine Rich & Kevin Knight, 'Artificial Intelligence', Tata McGraw Hill Edition, 2 nd Edition.

2.Stuart J. Russell,Artificial Intelligence: A Modern Approach,2nd Edition.

8. Reference Text Books

1. Patrick Henry Winston, 'Artificial Intelligence', Pearson Education.
2. Russel and Norvig, 'Artificial Intelligence', Pearson Education/ PHI.

9. URLs and Other E-Learning Resources

URLs:

- https://nptel.ac.in/courses/106105077/
- https://nptel.ac.in/courses/106105079/
- https://ocw.mit.edu/courses/electrical...and...artificial-intelligence.../lecture-videos/

Journals:

- International Journal on Artificial Intelligence Tools
- Journal of Artificial Intelligence Research
- Applied Artificial Intelligence

10. Lecture Schedule / Lesson Plan

Topic	No. of Periods	
	Theory	Tutorial
UNIT - I : Introduction to artificial intelligence		
Introduction	1	1
History	1	
Intelligent systems	1	
Foundations of AI	1	1
Applications	2	

tic-tac-toe game playing	2	
Current trends in AI	2	
UNIT - II: Problem solving and game playing		
Problem solving: state-space search and control strategies	2	1
Introduction, general problem solving	1	
Characteristics of problem	1	1
Exhaustive searches	2	
Heuristic search techniques	2	
Iterative-deepening a*	2	
Problem reduction	2	
Constraint satisfaction	2	
Game playing: Introduction	1	1
Game playing	2	
Alpha-beta pruning	2	
Two-player perfect information games	1	
UNIT - III: Logic Concepts		
Introduction	1	1
Propositional calculus	1	
Proportional logic	1	
Natural deduction system	1	
Axiomatic system	1	1
Semantic tableau system in proportional logic	1	
Resolution in proportional logic	1	
Predicate logic	1	
UNIT - IV: Knowledge representation		
Introduction	1	1
Approaches to knowledge representation	1	
Knowledge representation using semantic network	1	
Extended semantic networks for KR	1	
Knowledge representation using frames	1	
Advanced knowledge representation techniques: Introduction	1	
Conceptual dependency theory	1	1
Script structure	1	
Semantic web	1	
UNIT - V: Expert system and applications		
Introduction phases in building expert systems	1	1
Expert system versus traditional systems	1	
Rule-based expert systems	1	
Blackboard systems truth maintenance systems	1	1
Application of expert systems	1	
List of shells and tools	1	
UNIT - VI: Uncertainty measure		
Introduction	2	1
Probability theory	2	

Bayesian belief networks	2	1
Certainty factor theory	1	
Total No.of Periods:	**60**	**13**

UNIT – I

Introduction to artificial intelligence

Syllabus:

Introduction, history, intelligent systems, foundations of AI, applications, tic-tac-toe game playing, current trends in AI.

Outcomes:

Student will be able to:

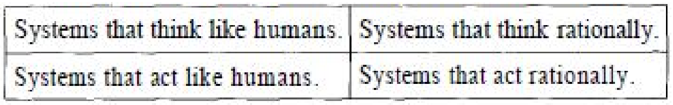
- define the concept of Artificial Intelligence
- explain brief history that contributed ideas and techniques to Artificial Intelligence
- outline current trends in AI
- interpret the steps in solving of tic-tac-toe problem

Introduction:

- ➢ The field of **artificial intelligence,** or AI, attempts to understand intelligent entities.
- ➢ Computers with human-level intelligence (or better) would have a huge impact on our everyday lives and on the future course of civilization.
- ➢ Computers have unlimited potential for intelligence.
- ➢ AI currently encompasses a huge variety of subfields, from general-purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases.

Definitions of artificial intelligence:

- ➢ Definitions of AI vary along following dimensions.
  - thought processes and reasoning
  - address behaviour
  - measure success in terms of human performance,
  - measure against an ideal concept of intelligence, which we will call **rationality**
- ➢ A system is "**rational"** if it does the right thing.
- ➢ Definitions are organized into four categories:



Systems that think like humans.	Systems that think rationally.
Systems that act like humans.	Systems that act rationally.

- **Systems that think like humans:** The exciting new effort to make computers think machines with minds.
- **Systems that think rationally:** The study of the computations that make it possible to perceive, reason, and act.
- **Systems that act like humans:** The study of how to make computers do things at which, at the moment, people are better.
- **Systems that act rationally**: The branch of computer science that is concerned with the automation of intelligent behaviour.

Acting humanly: The Turing Test approach:

- **The Turing Test,** proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence.
- Turing defined intelligent behaviour as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator.
- The test he proposed is that the computer should be interrogated by a human via a teletype, and passes the test if the interrogator cannot tell if there is a computer or a human at the other end.
- The computer would need to possess the following capabilities:
  - **natural language processing** to enable it to communicate successfully in English (or some other human language);
  - **knowledge representation** to store information provided before or during the interrogation;
  - **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 extrapolate patterns.
- Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because physical simulation of a person is unnecessary for intelligence.

Thinking humanly: The cognitive modelling approach:

- To say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds.
- There are two ways for this:
  - through introspection—trying to catch our own thoughts as they go by.
  - through psychological experiments

- Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program's input/output and timing behaviour matches human behaviour, that is evidence that some of the program's mechanisms may also be operating in humans.

Thinking rationally: The laws of thought approach:

- The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes.
- His famous syllogisms provided patterns for argument structures that always gave correct conclusions given correct premises. For example, "Socrates is a man; all men are mortal; therefore Socrates is mortal."
- These laws of thought were supposed to govern the operation of the mind, and initiated the field of logic.
- There are two main obstacles to this approach.
  - First, 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 100% certain.
  - Second, there is a big difference between being able to solve a problem "in principle" and doing so in practice.

Acting rationally: The rational agent approach:

- Acting rationally means acting so as to achieve one's goals, given one's beliefs.
- An agent is just something that perceives and acts.
- In this approach, AI is viewed as the study and construction of rational agents. In the ‘laws of thought" approach to AI, the whole emphasis was on correct inferences.
- Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals, and then to act on that conclusion.
- Correct inference is only a useful mechanism for achieving rationality, and not a necessary one.

Foundations of Artificial Intelligence:

- AI itself is a young field; it has inherited many ideas, viewpoints, and techniques from other disciplines.

- Over 2000 years of tradition in philosophy, theories of reasoning and learning have emerged, along with the viewpoint that the mind is constituted by the operation of a physical system.
- From over 400 years of mathematics, we have formal theories of logic, probability, decision making, and computation.
- From psychology, we have the tools with which to investigate the human mind, and a scientific language within which to express the resulting theories.
- From linguistics, we have theories of the structure and meaning of language.
- From computer science, we have the tools with which to make AI a reality.

Philosophy (428 B.C.-present)

- We have the idea of a set of rules that can describe the working of (at least part of) the mind, the next step is to consider the mind as a physical system.
- Mental processes and consciousness are therefore part of the physical world, but inherently unknowable; they are beyond rational understanding.
- Some philosophers critical of AI have adopted exactly this position.
- Barring these possible objections to the aims of AI, philosophy had thus established a tradition in which the mind was conceived of as a physical device operating principally by reasoning with the knowledge that it contained.
- All knowledge can be characterized by logical theories connected, ultimately, to observation sentences that correspond to sensory inputs.
- Confirmation theory of Rudolf Carnap and Carl Hempel attempted to establish the nature of the connection between the observation sentences and the more general theories—understand how knowledge can be acquired from experience.

Mathematics (c. 800-present)

- AI used mathematical formalization in three main areas: computation, logic, and probability.
- First-order logic is used today as the most basic knowledge representation system.
- Theory of reference that shows how to relate the objects in a logic to objects in the real world.
- There are some functions on the integers that cannot be represented by an algorithm—that is, they cannot be computed.

- This motivated Alan Turing (1912-1954) to try to characterize exactly which functions are capable of being computed.
- Church-Turing thesis, which states that the Turing machine (Turing, 1936) is capable of computing any computable function, is generally accepted as providing a sufficient definition.

Psychology (1879-present)

- The view that the brain possesses and processes information, is the principal characteristic of cognitive psychology,
- The three key steps of a knowledge-based agent:
    - The stimulus must be translated into an internal representation,
    - The representation is manipulated by cognitive processes to derive new internal representations,
    - These are in turn retranslated back into action.

Computer engineering (1940-present)

- For artificial intelligence to succeed, we need two things: intelligence and an artifact.
- The computer has been unanimously acclaimed as the artifact with the best chance of demonstrating intelligence.
- AI also owes a debt to the software side of computer science, which has supplied the operating systems, programming languages, and tools needed to write modern programs.
- AI has pioneered many ideas that have made their way back to "mainstream" computer science, including time sharing, interactive interpreters, the linked list data type, automatic storage management, and some of the key concepts of object-oriented programming and integrated program development environments with graphical user interfaces.

Linguistics (1957-present)

- Understanding language requires an understanding of the subject matter and context, not just an understanding of the structure of sentences.
- Much of the early work in knowledge representation (the study of how to put knowledge into a form that a computer can reason with) was tied to language and informed by research in linguistics, which was connected in turn to decades of work on the philosophical analysis of language.
- Modern linguistics and AI were "born" at about the same time, so linguistics does not play a large foundational role in the growth of AI.

Instead, the two grew up together, intersecting in a hybrid field called computational linguistics or natural language processing, which concentrates on the problem of language use.

The History of Artificial Intelligence

The gestation of artificial intelligence (1943-1956)

- The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943). They drew on three sources:
    - knowledge of the basic physiology
    - function of neurons in the brain
    - the formal analysis of propositional logic
- Turing's theory of computation: They proposed a model of artificial neurons in which each neuron is characterized as being "on" or "off," with a switch to "on" occurring in response to stimulation by a sufficient number of neighbouring neurons.
- Claude Shannon (1950) and Alan Turing (1953) were writing chess programs for von Neumann-style conventional computers.
- At the same time, two graduate students in the Princeton mathematics department, Marvin Minsky and Dean Edmonds, built the first neural network computer in 1951.

Early enthusiasm, great expectations (1952-1969)

- Newell and Simon's early success was followed up with the General Problem Solver, or GPS. Unlike Logic Theorist, this program was designed from the start to imitate human problem-solving protocols.
- Within the limited class of puzzles it could handle, it turned out that the order in which the program considered subgoals and possible actions was similar to the way humans approached the same problems. Thus, GPS was probably the first program to embody the "thinking humanly" approach.
- Herbert Gelernter (1959) constructed the Geometry Theorem Prover. Like the Logic Theorist, it proved theorems using explicitly represented axioms.
- Gelernter soon found that there were too many possible reasoning paths to follow, most of which turned out to be dead ends.
- Arthur Samuel wrote a series of programs for checkers (draughts) that eventually learned to play tournament-level checkers. Along the way, he disproved the idea that computers can only do what they are told

to, as his program quickly learned to play a better game than its creator.

- McCarthy defined the high-level language Lisp, which was to become the dominant AI programming language.
- Rosenblatt proved the famous perceptron convergence theorem, showing that his learning algorithm could adjust the connection strengths of a perceptron to match any input data.

A dose of reality (1966-1974)

- Most of the early AI programs worked by representing the basic facts about a problem and trying out a series of steps to solve it, combining different combinations of steps until the right one was found.
- The early programs were feasible only because micro worlds contained very few objects.
- Before the theory of NP-completeness was developed, it was widely thought that "scaling up" to larger problems was simply a matter of faster hardware and larger memories.
- Resolution theorem proving, was soon dampened when researchers failed to prove theorems involving more than a few dozen facts.
- A two-input perceptron could not be trained to recognize when its two inputs were different.
- Although their results did not apply to more complex, multilayer networks, solved the problem.
- The new back-propagation learning algorithms for multilayer networks that were to cause an enormous resurgence in neural net research.

Knowledge-based systems: The key to power? (1969-1979)

- General-purpose search mechanism performs elementary reasoning steps to find complete solutions. Such approaches are called **weak methods**, because they use weak information about the domain.
- The only way around this is to use knowledge more suited to making larger reasoning steps and to solving typically occurring cases in narrow areas of expertise.
- The DENDRAL program solves the problem of inferring molecular structure from the information provided by a mass spectrometer.
- The input to the program consists of the elementary formula of the molecule, and the mass spectrum giving the masses of the various fragments of the molecule generated when it is bombarded by an electron beam.

- The naive version of the program generated all possible structures consistent with the
- formula, and then predicted what mass spectrum would be observed for each.
- The significance of DENDRAL was that it is the first successful knowledge-intensive
- System.
- MYCIN diagnoses blood infections. With about 450 rules, MYCIN was able to perform as well as some experts, and considerably better than junior doctors.
- It also contained two major differences from DENDRAL.:
  - First, unlike the DENDRAL rules, no general theoretical model existed from which the MYCIN rules could be deduced. They had to be acquired from extensive interviewing of experts, who in turn acquired them from direct experience of cases.
  - Second, the rules had to reflect the uncertainty associated with medical knowledge. MYCIN incorporated a calculus of uncertainty called certainty factors.

AI becomes an industry (1980-1988)

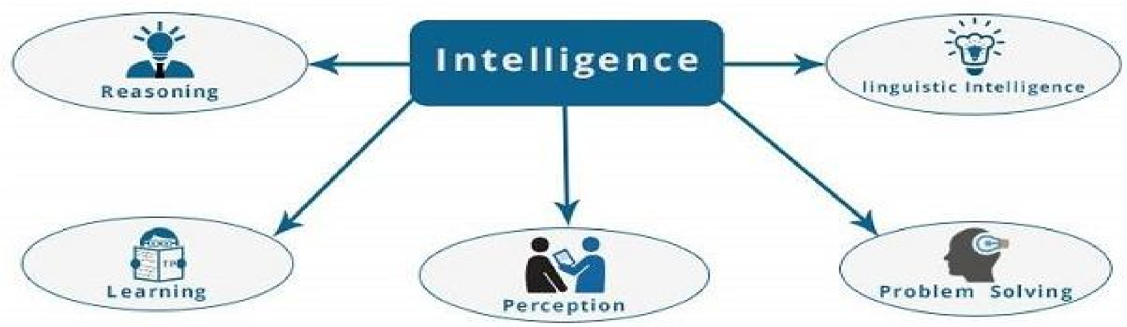
- In 1981, the Japanese announced the "Fifth Generation" project, a 10-year plan to build intelligent computers running Prolog in much the same way that ordinary computers run machine code.
- It has the ability to make millions of inferences per second.
- The project proposed to achieve full-scale natural language understanding, among other ambitious goals.
- The booming AI industry also included companies such as Carnegie Group, Inference, Intellicorp, and Teknowledge that offered the software tools to build expert systems, and hardware companies such as Lisp Machines Inc., Texas Instruments, Symbolics, and Xerox that were building workstations optimized for the development of Lisp programs.

The return of neural networks (1986-present)

- Back-propagation learning was applied to many learning problems in computer science and psychology, and the widespread dissemination of the results in **Parallel Distributed Processing.**
- In recent years, approaches were based on **hidden Markov models (HMMs)** which are based on a rigorous mathematical theory. These are generated by a process of training on a large corpus of real speech data.

Intelligent Systems

- **Intelligence:**The ability of a system to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize, and adapt new situations.
- **Intelligent systems** are technologically advanced machines that perceive and respond to the world around them.
- An **intelligent system** is a computer-based system that can represent reason about, and interpret data.
- An **intelligent system** is a system with artificial intelligence.



- **Reasoning** − It is the set of processes that enables us to provide basis for judgement, making decisions, and prediction.
- **Learning** − It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something.
- **Problem solving**- It is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal are available.
- **Perception** − It is the process of acquiring, interpreting, selecting, and organizing sensory information. Perception presumes sensing. In humans, perception is aided by sensory organs. In the domain of AI, perception mechanism puts the data acquired by the sensors together in a meaningful manner.
- **Linguistic Intelligence** − It is one's ability to use, comprehend, speak, and write the verbal and written language. It is important in interpersonal communication.

Applications of AI

- **Air Operations Division (AOD) -** uses AI for the rule based expert systems.
  - The AOD has use for artificial intelligence for surrogate operators for combat and training simulators, mission management aids, support systems for tactical decision making,

and post processing of the simulator data into symbolic summaries.

- Airplane simulators are using artificial intelligence in order to process the data taken from simulated flights.
- Simulated aircraft warfare computers are able to come up with the best success scenarios in these situations

- **Computer Science:** AI researchers have created many tools to solve the most difficult problems in computer science.
    - time sharing ,
    - interactive interpreters,
    - graphical user interfaces and the computer mouse,
    - rapid development environments,
    - the linked list data structure,
    - automatic storage management,
    - symbolic programming,
    - functional programming,
    - dynamic programming
    - object-oriented programming
- **Education:** Number of companies that create robots to teach subjects to children ranging from biology to computer science.
    - **Intelligent tutoring systems**: An ITS called **SHERLOCK** teaches Air Force technicians to diagnose electrical systems problems in aircraft. Another example is **DARPA**, Defense Advanced Research Projects Agency, which used AI to develop a digital tutor to train its Navy recruits in technical skills in a shorter amount of time.
- **Finance: Algorithmic Trading** involves the use of complex AI systems to make trading decisions at speeds several orders of magnitudes greater than any human is capable of, often making millions of trades in a day without any human intervention. Such trading is called **High-frequency Trading**. Many banks, funds, and proprietary trading firms now have entire portfolios which are managed purely by AI systems.
- **Hospitals and medicine:** used as clinical decision support systems for medical diagnosis.
    - Computer-aided interpretation of medical images help scan digital images, e.g. from computed tomography, for typical appearances and to highlight conspicuous sections, such as possible diseases. A typical application is the detection of a tumor.
    - Heart sound analysis

  - Companion robots for the care of the elderly
  - Mining medical records to provide more useful information.
  - Design treatment plans.
  - Assist in repetitive jobs including medication management.
  - Provide consultations.
  - Drug creation[27]
  - Using avatars in place of patients for clinical training[28]
  - Predict the likelihood of death from surgical procedures
  - Predict HIV progression
- **Media and E-commerce:** Typical use case scenarios include:
  - analysis of images using object recognition or face recognition techniques,
  - analysis of video for recognizing relevant scenes, objects or faces.
  - facilitation of media search,
  - creation of a set of descriptive keywords for a media item,
  - media content policy monitoring (such as verifying the suitability of content for a particular TV viewing time),
  - speech to text for archival or other purposes, and the detection of logos,
  - products or celebrity faces for the placement of relevant advertisements
  - AI is also widely used in E-commerce Industry for applications like Visual search, Visually similar recommendation, Chatbots, Automated product tagging etc
- **Music:** At Sony CSL Research Laboratory, their Flow Machines software has created pop songs by learning music styles from a huge database of songs. By analyzing unique combinations of styles and optimizing techniques, it can compose in any style.
- **News, publishing and writing:** Artificial intelligence is used to turn structured data into intelligent comments and recommendations in natural language. We can be able to write financial reports, executive summaries, personalized sales or marketing documents and more at a speed of thousands of pages per second and in multiple languages including English, Spanish, French & German.
- **Online and telephone customer service:** Artificial intelligence is implemented in automated online assistants which uses natural language processing.
- **Sensors:** Artificial Intelligence has been combined with many sensor technologies, such as Digital Spectrometry TM which enables many applications such as at home water quality monitoring.

- **Telecommunications maintenance:** Many telecommunications companies make use of heuristic search in the management of their workforces, for example BT Group has deployed heuristic search in a scheduling application that provides the work schedules of 20,000 engineers.
- **Toys and games:** AI has been applied to video games, for example video game bots, which are designed to stand in as opponents where humans aren't available or desired.
- **Transportation:** Fuzzy Logic controllers have been developed for automatic gearboxes in automobiles. For example, the 2006 Audi TT, feature the DSP transmission which utilizes Fuzzy Logic.

AI trends in various sectors

- **Healthcare:**
  - AI and ML technology has been particularly useful in the healthcare industry because it generates massive amounts of data to train with and enables algorithms to spot patterns faster than human analysts.
  - Medecision developed an algorithm that detects 8 variables in diabetes patients to determine if hospitalization is required.
  - An app called BiliScreen utilizes a smartphone camera, ML tools, and computer vision algorithms to detect increased levels of bilirubin in the sclera (white portion) of a person’s eye, which is used to screen people for pancreatic cancer. This cancer has no telltale symptoms, hence it has one of the worst prognoses of all cancers.
  - NuMedii, a biopharma company, has developed a platform called Artificial Intelligence for Drug Discovery (AIDD), which uses big data and AI to detect the link between diseases and drugs at the systems level.
  - GNS Healthcare uses ML algorithms to match patients with the most effective treatments for them.
- **Entertainment:**
  - A familiar application of AI in everyday life is seen with services like Netflix or Amazon, wherein ML algorithms analyze the user’s activity and compare it with that of other users to determine which shows or products to recommend. The algorithms are becoming intelligent with time—to the extent of understanding that a user may want to buy a product as a gift and not for him/her, or that different family members have different watching preferences.

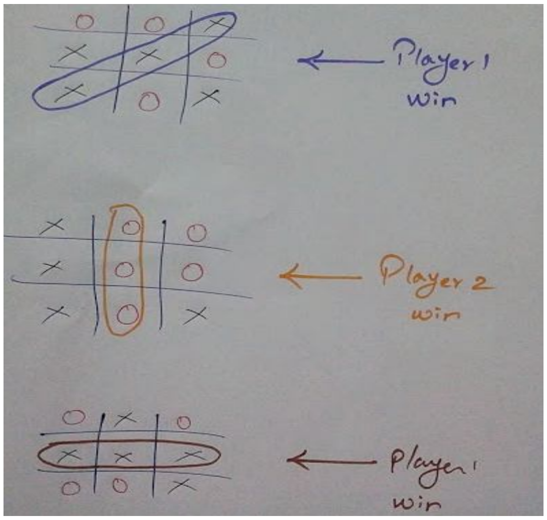
- **Finance:**
  - Financial services companies use AI-based natural language processing tools to analyze brand sentiment from social media platforms and provide actionable advice.
  - Investment companies like Aidya and Nomura Securities use AI algorithms to conduct trading autonomously and robo-traders to conduct high-frequency trading for greater profits, respectively.
  - Fintech firms like Kensho and ForwardLane use AI-powered B2C robo-advisors to augment rebalancing decisions and portfolio management performed by human analysts. Wealthfront uses AI algorithms to track account activity and help financial advisors customize their advice.
  - Chatbots, powered by natural language processing, can serve banking customers quickly and efficiently by answering common queries and providing information promptly.
  - Fraud detection is an important application of AI in financial services. For example, Mastercard uses Decision Intelligence technology to analyze various data points to detect fraudulent transactions, improve real-time approval accuracy, and reduce false declines.
- **Data security:**
  - Cyber attacks are becoming a growing reality with the move to a digital world. There are also concerns about AI programs themselves turning against systems.
  - Automatic exploit generation (AEG) is a bot that can determine whether a software bug, which may cause security issues, is exploitable. If a vulnerability is found, the bot automatically secures it. AEG systems help develop automated signature generation algorithms that can predict the likelihood of cyberattacks.
  - PatternEx and MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) have developed an AI platform called AI2 which claims to predict cyber attacks better than existing systems. The platform uses Active Contextual Modeling, a continuous feedback loop between a human analyst and the AI system, to provide an attack detection rate that is better than ML-only solutions by a factor of 10.
  - Deep Instinct, an institutional intelligence company, says that malware code varies between 2%-10% in every iteration and

that its AI model is able to handle the variations and accurately predict which files are malware.

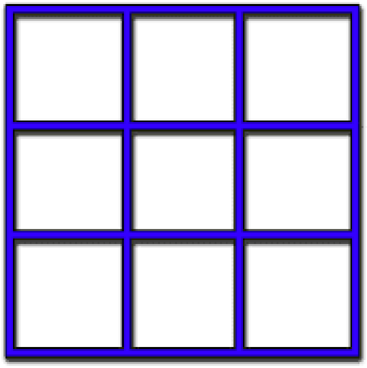
- **Manufacturing:**
    - Landing AI claims to have created machine-vision tools to find microscopic defects in objects like circuit boards using an ML algorithm trained using tiny volumes of sample images. In the future, self-driving robots may be created which can move finished goods around without endangering anyone or anything around.
    - Robots in factories are often stationary but are still in danger of crashing into objects around it. A new concept called collaborative robots or “cobots, enabled by AI, can take instructions from humans, including instructions that the robot has not been previously exposed to, and work productively with them.
    - AI algorithms can influence the manufacturing supply chain by detecting the patterns of demand for products across geographies, socioeconomic segments, and time, and predicting market demand. This, in turn, will affect inventory, raw material sourcing, financing decisions, human staffing, energy consumption, and maintenance of equipment.
    - AI tools help in predicting malfunctions and breakdown of equipment and taking or recommending preemptive actions as well as tracking operating conditions and performance of factory tooling.
- **Automotive industry**
    - Tesla introduced TeslaBot, an intelligent virtual assistant integrated with Tesla models S and X, allows users to interact with their car from their phone or desktop.
    - Uber AI Labs is working on developing self-driven cars with the help of the best engineers and scientists. Uber has already tested a batch of self-driving cars in 2016.
    - Nvidia has partnered with Volkswagen to develop “intelligent co-pilot systems” in cars that will enable safety warnings, gesture control, and voice and facial recognition.
    - Ericsson predicts that 5G technology will improve vehicle-to-vehicle communication wherein sensors will be implanted in airport runways, railways, and roads.

Tic Tac Toe Game playing

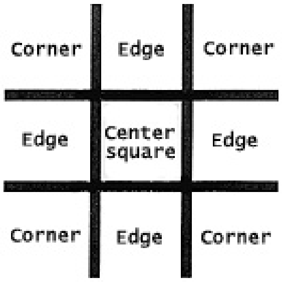
- The game Tic Tac Toe is also known as Noughts and Crosses or Xs and Os ,the player needs to take turns marking the spaces in a 3x3 grid with their own marks, if 3 consecutive marks (Horizontal, Vertical, Diagonal) are formed then the player who owns these moves get won.
  Assume ,
  Player 1 - X
  Player 2 – O



- **Board's Data Structure:**



- The cells could be represent as Center square, Corner,Edge as like below:



Corner	Edge	Corner
Edge	Center square	Edge
Corner	Edge	Corner

Unit- I
Assignment-Cum-Tutorial Questions

Objective Questions

1. Define Artificial Intelligence.
2. A system is said to be ____________ if it does the right thing, given known facts.
3. The study of how to make computers do things at which, at the moment, people are better-is a characteristic of_____. [ ]

   (a) Systems that think like humans (b) Systems that think rationally
   (c) Systems that act like humans (d) Systems that act rationally
4. The "Turing Test approach" is used to test whether a system ___.[ ]

   (a) think like humans (b) think rationally
   (c) act like humans (d) act rationally
5. The "Cognitive Modelling" approach is used to test whether a system ____.

   (a) think like humans (b) think rationally [ ]
   (c) act like humans (d) act rationally
6. The "Laws of Thought" approach is used to test whether a system ____.

   (a) think like humans (b) think rationally [ ]

   (c) act like humans (d) act rationally
7. For artificial intelligence to succeed, we need_____. [ ]

   (a) intelligence (b) artifact (c) both a & b (d) none
8. Understanding language requires an understanding of ______. [ ]

   (a) subject matter (b) context (c) structure of sentences (d) only a & b
9. GPS was probably the first program to embody ________ approach.

   (a) thinking humanly (b) acting humanly [ ]
   (c) thinking rationally (d) acting rationally
10. The _______program solves the problem of inferring molecular structure from the information provided by a mass spectrometer. [ ]

    (a) MYCIN (b) DENDRAL (c) both a & b (d) none

11. _________ diagnoses blood infections. [ ]

(a) MYCIN (b) DENDRAL (c) both a & b (d) none

12. ___________teaches Air Force technicians to diagnose electrical systems problems in aircraft. [ ]

(a) SHERLOCK (b) DARPA (c) DENDRAL (d) none

13. DARPA stands for____________.

SECTION-B

SUBJECTIVE QUESTIONS

1. Summarize AI definition categories?
2. Illustrate the capabilities that a computer must possess to pass Turing Test?
3. Explain the areas from which Artificial Intelligence laid its foundation?
4. Explain the history of Artificial Intelligence?
5. List the applications of Artificial Intelligence?
6. Outline the current trends in Artificial Intelligence?
7. What is an Intelligent System? Explain its characteristics?
8. Interpret the steps to solve tic-tac-toe problem.

UNIT - II

Problem solving and Game playing

Syllabus:

Problem solving: state-space search and control strategies: Introduction, general problem solving, characteristics of problem, exhaustive searches, heuristic search techniques, iterative-deepening a*, problem reduction, constraint satisfaction.

Game playing: Introduction, game playing, alpha-beta pruning, two-player perfect information games.

Outcomes:

Student will be able to:

General Problem Solving

- To build a system to solve a particular problem, we need to do four things:

  1. **Define the problem precisely**. This definition must include precise specifications of what the initial situation(s) will be as well as what final situations constitute acceptable solutions to the problem.

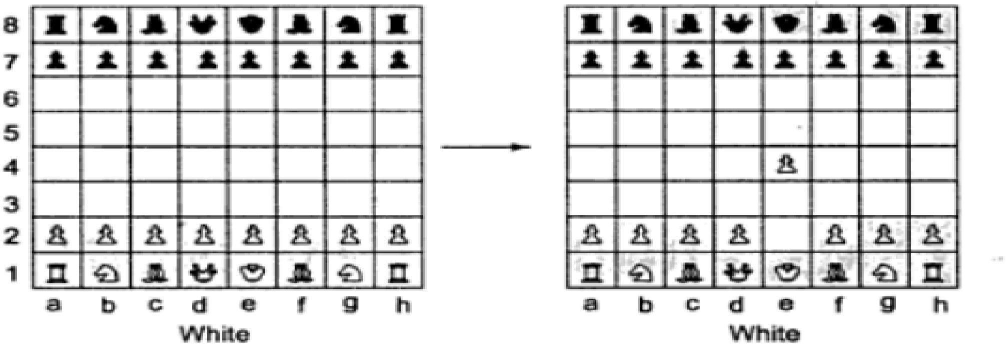
  2 **Analyse the problem**. A few very important features can have an immense impact on the appropriateness of various possible technique(s)for solving the problem.

  3. **Isolate and represent** the task knowledge that is necessary to solve the problem.

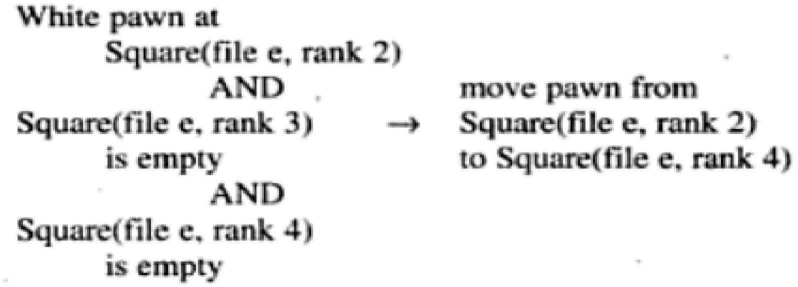
  4. **Choose the best problem-solving technique**(s) and apply it (them) to the particular problem.

State-Space Search

- To build a program that could "Play chess," we would first have to specify the starting position of the chess hoard, the rules that define the legal moves, and the board positions that represent a win for one side or the other.
- The starting position can he described as an 8-by-8 array where each position contains a symbol standing for the appropriate piece in the official chess opening position.
- We can define as our goal any board position in which the opponent does not have a legal move and his or her king is under attack.
- The legal moves provide the way of getting from the initial state to a goal state. They can he described easily as a set of rules consisting of two parts: a left side that serves as a pattern to he matched against the current board position and a right side that describes the change to be made to the board position in reflect the move.



- We have to write a very large number of rules since there have to be a separate rule for each of the 10^{120} possible board positions. Using so many rules poses two serious practical difficulties:
  - No person could ever supply a complete set of such rules. It would take too long and could certainly not be done without mistakes.
  - No program could easily handle all those rules. Although hashing scheme could be used to find the relevant rules for each move fairly quickly, just storing that many rules poses serious difficulties.
- We should write the rules describing the legal moves in as general a way possible. The convenient notation for describing patterns and substitutions is as follows:



```
White pawn at
        Square(file e, rank 2)
                 AND                 move pawn from
Square(file e, rank 3)        →      Square(file e, rank 2)
        is empty                     to Square(file e, rank 4)
                 AND
Square(file e, rank 4)
        is empty
```

- A problem can be defined in a **"State Space",** where each state corresponds to a legal position of the board.
  - We can start at an initial state using a set of rules to move from one state to another, and attempting to end up in one of a set of final states.
- The state space representation forms the basis of most of the AI methods. Its structure corresponds to the structure of problem solving in two important ways:
  - It allows formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.

- It permits us to define the process of solving a particular problem as a combination of known techniques) each represented as a role defining a single step in the space) and search, the general technique of exploring the space to try to find some path from the current state to a goal state. Search is a very important process in the solution of hard problems for which no more direct techniques are available.

Water Jug Problem

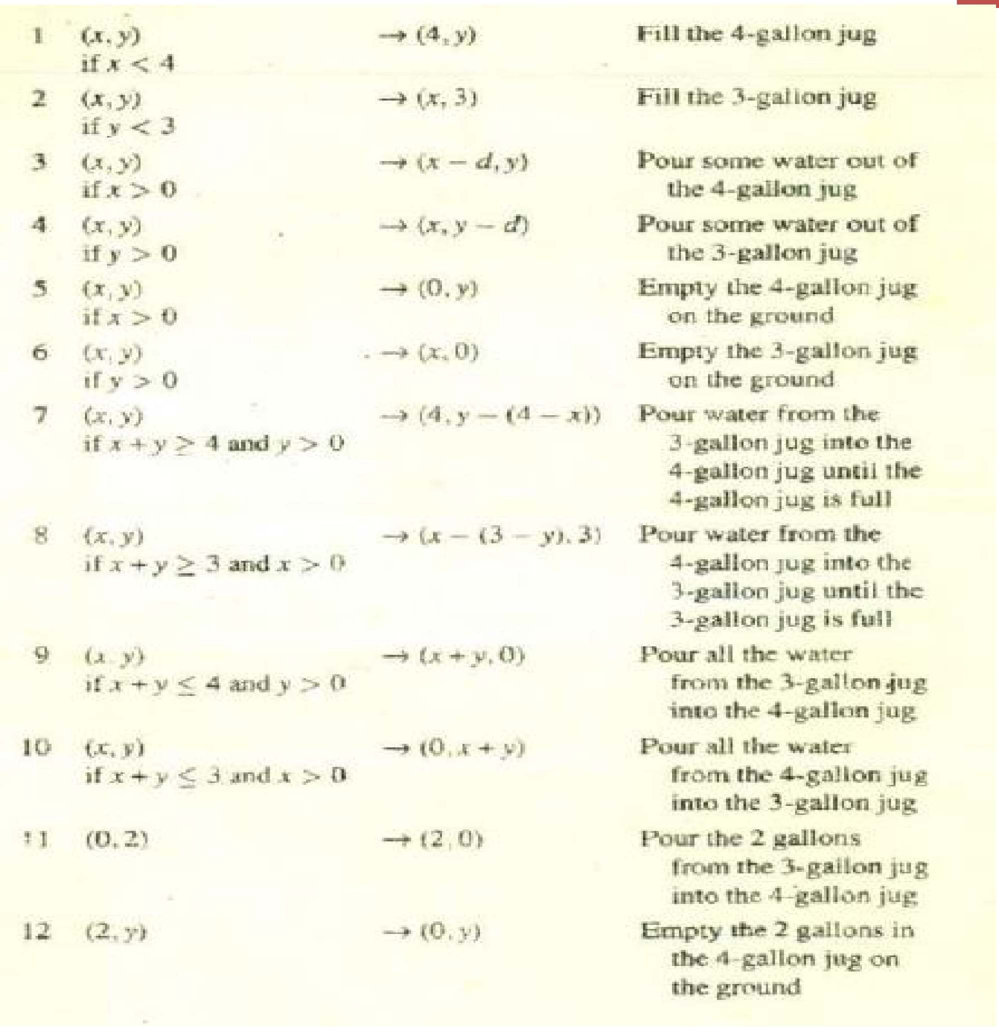
- Consider the following problem: A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?
- **State Representation and Initial State** – we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note $0 \le x \le 4$, and $0 \le y \le 3$.
- Our Initial state: (0,0) Goal state = (2,y) where $0 \le y \le 3$.
- To solve this we have to make some assumptions not mentioned in the problem. They are:

  1. We can fill a jug from the pump.

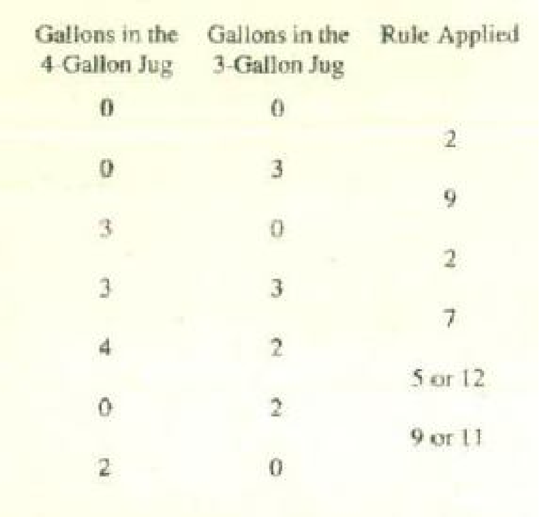
  2. We can pour water out of a jug to the ground.

  3. We can pour water from one jug to another.

  4. There is no measuring device available. To solve the water jug problem, all we need is a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen. The appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see ii ii corresponds to a goal state. As long as it does not, the cycle continues. Clearly the speed with which the problem gets solved depends on the mechanism that is used to select the next operation to be performed.
- We must define a set of operators that will take us from one state to another:



1	(x, y) if $x < 4$	$\rightarrow (4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	$\rightarrow (x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	$\rightarrow (x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow (x, y - d)$	Pour some water out of the 3-gallon jug
5	(x, y) if $x > 0$	$\rightarrow (0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	$\rightarrow (x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow (4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow (x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow (x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow (0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	$\rightarrow (2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	$\rightarrow (0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground



Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

- The first step toward the design of a program to solve a problem must be the creation of a formal and manipulable description of the problem itself. We should be able to write programs that can themselves produce such formal descriptions from informal ones. This process is called **Operationalization.**
- In order to provide a formal description of a problem, we must do the following:
    - **Define a state space** that contains all the possible configurations of the relevant objects (and perhaps some impossible ones). It is, of course, possible to define this space without explicitly enumerating all of the states it contains.
    - Specify one or more states within that space that describes possible situations from which the problem-solving process may start. These states are called the **Initial states.**
    - Specify one or more states that would be acceptable as solutions to the problem. These states are called **goal states.**
    - Specify a **set of rules** that describe the actions (operators) available.

- 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. Thus the process of search is fundamental to the problem-solving process.

Control Strategies

- It is important to decide which rule to apply next during the process of searching for a solution to a problem. This question arises when more than one will have its left side match the current state.
- The decisions made will have a crucial impact on how quickly a problem is finally solved.
- The following are the requirements of a control strategy:
    - The first requirement of a good control strategy is that **it causes motion**. Consider again the water jug problem. Suppose we implemented the simple control strategy of starting each time at the top of the list of rules and choosing the first applicable one. If we did that, we would never solve the problem. We would continue indefinitely filling the 4-gallon jug with water. Control strategies that do not cause motion will never lead to a solution.
    - The second requirement of a good control strategy is that it **be systematic**. Let us consider a simple control strategy for the water jug problem. On each cycle, choose at random from among the applicable rules. This strategy is better than the first. It causes motion. It will lead to a solution eventually. Sometimes

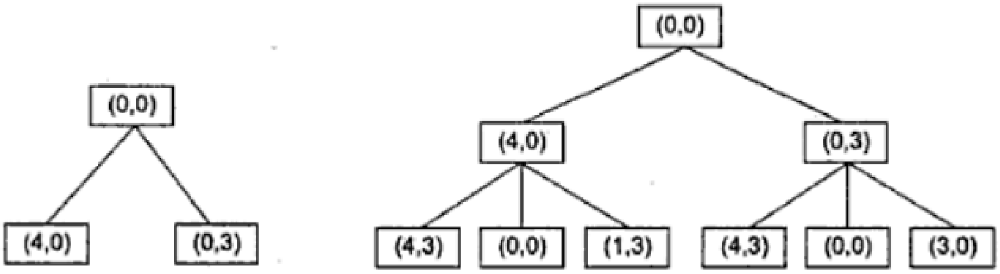
we arrive at the **same state several times** during the process and to use many more steps than are necessary. Because the control strategy is not systematic, we may explore a particular useless sequence of operators several times before we finally find a solution. The requirement that a control strategy be systematic corresponds to the need for **global motion (over the course of several steps) as well as for local motion** (over the course of a single step).

Breadth First Search

- For each leaf node, generate all its successors by applying all the rules that are appropriate. Continue this process until some rule produces a goal state. This process, called **Breadth-First search**.

Algorithm: Breadth-First Search

```
1. Create a variable called NODE-LIST and set it to the initial state.
2. Until a goal state is found or NODE-LIST is empty do:
    (a) Remove the first element from NODE-LIST and call it E. If
    NODE-LIST          was empty, quit.
    (b) For each way that each rule can match the state described in
    E do:
        i. Apply the rule to generate a new state.
        ii. If the new state is a goal state, quit and return this
        state.
        iii. Otherwise, add the new state to the end of' NODE-
        LIST.
```



- **Advantages of Breadth-First Search:**
  - Breadth-first search will not get trapped exploring a blind alley.
  - If there is a solution, then breadth-first search is guaranteed to find it. Furthermore, if there are multiple solutions, then a minimal solution will be found. This is guaranteed by the fact that longer paths are never explored until all shorter ones have already been examined.

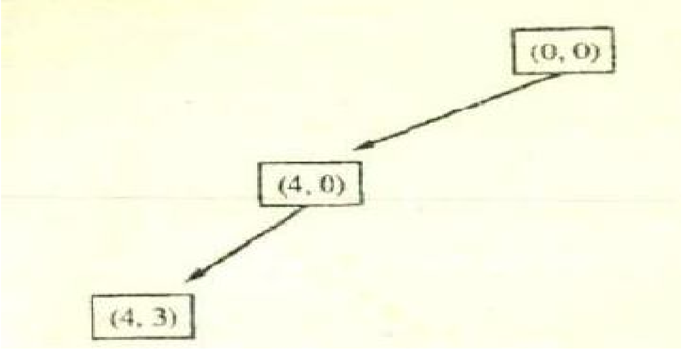
- **Disadvantages of Breadth-First Search:**
  - All of the tree that has so far been generated must be stored which requires more memory.
  - In breadth-first search, all parts of the tree must be examined to level *n* before any nodes on level n + 1 can be examined. This is particularly significant if many acceptable solutions exist.

Depth First Search

- We could pursue a single branch of the tree until it yields a solution or until a decision to terminate the path is made.
- It makes sense to terminate a path if it reaches a dead-end, produces a previous state, or becomes longer than some prespecified "futility" limit.
- In such a case backtracking occurs. The most recently created state from which alternative moves are available will be revisited and a new state will he created. This form of backtracking is called **Chronological Backtracking** because the order in which steps arc undone depends only on the temporal sequence in which the steps were originally made.
- The search procedure we have just described is called **Depth-First search.**

Algorithm: Depth-First Search

```
1. If the initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signalled:
        (a) Generate a successor, E, of the initial state. It there are no
        more successors,
              signal failure.
        (b) Call Depth-First Search with E as the initial state.
         (c) If success is returned, signal success. Otherwise continue in
this loop.
```



- **Advantages of Depth-First Search:**
  - Depth-first search requires less memory since only the nodes on the current path are stored.
  - By chance (or if care is taken in ordering the alternative successor states), depth first search may find a solution without examining much of the search space at all. Depth-first search can stop when one of solution is found.
- **Disadvantages of Depth-First Search:**
  - Depth- first searching may follow a single unfruitful path for a very long time.
  - Depth-first search, may find a long path to a solution in one part of the tree, when a shorter path exists in some other unexplored part of the tree.

Problem Characteristics

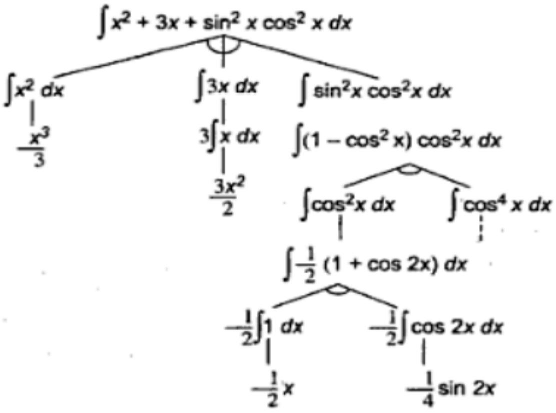
- *A* **heuristic** *is* a technique that improves the efficiency of a search process, possibly by sacrificing claims such as completeness. Heuristics are like tour guides. They are good to the extent that they point in generally interesting directions: they are bad to the extent that they may miss points of interest to particular individuals. But, on the average, they improve the quality of the paths that are explored.
- Heuristic search is a very general method applicable to a large class of problems; it encompasses a variety of specific techniques, each of which is particularly effective for a small class of problems. In order to choose the most appropriate method (or combination of methods for a particular problem, it is necessary to analyze the problem along several key dimensions:
  - Is the problem decomposable into a set of (nearly) independent smaller or easier sub problems?
  - Can solution steps be ignored or at least undone if they prove unwise?
  - Is the problem's universe predictable?
  - Is good solution to the problem obvious without comparison to all other possible solutions?
  - Is the desired solution a state of the world or a path to a state?
  - Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
  - Can a computer that is simple given the problem return the solution, or will the solution of the problem require interaction n between the computer and a person?

1. Is the Problem Decomposab1e?

- Suppose we want to solve the problem of computing the expression:

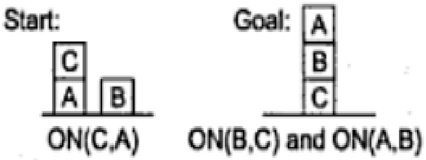
$$\int (x^2 + 3x + \sin^{2*} \cdot \cos^2 x)\, dx$$

- We can solve this problem by breaking it down into three smaller problems each of which we can then solve by using a small collection of specific rules.
- The figure shows the problem tree that will be generated by the process of problem decomposition as it can be exploited by a simple recursive integration program that works as follows:
- At each step. it checks to see whether the problem it is working on is immediately solvable. If so, then the answer is returned directly.
- If the problem is not easily solvable, the integrator checks to see whether it can decompose the problem into smaller problems. If it can, it creates those problems and calls itself recursively on them. Using this technique of problem decomposition we can often solve very large problems easily.

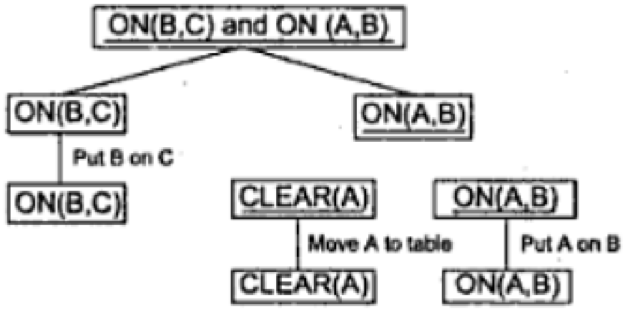


- **Example-Blocks World Problem:** Assume that the following operators are available-

1. CLEAR (x) [block x has nothing on it] → ON (x, Table) [pick up x and put it on the table]
2. CLEAR (x) and CLEAR (y) → ON (x, y) [put x on y]



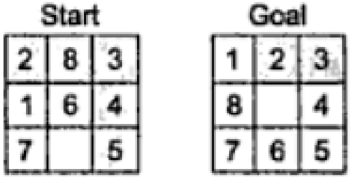
- Applying the technique of problem decomposition to this simple blocks world example would lead to a solution tree such as that shown below:



- In the figure, goals are underlined. States that have been achieved are not underlined. The idea of this solution is to reduce the problem of getting B on C and A on B to two separate problems.
- The **first** of these new problems, getting B on C. is simple, given the start state. Simply put B on C. The second sub goal is not quite so simple. Since the only operators we have allow us to pick up single blocks at a time, we have to clear off B by removing C before we can pick up A and put it on B This can easily be done.
- However, if we now try to combine the two sub solutions into one solution, we will fail. Regardless of which one we do first, we will not be able to do the second as we had planned.
- In this problem, the two sub problems are not independent. They interact and those interactions must be considered in order to arrive at a solution for the entire problem.

2. Can Solution Steps Be Ignored or Undone?

- **Theorem Proving:** We proceed by first proving lemma that we think will be useful. Eventually, we realize that the lemma is no help at all.
- Everything we need to prove the theorem is still true. Any rules that could have been applied can still be applied.
- We can just proceed as we should have in the first place. All we have lost is the effort that was spent exploring the blind alley.
- **The 8-Puzzle**: The 8-puzzle is a square tray in which are placed eight square tiles. The remaining ninth square is uncovered. Each tile has a number on it. A tile that is adjacent to the blank space can he slid into that space. A game consists of a starting position and a specified goal position. The goal is to transform the starting position into a goal position by sliding the tiles around.

- 

Start

2	8	3
1	6	4
7		5

Goal

1	2	3
8		4
7	6	5

- In attempting to solve the 8-puzzle, we might make a stupid move. We might start by sliding tile 5 into the empty space. Having done that, we cannot change our mind and immediately slide tile 6 into the empty space since the empty space will essentially have moved.
- But we can backtrack and undo the first move, sliding tile 5 back to where it was. Then we can move tile 6. Mistakes can still be recovered from but not quite easily. An additional step must be performed to undo each incorrect step,
- The control mechanism for an 8.puzzle solver must keep track of the order in which operations are performed so that the operations can be undone one at a time if necessary.
- **Chess:** Now consider again the problem of playing chess. Suppose a chess-playing program makes a stupid move and realizes it a couple of moves later. It cannot simply play as though it had never made the stupid move. Nor can it simply back up and start the game over from that point. All it can do is to try to make the best of the current situation and go on from there.
- The three problems—theorem proving, the 8-puzzle, and chess—illustrate the differences between three important classes of problems:

  • **Ignorable**(e.g... theorem proving), in which solution steps can be ignored

  • **Recoverable**(e.g... 8-puzzle), in which solution steps can be undone.

  • **Irrecoverable**(e.g... chess), in which solution steps cannot be undone
- The **recoverability of a problem** plays an important role in determining the complexity of the control structure necessary for the problem's solution.
  - **Ignorable problems** can be solved using a simple control structure that never backtracks. Such a control structure is easy to implement.
  - **Recoverable problems** can be solved by a slightly more complicated control strategy that does sometimes make mistakes. Backtracking will be necessary to recover from such mistakes, so the control structure must be implemented using a push-down stack, in which decisions are recorded in case they need to be undone later.

- **Irrecoverable problems**, on the other hand, will need to be solved by a system that expends a great deal of effort making each decision since the decision must be final

3. Is the Universe Predictable?

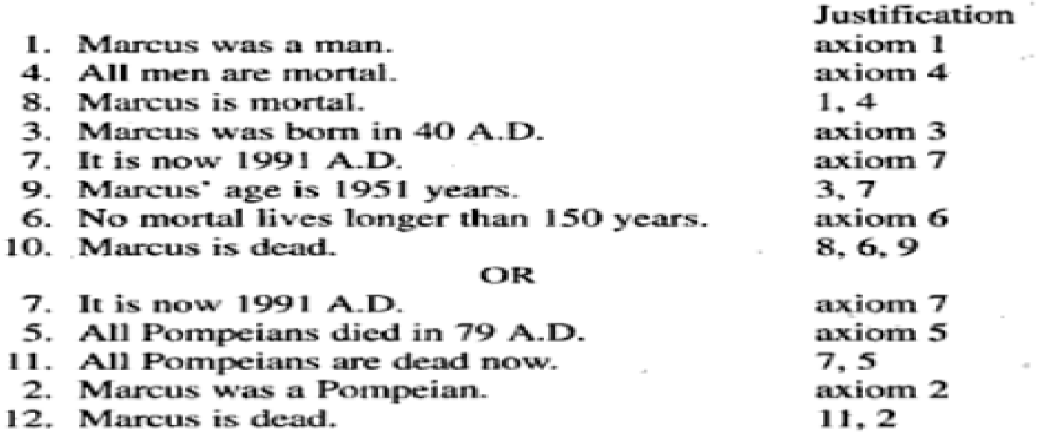
- **8-puzzle:** Every time we make a move, we know exactly what will happen. This means that it is possible to plan an entire sequence of moves and be confident that we know what the resulting state will be. We can use planning to avoid having to undo actual moves, although it still be necessary to backtrack past those moves one at a time during the planning process. Thus a control structure that allows backtracking will be necessary.
- **Play Bridge:** One of the decisions we will have to make is which card to play on the first trick. What we would like to do is to plan the entire hand before making that first play. But now it is not possible to do such planning with certainty since we cannot know exactly where all the cards are what the other players will do on their turns. The best we can do is to investigate several plans and use probabilities of the various outcomes to choose a plan that has the highest estimated probability of leading to a good score on the hand.
- **Certain–outcome problems** (e.g... 8-puzzle): The open-loop approach will work fine since the result of an action can he predicted perfectly. Thus, planning can be used to generate a sequence of operators that is guaranteed to lead to a solution.
- **Uncertain–outcome problems** (e.g... Bridge): Planning can at best generate a sequence of operators that has a good probability of leading to a solution. To solve such problems, we need to allow for a process of plan revision to take place as the plan is carried out and the necessary feedback is provided. In addition to providing no guarantee of art actual solution, planning for uncertain -outcome problems has the drawback that it is often very expensive since the number of solution paths that need to be explored increases exponentially with the number of points at which the outcome cannot be predicted.

4. Is a Good Solution Absolute or Relative?

- **Simple facts problem:** Consider the problem of answering questions based on a database of simple facts, such as the following:

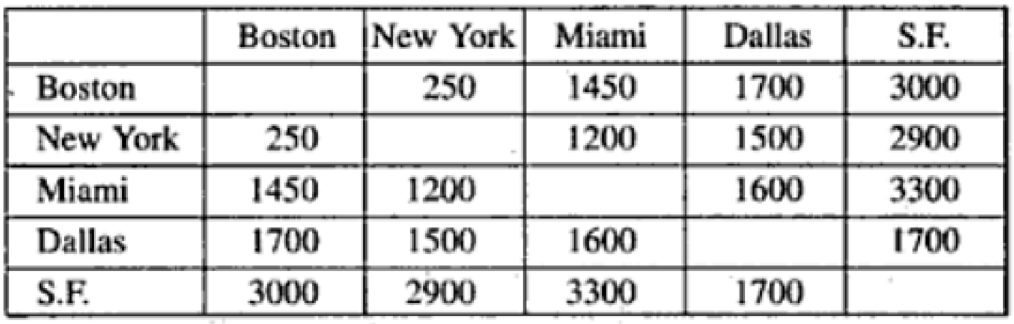
  1. Marcus was a man.
  2. Marcus was a Pompeian.
  3. Marcus was born in 40 A.D.
  4. All men are mortal.
  5. All Pompeians died when the volcano erupted in 79 A.D.
  6 No mortal lives longer than 150 years.
  7.11 is now 1991 A.D.

- Suppose we ask the question **'Is Marcus alive?'.** By representing each of these facts in a formal language such as predicate logic, and then using formal inference methods we can fairly easily derive an answer to the question. Since all we are interested in is the answer to the que7sion, it does not matter which path we follow. If we do follow one path successfully to the answer, there is no reason to go back and see it some other path might also lead to a solution.



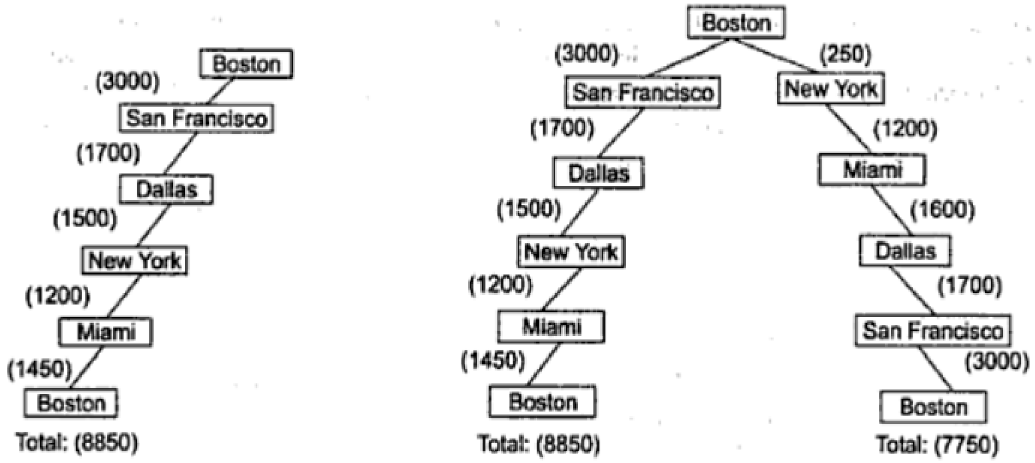
		Justification
1.	Marcus was a man.	axiom 1
4.	All men are mortal.	axiom 4
8.	Marcus is mortal.	1, 4
3.	Marcus was born in 40 A.D.	axiom 3
7.	It is now 1991 A.D.	axiom 7
9.	Marcus' age is 1951 years.	3, 7
6.	No mortal lives longer than 150 years.	axiom 6
10.	Marcus is dead.	8, 6, 9
	OR	
7.	It is now 1991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

- **Travelling salesman problem:** Our goal is to find the shortest route that visits each city exactly once. Suppose the cities to be visited and the distances between them are as shown:



	Boston	New York	Miami	Dallas	S.F.
Boston		250	1450	1700	3000
New York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
S.F.	3000	2900	3300	1700	

- One place the salesman could start is Boston. In that case, one path that might be followed is shown, which is 8850 miles long. But is this the solution to the problem? The answer is that we cannot be sure unless we also try all other paths to make sure that none of them is shorter. In this case, the first path is definitely not the solution to the salesman's problem.



- **Best-path problems**: These are, in general, computationally harder than Any-path problems. No heuristic that could possibly miss the best solution can be used.
- **Any-path problems**: can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore. If the heuristics are not perfect, the search for a solution may not be as direct as possible. So a much more exhaustive search will be performed.

5. Is the Solution a State or a Path?

- **Consistent interpretation for the sentence (NLU):** There are several components of this sentence, each of which, in isolation, may have more than one interpretation.

  “The Bank president ate a dish of pasta salad with the fork”
- Some of the sources of ambiguity in this sentence are the following:
  - The word "bank" may refer either to a financial institution or to a side of a river. But only one of these may have a president.
  - The word "dish" is the object of the verb "eat." It is possible that a dish was eaten. But it is more likely that the pasta salad in the dish was eaten.
  - Pasta salad is a salad containing pasta. But there are other ways meanings can be formed from pair of nouns. For example, dog food does not normally contain dog.
- Some search may be required to find a complete interpretation for the sentence. But to solve the problem of finding the interpretation we need to produce only the interpretation itself. No record of the processing by which the interpretation was found is necessary.
- **Water Jug Problem:** It is not sufficient to report that we have solved the problem and that the final state (2, 0). For this kind of problem,

what we really must report is not the final state but the path that we found to that state. Thus a statement of a solution to this problem must be a sequence of operations called a **"plan"** that produces the final state.

- The difference between these problems is:
  - Problems whose solution is a **state.**
  - Problems whose solution is a **path to a state.**
- In water jug problem, we must re-describe the states so that each state represents a partial path to a solution rather than just a single state of the world.

6. What Is the Role of Knowledge?

- **Playing chess**: Suppose you had unlimited computing power available. The knowledge that would be required by a perfect program is very little—just the rules for determining legal moves and some simple control mechanism that implements a search procedure.
- Additional knowledge about such things as good strategy, and tactics could of course help considerably to constrain the search and speed up the execution of the program.
- **Scanning daily newspapers**: to decide which are supporting the Democrats and which are supporting the Republicans in some upcoming election. Again assuming unlimited computing power, how much knowledge would be required by a computer trying to solve this problem? It would have to know such things as:

  • The names of the candidates in each party.

  • The fact that if the major thing you want to see done is have taxes lowered, you are probably supporting the Republicans.

  • The fact that it the major thing you want to see done is improved education for minority students, you are probably supporting the Democrats.

  • The fact that if you are opposed to big government you are probably supporting the Republicans.
- These two problems, chess and newspaper story understanding, illustrate the difference between problems for which a lot of knowledge is important only to constrain the search for a solution and those for which a lot of knowledge is required even to be able to recognize a solution.

7. Does the Task Require Interaction with a Person?

- Sometimes it is useful to program computers to solve problems in ways that the majority of people would not be able to understand. This is fine if the level of the interaction between the computer and its human users is **program-in solution-out**.

- But increasingly we are building programs that require intermediate interaction with people, both to provide additional input to the program and to provide additional reassurance to the user.
- We must distinguish between two types of problems:
- **Solitary:** in which the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation of the reasoning process
- **Conversational**: in which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both.

8. Problem Classification

- **Generic control strategy-Classification:** The task here is to examine an input and then decide which of a set of known classes the input is an instance of. Most diagnostic tasks, including medical diagnosis as well as diagnosis of faults in mechanical devices are examples of classification.
- **Propose and Refine**: Many design and planning problems can be attacked with this strategy.

Heuristic Search Techniques

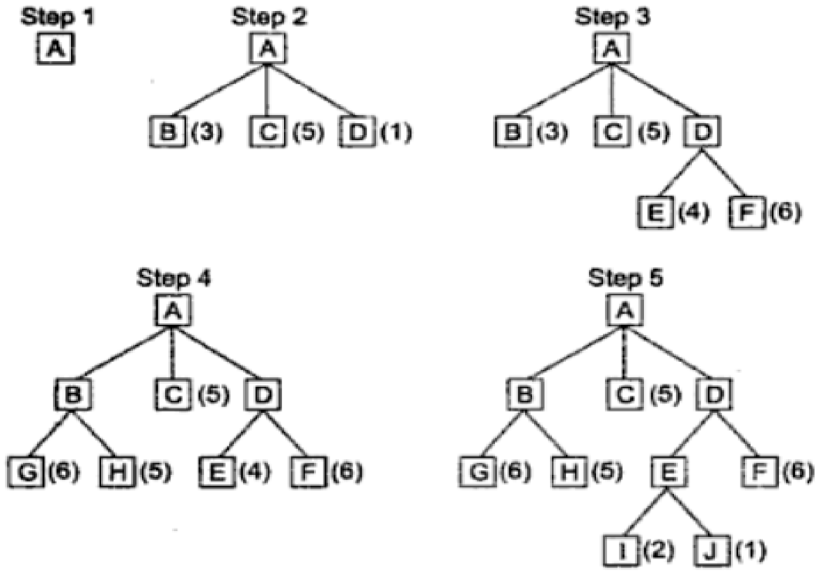
- A framework for describing search methods is provided and several general-purpose search techniques are discussed.
- There are many varieties of heuristic search. They can be described independently of any particular task or problem domain. But when applied to particular problems, their efficacy is highly dependent on the way they exploit domain-specific knowledge
- Over last three decades of AI research, these techniques continue to provide the framework into which domain-specific knowledge can be placed, either by hand or as a result of automatic learning. Thus they continue to form the core of most AI systems.

Best-First Search

- Best-first search, is a way of combining the advantages of both depth-first and breadth-first search into a single method.
- Depth-first search is good because it allows a solution to be found without all competing branches having to be expanded. Breadth-first search is good because it does not get trapped on dead-end paths.
- One way of combining the two is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does.
- At each step of the best-first search process, we select the most promising of the nodes we have generated so far. This is done by

applying an appropriate heuristic function to each of them, We then expand the chosen node by using the rules to generate its successors.

- If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far. Again the most promising node is selected and the process continues. A bit of depth first searching occurs as the most promising branch is explored.
- But eventually, it a solution is not found, that branch will start to look less promising than one of the top-level branches that had been ignored. At that point, the now more promising, previously ignored branch will be explored. But the old branch is not forgotten, Its last node remains in the set of generated but unexpanded nodes. The search can return to it whenever all the others get bad enough, that it is again the most promising path.
- Initially, there is only one node, so it will be expanded. Doing so generates three new nodes. The heuristic function, is an estimate of the cost of getting to a solution from a given node, is applied to each of these new nodes.
- Since node D is the most promising, it is expanded next, producing two successor nodes, E and F. Now the heuristic function is applied to them.
- Now another path that going through node B looks more promising, so it is pursued, generating nodes G and H. But again when these new nodes are evaluated they look less promising than another path, so attention is returned to the path through D to E. E is then expanded, yielding nodes I and J.
- At the next step, J will be expanded, since it is the most promising. This process can continue until a solution is found.



- In best-first search, one move is selected, but the others are kept around so that they can be revisited later if the selected path becomes less promising.
- Further, the best available state is selected in best first search, even it that state has a value that is lower than the value of the state that was just explored.
- In a best-first search of a tree, it is sometimes important to search a graph instead so that duplicate paths will not be pursued.
- An algorithm to do this will operate by searching a directed graph in which each node represents a point in the problem space. Each node will contain, in addition to a description of the problem state it represents, an indication of how promising it is; a parent link that points back to the best node from which it came, and a list of the nodes that were generated from it.
- The parent link will make it possible to recover the path to the goal once the goal is found. The list of successors will make it possible, if a better path is found to an already existing node, to propagate the improvement down to its successors.
- We will call a graph of this sort an **OR graph**, since each of its branches represents an alternative problem-solving path.
- To implement such a graph-search procedure, we will need to use two lists of nodes:
  - **OPEN**—nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined (i.e.. had their successors generated) OPEN is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.
  - **CLOSED**— -nodes that have already been examined. We need to keep these nodes in memory if we want to search a graph rather than a tree, since whenever a new node is generated; we need to check whether it has been generated before.
- We will also need a heuristic function that estimates the merits of each node we generate. The function f^1 that gives the true evaluation of the node.
- We define this function as the sum of two components: **g and $h^{1.}$**
- The **function g** is a measure of the cost of getting from the initial state to the current node.
- The **function h^1** is an estimate of the additional cost of getting from the current node to a goal state.
- The combined **function f^1** represents an estimate of the cost of getting from the initial state to a goal state along the path that generated the

current node. If more than one path is generated the node, then the algorithm will record the best one.

- This process can be summarized as follows:

  Algorithm: Best-First Search

  ```
  1. Start with OPEN containing just the initial state.
  2. Until a goal is found or there are no nodes left on OPEN do:
  (a) Pick the best node on OPEN.
  (b) Generate its successors.
  (c) For each successor do:
          i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.
          ii. If it has been generated before, change the parent if this new path is, better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.
  ```

- The best first search algorithm is a simplification of an algorithm called **A*.**
- This algorithm uses the same f^1, g, and h^1 functions, as well as the lists OPEN and CLOSED.

Iterative Deepening:

- A number of ideas for searching two-player game trees have led to new algorithm for single-agent heuristic search. One such idea is **“iterative deepening”** originally used in **CHESS**.
- Rather than searching to a fixed depth in the game tree, CHESS first searched only a single ply, applying its static evaluation function to the result of each of its possible moves.
- It then initiated a new minimax search, to a depth of two ply. This was followed by a three-ply search, then a four-ply search etc.
- The name "iterative deepening" derives from the fact that on each iteration, the tree is searched one level deeper.
- Since it is impossible to know in advance how long a fixed-depth tree search will take, a program may find itself running out of time. With iterative deepening, the current search can be aborted at any time and the best move found by the previous iteration can be played.
- Iterative deepening can also be used in improve the performance of the A* search algorithm.

Algorithm: Iterative-Deepening-A*

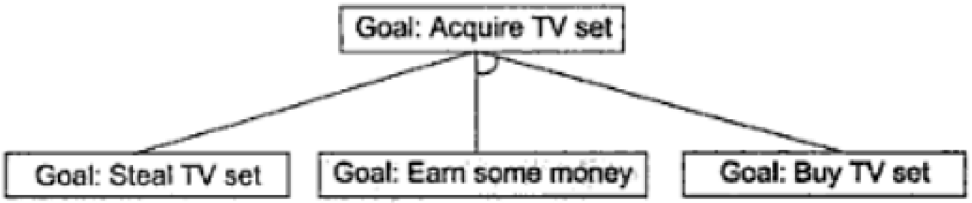
```
1. Set THRESHOLD = the heuristic evaluation of the start state.
2. Conduct a depth-first search, pruning any branch when its total cost function (g + h1) exceeds THRFSHOLD. If a solution path is found during the search, return it.
```

3. Otherwise, increment THRESHOLD by the minimum amount it was exceeded during the previous step, and then go to Step 2.

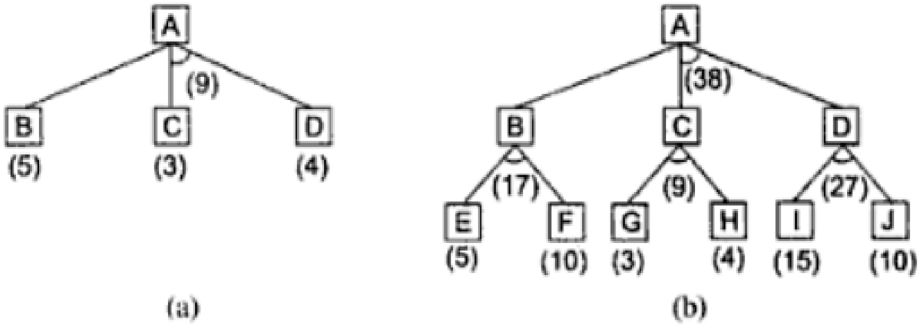
- Iterative-Deepening-A* (IDA*) is guaranteed to find an optimal solution provided that h^1 is an admissible heuristic.

Problem Reduction: AND-OR Graphs

- **AND-OR graph** (or tree), is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then he solved.
- This decomposition, or reduction, generates arcs that we call AND arcs. One AND arc may point to any number of successor nodes all of which must be solved in order for the arc to point to a solution.
- Several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved. This is why the structure is called not simply an AND graph but rather an AND-OR graph. AND arcs are indicated with a line connecting all the components.



- In order to find solutions in an AND-OR graph, we need an algorithm with the ability to handle the AND arcs appropriately. This algorithm should find a path from the starting node of the graph to a set of nodes representing solution states.
- The top node, A, has been expanded, producing two arcs, one leading to B and one leading to C and D. The numbers at each node represent the value of **f^1** at that node. Assume, that every operation has a uniform cost, so each arc with a single successor has a cost of I and each AND arc with multiple successors has a cost of 1 for each of its components.
- If we look just at the nodes and choose for expansion the one with the lowest; **f^1** value, we must select C.
- But using the information now available, it would be better to explore the path going through B since to use C we must also use D. (for a total cost of 9 (C+D+2) compared to the cost of 6 that we get by going through B.
- The problem is that the choice of which node to expand next must depend not only on the **f^1** value of that node but also on whether that node is part of the current best path from the initial node.



- The most promising single node is (G with an **f¹** value of 3). It is even part of the most promising arc G-H, with a total cost of 9. But that are is not part of the current best path since to use it we must also use the arc I-J with a cost of 27.
- The path from A through B, to E and F is better, with a total cost of 18. So we should not expand G next: rather we should examine either E or F.
- In order to describe an algorithm for searching an AND-OR graph, we need to exploit a value called **FULILITY**. If the estimated cost of a solution becomes greater than the value of FUTILITY, then we abandon the search.
- FUTILITY should be chosen to correspond to a threshold such that any solution with a cost above it is too expensive to be practical, even if it is found.

Algorithm: Problem Reduction

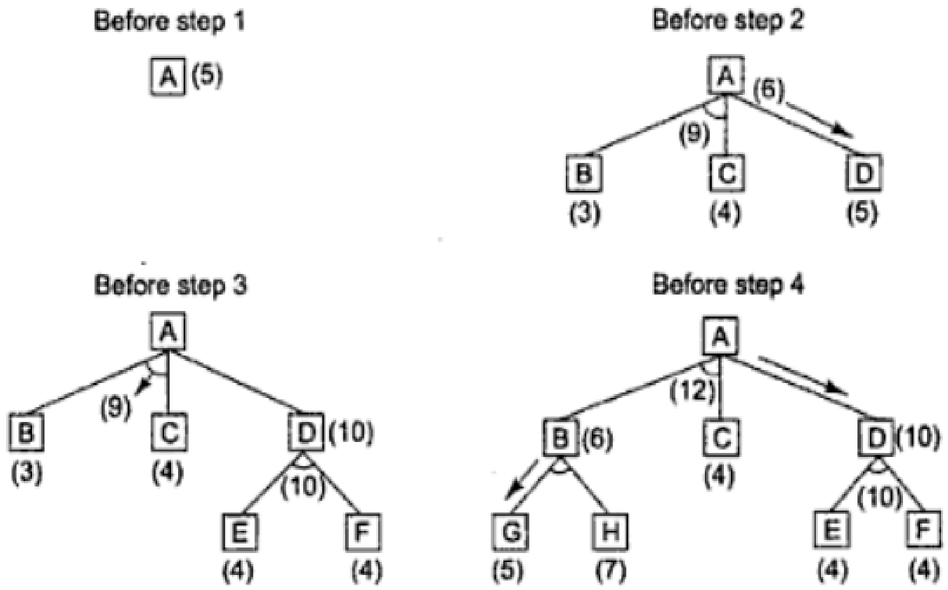
1 Initialize the graph to the starting node.

2. Loop until the starling node is labelled SOLVED or until its cost goes above FUTILITY:

(a) Traverse the graph, starting at the initial node and following the current best path, and accumulate the set of nodes that are or that path and not yet been expanded or labelled as solved.

b) Pick one of these unexpanded nodes and expand it. If there are no successors, assign FUTILITY as the value of this node, otherwise, add its successors to the graph and for each of them compute **f¹** .If **f¹** of any node is 0, mark that node as SOLVED.

(c) Change the **f¹** estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backward through the graph. If any node contains a successor arc whose descendants are all solved, label the node itself as SOLVED. At each node that is visited while going up the graph, decide which of its successor arcs is the most promising and mark it as part of the current best path. But now expanded nodes must be re-examined so that the best current path can he selected. Thus it is important that their **f¹** values be the best estimates available.



- At step 1, A is the only node, so it is at the end of the current best path. It is expanded, yielding nodes B, C and D. The arc to D is labelled as the most promising one emerging from A, since it costs *6* compared to B and C, which costs 9. (Marked arcs are indicated in the figures by arrows.)
- In step 2, node D is chosen for expansion. This process produces one new arc, the AND arc to E and F, with a combined cost estimate of 10. So we update the f¹ value of D to 10.
- Going back one more level, we can see that this makes the AND arc **B-C** better than the arc to D, so it is labelled as the current best path.
- At step 3, we traverse that arc from A and discover the unexpanded nodes B and C. If we want to find a solution along this path, we will have to expand both B and *C* eventually, so let's choose to explore B first. This generates two new arcs, the ones to G and to H.
- Propagating their f¹ values backward, we update f¹ of B to 6. This requires updating the cost of the AND arc B-C to 12 (6+4+2).

- After doing that, the arc to D is again the better path from A, so we record that as the current best path and either node E or node F will be chosen for expansion at step 4.
- This process continues until either a solution is found or all paths have led to dead ends, indicating that there is no solution.

The AO* Algorithm

- The problem reduction algorithm is a simplification of an AO* algorithm.
- The AO* algorithm will use a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far.
- Each node in the graph will point both down to its immediate successors and up to its immediate predecessors. Each node in the graph will also have associated with it an **h^1 value, an estimate of the cost of a path from itself to a set of solution nodes**. So h^1 will serve as the estimate of goodness of a node.

Algorithm: AO*

1. Let GRAPH consists of only the node representing the initial state. (Call this node INIT). Compute h^1 (INIT).

2. Until INIT is labelled SOLVED or until INIT's h^1 value becomes greater than FUTILITY, repeat the following procedure:

```
    (a) Trace the labelled arcs from INIT and select for expansion one of the yet unexpanded nodes that occurs on this path. Call the selected node NODE.
    (b) Generate the successors of NODE. If there are none, then assign FUTILITY as the $h^1$ value of NODE. 'This is equivalent to saying that NODE is not solvable. If there are successors, then for each one called SUCCESSOR that is not also an ancestor of NODE do the following:
        i. Add SUCCESSOR to GRAPH.
        ii. If SUCCESSOR is a terminal node, label it SOLVED and assign it, an $h^1$ value of 0.
        iii. If SUCCESSOR is not a terminal node, compute its $h^1$ value.
    (c) Propagate the newly discovered information up the graph by doing the following: Let S the a set of nodes that have been labelled SOLVED or whose $h^1$ values have been changed and so need to have values propagated back to their parents. Initialize S to NODE. Until S is empty, repeat the following procedure:
        i. If possible, select from S a node none of whose descendants in GRAPH occurs in S. If there is no such node, select any node from S. Call this node CURRENT, and remove it from S.
        ii. Compute the cost of each of the arcs emerging from CURRENT. The cost of each arc is equal to the sum of the $h^{1'}$ values of each of the nodes at the end of the arc plus whatever
```

the cost of the arc itself is. Assign as CURRENT's new h1 value the minimum or the costs just computed for the arcs emerging from it.

iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost as computed in the previous step.

iv. Mark CURRENT SOLVED if all of the nodes connected to it through the new labelled arc have been labelled SOLVED.

v. If CURRENT has been labelled SOLVED or if the cost of CURRENT was just changed, then its new status must be propagated back up the graph. So add all of the ancestors of CURRENT to S.

Constraint Satisfaction

- Many problems in AI can be viewed as problems of Constraint Satisfaction in which goal is to discover some problem state that satisfies a given set of constraints.
- Examples of this sort of problem include crypt arithmetic puzzles.
- Design tasks can also be viewed as Constraint Satisfaction problems in which a design must be created within fixed limit, on time, cost, and materials.
- By viewing a problem as one of constraint satisfaction, its often possible to reduce substantially the amount of search that, is required as compared with a method that attempts to form partial solutions.
- A constraint satisfaction approach to solving this problem avoids making guesses on particular assignments of numbers to letters. Instead, the initial set of constraints, which says that each number may correspond to only one letter and that the sums of the digits must be as they are given in the problem, is first augmented to include restrictions that can be inferred from the rules of arithmetic.
- Then, although guessing may still be required, the number of allowable guesses is reduced.
- Constraint satisfaction is a search procedure that operates in a space of Constraint sets.
- The initial state contains the constraints that are originally given in the problem description. A goal state is any state that has been constrained -enough." where "enough' must be defined for cacti problem.
- Constraint satisfaction is a two-step process. First, constraints are discovered and propagated as far as possible throughout the system. Then, if there is still not a solution, search begins. A guess about something is made and added as a new constraint. Propagation can then occur with this new constraint, and so forth.

- The first step, propagation, arises from the fact that there are usually dependencies among the constraints. These dependencies occur because many constraints involve more than one object and many objects participate in more than one constraint.
- So, for example, assume we start with one constraint: N = E + I. Then, if we added the constraint N = 3, we could propagate that to get a stronger constraint on E, namely that E = 2.
- Constraint propagation terminates for one of two reasons.
    - First, a contradiction may be detected. If this happens, then there is no solution consistent with all the known constraints. If the contradiction involves only those constraints that were given as part of the problem specification (as opposed to ones that were guessed during problem solving), then no solution exists.
    - The second possible reason for termination is that the propagation has run out of steam and there are no further changes that can be made on the basis of current knowledge. If this happens and a solution has not yet been adequately specified, then search is necessary to get the process moving again.
    - At this point, the second step begins. Some hypothesis about a way to strengthen the constraints must be made. In case of the crypt arithmetic problem, for example this usually means guessing a particular value for some letter. Once this has been done, constraint propagation can begin again from this new state. If a solution is found, it can be reported. If still more guesses are required, they can be made. If a contradiction is detected, then backtracking can he used to try a different guess and proceed with it.

Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this, first set OPEN to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty:

(a) Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.

(b) If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to 0PEN all objects that share any constraints with OB.

(c) Remove OB from OPEN.

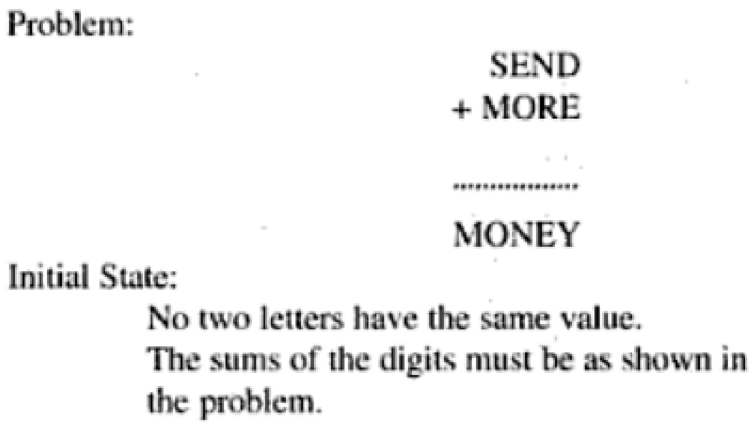
2. If the union of the Constraints discovered above defines a solution, then quit and report the solution.

3. If the union of the constraints discovered above defines a contradiction then return failure.

4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:

(a) Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.

(b) Recursively inv006Fke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.



Problem:

```
  SEND
+ MORE
..................
 MONEY
```

Initial State:

No two letters have the same value.
The sums of the digits must be as shown in the problem.

- The goal slate is problem state in which all letters have been assigned a digit in such a way that all the initial constraints are satisfied.
- The solution process proceeds in cycles. At each cycle, two significant things are done :
  - Constraints are propagated by using rules that correspond to the properties of arithmetic.
  - A value is guessed for some letter whose value is not yet determined.
- In the first step, it does not usually matter a great deal what order the propagation is done in, since all available propagations will be performed before the step ends.
- In the second step, the order in which guesses are tried may have a substantial impact on the degree of search that is necessary.
- A few useful heuristics can help to select the best guess to try first. For example, if there is a letter that has only two possible values and other with possible values, there is a better chance of guessing right on the first than on the second.
- Another useful heuristic is that if there is a letter that participates in many constraints then it is a good idea to prefer it to a letter that participates in a few. A guess on such a highly constrained letter will

usually lead quickly either to a contradiction (if it's wrong) or to the generation of many additional constraints (if it is right). A guess on a less constrained letter, on the other hand, provides less information.

- ➢ Let Cl, C2, C3, and C4 indicate the carry bits out of the columns, numbering from the right. Initially, rules for propagating constraints generate the following additional constraints:

  • M 1, since two single-digit numbers plus a carry cannot total more than 19; S=8 or 9, since S+M+C3>9 (to generate the carry) and M=1, S+l+C3>9. So S + C3 > 8 and C3 is at most 1.

  • O=0, since S + M(l)+C3(<= 1) must be at least 10 to generate a carry and it can be at most 11. But M is already 1. So O must be 0.

  • N = E or E+1, depending on the value of C2. But N cannot have the same value as E. So, N=E+l and C2 is 1.

  • In order for C2 to be1, the sum of N+ R+CI must be greater than 9, so N+ R must be greater than 18.

  • N + R cannot be greater than 18, even with a carry in, so E cannot be 9.

- ➢ At this point, let us assume that no more constraints can be generated. Then, to make progress from here, we must guess. Suppose E is assigned the value 2. (We chose to guess a value for E because it occurs three times and thus interacts highly with the other letters.) Now the next cycle begins. The constraint propagator now observes that:

  • N =3, since N = E+ 1.

  • R= 8 o 9, s ince R+ N (3)+C1 (1 or 0)=2 or 12. But since N is already 3, the sum of these non negative numbers cannot be less than 3. Thus R + 3 + (0 or 1)= 12 and R = 8 or 9.

  • 2 + D= Y or 2 + D = 10 + Y, from the sum in the rightmost column.

Game Playing

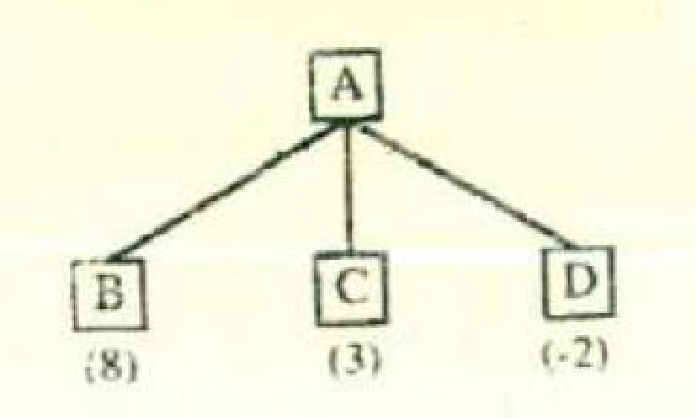
- ➢ **Charles Babbage**, the nineteenth-century computer architect, thought about programming his **Analytical Engine** to play chess and later of building a machine to play tic-tac-toe.
- ➢ **Claude Shannon** wrote a paper in which he described mechanisms that could be used in a program to play chess.
- ➢ **Alan Turing** described a chess-playing program, but he never built it.
- ➢ **Arthur Samuel,** succeeded in building the first significant, operational game-playing program. His program played checkers and, in addition to simply playing the game, could learn from its mistakes and improve its performance.
- ➢ There were two reasons that games appeared to be a good domain in which to explore machine intelligence:

- They provide a structured task in which it is very easy to measure success or failure.
- They did not obviously require large amounts of Knowledge. They were thought to be solvable by straightforward search from the starting state to a winning position.
- A program that simply does a straightforward search of the game tree will not be able to select even its first move during the lifetime of its opponent. Some kind of heuristic search procedure is necessary.
- We use **Generate-and-Test procedures** in which the testing is done after varying amounts of work by the generator.
    - At one extreme, the generator generates entire proposed solutions, which the tester then evaluates.
    - At the other extreme, the generator generates individual moves in the search space, each of which is then evaluated by the tester and the most promising one is chosen.
- To improve the **effectiveness** of a search-based problem-solving program, two things can be done:
    - Improve the generate procedure so that only good moves (or paths) are generated.
    - Improve the test procedure so that the best moves (or paths) will be recognized and explored first.
- The ideal way to use a search procedure to find a solution to a problem is to generate moves through the problem space until a goal state is reached.
- In order to choose the best move, the resulting board positions must be compared to discover which is most advantageous. This is done using a **static evaluation function**, which uses whatever information it has to evaluate so that the best next move can be chosen.

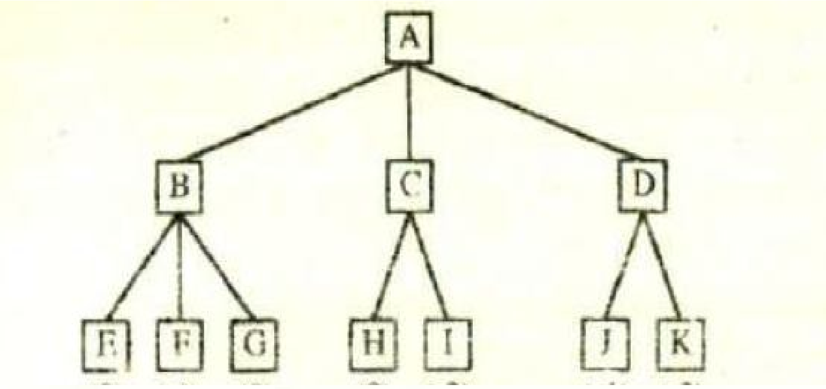
The Minimax Search Procedure

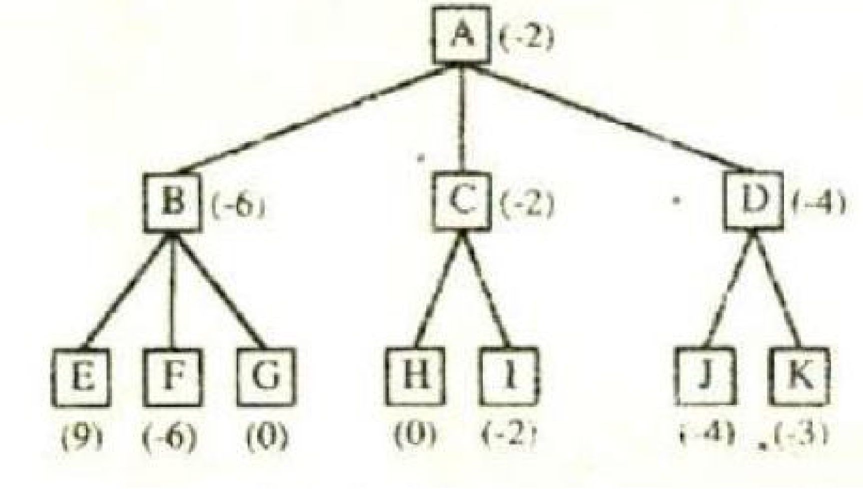
- The minimax search procedure is a depth-first, depth-limited search procedure.
- The idea is to start at the current position and use the plausible-move generator to generate the set of possible successor positions.
- We apply the static evaluation function to those positions and simply choose the best one. Then we can back that value up to the starting position to represent our evaluation of it.
- Our goal is to **maximize** the value of the static evaluation function of the next board position.
- Assume a static evaluation function that returns values ranging from - 10 to 10, with 10 indicating a win for us, and -10 a win for the opponent, and 0 an even match.

- Since our goal is to maximize the value of the heuristic function, we choose to move to B. Backing B's value up to A, we can conclude that A’s value is 8.



- After our move, the situation would appear to be very good. But, if we look one move ahead, we will see that one of our pieces also gets captured and so the situation is not as favourable as it seemed.
- So we would like to look ahead to see what will happen to each of the new game positions at the next move which will be made by the opponent.
- Instead of applying the static evaluation function to each of the positions that we just generated, we apply the plausible-move generator, generating a set of successor positions for each position.





- But now we must take into account that the opponent gets to choose which successor moves to make and thus which terminal value should he backed up to the next level.
- Suppose we made move B. Then the opponent must choose among moves E, F, and G. The opponents goal is to minimize the value of the evaluation function, so he or she can be expected to choose move F. This means that if we make move B, the actual position in which we will end up one move later is very bad for us. If node E is selected, it is very good for us. Since at this level we are not the ones to move, we will not get to choose it.
- At the opponent's choice, the minimum value was chosen and backed up. At the level representing our choice, the maximum value was chosen.
- Once the values from the second ply are backed up, it becomes clear that the correct move for us to make at the first level, given the information, is C, since there is nothing the opponent can do from there to produce a value worse than –2.
- This process can he repeated for as many ply as time follows, and more accurate evaluations that are produced can be used to choose the correct move at the top level.
- The alternation of maximizing and minimizing at alternate ply when evaluations are being pushed back up corresponds to the opposing strategies of the two players and hence this method is called. **minimax.**
- The recursive procedure that relies on two auxiliary procedures that are specific to the game being played:
  - **MOVEGEN(Position,Player)** -The plausible-move generator, which returns the list of nodes representing the moves that can be made by player in Position. We call the two players PLAYER-ONE and PLAYER-TWO. In a chess program, we might use the names BLACK and WRITE instead.
  - **STATIC(Position,Player)**—The static evaluation function, which returns a number representing the goodness of position from the standpoint of Player.
- As with any recursive program. a critical issue in the design of the MINIMAX procedure is when to stop the recursion and simply call the static evaluation function.
- The following are the factors that may influence this decision. They include:
  - Has one side Won?
  - How many ply have we already explored?
  - How promising is this path?

- How much time is left?
- How stable is the configuration?

➢ For the general MINIMAX, we use a function **DEEP-ENOUGH**, which is assumed to evaluate all of these factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise.

➢ The implementation of DEEP-ENOUGH will take two parameters: **Position and Depth**.

➢ It will ignore its Position parameter and simply return TRUE if its Depth parameter exceeds a constant cut off value.

➢ M1NIMAX as a recursive procedure needs to return two results:

- The backed-up value of the path it chooses.
- The path itself. We return the entire path even though probably only the first element, representing the best move from the current position, is actually needed.

➢ MINIMAX returns a structure containing both results: **VALUE and PATH.**

➢ We define the MININMAX procedure as a recursive function, called initially that takes three parameters.

- A board position.
- The current depth of the search, and
- The player to move.

➢ So the initial call to compute the best move from the position CURRENT should be :

- MI1N1MAX(CURRENT, O PLAYER-ONE) if PLAYER-ONE is to move,
- MINIMAX (CURRENT, 0, PLAYER-TWO) if PLAYER TWO is to move.

Algorithm: MINIMAX (Position, Depth, Player)

1. If *DEEP-ENOUGH (Position, Depth) then* return the structure.

    VALUE= STATIC (Position, Player);

    PATH = nil

This indicates that there is no path from this node and that its value is that determined by the static evaluation function,

2. Otherwise, generate one more ply of the tree by calling the function **MOVEGEN (Position, Player**) and setting SUCCESSORS to the list it returns.

3. If SUCCESSORS is empty, then there are no moves to be made, so return the same structure that would have been returned it DEEP-ENOUGH had returned true.

4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This *is* done as follows.

Initialize BEST-SCORE to the minimum value that STATIC can return It will be updated in reflect the best score that can be achieved by an element of SUCCESSORS**.**

For each element *SUCC* of SUCCESSORS, do the following:

(a)Set RESULT-SUCC to

MINIMAX(SUCC, *Depth + i, OPPOSITE(Player))*

(b)Set NEW_VALUE to –VALUE (RESULT-SUCC).

(c) If NFW-VALUE > BEST-SCORE, then we have found a successor that better than any that have been examined so far. Record this by doing the following:

i. Set BEST-SCORE to NEW-VALUE.

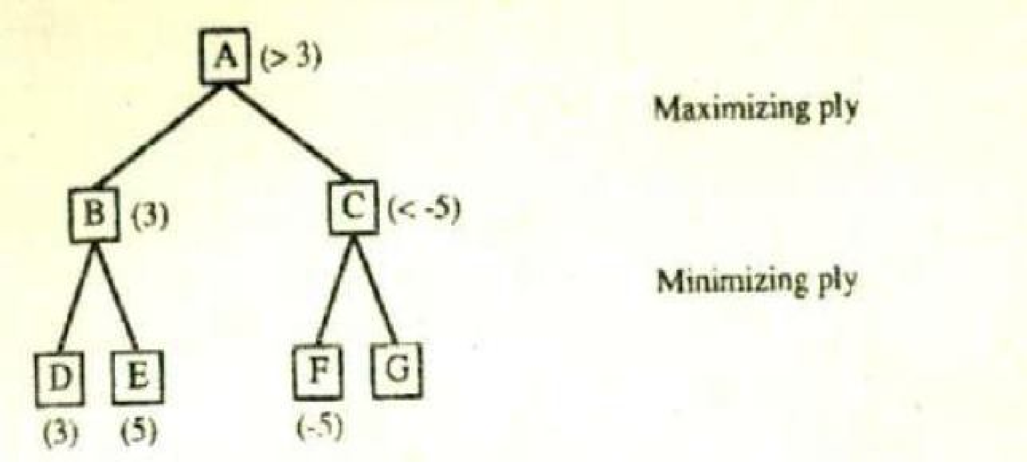
ii. The best known path is now from CURRENT to SUCC and then on to the appropriate path down from SUCC as determined by the recursive call to MINIMAX. So set BEST-PATH to the result of attaching SUCC to the front of PATH(RESUIT-SUCC);

5. Now that all the successors have been examined, we know the value of Position as well as which Path to take from it. So return the structure:

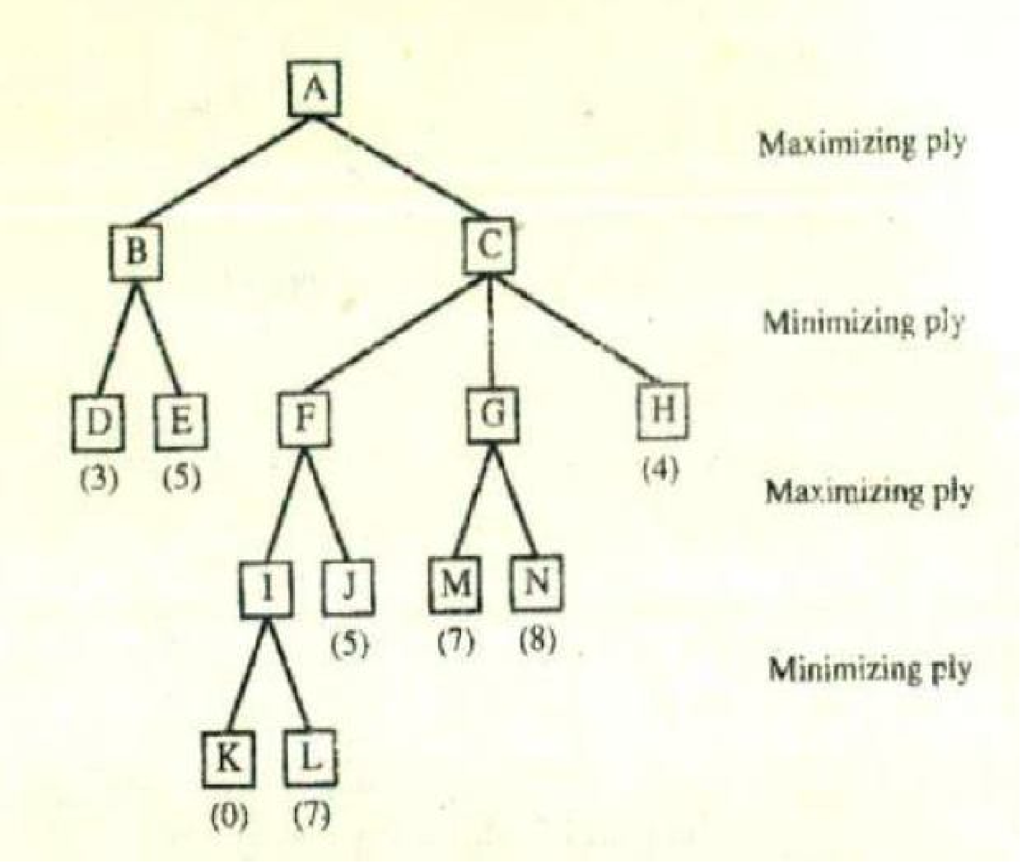
VALUE = BEST-SCORE
PATH = BEST- PATH

Alpha-Beta Cutoffs

- The minimax procedure is a depth-first process. One path is explored as far as time allows, the static evaluation function is applied to the game positions at the last step of the path, and the value can then be passed up the path one level at a time.
- The efficiency of depth-first procedures can be improved by using **branch-and-bound** techniques in which partial solutions that are clearly worse than known solutions can be abandoned early.
- For this, we require to store of the length of the best path found so far. If a later partial path outgrew that bound, it was abandoned.
- It is necessary to modify our search procedure to handle both maximizing and minimizing players.
- It is also necessary to modify the branch-and-bound strategy to include two bounds, one for each of the players. This modified strategy is called "**alpha-beta pruning"**. It requires the maintenance of two threshold values:
  - a lower bound on the value that a maximizing node may ultimately be assigned called **alpha.**
  - an upper bound on the value that a minimizing node may be assigned called **beta.**



- After examining node F, we know that the opponent is guaranteed a score of —5 or less at C (since the opponent is the minimizing player). But we also know that we are guaranteed a score of 3 or greater at node A, which we can achieve if we move to B.
- Any other move that produces a score of less than 3 is worse than the move to B, and we can ignore it.
- After examining only F, we are sure that a move to C is worse (it will be less than or equal to —5) regardless of the score of node G. Thus we need not bother to explore node G at all.



- **Alpha Value:** In searching the tree, the entire sub tree headed by B is searched, and we discover that at A we can expect a score of at least 3. When this alpha value is passed down to F, it will enable us to skip the exploration of L.
- The reason is as follows: After K is examined; we see that I is guaranteed a maximum score of 0, which means that F is guaranteed a minimum of 0. But this is less than alpha's value of 3, no more branches of I need be considered.
- The maximizing player already knows not to choose to move to C and then to I since, if that move is made, the resulting score will be no better than 0 and a score of 3 can be achieved by moving to B instead. Nov,
- **Beta Value:** After cutting off further exploration of I, J is examined yielding a value of 5, which is assigned as the value of F. This value becomes the value of beta at node C. It indicates that C is guaranteed to get a 5 or less.
- Now, we must expand G. First M is examined and it has a value of 7, which is passed back to G as its tentative value. But now 7 is compared to beta (5). It is greater, and the player whose turn it is at node C is trying to minimize. So this player will not choose G, which would lead to a score of at least 7, since there is in alternative move to F, which will lead to a score of 5. Thus it is not necessary to explore any of the other branches of G.
- The function **MINIMAX-A-B**, which requires four arguments: **Position, Depth, Use-Thresh, and Pass-Thresh.**
- The initial call, to choose a move for PLAYER-ONE from the position CURRENT should be

  MINIMAX-A-B(CURRENT
  0,
  PLAYER-ONE,
  maximum value STATIC can compute
  minimum value STATIC can compute)

  The initial values for Use-Thresh and Pass-Thresh represent the worst values that each side could achieve.

Algorithm: MINIMAX-A-B((Position, Depth, Player, Use-Thresh, Pass-Thresh)

1. If *DEEP-ENOUGH (Position, Depth) then* return the structure.

   VALUE= STATIC (Position, Player);

   PATH = nil

   This indicates that there is no path from this node and that its value is that determined by the static evaluation function,

2. Otherwise, generate one more ply of the tree by calling the function **MOVEGEN (Position, Player**) and setting SUCCESSORS to the list it returns.

3. If SUCCESSORS is empty, then there are no moves to be made, so return the same structure that would have been returned it DEEP-ENOUGH had returned true.

4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This *is* done as follows.

Initialize BEST-SCORE to the minimum value that STATIC can return It will be updated in reflect the best score that can be achieved by an element of SUCCESSORS**.**

For each element *SUCC* of SUCCESSORS, do the following:

(a) Set RESULT-SUCC to

MINIMAX-A-B (SUCC, Depth + 1, OPPOSITE(Player),
-Pass-Thresh, -Use-Thresh).

(b) Set NEW-VALUE to —VALUE (RESULT-SUCC).

(c) If NEW-VALUE > Pass-Thresh, then we have found a successor that is better than any that have been examined so far. Record this by doing the following:

i. Set Pass- Thresh to NEW-VALUE.

ii. The best known path is now from CURRENT to SUCC and then on to the appropriate path from SUCC as determined by the recursive call to MINIMAX-A-B. So set BEST-PATH to the result of attaching SUCC to the front of PATH (RESULT-SUCC).

(d) If Pass-Thresh (reflecting the current best value) is not better than Use-Thresh, then we should stop examining this branch. But both thresholds and values have been inverted. So if Pass-Thresh >= (Use-Thresh, then return immediately with the value

VALUE = Pass-Thresh
PATH = BEST-PATH

5. Return the structure:

VALUE Pass-Thresh
PATH = BEST-PATH

UNIT-I
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. Define the term “state space”.

2. Define the term “Operationalization”.

3. *A* ___________ *is* a technique that improves the efficiency of a search process. [ ]

   (a) Heuristic (b) Control Strategy (c) GPS (d) none

4. List the steps for solving the problem.

5. In Theorem Proving, the solution steps can be________. [ ]

   (a) Ignored (b) Recoverable (c) Irrecoverable (d) none

6. Problems in which solution steps cannot be undone are ___ [ ]

   (a) Ignored (b) Recoverable (c) Irrecoverable (d) none

7. Travelling Salesman problem is an example of________. [ ]

   (a) Best-Path (b) Any-Path (c) Both (d) none

8. ___________ are the problems in which the computer is given a problem description and produces an answer with no intermediate communication. [ ]

   (a) Conversational (b) Solitary (c) Ignorable (d) None

9. ________is the list of nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined. [ ]

   (a) OPEN (b) CLOSED (c) NODES (d) none

10. The function ___ is a measure of the cost of getting from the initial state to the current node. [ ]

   (a) f^1 (b) g (c) h^1 (d) none

11.___________ is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then he solved. [ ]

   (a) OR graph (b) AND graph (c) AND-OR graph (d) none

12.___________ function uses whatever information it has to evaluate so that the best next move can be chosen. [ ]

(a) static evaluation (b) dynamic evaluation

(c) Threshold evaluation (d) none

13.DEEP-ENOUGH will take two parameters: _______ and __________.

14. MINIMAX returns a structure containing both results: _______ and ____________.

15. A lower bound on the value that a maximizing node may ultimately be assigned called _________. [ ]

(a) alpha (b) beta (c) gamma (d) none

16. An upper bound on the value that a minimizing node may be assigned called ___________. [ ]

(a) alpha (b) beta (c) gamma (d) none

SECTION-B

SUBJECTIVE QUESTIONS

1. List the steps for General Problem Solving.

2. Give an example of a problem for which breadth -first search would work better than depth-first search. Give an example of a problem for which depth-first search would work better than breadth-first search.

3. Describe the state space of water jug problem and also explain its solution.

4. List the steps necessary to provide a formal description of a problem.

5. What are the requirements of a control strategy? Develop an algorithm for:

(i) Breadth First Search (ii) Depth First Search

6. Summarize the advantages and disadvantages of control strategies.

7. Outline the factors that are necessary for analyzing a problem to choose most appropriate heuristic method.

8. Explain non-decomposable problem with suitable example.

9. Distinguish between Ignorable, Recoverable and Irrecoverable problems with necessary examples.

10. Distinguish between:

(i) Certain Outcome Vs Uncertain Outcome problems

(ii) Best path and Any-path problems

(iii) Problems whose solution is a state and whose solution is a path to state.

(iv) Solitary Vs Conversational problems

11. Explain an algorithm for Best-first Search.

12. Explain Problem Reduction using AND-OR graph with an algorithm.

13. Explain AO^* algorithm.

14. Explain the procedure of Constraint Satisfaction with an example.

15. Explain Minimax procedure with an example.

16. Explain Alpha-Beta Pruning with an example.

17. Solve the Criptarithmetic problem using Constraint Satisfaction:

CROSS
ROADS

____

DANGER

____

UNIT - III: Logic Concepts

Syllabus:

Introduction, propositional calculus, proportional logic, natural deduction system, axiomatic system, semantic tableau system in proportional logic, resolution in proportional logic, predicate logic.

Outcomes:

Student will be able to:

Introduction

- One particular way of representing facts is the language of logic.
- The logical formalism is appealing because it immediately suggests a powerful way of deriving new knowledge from old called as "mathematical deduction".
- In this formalism, we can conclude that a new statement is true by proving that it follows from the statements that are already known.
- The way of demonstrating the truth of an already believed proposition can be extended to include deduction as a way of deriving answers to questions and solutions to problem.
- we use the following standard logic symbols:
    - →(Implication)
    - ¬ (Not)
    - Λ (And)
    - V (Or)
    - ∃ (there exists)
    - ∀ (For All)

Representing Simple Facts in Logic

- Propositional logic is appealing because it is simple to deal with and a decision procedure for it exists.
- We can easily represent real-world facts as logical propositions written as well formed formulas (wff's) in Propositional logic.
- Some Simple Facts in Propositional logic:

- It is raining

  RAINING
- It is sunny

  SUNNY
- It is windy

  WINDY
- If it is raining, then it is not sunny.

  RAINING→¬SUNNY

➢ Suppose we want to represent the fact stated by the sentence:

- Socrates is a man

  We could write SOCRATESMAN. It is represented as:

  MAN (SOCRATES)
- Plato is a man

  MAN (PLATO)

➢ Now the structure of the representation reflects the structure of the knowledge itself. But to do that, we need to be able to use “**predicates”** applied to arguments.

➢ To represent the following sentence:

All men are mortal

We can’t represent this as: **MORTALMAN**

But that fails to capture the relationship between any individual being a man and that individual being a mortal. To do that, we need **variables and quantification.**

➢ So we use **first-order predicate logic** as a way of representing knowledge because it permits representations of things that cannot be represented in propositional logic.

➢ In predicate logic, we can represent real-world facts as statements written as wff's.

➢ The major motivation for choosing to use logic is that it we use logical statements as a way of representing knowledge, and then we have a good way of reasoning with that knowledge.

- Determining the validity of a proposition in propositional logic is straightforward, although it may be computationally hard.
- Before we use predicate logic as a medium for representing knowledge, we need to check whether it also provides a good way of reasoning with the knowledge.
    - It provides a way of deducing new statements from old ones.
    - Unfortunately, unlike propositional logic, it does not possess a decision procedure.
    - There do exist procedures that will find a proof of a proposed theorem, but first-order predicate logic is not decidable, it is semi-decidable.
    - A simple such procedure is to use the rules of inference to generate theorems from the axioms in some orderly fashion. This method is not particularly efficient, however, and we will want to try to find a better one.

Use of predicate logic:

- Let's now explore the use of predicate logic as a way of representing knowledge by looking at a following example:

  Consider the following set of sentences:

  1. Marcus was a man.
  2. Marcus was a Pompeian.
  3. All Pompeians were Romans.
  4. Caesar was a ruler.

  S, All Romans were either loyal to Caesar or hated him.

  6. Everyone is loyal to someone.
  7. People only try to assassinate rulers they are not loyal to.
  8. Marcus tried to assassinate Caesar.

- The facts described by these sentences can be represented as a set of wff's in predicate logic as follows:

  1. Marcus was a man.

  man (Marcus)

This representation captures the critical fact of Marcus being a man. It fails to capture some of the information in the English sentence, the notion of past tense.

2. Marcus was a Pompeian.

Pompeian (Marcus)

3. All Pompeians were Romans.

$\forall(x)\text{: }\textbf{Pompeian(x)} \rightarrow \textbf{Roman(x)}$

4. Caesar was a ruler.

ruler (Caesar)

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

$\forall(x)\text{: }\textbf{Roman(x)} \rightarrow \textbf{loyalto(x, Caesar) V hate(x , Caesar)}$

6. Everyone is loyal to someone.

$\forall\textbf{(x) : }\exists(y)\text{: }\textbf{loyalto(x, y)}$

7. People only try to assassinate rulers they are not loyal to

$\forall(x) : \exists(y)\text{:person}(x) \wedge \text{ruler}(y) \wedge \text{tryassassinate}(x, y) \rightarrow \neg\textbf{loyalto(x, y)}$

8. Marcus tried to assassinate Caesar

tryassassinate(Marcus, Caesar)

- Now suppose that we want to use these statements to answer the question:

Was Marcus loyal to Caesar?

Let's try to produce a formal proof, reasoning backward from the desired goal:

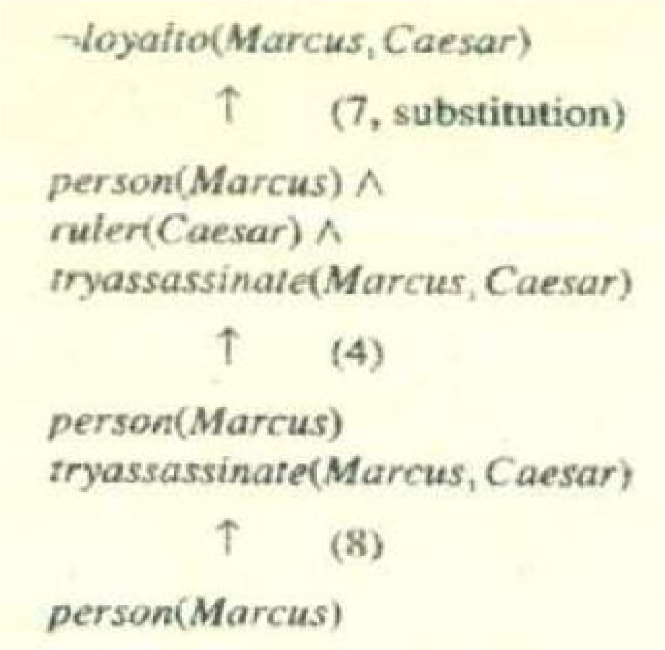
$\neg\textbf{loyalto (Marcus, Caesar)}$

To prove this, let us add a fact:

All men are people

Although we know that Marcus was a man, we do not have any way to conclude that Marcus was a person. So, we need the representation of another fact to our system,

$\square\textbf{(x): man(x)} \rightarrow \textbf{person(x)}$



$\neg loyalto(Marcus, Caesar)$

↑ (7, substitution)

$person(Marcus) \wedge$
$ruler(Caesar) \wedge$
$tryassassinate(Marcus, Caesar)$

↑ (4)

$person(Marcus)$
$tryassassinate(Marcus, Caesar)$

↑ (8)

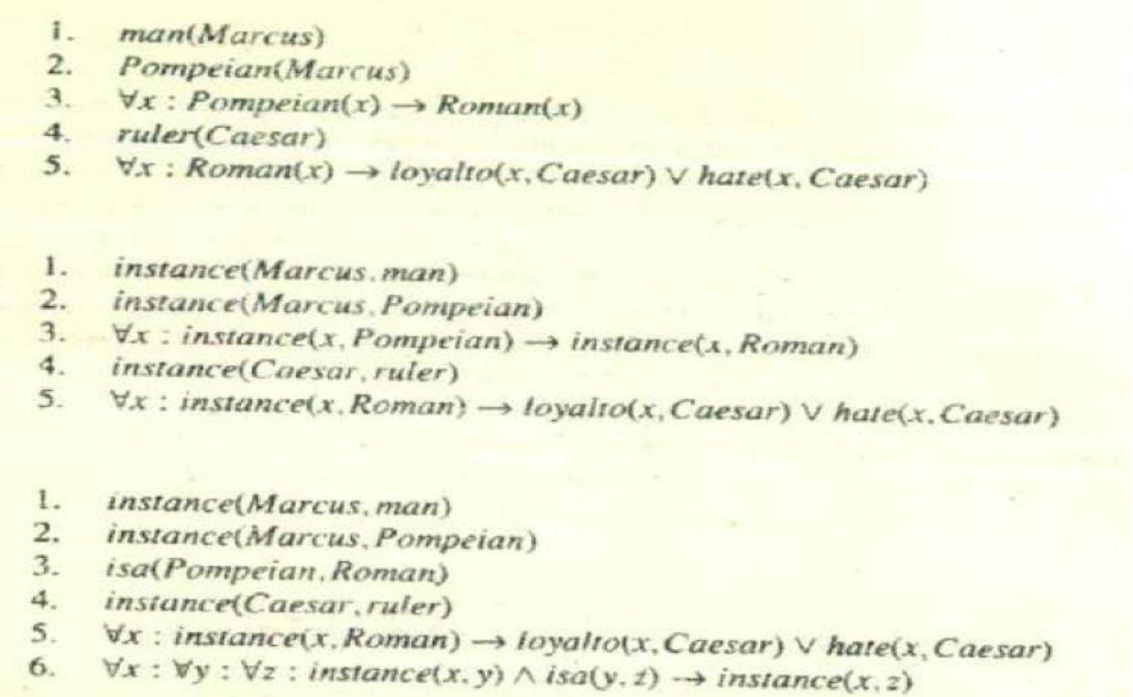
$person(Marcus)$

Representing Instance and isa Relationships

- The specific attributes **instance and isa** play an important role in the useful form of reasoning, **property inheritance**.
- Let us represent the following facts in three ways:

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.

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



1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\forall x : Pompeian(x) \rightarrow Roman(x)$
4. $ruler(Caesar)$
5. $\forall x : Roman(x) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. $instance(Marcus, man)$
2. $instance(Marcus, Pompeian)$
3. $\forall x : instance(x, Pompeian) \rightarrow instance(x, Roman)$
4. $instance(Caesar, ruler)$
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. $instance(Marcus, man)$
2. $instance(Marcus, Pompeian)$
3. $isa(Pompeian, Roman)$
4. $instance(Caesar, ruler)$
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$
6. $\forall x : \forall y : \forall z : instance(x, y) \wedge isa(y, z) \rightarrow instance(x, z)$

- Second representation: The predicate ***instance*** *is* a binary one, whose first argument is an object and whose second argument is a class to which the object belongs. But these representations do not use an explicit ***isa*** predicate.
- In subclass relationships, such as that between Pompeians and Romans, the implication rule there states that it an object is an instance of the subclass *Pompeian* then it is an instance of the superclass *Roman.*
- This rule is equivalent to the standard set-theoretic definition of the subclass-superclass relationship.
- Third representation: It contains representations that use both the ***instance* and *isa* predicates** explicitly.
- The use of the *isa* predicate simplifies the representation of sentence 3, but it requires that one additional axiom. This additional axiom describes how an *instance* relation and an *isa* relation can be combined to derive a new *instance* relation.

Propositional Logic:

Consider the following set of facts:

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was boni in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 AD.
6. No mortal lives longer than 150 years.
7. It is now 1991.
8. Alive means not dead.
9. If someone dies, then he is dead at all later times.

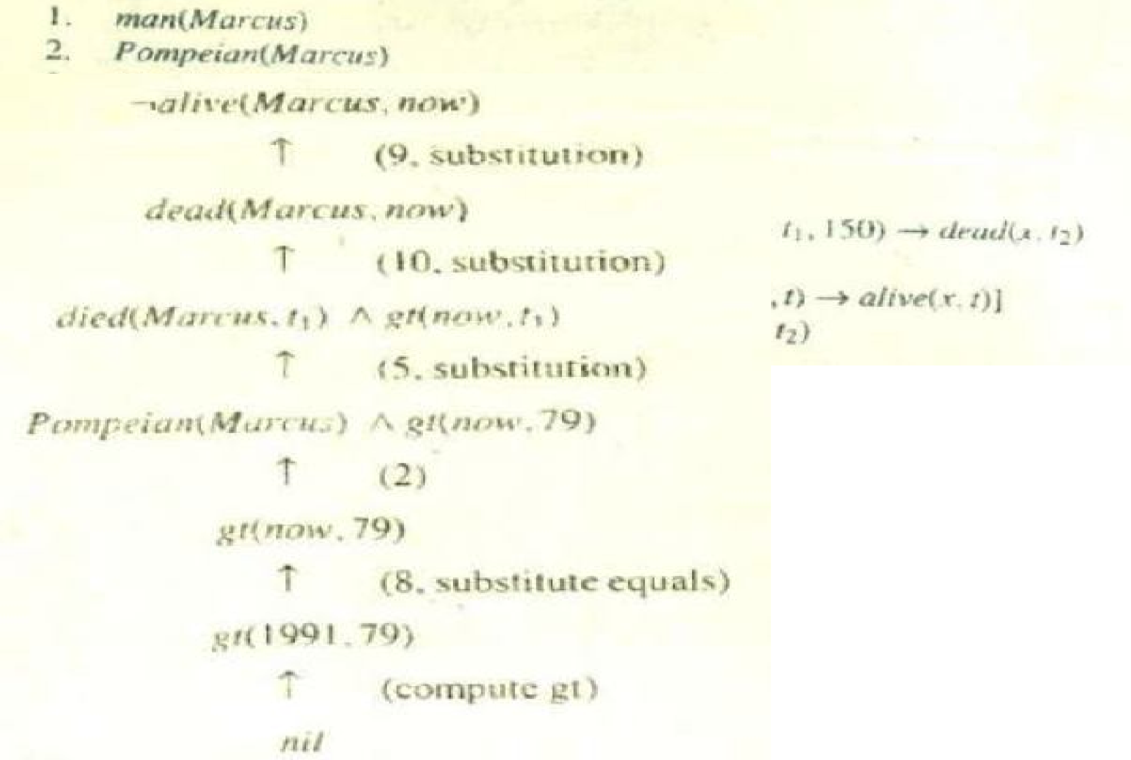
Now let's attempt to answer the question:

"Is Marcus alive?"

By proving:

¬ alive(Marcus, now)

The given facts are represented as follows:



1. $man(Marcus)$
2. $Pompeian(Marcus)$

$\neg alive(Marcus, now)$

$\uparrow$ (9, substitution)

$dead(Marcus, now)$

$\uparrow$ (10, substitution)

$died(Marcus, t_1) \wedge gt(now, t_1)$

$\uparrow$ (5, substitution)

$Pompeian(Marcus) \wedge gt(now, 79)$

$\uparrow$ (2)

$gt(now, 79)$

$\uparrow$ (8, substitute equals)

$gt(1991, 79)$

$\uparrow$ (compute gt)

nil

$t_1, 150) \rightarrow dead(x, t_2)$

$, t) \rightarrow alive(x, t)]$

$t_2)$

Resolution:

- Resolution is such a procedure, which gains its efficiency from the fact that it operates on statements that have been converted to a very convenient standard form.
- Resolution produces proofs by ***refutation***. To prove a statement (to show that it is valid), resolution attempts to show that the negation of the statement produces a contradiction with the known statements (i.e.. that it is unsatisfiable).
- Resolution is a technique for proving theorems in the propositional or predicate calculus that has been a part of AI problem-solving.
- Resolution describes a way of finding contradictions in a database of clauses with minimum use of substitution. Resolution refutation proves a theorem by negating the statement to be proved and adding this negated goal to the set of axioms that are known (have been assumed) to be true.
- It then uses the resolution rule of inference to show that this leads to a contradiction. Once the theorem prover shows that the negated goal is inconsistent with the given set of axioms, it follows that the original goal must be consistent. This proves the theorem.
- Resolution refutation proofs involve the following steps:

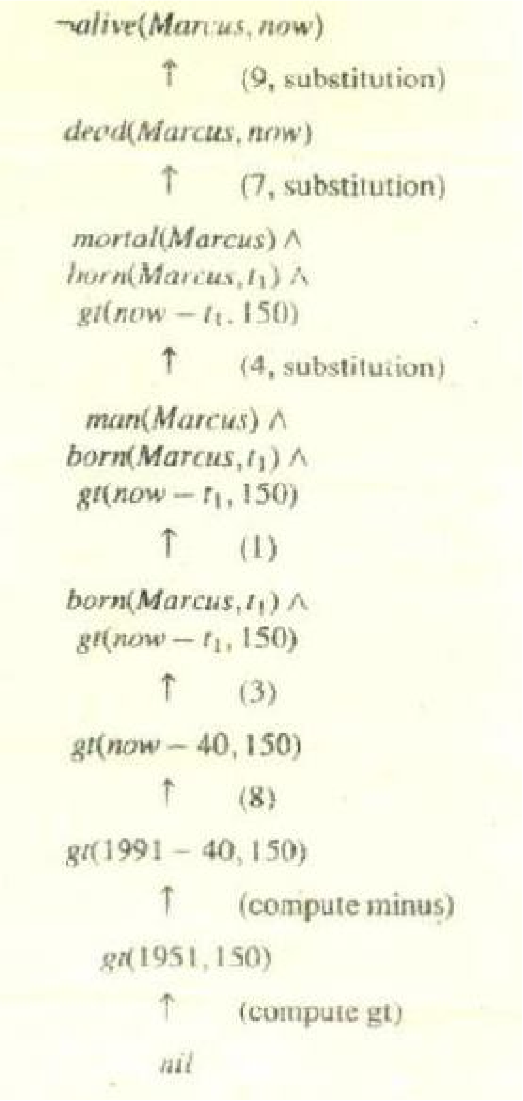
  1. Put the premises or axioms into clause form.

  2. Add the negation of what is to be proved, in clause form, to the set of axioms.

  3. Resolve these clauses together, producing new clauses that logically follow from them.

  4. Produce a contradiction by generating the empty clause.

  5. The substitutions used to produce the empty clause are those under which the opposite of the negated goal is true.



$\neg alive(Marcus, now)$

↑ (9, substitution)

$dead(Marcus, now)$

↑ (7, substitution)

$mortal(Marcus) \wedge$
$born(Marcus, t_1) \wedge$
$gt(now - t_1, 150)$

↑ (4, substitution)

$man(Marcus) \wedge$
$born(Marcus, t_1) \wedge$
$gt(now - t_1, 150)$

↑ (1)

$born(Marcus, t_1) \wedge$
$gt(now - t_1, 150)$

↑ (3)

$gt(now - 40, 150)$

↑ (8)

$gt(1991 - 40, 150)$

↑ (compute minus)

$gt(1951, 150)$

↑ (compute gt)

nil

Conversion to Clause Form:

- Consider the following fact:

  All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy.

∀x:[Roman(X)Λknow(x,Marcus)]→[hate(x,Caesar)V(∀y:∃z:hate(y, z)→thinkcrazy(x, y))]

Algorithm: Conversion to clause form:

1. Eliminate→, using the fact that **a→b** is equivalent to**¬ aVb**. Performing this transformation, we get:

$\forall x:\neg[\text{Roman}(X)\wedge \text{know}(x,\text{Marcus})]\mathbf{V}[\text{hate}(x,\text{Caesar})V(\forall y:\neg(\exists z:\text{hate}(y, z))\mathbf{V}\text{thinkcrazy}(x, y))]$

2. Reduce the scope of each **¬** to a single term, using the fact that $\neg(\neg P)=P$ and deMorgan's laws [which say that $(\neg(a\wedge b)= \neg aV \neg b$ and $(\neg(a \text{ V } b)= \neg a\wedge \neg b)$ and the standard correspondences between quantifiers:- $\forall x: P(x)= \exists x: \neg P(x)$ and $\exists x: P(x)= \forall x: \neg P(x)$ Performing this , we get:

$\forall x:[\neg\text{Roman}(X)\mathbf{V}\neg\text{know}(x,\text{Marcus})]\mathbf{V}[\text{hate}(x,\text{Caesar})V(\forall y:\forall z:\neg\text{hate}(y, z))V(\text{thinkcrazy}(x, y))]$

3. Standardize variables so that each quantifier binds a unique variable. Since variables are just dummy names, this process cannot affect the truth value of the wff. For example, the formula:

$\forall x : P(x) \text{ V } \forall x: Q(x)$ would be converted to:

$\forall x : P(x) \text{ V } \forall y: Q(y)$

4. Move all quantifiers to the left of the formula without changing their relative order. This is possible since there is no conflict among variable names. Performing this, we get:

$\forall x:\forall y:\forall z:[\neg\text{Roman}(X)\mathbf{V}\neg\text{know}(x,\text{Marcus})]\mathbf{V}[\text{hate}(x,\text{Caesar})V(\neg\text{hate}(y, z)\mathbf{V}\text{thinkcrazy}(x, y))]$

At this point, the formula is known as **Prenex normal form**. It consists of a prefix of quantifiers followed by a matrix, which is quantifier-free.

5. Eliminate existential quantifiers.

For example, the formula

∃y:President(y) can be transformed into the formula:

President(S1) where S1 is a function with no arguments that somehow produces a value that satisfies **President.**

If existential quantifiers occur within the scope of universal quantifiers, then the value that satisfies the predicate may depend on the values of the universally quantified variables. For example:

$$\forall x: \exists y: \text{father-of}(y, x)$$

The value of' y that satisfies father-of depends on the particular value of x, we must generate functions with the same number of arguments as the number of universal quantifiers in whose scope the expression occurs. So this would be transformed into:

$$\forall x: \text{father-of}(S2(x), x))$$

These generated functions are called **Skolem Functions**. Sometimes ones with no arguments are called **Skomlem constants**.

6. Drop the prefix. All remaining variables are universally quantifled, so the prefix can just be dropped and any proof procedure we use can simply assume that any variable it sees is universally quantified.

$[\neg \text{Roman}(X) \lor \neg \text{know}(x,\text{Marcus})] \lor [\text{hate}(x,\text{Caesar}) \lor$

$(\neg \text{hate}(y, z) \lor \text{thinkcrazy}(x, y))]$

7. Convert the matrix into a conjunction of disjuncts. Since there are no and's, it is only necessary to exploit the associative property of (i.e., $a \lor (b \lor c) = (a \lor b) \lor c$ and simply remove the parentheses.

$\neg \text{Roman}(X) \lor \neg \text{know}(x,\text{Marcus}) \lor \text{hate}(x,\text{Caesar}) \lor (\neg \text{hate}(y, z)$

$\lor \text{thinkcrazy}(x, y)$

8 Create a separate clause corresponding to each conjunct. In order for a wff to be true, all the clauses that are generated from it must be true. If we are going to be working with several wff's, all the clauses generated by each of them can now be combined to represent the same set of facts as were represented by the original wff's.

9. Standardize apart the variables in the set of clauses.

$(\forall x:P(x) \land Q(x)) = \forall x:P(x) \land \forall x:Q(x)$

Since each clause is a separate conjunct and since all the variables are universally quantified, there is no relationship between the variables of two clauses, even if they were generated from the same wff.

The Basis of Resolution

- The resolution procedure is a simple iterative process: at each step two clauses, called the parent clauses, are compared (resolved), yielding a new clause that has been inferred from them. The new clause represents ways that the two parent clauses interact with each other. Suppose there are two clauses in the system:

winter V summer

¬winter V cold

- Now we observe that precisely one of winter and ¬winter will be true at any point.
- If winter is true, then cold must be true to guarantee the truth of the second clause.
- If ¬winter is true, then summer must be true to guarantee the truth of the first clause.
- Thus we see that from these two clauses we can deduce:

summer V cold

- Resolution operates by taking two clauses that each contains the same literal, in this example, winter. The literal must occur in positive form in one clause and in negative form in the other. The resolvent is obtained by combining all of the literals of the two parent clauses except the ones that cancel.
- If the clause that is produced is the empty clause, then a contradiction has been found. For example. the two clauses:

Winter

¬winter

will produce the empty clause.

- In predicate logic, the situation is more complicated since we must consider all possible ways of substituting values for the variables.

Resolution in Propositional Logic

- In propositional logic, the procedure for producing a proof by resolution of proposition *P* with respect to a set of axioms *F is* the following

Algorithm: Propositional Resolution

1. Convert all the propositions of *F* to clause form.

2. Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in step 1.

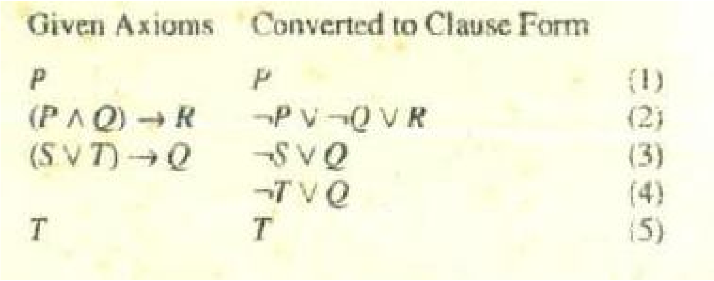
3. Repeat until either a contradiction is found or no progress can be made:

   (a) Select two clauses. Call these the parent clauses.

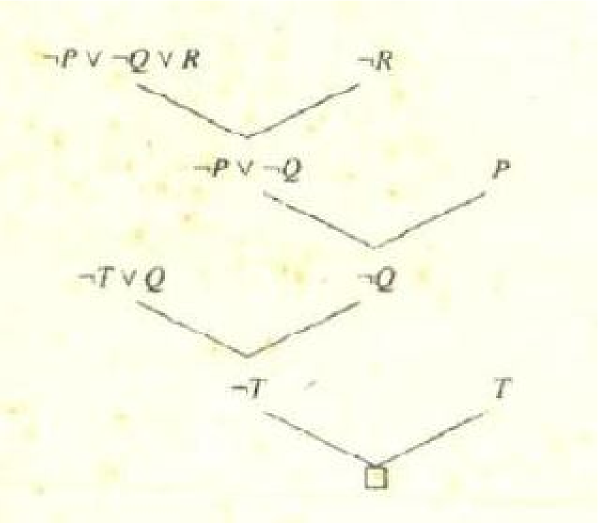
   (b) Resolve them together. The resulting clause, called the **resolvent,** will be the disjunction of all of the literals of both of the parent clauses with the following exception: It there are any pairs of literals *L* and **¬**L such that one of the parent clauses contains L and the other contains **¬**L ,then select one such pair and eliminate both *L* and **¬**L from the resolvent.

   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

- Suppose we are given the axioms shown below and we want to prove R. First we convert the axioms to clause form. Then we negate R. producing **¬**R, which is already in clause form.
- Then we begin selecting pairs of clauses to resolve together. Although any pair of clauses can be resolved, only those pairs that contain complementary literals will produce a resolvent that is likely to lead to the goal of producing the empty clause.
- We begin by resolving with the clause **¬**R .
- One way of viewing the resolution process is that it takes a set of clauses that are all assumed to be true and, based on information provided by the others, it generates new clauses that represent restrictions on the way each of those original clauses can be made true.
- A contradiction occurs when a clause when there is no way it can be true. This is indicated by the generation of the empty clause.



Given Axioms	Converted to Clause Form	
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	(5)



The Unification Algorithm

- In predicate logic, this matching process is more complicated since the arguments of the predicates must be considered.
- For example, man(John) and **¬**man(John) is a contradiction, while man(John) and **¬**man{Spot) is not.
- Thus, in order to determine contradictions, we need a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical. The straightforward recursive procedure, called the “**unification algorithm”** does this.
- **Basic idea of unification**: It is very simple. To attempt to unify two literals, we first check if their initial predicate symbols are the same. If

so, we can proceed. Otherwise, there is no way they can he unified, regardless of their arguments.

- For example, the two literals:

  trytoassassinate(Marcus, Caesar)

  hate(Marcus, Caesar)

  cannot be unified.
- If the predicate symbols match, then we must check the arguments one pair at a time. If the first matches, we can continue with the second, and so on. To test each argument pair, we can simply call the unification procedure recursively.
- The matching rules are simple: Different constants or predicates cannot match; identical ones can. A variable can match another variable, any constant, or a predicate expression with the restriction that the predicate expression must not contain any instances of the variable being matched.
- We must find a single, consistent substitution for the entire literal, not separate ones for each piece of it. To do this, we must take each substitution that we find and apply it to the remainder of the literals before we continue trying to unify them.
- For example, suppose we want to unify the expressions

  P(x, x)

  P(y, z)
- The two instances of P match. Next we compare x and y, and decide that if we substitute **y for x**, they could match. We will write that substitution as: **y/x.**
- But now, if we simply continue and match x and z, we produce the substitution **z/x.** But we cannot substitute both y and z for x. The problem can be solved as follows:
- What we need to do after finding the first substitution **y/x** is to make that substitution throughout the literals, giving:

  P(y, y)

  P(y, z)

- Now we can attempt to unify arguments y and z, which succeeds with the substitution **z/y.** The entire unification process has now succeeded with a substitution that is the composition of the two substitutions: **(z/y) (y/x).**
- In general, substitutions: **(a1/a2, a3/a4,....)(b1/b2, b3/b4....)** means to apply all the substitutions of the right most list, then take the result and apply all the ones of the next list, and so forth, until all substitutions have been applied.
- The objective of the unification procedure is to discover at least one substitution that causes two literals to match.
- For example, the literals:

  hate (x, y)

  hate (Marcus, z)

  could be unified with any of the following substitutions:

  (Marcus/x, z/y)

  (Marcus/x, y/z)

  (Marcus/x, Caesar/y, Caesar/z)

  (Marcus/x, Polonius/y, Polonius/z)
- We describe a procedure **Unify (LI, L2),** which returns as its value a list representing the composition of the substitutions that were performed during the match. The empty list, NIL, indicates that a match was found without any substitutions. The list consisting of the single value FAIL indicates that the unification procedure failed.

Algorithm: Unify (L1, L2)

```
1. If L1 and L2 are both variables and constants, then:
    (a) If L1 and L2 are identical, then return NIL.
    (b) Else if L1 is a variable, then if LI occurs in L2 then return {FAIL},
        else return (L2/L1).
    (c) Else if L2 is a variable then if L2 occurs in L1 then return {FAIL},
        else return (LI/L2)
    (d) Else return {FAIL}.
```

```
2. If the initial predicate symbols in LI and L2 are not identical, then return {FAIL}.
3. If LI and L2 have a different number of arguments, then return {FAIL}.
4. Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the substitutions used to unify L1 and L2).
5. For $i \leftarrow 1$ to number of arguments in LI:
        (a) Call Unify with the ith argument of LI and ith argument of L2, putting result in S.
        (b) If contains FAIL, then return {FAIL}.
        (c) If S not equal to NIL then:
                i. Apply S to the remainder of both L1 and L2.
                ii. SUBST= APPEND(S, SUBST).
6. Return SUBST.
```

Resolution in Predicate Logic

- With Unification, we now have an easy way of determining that two literals are contradictory, if one of them can be unified with the negation of the other.
- So, for example, man(x) and **¬**man(Spot) are contradictory, since man(x) and man(Spot) can be unified, with substitution x/spot.
- In order to use resolution for expressions in the predicate logic, we use the unification algorithm to locate pairs of literals that cancel out.
- For example, suppose we want to resolve two clauses:

  man(Marcus)

  ¬man(x1) V mortal(x1)

  The literal man(Marcus)can be unified with the literal **¬**man(x1) with the substitution **Marcus/x1.** we can now conclude only that **mortal(Marcus)** must be true.
- So the resolvent generated by clauses 1 and 2 must be **mortal (Marcus)**, which we get by applying me result of the unification process to the resolvent. The resolution process can then proceed from

there to discover whether **mortal (Marcus)** leads to a contradiction with other available clauses.

Algorithm: Resolution in predicate logic

1. Convert all the statements of F to clause form.

2. Negate *P* and convert the result to clause form. Add it to the-set of clauses obtained in 1.

3. Repeat until a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.

(a) Select two clauses. Call these the parent clauses.

(b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals TI and **¬**T2 such that one of the parent clauses contains T1 and the other contains **¬**T2 and if T1 and T2 are unifiable, then neither T1 nor T2 should appear in the resolvent. We call T1 and T2 **Complementary Literals**. Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.

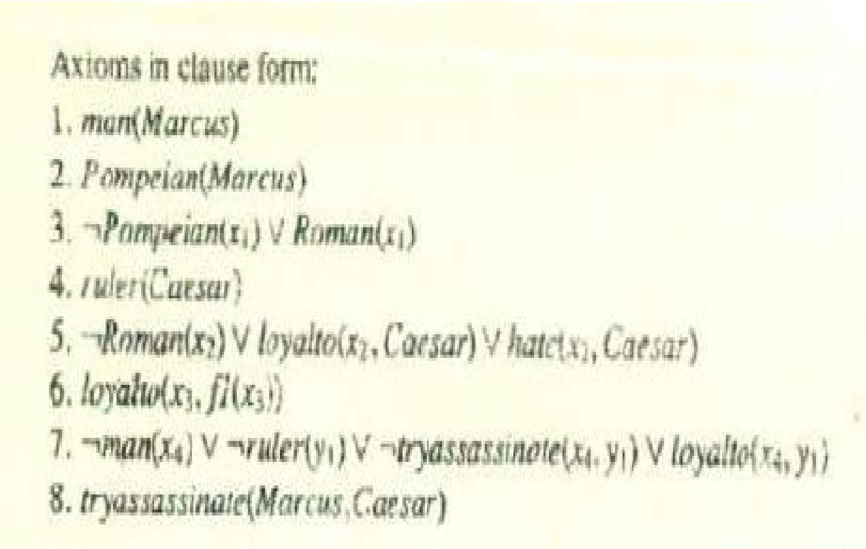
(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of c1auses available to the procedure.

- There are systematic strategies for making the choice of clauses to resolve together each step so that we will find a contradiction if one exists. This can speed up the process considerably.
  - Only resolve pairs of clauses that contain complementary literals, since only such resolutions produce new clauses that are harder to satisfy than their parents. To facilitate this, index clauses by the predicates they contain, combined with an indication of whether the predicate is negated Then, given a particular clause, possible resolvent that contain a

complementary occurrence of one of its predicates can be located directly.

- Eliminate certain clauses as soon as they are generated so that they cannot participate in later resolutions. Two kinds of clauses should be eliminated:
    - Tautologies (which can never be unsatisfied) and
    - Clauses that are subsumed by other clauses. For example, PVQ can be subsumed by P.
- Whenever possible, resolve either with one of- the clauses that is part of the statement we are trying to refute or with a clause generated by a resolution with such a clause. This is called the **“set-of-support strategy”.**
- Whenever possible, resolve with clauses that have a single literal. Such resolutions generate new clauses with fewer literals than the larger of their parent clauses and thus are probably closer to the goal of a resolvent with zero terms. This method is called the **“unit-preference-strategy”.**

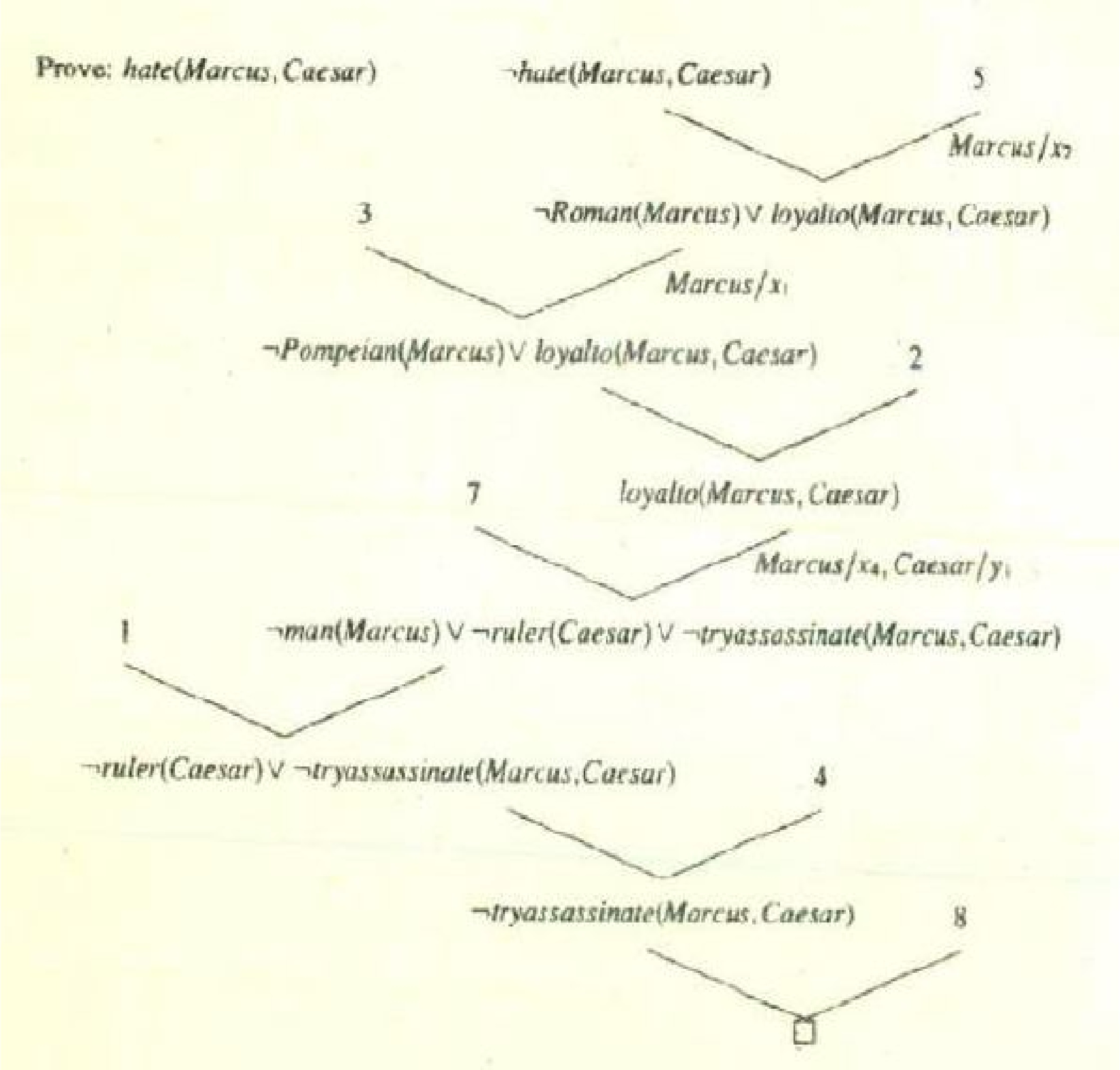
Example 1:



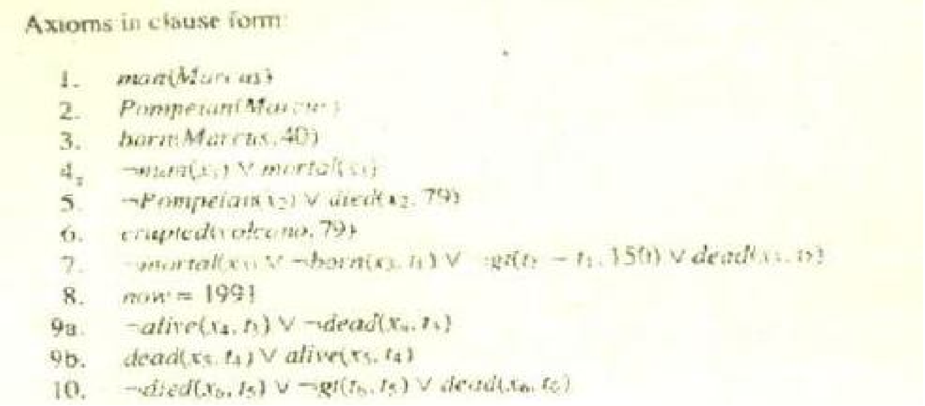
Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, f1(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

Resolution Proof:


Prove: *hate*(*Marcus*, *Caesar*)
¬*hate*(*Marcus*, *Caesar*)
5
Marcus/x_2
3
¬*Roman*(*Marcus*) ∨ *loyalto*(*Marcus*, *Caesar*)
Marcus/x_1
¬*Pompeian*(*Marcus*) ∨ *loyalto*(*Marcus*, *Caesar*)
2
7
loyalto(*Marcus*, *Caesar*)
Marcus/x_4, *Caesar*/y_1
1
¬*man*(*Marcus*) ∨ ¬*ruler*(*Caesar*) ∨ ¬*tryassassinate*(*Marcus*, *Caesar*)
¬*ruler*(*Caesar*) ∨ ¬*tryassassinate*(*Marcus*, *Caesar*)
4
¬*tryassassinate*(*Marcus*, *Caesar*)
8
□

Example 2:


Axioms in clause form:

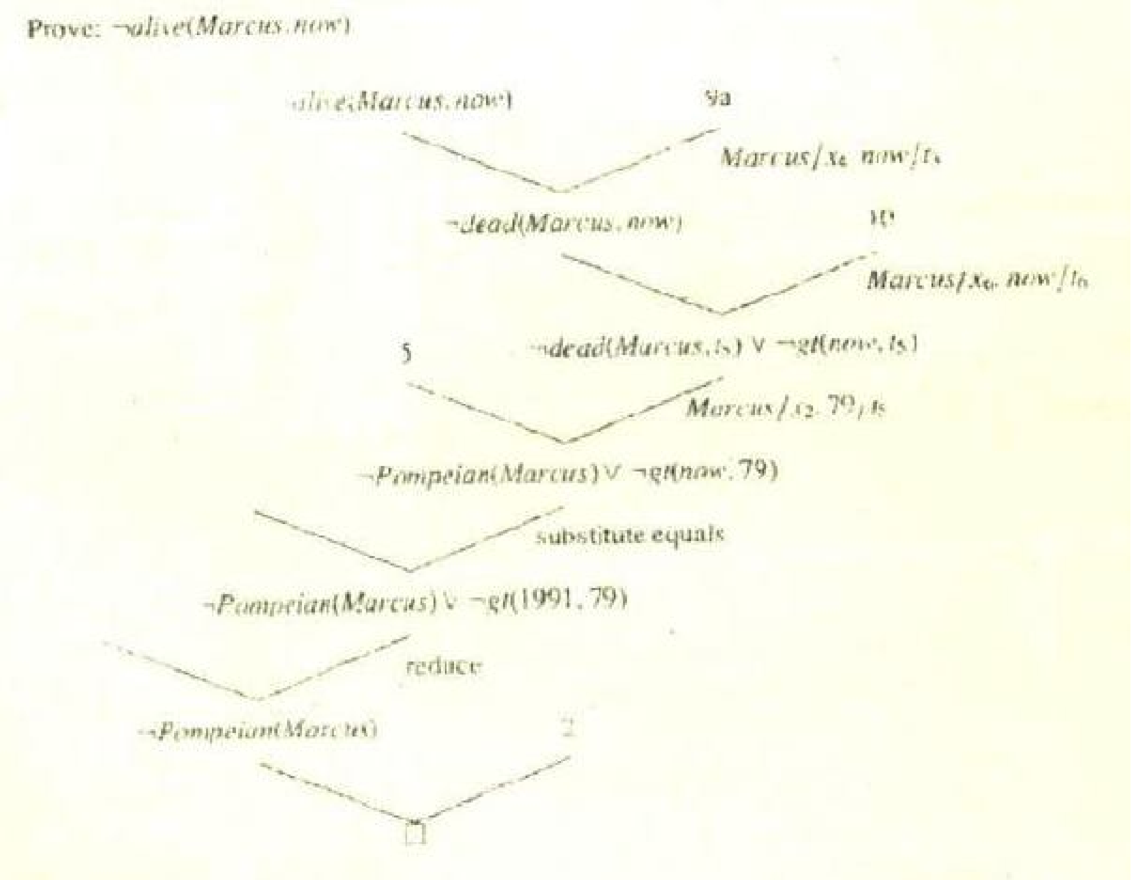
1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $\neg man(x_1) \vee mortal(x_1)$
5. $\neg Pompeian(x_2) \vee died(x_2, 79)$
6. $erupted(volcano, 79)$
7. $\neg mortal(x_3) \vee \neg born(x_3, t_1) \vee \neg gt(t_2 - t_1, 150) \vee dead(x_3, t_2)$
8. $now = 1991$

9a. $\neg alive(x_4, t_3) \vee \neg dead(x_4, t_3)$

9b. $dead(x_5, t_4) \vee alive(x_5, t_4)$

10. $\neg died(x_6, t_5) \vee \neg gt(t_6, t_5) \vee dead(x_6, t_6)$

Resolution Proof:



Natural Deduction:

- We used resolution as an easily implementable proof procedure that relies for its simplicity on a uniform representation of the statements it uses.
- Unfortunately, in uniformity, everything looks the same. Since everything looks the same, there is no easy way to select those statements that are the most likely to be useful in solving a particular problem.

- In converting everything to clause form, we often lose valuable heuristic information that is contained in the original representation of the facts.
- For example, suppose the fact: **All judges who are not crooked are well-educated**, can be represented as:

  $$\forall x : judge(x) \vee \neg crooked(x) \rightarrow educated(x)$$

  In this form, the statement suggests a way of deducing that someone is educated. But when the same statement is converted to clause form:

  $$\neg judge(x) \vee crooked(x) \vee educated(x)$$

  It is a way of deducing that someone is not a judge by showing that he is not crooked and not educated. Of course, in a logical sense, it is. But it is almost certainly not the best way, or even a very good way, to go about showing that someone is not a Judge. The heuristic information contained in the original statement has been lost in the transformation.
- Another problem with the use of resolution as the basis of a theorem-proving system is that people do not think in resolution. Thus it is very difficult for a person to interact with a resolution theorem prover, either to give it advice or to be given advice by it. Since proving very hard things is something that computers still do poorly, it is important from a practical standpoint that such interaction be possible.
- **Natural deduction** is describes a melange of techniques used in combination to solve problems that are not tractable by any one method alone.
- One common technique is to arrange knowledge, not by predicates, but rather by the objects involved in the predicates.

Unit- III
Artificial Intelligence: Logic Concepts

Assignment-Cum-Tutorial Questions

I) Objective Questions

1. List standard logic symbols.
2. Represent **"It is RAINING"** in propositional logic.
3. Real-world facts written as logical propositions are called ____________.

4. Represent **"Marcus was a man"** in propositional and predicate logic.
5. In _______________ logic decision procedure does not exist.[ ]

   (a) Propositional (b) Predicate (c) First order (d) none

6. ____________ and __________ attributes play an important role in process of reasoning.

7. ______________property uses the attributes isa and instance.[ ]

   (a) Polymorphism (b) Inheritance (c) Reasoning (d) both b & c

8. The attribute **instance** is _________ predicate. [ ]

   (a) Unary (b) Binary (c) Ternary (d) none

9. Resolution produces proofs by _________________**.** [ ]

   (a) Contradiction (b) Reasoning (c) Refutation (d) none

10. Define the term "Resolution".

11. Moving all quantifiers to the left of the formula without changing their relative order is called ___________________.

12. After resolving the below facts, we get___________________.

    winter V summer

    ¬winter V cold

13. The resultant clause after resolution is called_____________. [ ]

    (a) Implicant (b) Resolvent (c) fact (d) instance

14. The substitutions in Unification algorithm are applied from __. [ ]

(a) right to left (b) left to right (c) any order (d) none

15. Resolving clauses that have a single literal first is called ______________ strategy.

16. Resolving either with one of- the clauses that is part of the statement we are trying to refute or with a clause generated by a resolution with such a clause- This is called the ___________ strategy.

SECTION-B

SUBJECTIVE QUESTIONS

1. Represent the following sentences in predicate logic:

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
S, All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.
9. Answer the question: **Was Marcus loyal to Caesar?** using the above facts.

10. Represent the following facts using isa and instance attributes:

11. Marcus was a man.
12. Marcus was a Pompeian.
13. All Pompeians were Romans.
14. Caesar was a ruler.
15. All Romans were either loyal to Caesar or hated him.
16. Enumerate the steps in the process of resolution.

17. Explain the steps of converting a fact to Clause form.

18. Convert the following fact into clause form:

All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy.

19. Explain the algorithm for Propositional resolution.

20. Explain Unification Algorithm.

21. Explain the algorithm for resolution in predicate logic.

22. Determine the possible substitutions for unifying the following facts:

hate (x, y)
hate (Marcus, z)

23. Perform resolution in predicate logic:

Axioms in clause form:

1. man(Marcus)
2. Pompeian(Marcus)
3. $\neg Pompeian(X_1) \lor Roman(X_1)$
4. Tuler(Caesar)
5. $\neg Roman(X_2) \lor loyalto(X_2, \mathrm{Caesar}) \lor \mathrm{hate}(\mathrm{X}_2, Caesar)$
6. Loyalto(X_3,fl(X_3))
7. $\neg man(X_4) \lor \neg \mathrm{ruler}(\mathrm{Y}_1) \lor \neg tryassas \sin note(X_4, Y_1) \lor loyalto(X_4, \mathrm{Y}_1)$
8. Tryassassinote(Marcus,Caesar)

24. Explain about Natural Deduction?

UNIT – IV

Knowledge representation

Syllabus:

Introduction, approaches to knowledge representation, knowledge representation using semantic network, extended semantic networks for KR, knowledge representation using frames.

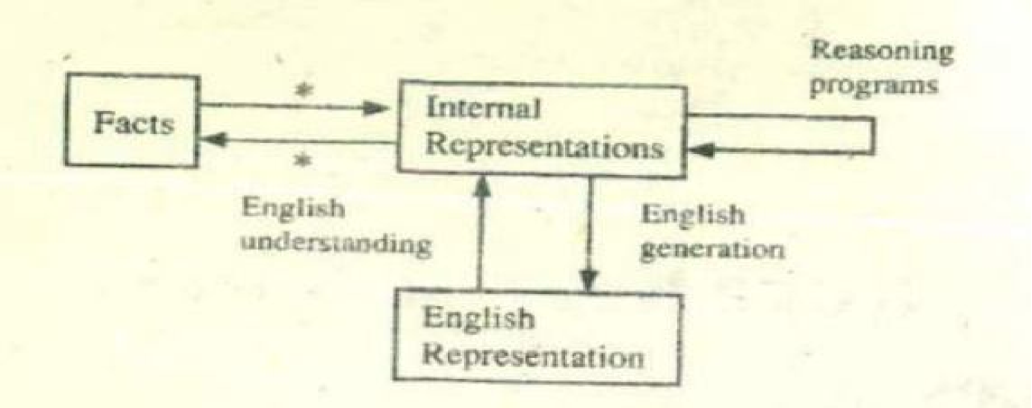
Advanced knowledge representation techniques: Introduction, conceptual dependency theory, script structure, semantic web.

Outcomes:

Student will be able to:

<u>Representations and Mappings</u>

- In order to solve the complex problems encountered in artificial intelligence; we need both a large amount of knowledge and some mechanisms for manipulating that knowledge to create solutions to new problems.
- A variety of ways of representing knowledge (facts) have been exploited in AI programs.
- We deal with two different kinds of entities:
    - **Facts:** truths in some relevant world, these are the things we want to represent.
    - **Representations of facts** in some chosen formalism. These are the things we will actually be able to manipulate.
- These entities are at two levels:
    - **The knowledge level**, at which facts (including each agent's behaviour. and current goals) are described.
    - **The symbol level**, at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.



- The two-way mappings exist between facts and representations called **“Representation mappings”**. The forward representation mapping maps from facts to representations. The backward representation mapping maps from representations to facts.
- One representation of facts is: **natural language (particularly English) sentences.** Regardless of the representation for facts that we use in a program, we may also need to be concerned with an English representation of those facts in order to facilitate getting information into and out of the system.
- In this case, we must have mapping function from English sentences to the representation we are actually going to use and back to sentences.
- Consider the English sentence:

  Spot is a dog.

  The fact represented by that English sentence can also be represented in logic as:

  dog (Spot)

  Suppose that we also have a logical representation of the fact that all dogs have tails:

  $\forall$x: dog (x)→hastail (x)

  Then, using the deductive mechanisms of logic, we may generate the new representation object:

  hastail(Spot)

Using an appropriate backward mapping function, we could then generate the English sentence:

Spot has a tail

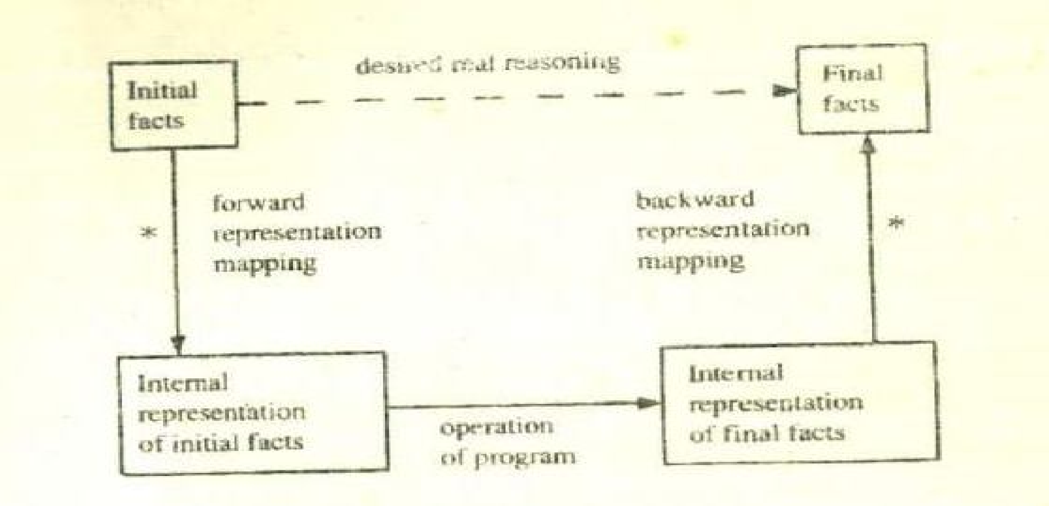
- The available mapping functions are not one-to-one; they are many . to-many relations. In other words, each object in the domain may map to several elements in the range, and several elements in the domain may map to the same element of the range.)
- Consider an example of the mappings involving English representations of facts. The two sentences:

  "All dogs have tails" and "Every dog has a tail"

  Both represent the same fact, namely that **every dog has at least one tail.**

  On the other hand, the former could represent either the tact that every dog has at least one tail or the fact that each dog has several tails.

  The latter may represent either the fact that every dog has at least one tail or the fact that there is a tail that every dog has.
- So, when we try to convert English sentences into some other representation, such as logical propositions, we must first decide what facts the sentences represent and then convert those facts into the new representation.

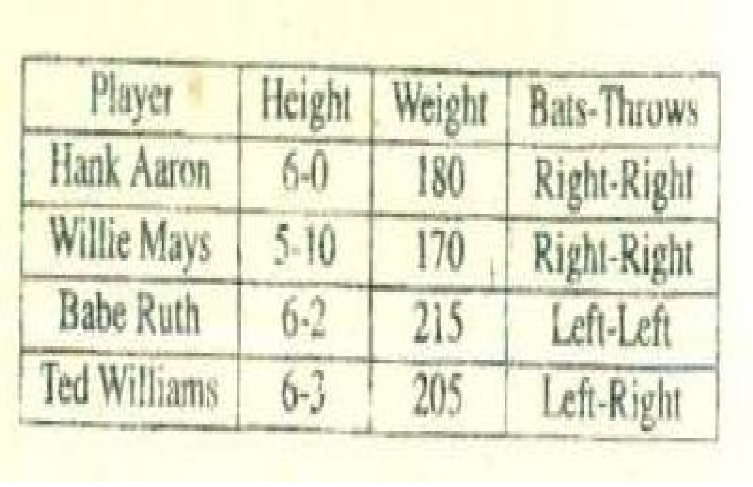

desired real reasoning
Initial facts
Final facts
*
forward representation mapping
backward representation mapping
*
Internal representation of initial facts
operation of program
Internal representation of final facts

Approaches to Knowledge Representation

- ➢ A good system for the representation of knowledge in a particular domain should possess the following four properties:
    - **Representational Adequacy:** the ability to represent all of the kinds of knowledge that are needed in that domain.
    - **Inferential Adequacy:** the ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.
    - **Inferential efficiency:** the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions.
    - **Acquisitional Efficiency:** the ability to acquire new information easily.
- ➢ But, no single system that optimizes all of the capabilities for all kinds of knowledge has yet been found. As a result, multiple techniques for knowledge representation exist. Many programs rely on more than one technique. Following are some knowledge representation techniques:

Simple Relational Knowledge:

- ➢ The simplest way to represent declarative facts is as a set of relations of the same sort used in database systems.

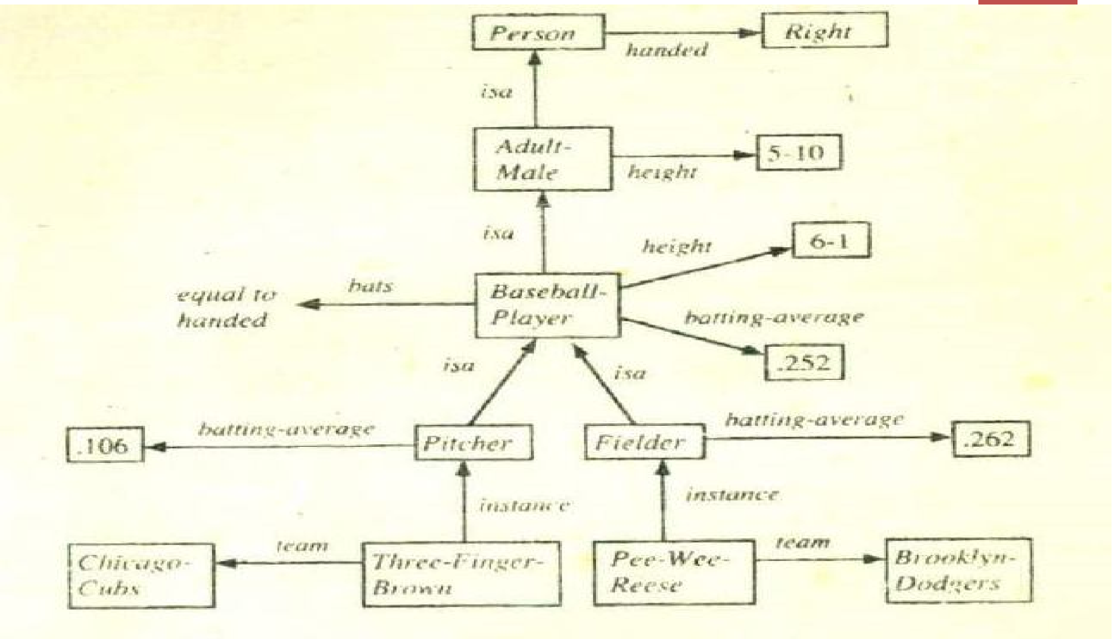


Player	Height	Weight	Bats-Throws
Hank Aaron	6-0	180	Right-Right
Willie Mays	5-10	170	Right-Right
Babe Ruth	6-2	215	Left-Left
Ted Williams	6-3	205	Left-Right

- The reason that this representation is simple is that standing alone it provides very weak inferential capabilities, but knowledge represented in this form may serve as the input to more powerful inference engines.
- The relational knowledge corresponds to a set or attributes and associated values that together describe the objects of the knowledge base.
- Providing support for relational knowledge is what database systems are designed to do.

Inheritable Knowledge

- Knowledge about objects, their attributes, and their values need not be as simple as that shown in relational knowledge.
- In particular, it is possible to augment the basic representation with inference mechanisms that operate on the structure of the representation.
- For this to be effective, the structure must be designed to correspond to the inference mechanisms that are desired.
- One of the most useful forms of inference is property **inheritance**, in which elements of specific classes inherit attributes and values from more general classes in which they are included.
- In order to support property inheritance, objects must be organized into classes and classes must be arranged in a generalization hierarchy.
- Lines represent attributes. Boxed nodes represent objects and values of attributes of objects. The arrows on the lines point from an object to its value along the corresponding attribute line. The structure called **slot-and-filler** structure.



- Each individual frame represents the collection of attributes and values associated with a particular node. This structure is called a **Semantic network or a collection of Frames.**

Baseball-Player isa: Adult-Male
bars: (EQUAL handed)
height: 6-1
batting-average: .252

Basic mechanism of inheritance:

Algorithm: Property Inheritance

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

1. Find O in the knowledge base.

2. If there is a value there for the attribute A, report that value.

3. Otherwise, see if there is a value for the attribute instance. If not, then fail.

4. Otherwise, move to the node corresponding to that value and look for a value for the attribute A. If one is found, report it.

5. Otherwise, do until there is no value for the isa attribute or until an answer is found:

(a) Get the value of the isa attribute and move to that node.

(b) See if there is value for the attribute A. If there is, repori it.

- We can apply this procedure to our example knowledge base to derive answers to the following queries:
  - **team (Pee-Wee-Reese) = Brooklyn-fladgers.** This attribute had a value stored explicitly in the knowledge base.
  - **batting-average (Three- Finger-Brown) = .106**. Since there is no value for batting average stored explicitly for Three Finger Brown, we follow the instance attribute to Pitcher and extract the value stored there.
  - **height (Pee-Wee-Reese) = 6-1**. This represents another default inference. Notice here that because we get to it first, the more specific tact about the height of baseball players overrides a more general fact about the height of adult males.
  - **bats (Three- Finger-Brown) =Right.** To get a value for the attribute bats required going up the isa hierarchy to the class Baseball-Player. But what we found there was not a value but a rule for computing a value. This rule required another value as input. So the entire process must be begun again recursively to find a value for **handed**. This time, it is necessary to go all the way up to Person to discover that the default value for handedness for people is Right. Now the rule for bats can be applied, producing the result Right.

Inferential Knowledge

- The power of traditional logic is necessary to describe the inferences that are needed.

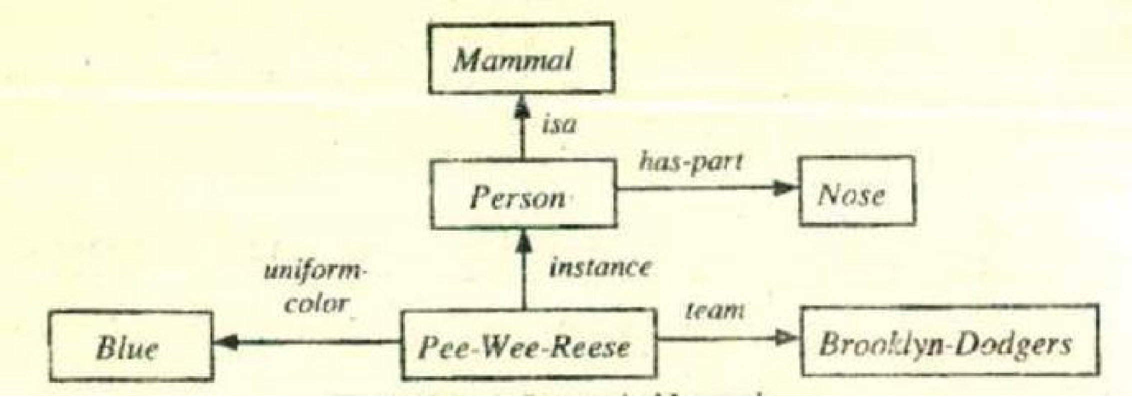
- This knowledge is useless unless there is also an inference procedure that can exploit it The required inference procedure is one that implements the standard logical rules of inference.
- The procedures, may reason forward from given facts to conclusions, or may reason backward from desired conclusions to given facts. One of the most commonly used of these procedures is **Resolution**, which exploits a proof by contradiction strategy.

Procedural Knowledge

- This kind of knowledge is operational.
- Procedural knowledge specifies what to do and when to do.
- Procedural knowledge can be represented in programs in many ways. The most common way is simply as code (in some programming language such as LISP) for doing something. The machine uses the knowledge when it executes the code to perform a task.
- But, this way of representing procedural knowledge gets low scores for the following properties:
    - **Inferential adequacy**, because it is very difficult to write a program that can reason about another programs behaviour.
    - **Acquisitional efficiency**, because the process of updating and debugging large pieces of code becomes unwieldy.
- The most commonly used technique for representing procedural knowledge in AI Programs is the use of **production rules.**

Semantic Nets

- The main idea behind semantic nets is that the meaning of a concept comes from the ways in which it is connected to other concepts.
- In a semantic net, information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationships among the nodes.



- This network contains examples of both the isa and instance relations, and domain-specific relations like team and uniform-color. In this network, we could use inheritance to derive the additional relation:

  has-part (Pee- Wee-Reese, Nose)
- Earlier, semantic nets were used to find relationships among objects by spreading activation out from each of two nodes and seeing where the activation met. This process is called **intersection search.**
- Using this process, it is possible to use the network to answer questions such as **"What is the connection between the Brooklyn Dodgers and blue?"** This kind of reasoning exploits one of the important advantages that slot-and-filler structures have over purely logical representations because it takes **advantage of the entity-based organization of knowledge** that slot-and-filler representations provide.

Representing Non binary Predicates:

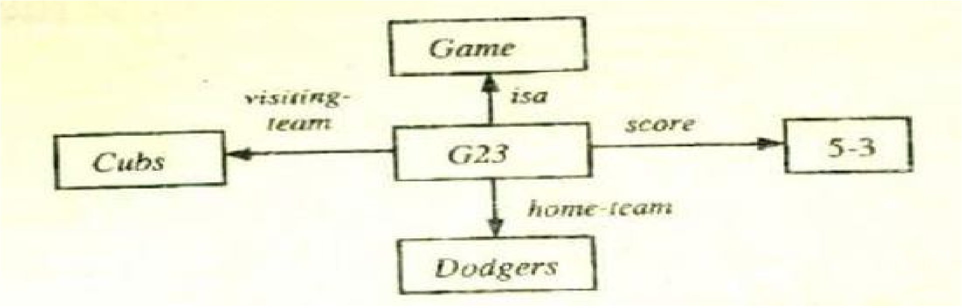
- Semantic nets are a natural way to represent relationships that would appear as ground instances of binary predicates in predicate logic.
- Some of the arcs can be represented in logic as:
  - isa(Person, mammal)
  - instance(Pee-Wee-Reese, Person)
  - team(Pee-Wee-Reese, Brooklyn- Dodgers)

- uniform color(Pee-Wee-Reese, Blue)

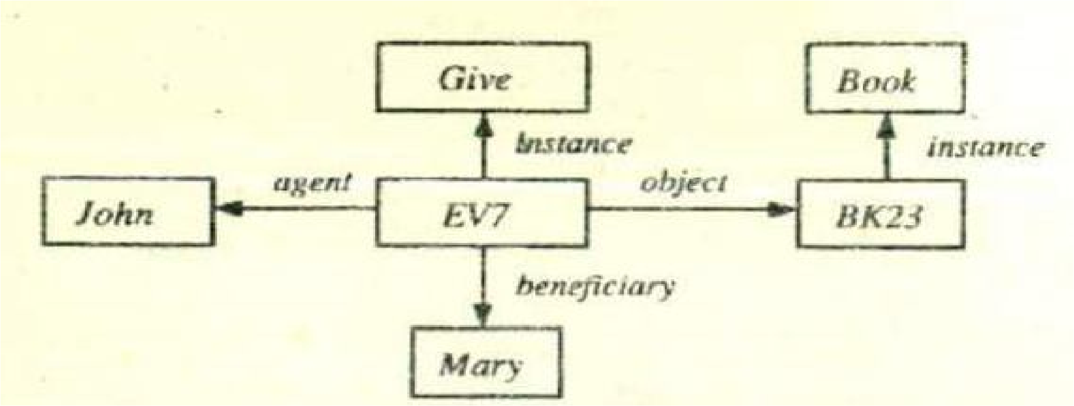
- Unary predicates in logic can be thought of as binary predicates using general-purpose predicates, such as **isa and instance**. For example:

  Man (Marcus) could be rewritten as **instance (Marcus, Man)**

- Three or more place predicates can also be converted to a binary form by creating one new object representing the entire predicate statement and then introducing binary predicates to describe the relationship to this new object of each of the original arguments.
- For example, **score (Cubs, Dodger, 5-3)**
  This can be represented in a semantic net by creating a node to represent the specific game and then relating each of the three pieces of information to it.

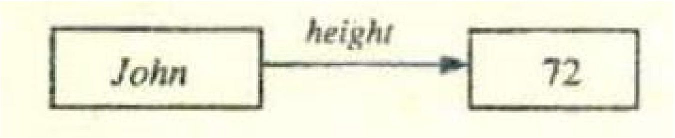


- The sentence: **John gave the book to Mary** could be represented by



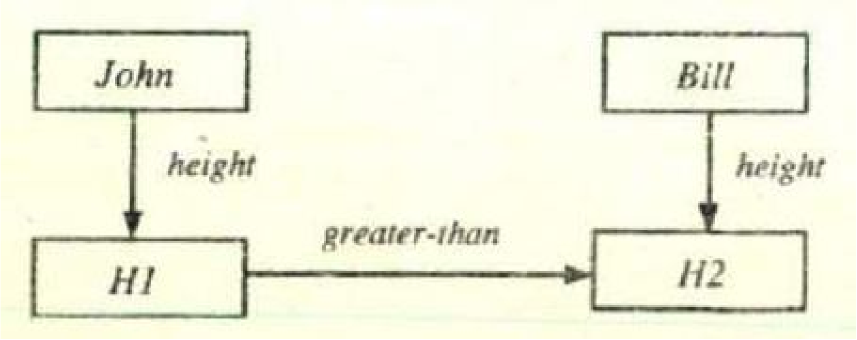
the network as follows:

- There should be a difference between a link that defines a new entity and one that relates two existing entities. Consider the net:



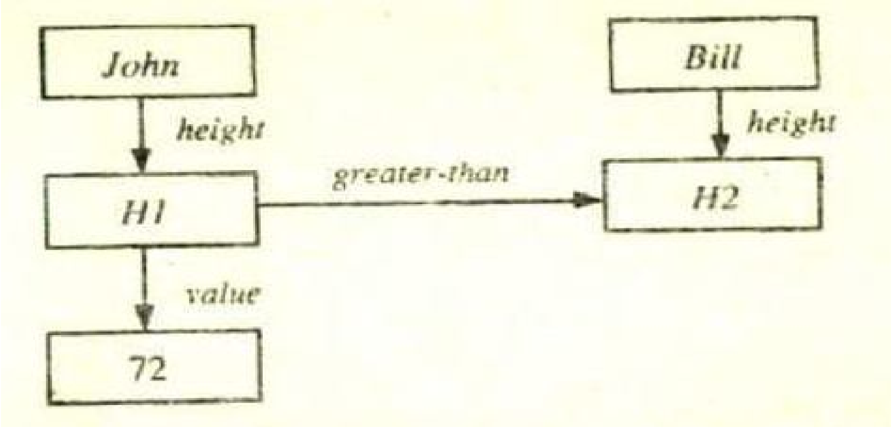
Both nodes represent objects that exist independently of their relationship to each other.

- Suppose we want to represent the fact that: **John is taller than Bill,** using the net:



The nodes HI and 112 are new concepts representing John's height and Bill's height, respectively. They are defined by their relationships to the nodes John and Hill.

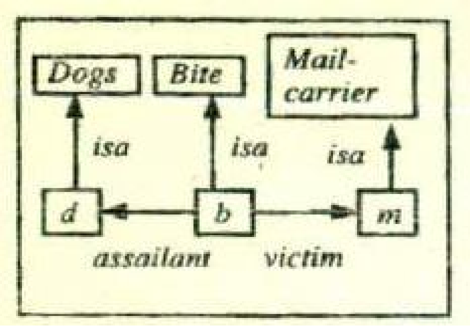
- Sometimes it is useful to introduce the arc value to make this distinction clear. Thus we might use the following net to represent the fact that **John is 6 feet tall and that he is taller than Bill.**



Extended semantic networks for KR

- **Partitioned Semantic Networks** allow for:
  - Propositions to be made without commitment to truth.
  - Expressions to be quantified.
- The basic idea is to break network into spaces which consist of groups of nodes and arcs and regard each space as a node.
- By partitioning the semantic net into a hierarchical set of spaces, each partition corresponds to the scope of one or more variables.
- Consider the statement: **The dog bit the mail carrier.**

  The nodes Dogs, Bite, and Mail-Carrier represent the classes of dogs, bitings and mail carriers. Respectively, while the nodes d, b and m represent a particular dog, a particular biting and a particular mail carrier. This fact can easily be represented by a single net with no partitioning.

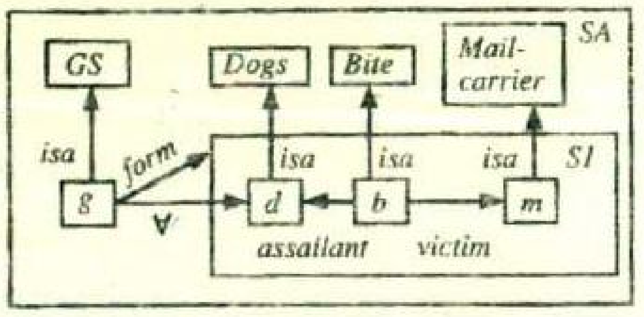


- But now suppose that we want to represent the fact:

  Every dog has bitten a mail carrier.

  ∀x: Dog(x) → ∃y: Mail-Carrier(y) Λ Bite(x, y)

  To represent this fact, it is necessary to encode the scope of the universally quantified variable x. This can be done using partitioning net.

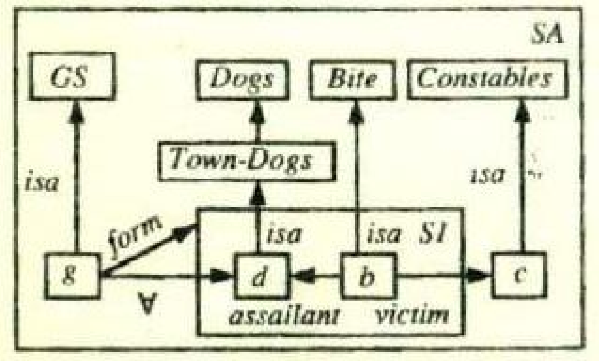


Node g is an instance of the special class GS of general statements about the world (i.e.. those with universal quantifiers). Every, element of GS has at least two attributes: a form, which states the relation that is being asserted, and one or more ∀ connections, one for each of the universally quantified variables. In this example, there is only one such variable d, which can stand for any element of the class Dogs. The other two variables in the form, b and m are existentially quantified. In other words, for every dog d, there exists a biting event b, and a mail carrier, m. such that d is the assailant of b and m is the victim.

- Consider the sentence:

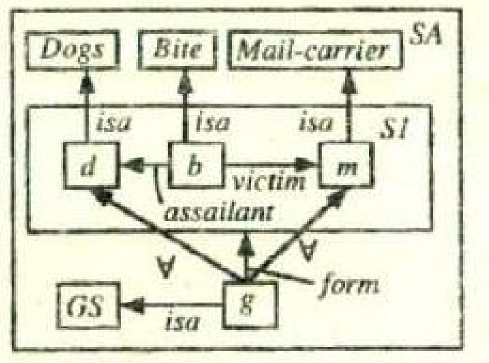
  Every dog in town has bitten the constable.

  In this net, the node c representing the victim lies outside the form of the general statement. Thus it is not viewed as an existentially quantified variable whose value may depend on the value of d.



- Consider the sentence:

Every dog has bitten every mail carrier.



In this case, g has two ∀ links: one pointing to d, which represents any dog, and one pointing to m, representing any mail carrier.

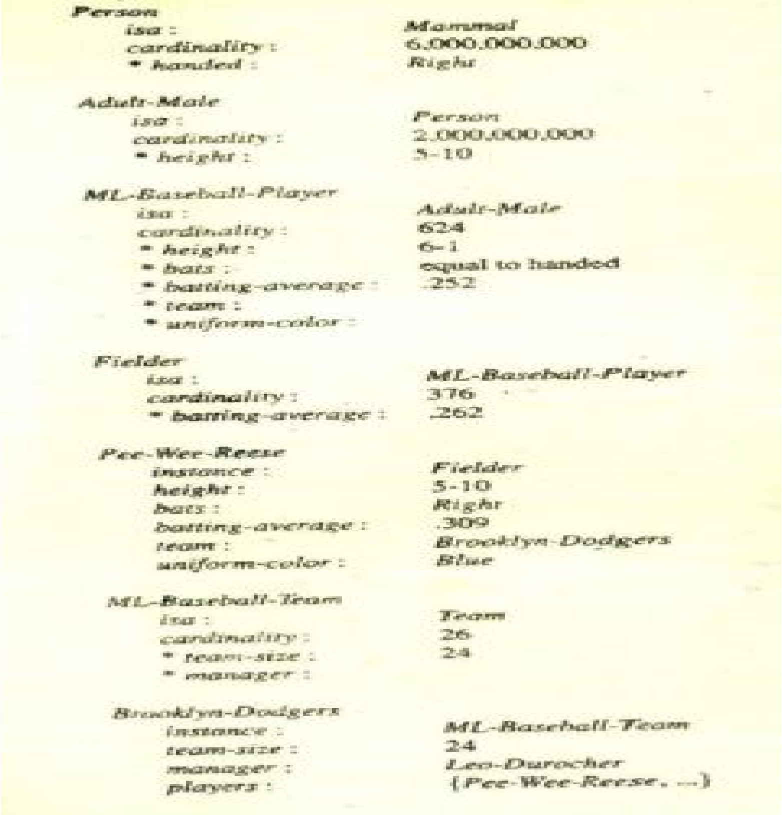
Space S is included in space SA. Whenever is search process operates in a partitioned semantic net, it can explore nodes and arcs in the space from which it starts and in other spaces that contain the starting point, but it cannot go downward, except in special circumstances, such as when a form arc is being traversed. From node d, it can be determined that d must be a dog. But if we were to start, at the node Dogs and search for all known instances of dogs by traversing isa links, we would not find d since it and the link to it are in the space S1, which is at a lower level than space S4, which contains Dogs. This is important, since d does not stand for a particular dog; it is merely a variable that can be instantiated with a value that represents a dog.

Frames

- A frame is a collection of attributes (usually called slots) and associated values (and possibly constraints on values) that describes some entity in the world.
- A single frame taken alone is rarely useful. Instead, we build **frame systems** out of collection of frames that are connected to each other. The value of an attribute of one frame may be another frame.

Frames as Sets and Instances:

- Set theory provides a good basis for understanding frame systems. Each frame represents either a class (a set) or an instance (an element of a class).
- Consider the following frame system:



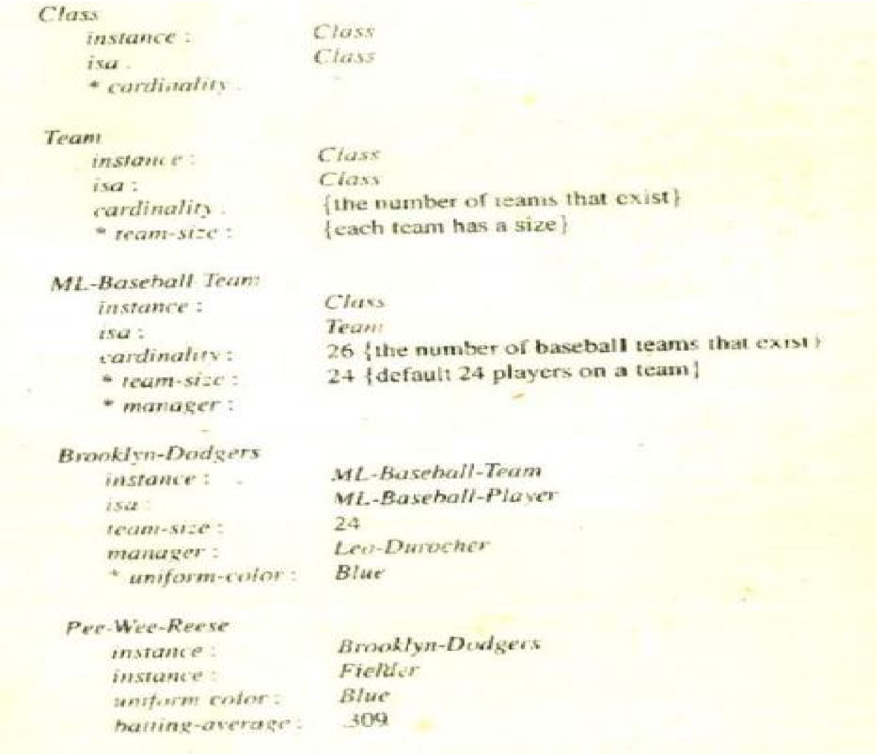
```
Person
    isa :                       Mammal
    cardinality :               6,000,000,000
    * handed :                  Right

Adult-Male
    isa :                       Person
    cardinality :               2,000,000,000
    * height :                  5-10

ML-Baseball-Player
    isa :                       Adult-Male
    cardinality :               624
    * height :                  6-1
    * bats :                    equal to handed
    * batting-average :         .252
    * team :
    * uniform-color :

Fielder
    isa :                       ML-Baseball-Player
    cardinality :               376
    * batting-average :         .262

Pee-Wee-Reese
    instance :                  Fielder
    height :                    5-10
    bats :                      Right
    batting-average :           .309
    team :                      Brooklyn-Dodgers
    uniform-color :             Blue

ML-Baseball-Team
    isa :                       Team
    cardinality :               26
    * team-size :               24
    * manager :

Brooklyn-Dodgers
    instance :                  ML-Baseball-Team
    team-size :                 24
    manager :                   Leo-Durocher
    players :                   {Pee-Wee-Reese, ...}
```

The frames Person, Adult-Male, ML-Baseball Player (corresponding to major league baseball players), Pitcher, and ML-Baseball-Team (for major league baseball team) are all classes. The frames Pee-Wee-Reese and Brooklyn-Dodgers are instances.

- ➢ The set of adult males is a subset of the set of people. The set of major league baseball players is a subset of the set of adult males, and so forth. The instance relation corresponds to the relation “**element-of”**. Pee Wee Reese is an element of the set of fielders. Thus he is also an element of all of the supersets of fielders, including major league baseball players and people.
- ➢ A class represents a set; there are two kinds of attributes that can be associated with it. They are:
    - attributes about the set itself, and
    - attributes that are to be inherited-by each element of the set

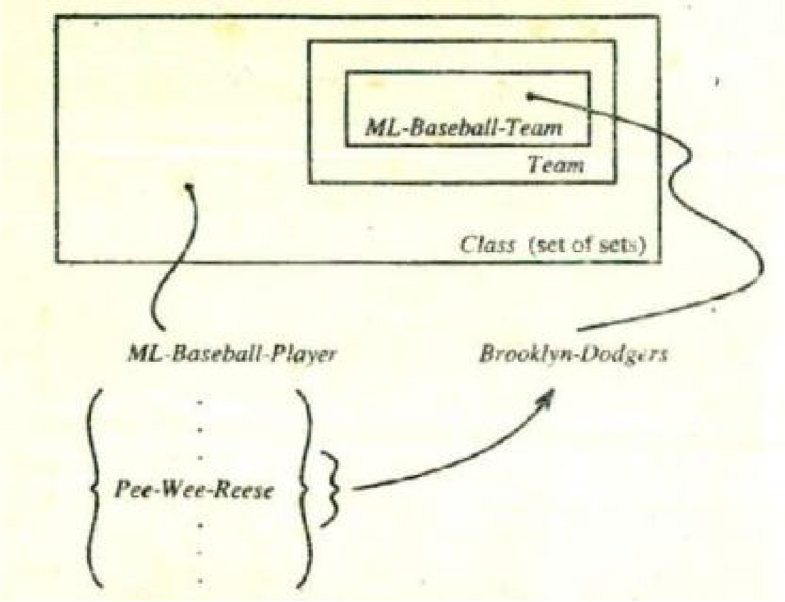
We indicate the difference between these two by prefixing the latter with an asterisk (*).For example, consider the class **ML-Basebalf-Player**.

- It is important to distinguish between **regular classes**, whose elements are individual entities, and **metaclasses**, which are special classes whose elements are themselves classes. A class is now an element of (instance) some class (or classes) as well as a subclass (isa) of one or more classes. A class inherits properties from the class of which it is an instance, just as any instance does. In addition, a class passes inheritable properties down from its super classes to its instances.



```
Class
    instance :            Class
    isa :                 Class
    * cardinality :

Team
    instance :            Class
    isa :                 Class
    cardinality :         {the number of teams that exist}
    * team-size :         {each team has a size}

ML-Baseball Team:
    instance :            Class
    isa :                 Team
    cardinality :         26 {the number of baseball teams that exist}
    * team-size :         24 {default 24 players on a team}
    * manager :

Brooklyn-Dodgers
    instance :            ML-Baseball-Team
    isa :                 ML-Baseball-Player
    team-size :           24
    manager :             Leo-Durocher
    * uniform-color :     Blue

Pee-Wee-Reese
    instance :            Brooklyn-Dodgers
    instance :            Fielder
    uniform color :       Blue
    batting-average :     .309
```

The most basic **metaclass** is the class **Class**. It represents the set of all classes. All classes are instances of it, either directly or through one of its subclasses. In the example, **Team** is a subclass (subset) of **Class** and **ML-Baseball-Team** is a subclass of **Team**. The class **Class** introduces the attribute cardinality, which is to be inherited by all instances of Class (including itself).

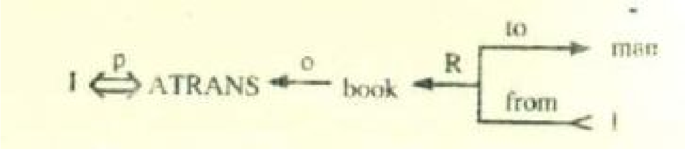


- Ways in which classes are related to each other:
  - Class1 can be a subset of Class2.
  - If Class2 is a metaclass, then Class1 can be an instance of Class2.
  - **Mutually-disjoint-with:** which relates a class to one or more other classes that are guaranteed to have no elements in common with it.
  - **Is-covered-by:** which relates a class to a set of subclasses, the union of which is equal to it. If a class is covered-by a set of mutually disjoint classes, then S is called a partition of the class.
- The following are the properties the frames would be able to represent and use in reasoning:
  - The classes to which the attribute can be attached
  - Constraints on either the type or the value of the attribute.
  - A value that all instances of a class must have by the definition of the class.
  - A default value for the attribute.
  - Rules for inheriting values for the attribute. The usual rule is to inherit down isa and instance links.

- Rules for computing a value separately from inheritance.

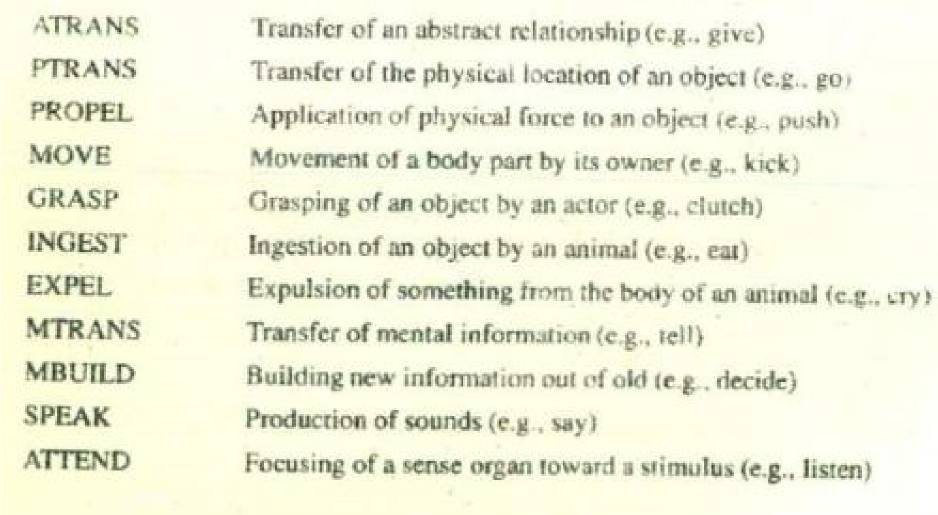
Conceptual Dependency

- Semantic networks and frame systems may have specialized links and inference procedures, but there are no hard and fast rules about what kinds of objects and links arc good in general for knowledge representation.
- Conceptual Dependency specifies what types of objects and relations are permitted.
- Conceptual Dependency (CD) is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences. The goal is to represent the knowledge in a way that:
  - Facilitates drawing inferences from the sentences.
  - Is independent of the language in which the sentences were originally stated.
- The CD representation of a sentence is built not out of primitives corresponding to the words used in the sentence, but rather out of conceptual primitives that can be combined to form the meanings of words in any particular language.
- Semantic nets provide only a structure into which nodes representing information at any level can be placed. Conceptual dependency provides both a **structure and a specific set of primitives**, at a particular level of granularity out of which representations of particular pieces information can be constructed.
- Consider a Simple Conceptual Dependency Representation of : **I gave the man a book,** where the symbols have the following meanings:



- Arrows indicate direction of dependency.
- Double arrow indicates two way links between actor and action.
- p indicates past tense.
- ATRANS is one of the primitive acts used by the theory. It indicates transfer of possession.
- O indicates the object case relation.
- R indicates the recipient case relation.

➢ In CD, representations of actions are built from a set of primitive acts:



ATRANS	Transfer of an abstract relationship (e.g., give)
PTRANS	Transfer of the physical location of an object (e.g., go)
PROPEL	Application of physical force to an object (e.g., push)
MOVE	Movement of a body part by its owner (e.g., kick)
GRASP	Grasping of an object by an actor (e.g., clutch)
INGEST	Ingestion of an object by an animal (e.g., eat)
EXPEL	Expulsion of something from the body of an animal (e.g., cry)
MTRANS	Transfer of mental information (e.g., tell)
MBUILD	Building new information out of old (e.g., decide)
SPEAK	Production of sounds (e.g., say)
ATTEND	Focusing of a sense organ toward a stimulus (e.g., listen)

➢ The set of CD building blocks is the set of allowable dependencies among the conceptualizations described in a sentence. There are tour primitive conceptual categories from which dependency structures can be built. These are:

ACTS	Actions
PPs	Objects (picture producers)
AAs	Modifiers of actions (action aiders)
PAs	Modifiers of PPs (picture aiders)

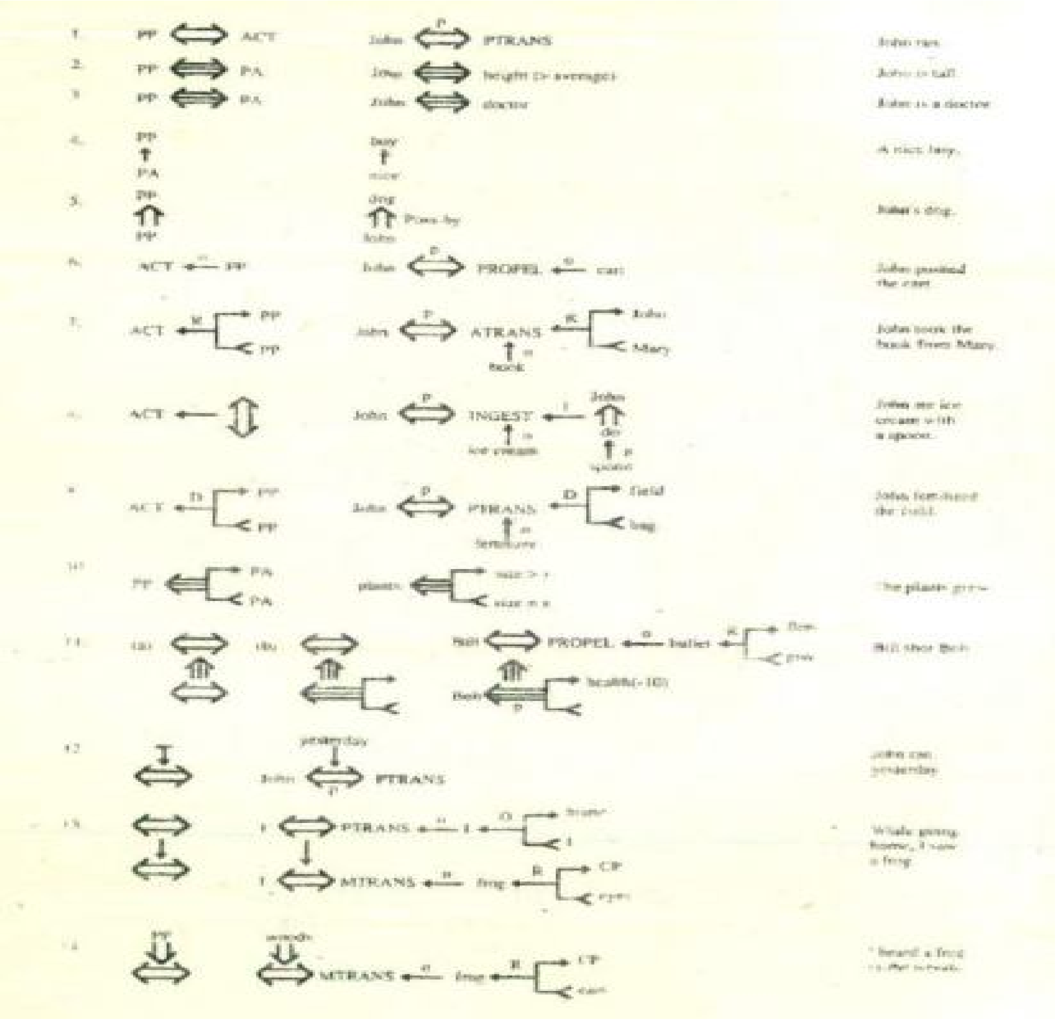
➢ The Dependencies of CD:

- Rule 1 describes the relationship between an actor and the event he or she causes. This is a two-way dependency since neither actor nor event can be considered primary. The letter p above the dependency link indicates past tense.
- Rule 2 describes the relationship between a PP and a PA that is being asserted to describe it.
- Rule 3 describes the relationship between two PPs, one of which belongs to the set defined by the other.
- Rule 4 describes the relationship a PP and an attribute that has already been predicated of it. The direction of the arrow is toward the PP being described.
- Rule 5 describes the relationship between two PPs, one of which provides a particular kind of information about the other. The three most common types of information to be provided in this way are:
  - possession (shown as POSSBY)
  - location (shown as LOC), and
  - physical containment (shown as CONT)
  - The direction of is again toward the concept being described
- Rule 6 describes the relationship between an ACT and the PP that is the object oh that ACT. The direction of the arrow is toward the ACT since the context of the specific ACT determines the meaning of the object relation.
- Rule 7 describes the relationship between an ACT and the source and the recipient of the ACT.
- Rule 8 describes the relationship between an ACT and the instrument with which it is performed. The instrument must always be a full conceptualization, not just a single physical object.
- Rule 9 describes the relationship between an ACT and its physical source and destination.
- Rule 10 represents the relationship between a PP and a state in which it started and another in which it ended.

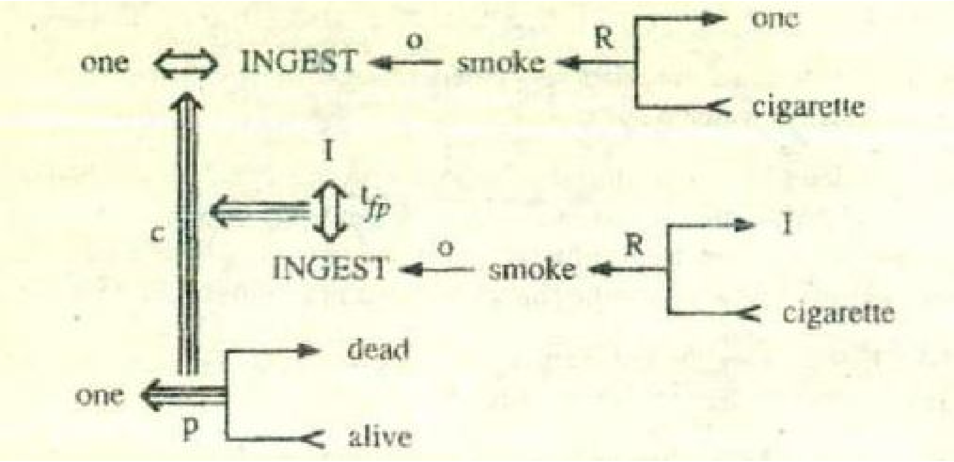
- Rule 10 describes the relationship between one conceptualization and another that causes it. The arrows indicate dependency of one conceptualization on another and point in the opposite direction of the implication arrows. The two forms of the rule describe the cause of an action and the cause of a state change.
- Rule 12 describes the relationship between a conceptualization and the time at which the event it describes occurred.
- Rule 13 describes the relation between one conceptualization and another that is the time of the first.
- Rule 14 describes the relationship between a conceptualization and the place at which it occurred.

➢ The set of conceptual tenses in a CD are:

p	Past
f	Future
t	Transition
t_x	Start transition
t_y	Finished transition
k	Continuing
?	Interrogative
nil	Present
delta	Timeless
c	Conditional

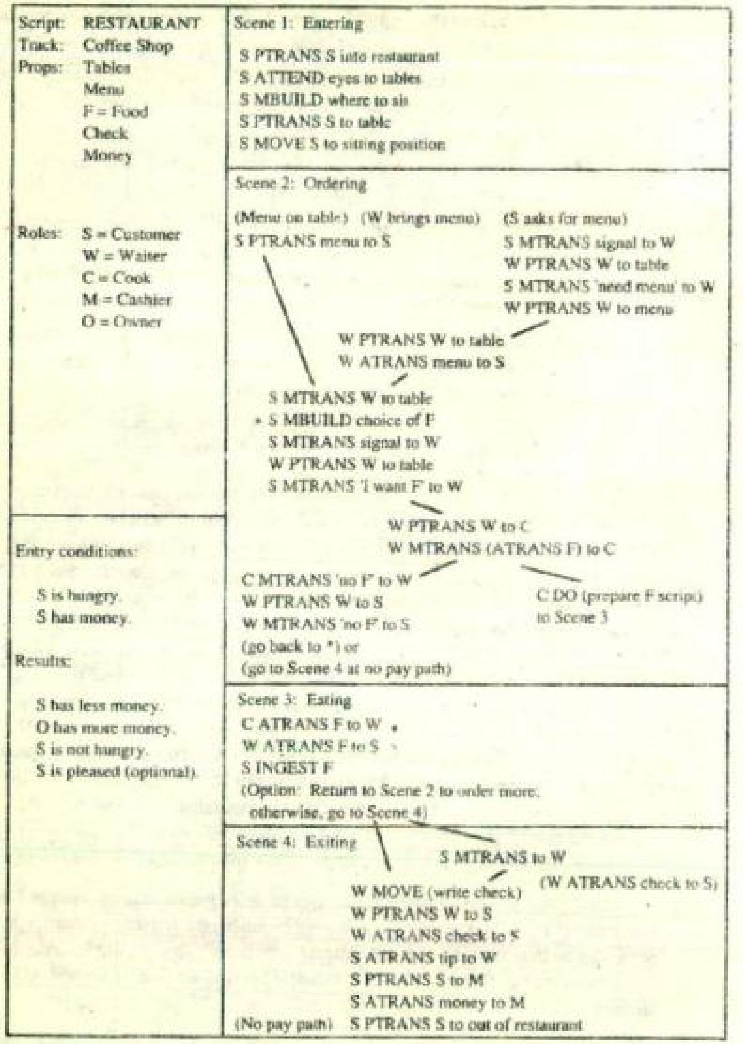


➢ Consider the sentence: **Since smoking can kill you, I stopped.**



Scripts

- CD is a mechanism for representing and reasoning about events. A script is a structure that describes a stereotyped sequence of events in a particular context.
- Script is a mechanism for representing knowledge about common sequences of events.
- A script consists of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available.
- Consider the following example of a restaurant script:



Script: RESTAURANT
Track: Coffee Shop
Props: Tables
Menu
F = Food
Check
Money

Roles: S = Customer
W = Waiter
C = Cook
M = Cashier
O = Owner

Entry conditions:

S is hungry.
S has money.

Results:

S has less money.
O has more money.
S is not hungry.
S is pleased (optional).

Scene 1: Entering

S PTRANS S into restaurant
S ATTEND eyes to tables
S MBUILD where to sit
S PTRANS S to table
S MOVE S to sitting position

Scene 2: Ordering

(Menu on table) (W brings menu) (S asks for menu)
S PTRANS menu to S
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'need menu' to W
W PTRANS W to menu
W PTRANS W to table
W ATRANS menu to S
S MTRANS W to table
* S MBUILD choice of F
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'I want F' to W
W PTRANS W to C
W MTRANS (ATRANS F) to C
C MTRANS 'no F' to W
W PTRANS W to S
W MTRANS 'no F' to S
(go back to *) or
(go to Scene 4 at no pay path)
C DO (prepare F script)
to Scene 3

Scene 3: Eating

C ATRANS F to W
W ATRANS F to S
S INGEST F
(Option: Return to Scene 2 to order more;
otherwise, go to Scene 4)

Scene 4: Exiting

S MTRANS to W
(W ATRANS check to S)
W MOVE (write check)
W PTRANS W to S
W ATRANS check to S
S ATRANS tip to W
S PTRANS S to M
S ATRANS money to M
(No pay path) S PTRANS S to out of restaurant

- The important components of a script are:
  - **Entry conditions:** Conditions that must, be satisfied before the events described in the script can occur.
  - **Result:** Conditions that will, be true after the events described in the script have occurred.
  - **Props:** Slots representing objects that are involved in the events described in the script. The presence of these objects can be inferred even if they are not mentioned explicitly.
  - **Role:** Slots representing people who are involved in the events described in the script.
  - **Track:** The specific variation on a more general pattern that is represented by this particular script. Different tracks of the same script will share many but not all components.
  - **Scenes:** The actual sequences of events that occur.

- Scripts are useful because, in the real world, there are patterns to the occurrence of events. These patterns arise because of causal relationships between events.
- Agents will perform one action so that they will then be able to perform another. The events described in a script form a giant causal chain.
- The beginning of the chain is the set of entry conditions which enable the first events of the script to occur. The end of the chain is the set of results which may enable later events or event sequences to occur. Within the chain, events are connected both to earlier events that make them possible and to later events that they enable.
- Scripts can also be used to indicate how events that were mentioned relate to each other.
- Before a particular script can be applied, it must be activated. There are two ways in which a script can be activated:
  - For **fleeting scripts** (ones that are mentioned briefly and may he referred to again but are not central to the situation), it may be

sufficient to store a pointer to the script so that it can be accessed later if necessary.

- For **non-fleeting scripts** it is appropriate to activate the script fully and to attempt to fill in its slots with particular objects and people involved in the current situation.

➢ Once a script has been activated, there are varieties of ways in which it can be useful in interpreting a particular situation. The most important of these is the **ability to predict events that have not explicitly been observed**.

For example:

> John went out to a restaurant last night. He ordered steak. When he paid for it, he noticed that he was running out of money. He hurried home since it had started to rain.

If you were then asked the question:

Did John eat dinner last night?

By using the restaurant script, a computer question-answerer would also be able to infer that John ate dinner, since the restaurant script could have been activated. Since all of the events in the story correspond to the sequence of events predicted by the script, the program could infer that the entire sequence predicted by the script occurred normally. Thus it could conclude that John ate.

➢ Scripts provide a way of building a **single coherent interpretation from a collection of observations.** Consider, for example:

> Susan went out to lunch. Site sat down at table and called the waitress. The waitress brought her a menu and she ordered a hamburger.

Now consider the question:

> **Why did the waitress bring Susan a menu?**

The script provides two possible answers to that question:

- Because Susan asked her to.
- So that Susan could decide what she wanted to eat.

Unit- IV
Assignment-Cum-Tutorial Questions

Objective Questions

1. The two different kinds of entities in Ai are ____________ and _____________.

2. The _________ mappings exist between facts and representations.

(a) One-way (b) two-way (c) no mapping (d) both (a) & (b) [ ]

3. The forward representation mapping maps from ____________ to _________________. [ ]

(a) representations, facts (b) facts, representations

(c) facts, facts (d) representations, representations

4. The ability to represent all of the kinds of knowledge that are needed in that domain is called _____________. [ ]

(a) Referential Efficiency (b) Representational Adequacy

(c) Inferential Efficiency (d) Acquisitional Efficiency

5. The ability to acquire new information easily is called ____. [ ]

(a) Referential Efficiency (b) Representational Adequacy

(c) Inferential Efficiency (d) Acquisitional Efficiency

6. The two important attributes of inheritance are _______ and ________.

7. Weak slot-and-filler structures are __________and _____________.

8. Strong slot-and-filler structures are __________and _____________.

9. Procedural knowledge is _____________.

(a) Declarative (b) Operational (c) progressive (d) none [ ]

10. Procedural Knowledge get low scores for the properties______

(a) Inferential Adequacy (b) Acquisitional Efficiency

(c) Inferential Efficiency (d) both (a) and (b) [ ]

11. The most commonly used technique for representing procedural knowledge in AI Programs is the use of ____________. [ ]

(a) Production rules (b) symbols (c) facts (d) both (b) & (c)

12. The structure in which information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationships among the nodes is called ___________.

13. Define the term "frame".

14. __________ theory provides a good basis for understanding frame systems. [ ]

(a) set (b) Graphics (c) Logic (d) none

15. The classes whose elements are themselves classes are called ______.

(a) Sub class (b) Base class (c) Meta class (d) Parent class [ ]

16. _____________ is a theory of how to represent the kind of knowledge about events that is usually contained in natural language sentences.

17. The primitive that represents transfer of an abstract relationship is _______.

(a) PTRANS (b) ATRANS (c) MOVE (d) GRASP [ ]

18. A ______________ is a structure that describes a stereotyped sequence of events in a particular context.

SECTION-B

Descriptive Questions

1. Enlist the four properties that a knowledge representation system must have?
2. Explain four knowledge representation techniques.
3. Enumerate the basic mechanism of retrieving a value of an attribute, using inheritance.
4. How non binary predicates are represented using semantic net. Explain with suitable example.
5. Represent the following facts using semantic nets:

John gave the book to Mary

John is 6 feet tall and that he is taller than Bill.

6. Represent the following facts using partitioned semantic nets:

The dog bit the mail carrier.
Every dog has bitten a mail carrier.
Every dog in town has bitten the constable.
Every dog has bitten every mail carrier.

7. Justify the statement- “Set theory provides a good basis for understanding frame systems”.
8. List the ways in which classes are related to each other in frames, with suitable example?
9. List the set of primitives and conceptual tenses used in Conceptual Dependency.

10. Explain the rules used in Conceptual Dependency.

11. Represent the following sentence in CD:

Since smoking can kill you, I stopped.

12. Describe the important components of a script, with a suitable example.

Artificial Intelligence

UNIT - V

Expert system and applications

Syllabus:

Introduction phases in building expert systems, expert system versus traditional systems, rule-based expert systems blackboard systems truth maintenance systems, application of expert systems, list of shells and tools.

Outcomes:

Student will be able to:

Introduction: Expert System

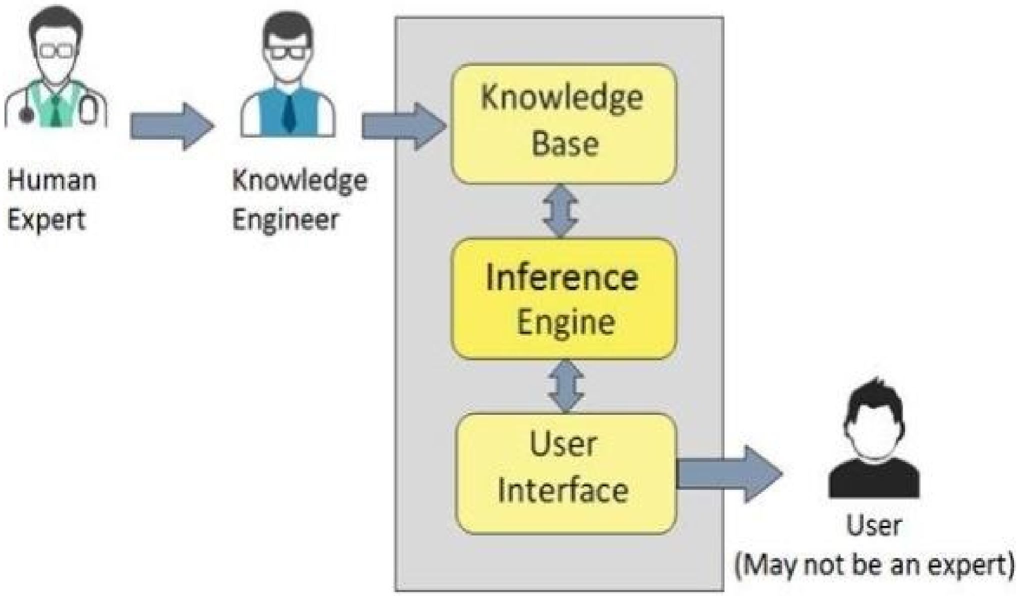
- The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.
- **Characteristics of Expert Systems:**
  - High performance
  - Understandable
  - Reliable
  - Highly responsive
- **Capabilities of Expert Systems:** The expert systems are capable of −
  - Advising
  - Instructing and assisting human in decision making
  - Demonstrating
  - Deriving a solution
  - Diagnosis
  - Explaining
  - Interpreting input
  - Predicting results
  - Justifying the conclusion
  - Suggesting alternative options to a problem
- **They are incapable of :**
  - Substituting human decision makers

- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

Components of Expert Systems

➢ The components of ES include −

- Knowledge Base
- Inference Engine
- User Interface



Knowledge Base:

➢ It contains domain-specific and high-quality knowledge.

➢ Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

➢ What is Knowledge?-The data is collection of facts. The information is organized as data and facts about the task domain. **Data, information,** and **past experience** combined together are termed as knowledge.

- **Components of Knowledge Base:** The knowledge base of an ES is a store of both, factual and heuristic knowledge.
  - **Factual Knowledge** − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
  - **Heuristic Knowledge** − It is about practice, accurate judgement, one's ability of evaluation, and guessing.
- **Knowledge representation:** It is the method used to organize and formalize the knowledge in the knowledge base. It is in the form of IF-THEN-ELSE rules.
- **Knowledge Acquisition:** The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base.
- The knowledge base is formed by readings from various experts, scholars, and the **Knowledge Engineers**. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills.
- He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Inference Engine:

- Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct solution.
- In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.
- In case of rule based ES, it −
  - Applies rules repeatedly to the facts, which are obtained from earlier rule application.
  - Adds new knowledge into the knowledge base if required.

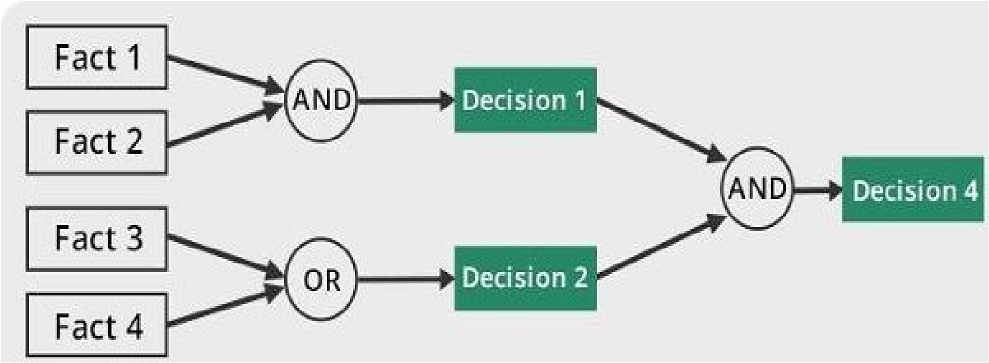
- Resolves rules conflict when multiple rules are applicable to a particular case.

➢ To recommend a solution, the Inference Engine uses the following strategies −

- Forward Chaining
- Backward Chaining

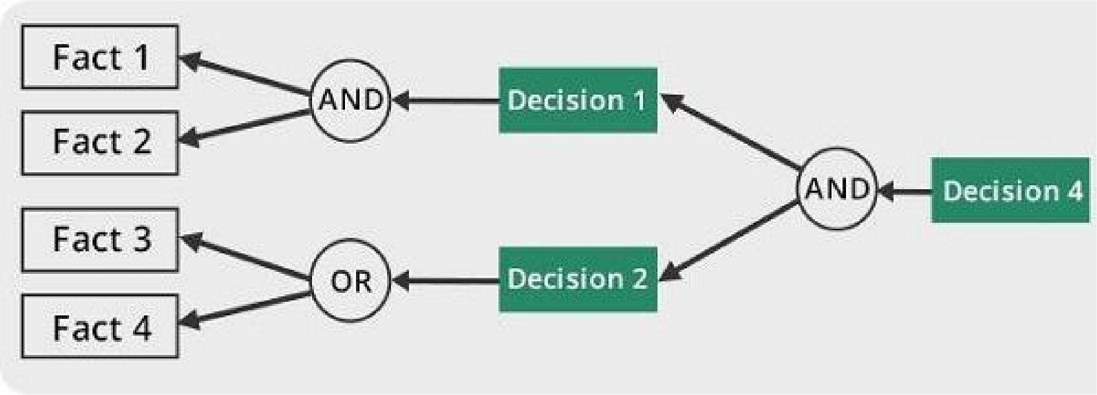
➢ **Forward Chaining:** It is a strategy of an expert system to answer the question, **"What can happen next?"**

- Here, the Inference Engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution.
- This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



➢ **Backward Chaining:** With this strategy, an expert system finds out the answer to the question, **"Why this happened?"**

- On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.



User Interface:

- User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence.
- It explains how the ES has arrived at a particular recommendation. The explanation may appear in the following forms −
  - Natural language displayed on screen.
  - Verbal narrations in natural language.
  - Listing of rule numbers displayed on the screen.
- The user interface makes it easy to trace the credibility of the deductions.
- **Requirements of Efficient ES User Interface:**
  - It should help users to accomplish their goals in shortest possible way.
  - It should be designed to work for user's existing or desired work practices.
  - Its technology should be adaptable to user's requirements; not the other way round.
  - It should make efficient use of user input.

Expert Systems Limitations

- No technology can offer easy and complete solution. Large systems are costly, require significant development time, and computer resources. ESs have their limitations which include −
  - Limitations of the technology
  - Difficult knowledge acquisition
  - ES are difficult to maintain
  - High development costs

Applications of Expert System

- The following table shows where ES can be applied.

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.

Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.

Expert System Technology

- There are several levels of ES technologies available. Expert systems technologies include −
  - **Expert System Development Environment** − The ES development environment includes hardware and tools. They are −
    - Workstations, minicomputers, mainframes.
    - High level Symbolic Programming Languages such as **LIS**t **P**rogramming (LISP) and **PRO**grammation en **LOG**ique (PROLOG).
    - Large databases.
  - **Tools** − They reduce the effort and cost involved in developing an expert system to large extent.
    - Powerful editors and debugging tools with multi-windows.
    - They provide rapid prototyping
    - Have Inbuilt definitions of model, knowledge representation, and inference design.
  - **Shells** − A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility. For example, few shells are given below −
    - Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.

- *Vidwan*, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

Benefits of Expert Systems

- **Availability** − They are easily available due to mass production of software.
- **Less Production Cost** − Production cost is reasonable. This makes them affordable.
- **Speed** − They offer great speed. They reduce the amount of work an individual puts in.
- **Less Error Rate** − Error rate is low as compared to human errors.
- **Reducing Risk** − They can work in the environment dangerous to humans.
- **Steady response** − They work steadily without getting motional, tensed or fatigued.

Phases in building Expert Systems

The process of ES development is iterative. Steps in developing the ES include −

- **Identify Problem Domain**
  - The problem must be suitable for an expert system to solve it.
  - Find the experts in task domain for the ES project.
  - Establish cost-effectiveness of the system.
- **Design the System**
  - Identify the ES Technology
  - Know and establish the degree of integration with the other systems and databases.
  - Realize how the concepts can represent the domain knowledge best.
- **Develop the Prototype**
  - From Knowledge Base: The knowledge engineer works to −

  - Acquire domain knowledge from the expert.
  - Represent it in the form of If-THEN-ELSE rules.

- **Test and Refine the Prototype**
  - The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
  - End users test the prototypes of the ES.
- **Develop and Complete the ES**
  - Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
  - Document the ES project well.
  - Train the user to use ES.
- **Maintain the System**

- Keep the knowledge base up-to-date by regular review and update.
- Cater for new interfaces with other information systems, as those systems evolve.

Expert system versus traditional systems

Expert System	**Traditional System**
The entire problem related expertise is encoded in data structures only, none is in programs.	Problem expertise is encoded in both program and data structures.
The use of knowledge is vital.	Data is used more efficiently than knowledge.
These are capable of explaining how a particular conclusion is reached and why requested information is needed during a process.	These are not capable of explaining a particular conclusion for a problem. These systems try to solve in a straight forward manner.
Problems are solved more efficiently	Not so efficient as an expert system
It uses the symbolic representations for knowledge i.e. the rules, different	These are unable to express in symbols. They just simplify the

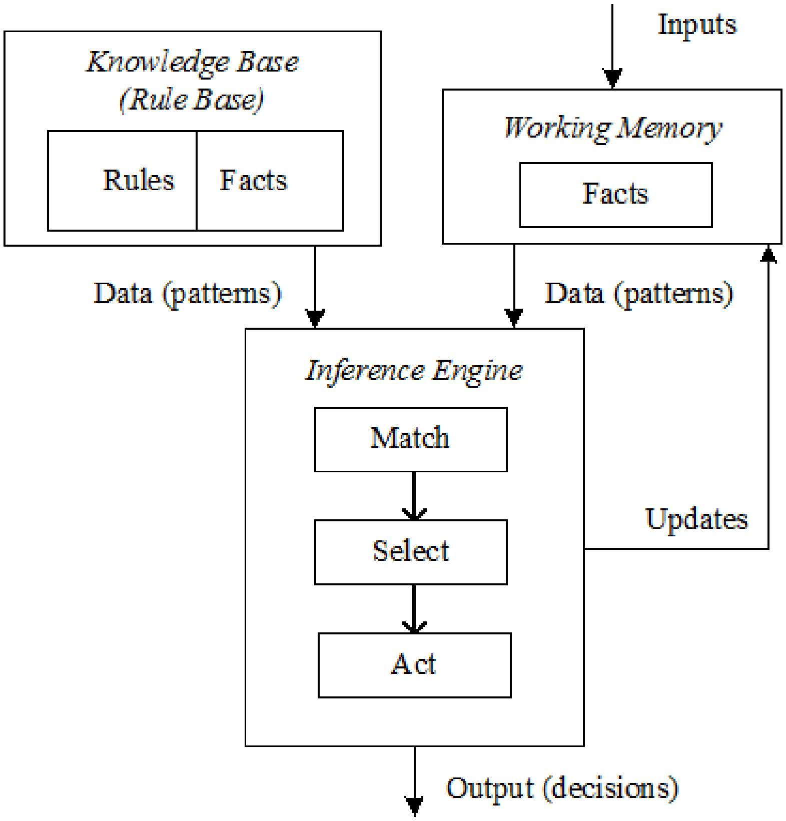
forms of networks, frames, scripts etc. and performs their inference through symbolic computations	problems in a straight forward manner and are incapable to express the "how, why" questions.
Problem solving tools those are present in expert system	No problem solving tools in specific.
Solution of the problem is more accurate.	Solution of the problem may not be more accurate.
Provide a clear separation of knowledge from its processing.	Do not separate knowledge from the control structure to process this knowledge.
Process knowledge expressed in the form of rules and use symbolic reasoning to solve problems in a narrow domain.	Process data and use algorithms, a series of well-defined operations, to solve general numerical problems.
Trace the rules fired during a problem-solving session and explain how a particular conclusion was reached and why specific data was needed.	Do not explain how a particular result was obtained and why input data was needed.
Permit inexact reasoning and can deal with incomplete, uncertain and fuzzy data.	W ork only on problems where data is complete and exact.
Enhance the quality of problem solving by adding new rules or adjusting old ones in the knowledge base. W hen new knowledge is acquired, changes are easy to accomplish.	Enhance the quality of problem solving by changing the program code, which affects both the knowledge and its processing, making changes difficult.

Rule-based Systems

- **Rule-based systems** are used as a way to store and manipulate knowledge to interpret information in a useful way. They are often used in artificial intelligence applications and research.
- An RBS consists of a knowledge base and an inference engine. The knowledge base contains rules and facts.

- A typical rule-based system has four basic components:

  - **A list of rules or rule base**, which is a specific type of knowledge base.
  - **An inference engine or semantic reasoner**, which infers information or takes action based on the interaction of input and the rule base. The interpreter executes a production system program by performing the following **match-resolve-act cycle:**
    - **Match:** In this first phase, the left-hand sides of all productions are matched against the contents of working memory. As a result a conflict set is obtained, which consists of instantiations of all satisfied productions. An instantiation of a production is an ordered list of working memory elements that satisfies the left-hand side of the production.
    - **Conflict-Resolution:** In this second phase, one of the production instantiations in the conflict set is chosen for execution. If no productions are satisfied, the interpreter halts.
    - **Act:** In this third phase, the actions of the production selected in the conflict-resolution phase are executed. These actions may change the contents of working memory. At the end of this phase, execution returns to the first phase.
  - **Temporary working memory-** set of facts.
  - A **user interface** or other connection to the outside world through which input and output signals are received and sent.



- The most common RBS modes of operation are:
  - forward chaining (stimulus driven)
  - backward chaining (goal directed)
- **Forward Chaining:** Forward chaining mode of operation means that a rule is triggered when changes in the working memory produce a situation that matches all of its antecedents.

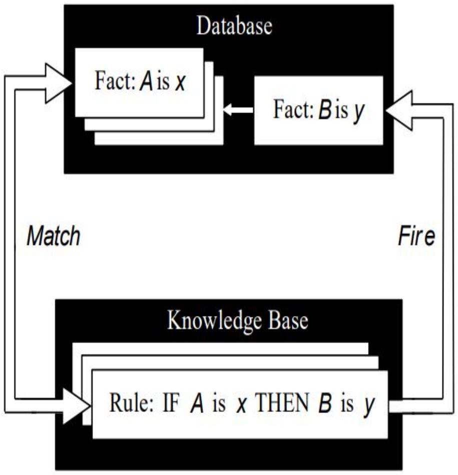
  Forward chaining is the process of inferring then-patterns from if-patterns that is consequents from antecedents. When an antecedent matches an assertion the antecedent is satisfied. When all antecedents of a rule are satisfied the rule is triggered. In deduction systems all triggered rules are allowed and may fire.

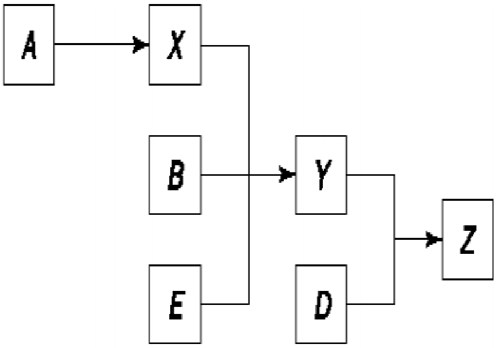
Forward Chaining Algorithm

Repeat

```
        For each rule do
            - Match all its antecedents to
                the facts from the Working memory
            - if all antecedents of a rule are matched,
                Execute is consequents
    until no rule produces a new assertion, or the goal is satisfied.
```

- **Backward Chaining:** The backward chaining mode of operation means that the systems begins with a goal and successively examines any rules with matching consequents. These candidate rules are considered one at a time. The unmet conditions are in turn reintroduced as new goals. The control procedure then shifts attention recursively toward the new goal. The effort terminates when the top goal is finally satisfied.
- **In a Rule-based System:**
  - The domain knowledge is represented by a set of IF-THEN production rules
  - Data is represented by a set of facts about the current situation.
  - The inference engine compares each rule stored in the knowledge base with facts contained in the database.
  - When the IF (condition) part of the rule matches a fact, the rule is fired and its THEN (action) part is executed.





Rule 1: IF Y is true AND D is true THEN Z is true

Rule 2: IF X is true AND B is true AND E is true THEN Y is true

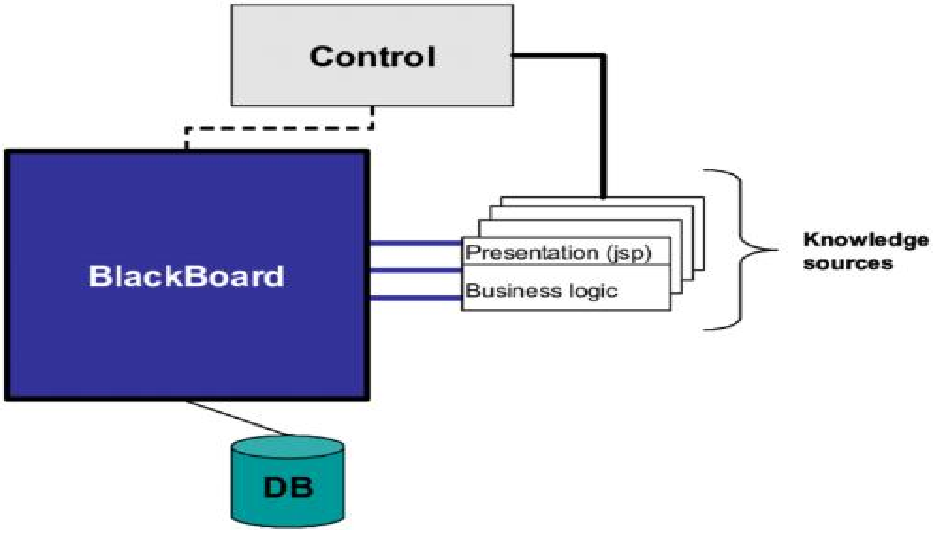
Rule 3: IF A is true THEN X is true

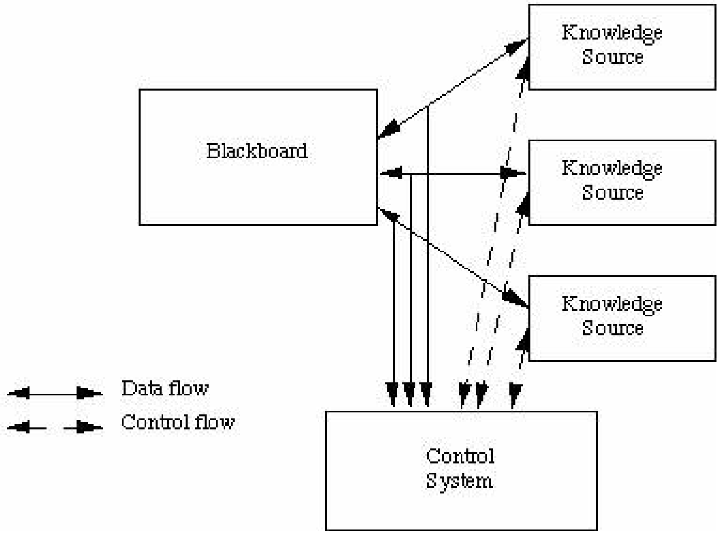
Blackboard System

- A blackboard system is an artificial intelligence approach based on the blackboard architectural model, where a common knowledge base, the "blackboard", is iteratively updated by a diverse group of specialist knowledge sources, starting with a problem specification and ending with a solution.
- Each knowledge source updates the blackboard with a partial solution when its internal constraints match the blackboard state. In this way, the specialists work together to solve the problem.
- The blackboard model was designed to handle complex, ill-defined problems, where the solution is the sum of its parts.
- A blackboard-system application consists of three major components:
  - The software specialist modules, which are called **knowledge sources (KSs)**. Like the human experts at a blackboard, each knowledge source provides specific expertise needed by the application.
  - **The blackboard**, a shared repository of problems, partial solutions, suggestions, and contributed information. The blackboard can be

thought of as a dynamic "library" of contributions to the current problem that have been recently "published" by other knowledge sources.

- **The control shell**, which controls the flow of problem-solving activity in the system. KSs need a mechanism to organize their use in the most effective and coherent fashion. In a blackboard system, this is provided by the control shell.





- The advantages of a blackboard include separation of knowledge into independent modules with each module being free to use the appropriate technology to arrive at the best solution with the most efficiency.

Justification-Based Truth Maintenance Systems (JTMS)

- The idea of a truth maintenance system or TMS is providing the ability to do **dependency-directed backtracking** and so to support **non-monotonic reasoning.**
- **Dependency-Directed Backtracking:**
  - We need to know a fact, F. which cannot be derived monotonically from what we already know, but which can be derived by making some assumption A which seems plausible.
  - So we make assumption A, derive F, and then derive some additional facts G and H from F. We later derive some other facts M and N, but they are completely independent of A and F.
  - A little while later, a new fact comes in that invalidates A. We need to rescind our proof of F, and also our proofs of G and H since they depended on F. But what about M and N? They didn't depend on F, so there is no logical need to invalidate them.
  - But if we use a conventional backtracking scheme, we have to back up past conclusions in the order in which we derived them. So we have to backup past M and N, thus undoing them, in order to get back to F, G, H and A.
  - To get around this problem, we need a slightly different notion of backtracking, one that is based on logical dependencies rather than the chronological order in which decisions were made. This new method is called **"Dependency-directed backtracking".**
- **Nonmonotonic reasoning**, in which the axioms/or the rules of inference are extended to make it possible to reason with incomplete

information. These systems preserve, however, the property that, at any given moment, a statement is either believed to be true, believed to be false, or not believed to be either.

- A TMS allows assertions to be connected via a spreadsheet-like network of dependencies.
- Consider an example: **ABC Murder story**.

Let Abbott, Babbitt, and Cabot be suspects in a murder case. Abbott has an alibi, in the register of a respectable hotel in Albany. Babbitt also has an alibi, for his brother-in-law testified that Babbitt was visiting him in Brooklyn at the time. Cabot pleads alibi too, claiming to have been watching a ski meet in the Catskills, but we have only his word for that. So we believe:

(1) That Abbott did not commit the crime,

(2) That Babbitt did not.

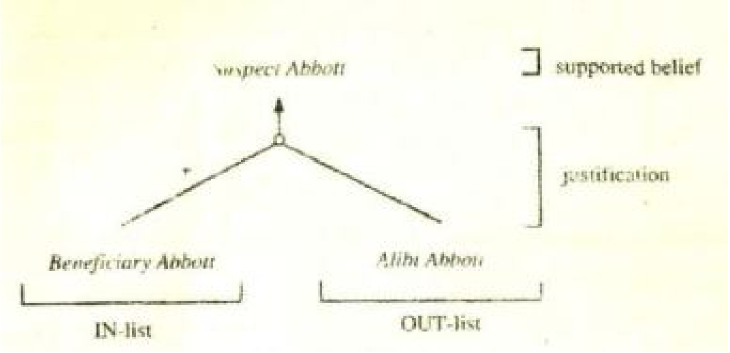
(3) That Abbott or Babbitt or Cabot did.

But presently Cabot documents his alibi—he had the good luck to have been caught by television in the sidelines at the ski meet. A new belief is:

(4) That Cabot did not.

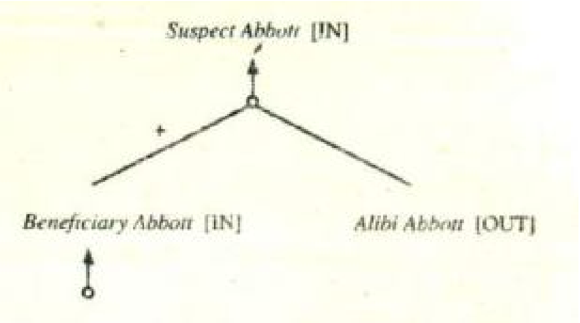
- Which has the weakest evidence? The basis for (1) in the hotel register is good, since it is a fine old hotel. The basis for (2) is weaker, since Babbitt's brother-in-law might be lying. The basis for (3) is perhaps twofold; that there is no sign of burglary and that only Abbott, Babbitt and Cabot seem to have stood to gain from the murder apart from burglary. This exclusion of burglary seems conclusive, but the other consideration does not: there could be some fourth beneficiary. For (4), finally, the basis is conclusive: the evidence from television. Thus (2) and (3) are the weak points. To resolve the inconsistency of (1) through (4) we should reject (2) or (3), thus either incriminating Babbitt or widening our net for some new suspect.

- Let us see how TMS works in ABC Murder story. Initially, we might believe that Abbott is the primary suspect because he was a beneficiary of the deceased and he had no alibi. There are three assertions here, a specific combination of which we now believe, although we may change our beliefs later. We can represent these assertions in shorthand as follows:
    - **Suspect** Abbott (Abbott is the primary murder suspect.)
    - **Beneficiary** Abbott (Abbott is a beneficiary of the victim.)
    - **Alibi** Abbott (Abbott was at an Albany hotel at the time.)
- A TMS dependency network offers a purely syntactic domain-independent way to represent belief and change it consistently.

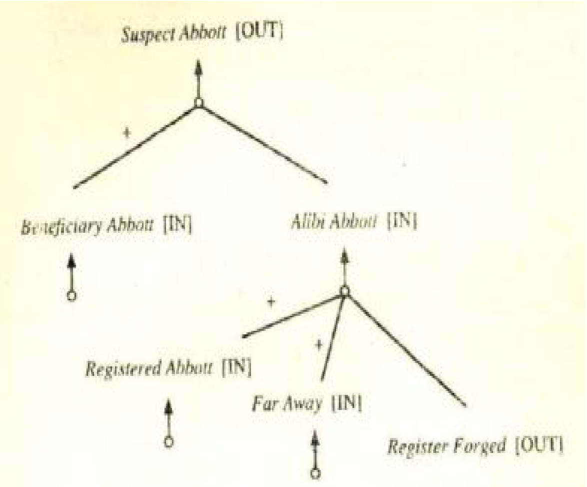


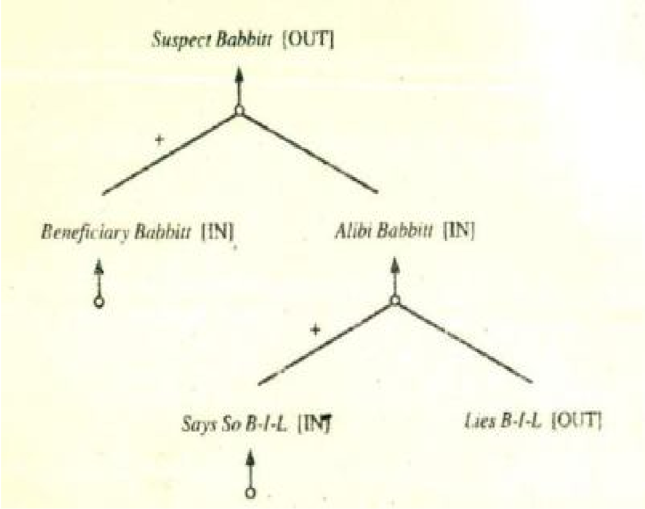
- The assertion **Suspect Abbott** has an associated TMS Justification. Each justification consists of two parts: **an IN-list and an OUT-List.** In the figure, the assertions on the IN-list are connected to the justification by "+" links, those on the OUT-list by “-"links. The justification is connected by an arrow to the assertion that it supports. In the justification shown, there is exactly one assertion in each list. Beneficiary Abbott is in the IN-list and Alibi Abbott is in the OUT-list. Such a justification says that Abbott should be a suspect just when it is believed that he is a beneficiary and it is not believed that he has an alibi.

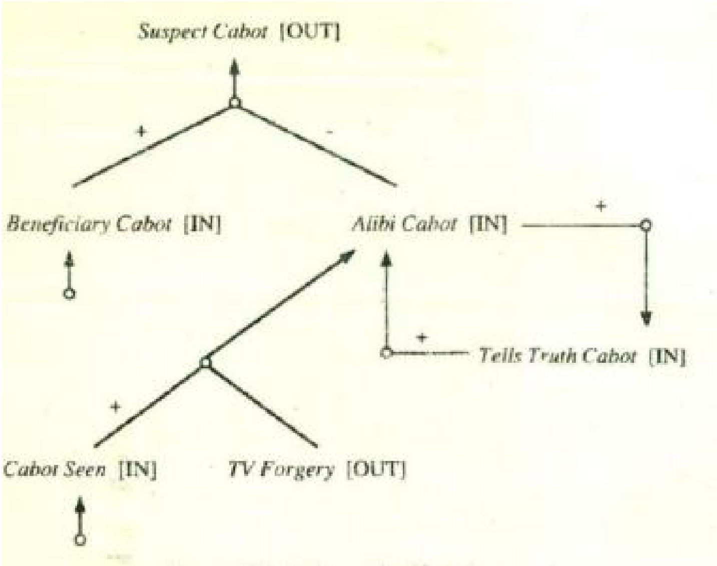
- Assertions (usually called nodes) in a TMS dependency network are believed when they have a **valid justification**. A justification is valid if every assertion in the IN-list is believed and none of those in the OUT list is. A justification is nonmonotonic if its OUT-list is not empty, or, recursively, if any assertion in its IN-list has a nonmonotonic justification. Otherwise, it is monotonic.
- In a TMS network, nodes are labelled with a belief status. If the assertion corresponding to the node should be believed, then in the TMS it is labelled **IN**. If there is no good reason to believe the assertion, then it is labelled **OUT**.
- The labelling task of a TMS is to label each node so that two criteria about the dependency network structure are met. The first criterion is consistency: every node labelled IN is supported by at least one valid justification and all other nodes are labelled OUT.
- **A justification is valid if every node in its IN-list is labelled IN and every node in it’s OUT-list is labelled OUT.**
- We are told that Abbott is a beneficiary. We have no further justification for this fact; we must simply accept it. For such facts, we give a premise justification: a justification with empty IN and OUT-lists. Premise justifications are always valid. The figure shows such a justification added to the network and a consistent labelling for that network, which shows Suspect Abbott labelled IN.



- That Abbot is the primary suspect represents an initial slate of the murder investigation. Subsequently, the detective establishes that Abbott is listed on the register of a good Albany hotel on the day of the murder. This provides a valid reason to believe Abbott's alibi. The figure shows the effect of adding such justification to the network. That Abbott was registered at the hotel. Registered Abbott, was told to us and has a premise justification and so is labelled IN. That the hotel is far away is also asserted as a premise. The register might have been forged, but we have no good reason to believe it was. Thus Register Forged lacks any justification and is labelled OUT. That Abbott was on the register of a far away hotel and the lack of belief that the register was forged will cause the appropriate forward rule to fire and create a justification for Alibi Abbott, which is thus labelled IN. This means that Suspect Abbott no longer has a valid justification and must be labelled OUT. Abbott is no longer a suspect.







- The key reasoning operations that are performed by a JTMS:
  - consistent labelling
  - contradiction resolution
- A set of important reasoning operations that a JTMS does not perform, includes:
  - Applying rules to derive conclusions.
  - Creating justifications for the results of applying rules (although justifications are created as part of contradiction resolution).
  - Choosing among alternative ways of resolving a contradiction.
  - Detecting contradictions.

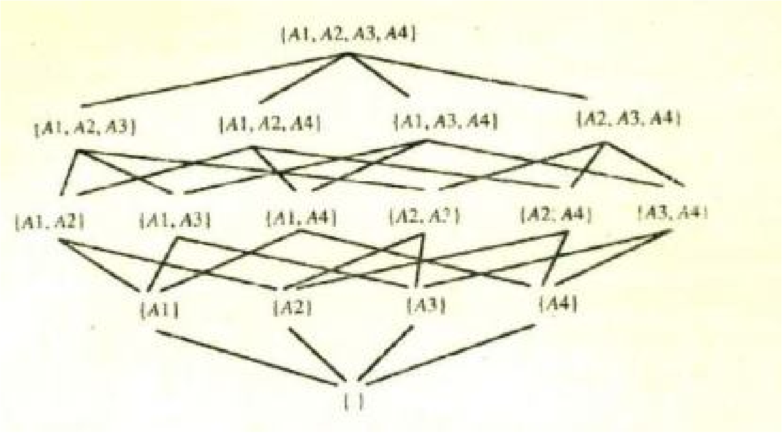
Logic-Based Truth Maintenance Systems (LTMS)

- In a JTMS, the nodes in the network are treated as atoms by the TMS, which assumes no relationships among them except the ones that are explicitly stated in the justifications.
- A JTMS has no problem simultaneously labelling both P and ⌐P, IN. For example, we could have represented explicitly both Lies B-I-L and Not Lies B-I-L and labelled both of them IN. No contradiction will be detected automatically. In an LTMS, on the other hand, a contradiction would be asserted automatically in such a case.

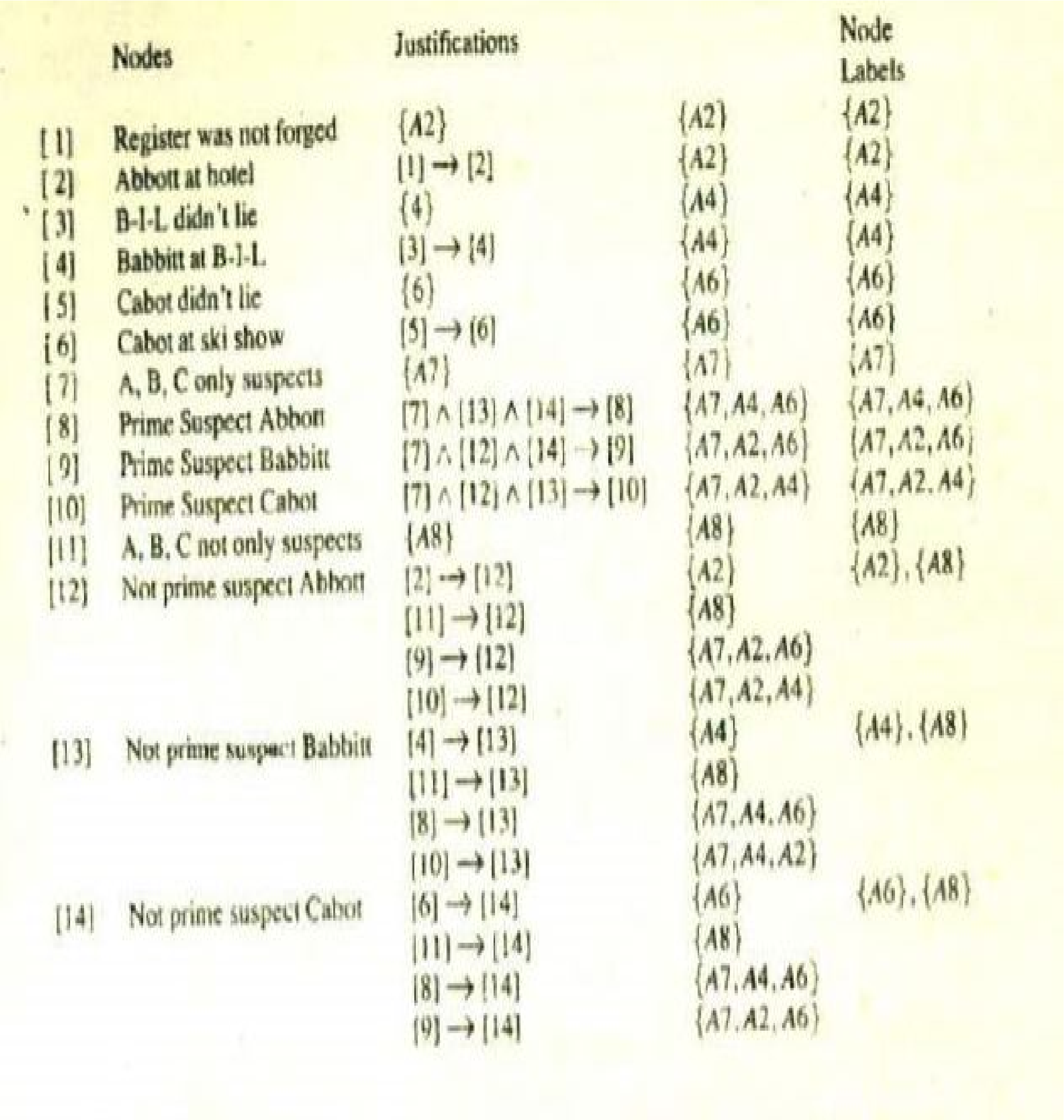
Assumption-Based Truth Maintenance Systems (ATMS)

- The ATMS is an alternative way of implementing nonmonotonic reasoning. In both JTMS and LTMS systems, a single line of reasoning is pursued at a time, and dependency-directed backtracking occurs whenever it is necessary to change the system's assumptions.
- In an ATMS, alternative paths are maintained in parallel. Backtracking is avoided the expense of maintaining multiple contexts, each of which corresponds to a set of consistent assumptions. As reasoning proceeds in an ATMS-based system, the universe of consistent contexts is pruned as contradictions are discovered.
- The remaining consistent contexts are used to label assertions, thus indicating the contexts in which each assertion has a valid justification.
- Assertions that do not have a valid justification in any consistent context can be pruned from consideration by the problem solver. As the set of consistent contexts gets smaller, so too does the set of assertions that can consistently be believed by the problem solver.
- An ATMS system works breadth-first, considering all possible contexts at once, while both JTMS and LTMS systems operate depth-first.
- The ATMS, uses a problem solver whose job is to:
    - Create nodes that correspond to assertions (both those that are given as axion1s and those that are derived by the problem solver).
    - Associate with each such node one or more justifications, each of which describes reasoning chain that led to the node.
    - Inform the ATMS of inconsistent contexts.
- The role of the ATMS system is to:
    - Propagate inconsistencies, thus ruling out contexts, that include sub contexts (sets of assertions) that are known to be inconsistent
    - Label each problem solver node with the contexts in which it has valid justification. This is done by combining contexts that correspond to the components of a justification.

Al Λ A2 Λ.... Λ An→C



- ➢ One problem with this approach is that given a set of n assumptions, the number of possible contexts that may have to be considered is 2^n.
- ➢ Consider how an ATMS-based problem solver works, in ABC Murder story. Again, our goal is to find a primary suspect. We need the assumptions:
  - Al. Hotel register was forged.
  - A2. Hotel register was not forged.
  - A3. Babbitt's brother-in-law tied.
  - A4. Babbitt's brother-in-law did not lie.
  - A5. Cabot lied.
  - A6. Cabot did not lie.
  - A7. Abbott, Babbitt, and Cabot are the only possible suspects.
  - A8. Abbott. Babbitt, and Cabot are not the only suspects

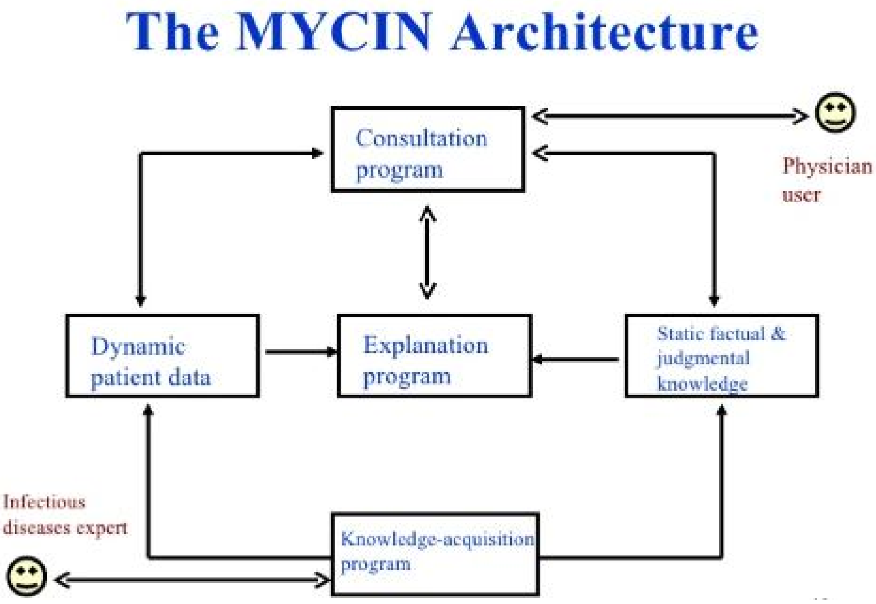


	Nodes	Justifications		Node Labels
[1]	Register was not forged	{A2}	{A2}	{A2}
[2]	Abbott at hotel	[1] → [2]	{A2}	{A2}
[3]	B-I-L didn't lie	{4}	{A4}	{A4}
[4]	Babbitt at B-I-L	[3] → [4]	{A4}	{A4}
[5]	Cabot didn't lie	{6}	{A6}	{A6}
[6]	Cabot at ski show	[5] → [6]	{A6}	{A6}
[7]	A, B, C only suspects	{A7}	{A7}	{A7}
[8]	Prime Suspect Abbott	[7] ∧ [13] ∧ [14] → [8]	{A7, A4, A6}	{A7, A4, A6}
[9]	Prime Suspect Babbitt	[7] ∧ [12] ∧ [14] → [9]	{A7, A2, A6}	{A7, A2, A6}
[10]	Prime Suspect Cabot	[7] ∧ [12] ∧ [13] → [10]	{A7, A2, A4}	{A7, A2, A4}
[11]	A, B, C not only suspects	{A8}	{A8}	{A8}
[12]	Not prime suspect Abbott	[2] → [12]	{A2}	{A2}, {A8}
		[11] → [12]	{A8}	
		[9] → [12]	{A7, A2, A6}	
		[10] → [12]	{A7, A2, A4}	
[13]	Not prime suspect Babbitt	[4] → [13]	{A4}	{A4}, {A8}
		[11] → [13]	{A8}	
		[8] → [13]	{A7, A4, A6}	
		[10] → [13]	{A7, A4, A2}	
[14]	Not prime suspect Cabot	[6] → [14]	{A6}	{A6}, {A8}
		[11] → [14]	{A8}	
		[8] → [14]	{A7, A4, A6}	
		[9] → [14]	{A7, A2, A6}	

MYCIN

- **MYCIN** was an backward chaining expert system that used artificial intelligence to identify bacteria causing severe infections, such as bacteremia and meningitis, and to recommend antibiotics, with the dosage adjusted for patient's body weight .
- MYCIN was developed over five or six years in the early 1970s at Stanford University. It was written in Lisp.

- MYCIN operated using fairly simple inference engine, and a knowledge base of ~600 rules. It would query the physician running the program via a long series of simple yes/no or textual questions.
- At the end, it provided a list of possible culprit bacteria ranked from high to low based on the probability of each diagnosis, its confidence in each diagnosis' probability, the reasoning behind each diagnosis.
- Mycin is a program that diagnoses infectious diseases. It reasons backward from its goal of determining the cause of a patient illness.
- It attempts to solve its goal of recommending a therapy for a particular patient by first finding the cause of the patient's illness.
- It uses its production rules to reason backward from goals to clinical observations.



DENDRAL

- Dendral was a project in artificial intelligence (AI) of the 1960s, and the computer software expert system that it produced. Its primary aim was to study hypothesis formation and discovery in science. For that, a specific task in science was chosen: help organic chemists in identifying unknown organic molecules, by analyzing their mass spectra and using knowledge of chemistry.

- The software program Dendral is considered the first expert system because it automated the decision-making process and problem-solving behaviour of organic chemists.
- It was written in the LISP programming language, which was considered the language of AI because of its flexibility.
- Dendral is a program that uses mass spectra or other experimental data together with knowledge base of chemistry, to produce a set of possible chemical structures that may be responsible for producing the data.
- A mass spectrum of a compound is produced by a mass spectrometer, and is used to determine its molecular weight, the sum of the masses of its atomic constituents.
- For example, the compound water (H2O), has a molecular weight of 18 since hydrogen has a mass of 1.01 and oxygen 16.00, and its mass spectrum has a peak at 18 units.
- Dendral uses input mass and the knowledge of atomic mass numbers and valence rules, to determine the possible combinations of atomic constituents whose mass would add up to 18.
- As the weight increases and the molecules become more complex, the number of possible compounds increases drastically.

R1

- The R1 (internally called XCON, for eXpert CONfigurer) program was a production-rule-based system written in OPS5 in 1978 to assist in the ordering of DEC's VAX computer systems by automatically selecting the computer system components based on the customer's requirements.
- R1 is a program that configures VAX-11/780 computer systems. Given a customer's order, it determines what, if any, modifications

have to be made to the order for reasons of system functionality and produces a number of diagrams showing how the various components on the order are to be associated.

- The program is currently being used on a regular basis by Digital Equipment Corporation's manufacturing organization. R1 is implemented as a production system.
- It uses Match as its principal problem solving method; it has sufficient knowledge of the configuration domain and of the peculiarities of the various configuration constraints that at each step in the configuration process, it simply recognizes what to do. Consequently, little search is required in order for it to configure a computer system.
- A typical VAX system includes many components from 50 to 150 in the following categories: CPU, memory control units, peripherals: tape drives, floppy disks, hard disks, size, printers, drivers for the peripherals, cabinets, and cables.
- There are many constraints because most permutations of components are not feasible. Only certain components can be attached to one another, and this limits the possible combinations that can be considered.
- Knowledge about the interconnections is represented by thousands of production rules. The control strategy is hierarchical and sub-problems are solved in order of the importance of their associated goal.
- XCON uses forward reasoning and is written in the OPS5 language.

PROSPECTOR

- PROSPECTOR is an expert system designed for decision-making problems in mineral exploration. It aids geologists in evaluating the favourability of an exploration site or region for occurrences of ore deposits of particular types.
- Once a site has been identified, PROSPECTOR can also be used for drilling-site selection.

- PROSPECTOR is written in INTERLISP, an advanced dialect of the LISP language.
- PROSPECTOR can reach a conclusion about a particular ore deposit. It gives a certainty value of the ore deposit. It as well provides the explanation text for the conclusion.
- When PROSPECTOR is used for the drilling site selection, it could also produce the favourability map of the site.

Unit- V
Artificial Intelligence: Expert system and applications
(Open Elective –I)

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Define the term “Expert System”.
2. The components of expert system are _______________, ________________and _______________.
3. Knowledge comprises of ____________. [ ]

   (a) Data (b) Information (c) Past Experience (d) All the above
4. The information that is widely accepted by the Knowledge Engineers and scholars in the task domain is called _________ knowledge.

   (a) Factual (b) Heuristic (c) Domain (d) none [ ]
5. Knowledge that is about practice, accurate judgment, one’s ability of evaluation, and guessing is called ______________ knowledge.

   (a) Factual (b) Heuristic (c) Domain (d) none [ ]
6. ___________categorizes and organizes the information in a meaningful way. [ ]

   (a) Knowledge Engineer (b) Human Expert (c) User (d) Tool
7. __________is a strategy of an expert system to answer the question, **“What can happen next?”** [ ]

   (a) Forward Chaining (b) Backward chaining (c) both (d) none
8. __________is a strategy of an expert system finds out the answer to the question, **“Why this happened?”** [ ]

   (a) Forward Chaining (b) Backward chaining (c) both (d) none
9. ___________ is an expert system without knowledge base. [ ]

   (a) Tool (b) Shell (c) Task (d) none
10. In an Expert System, the entire problem related expertise is encoded in _____________________. [ ]

    (i) Data Structures

    (ii) (ii) Programs

    (a) Only (i) (b) Only (ii) (c) Both (i) and (ii) (d) none

11. In a traditional system, the entire problem related expertise is encoded in ____________________. [ ]

    (i) Data Structures

    (ii) (ii) Programs

    (a) Only (i) (b) Only (ii) (c) Both (i) and (ii) (d) none

12. The knowledgebase in a Rule-base system consists of _____. [ ]

    (a) Rules (b) Facts (c) Both a & b (d) productions

13. Truth maintenance system supports ___________reasoning. [ ]

    (a) Monotonic (b) Non-Monotonic (c) Both a & b (d) none

14. **MYCIN** is a _____________ expert system. [ ]

15. (a) Forward Chaining (b) Backward chaining (c) both (d) none

16. **DENDRAL was written in the _________ programming language**.

    (a) PROLOG (b) LISP (c) FORTRAN (d) PYTHON [ ]

SECTION-B

Descriptive Questions

1. List the characteristics and capabilities of Expert System.
2. Explain the components of an expert system.
3. Distinguish between Forward chaining and Backward chaining.
4. Enlist the application of Expert systems.
5. Describe the phases of developing an Expert system.
6. What do you mean by expert system technology? Explain.
7. Distinguish Expert system and Traditional system.
8. Explain about Rule-based Systems.
9. Explain Justification-based Truth maintenance system.
10. Write short notes on:

    (i) MYCIN

    (ii) DENDRAl

    (iii) R1

UNIT - VI: Uncertainty measure

Syllabus:

Introduction, probability theory, Bayesian belief networks, certainty factor theory, Dempster- Shafer theory.

Outcomes:

Student will be able to:

Introduction:

- In several representation techniques, at any given point, a particular fact is believed to be true, believed to be false, or not considered one way or the other.
- For some kinds of problem solving, though, it is useful to be able to describe beliefs that are not certain but for which there is some supporting evidence.
- Let's consider two classes of such problems.
    - The first class contains problems in which there is genuine **randomness in the world**. Playing card games such as bridge and blackjack is good example of this class. Although in these problems, it is not possible to predict the world with certainty, some knowledge about the likelihood of various outcomes is available, and we would like to be able to exploit it.
    - The second class contains problems in which the **relevant world is not random:** it behaves "normally" unless there is some kind of exception. The difficulty is that there are many more possible exceptions. Many common sense tasks fall into this category. For problems like this, **statistical measures** may serve a very useful function as summaries of the world; rather than enumerating all the possible exceptions, we can use a numerical summary that tells us how often an exception of some sort can be expected to occur.

Probability and Bayes' Theorem

- An important goal for many problem-solving systems is to collect evidence as the system goes along and to modify its behaviour on the basis of the evidence.
- To model this behaviour, we need a statistical theory of evidence. **Bayesian statistics** is such a theory. The fundamental notion of Bayesian statistics is that of **conditional probability.**

  P(H/E) - the probability of hypothesis H given that we have observed evidence E

- To compute this, we need to take into account the **prior probability** of H (the probability that we would assign to H if we had no evidence) and the extent to which E provides evidence of H.
  - **P(H_i/E)** =the probability that hypothesis H_i is true given evidence E.
  - **P(E/H_i)** =the probability that we will observe evidence E given that hypothesis i is true.
  - **P(H_i)** =the a Priori probability that hypothesis i is true in the absence of any specific evidence. These probabilities are called prior probabilities or priors.
  - **k** =the number of possible hypotheses
- Bayes theorem then states that:

$$P(H_i|E) = \frac{P(E|H_i) \cdot P(H_i)}{\sum_{n=1}^{k} P(E|H_n) \cdot P(H_n)}$$

- Suppose, for example, that we are interested in examining the geological evidence at a particular location to determine whether that would be a good place to dig to find a desired mineral. If we know the prior probabilities of finding each of the various minerals and we know the probabilities that if a mineral is present then certain physical characteristics will be observed, then we can use Bayes formula to compute, from the evidence we collect, how likely it is that the various minerals are present. This is, in fact, what is done by the PROSPECTOR program which has been used successfully to help locate deposits of several minerals, including copper and uranium.

- The key to using Bayes theorem as a basis for uncertain reasoning is to recognize exactly what it says. When we say P(A/B), we are describing the conditional probability of A given that the only evidence we have is B. If there is also other relevant evidence, then it too must be considered.
- Suppose, for example, that we are solving a medical diagnosis problem. Consider the following assertions:

  S: patient has spots
  M: patient has measles
  F: patient has high fever

  Without any additional evidence, the presence of spots serves as evidence in favour of measles. It also serves as evidence of fever since measles would cause fever. But suppose we already know that the patient has measles. Then the additional evidence that he has spots actually tells us nothing about the likelihood of fever. Alternatively, either spots alone or fever alone would constitute evidence in favour of measles. If both are present, we need to take both into account in determining the total weight of evidence. But, since spots and fever are not independent events, we cannot just sum their effects. Instead, we need to represent explicitly the conditional probability that arises from their conjunction.
- In general, given a prior body of evidence e and some new observation E, we need to compute:

$$P(H|E,e) = P(H|E) \cdot \frac{P(e|E,H)}{P(e|E)}$$

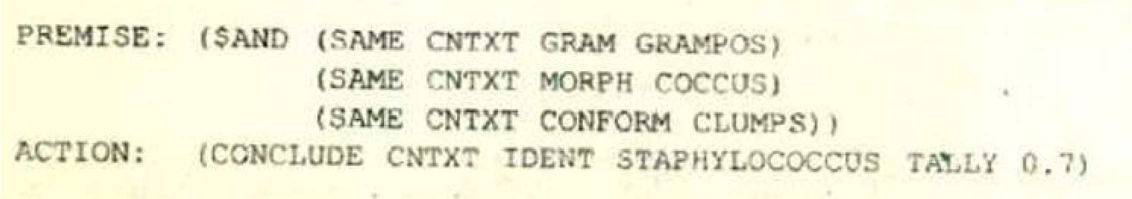
- The size of the set of joint probabilities that we require in order to compute this function grows as 2^n if there are n different propositions being considered.
- Drawbacks of Bayes Theorem:
  - The knowledge acquisition problem; too many probabilities have to be provided.
  - The space that would be required to store all the probabilities is too large.
  - The time required to compute the probabilities is too large.
- Despite these problems, **Bayesian statistics** provide an attractive basis for an uncertain reasoning system. The mechanisms that use this technique are:
  - Attaching certainly factors to rules
  - Bayesian networks
  - Dempster-Shafer theory

Certainty Factor Theory

- MYCIN represents most of its diagnostic knowledge as a set of rules. Each rule has associated with it a **certainty factor**, which is a measure of the extent to which the evidence that is described by the antecedent of the rule supports the conclusion that is given in the rule's consequent.

  If: (1) the stain of the organism is gram-positive, and
  (2) the morphology of the organism is coccus, and
  (3) the growth conformation of the organism is Clumps,
  then there is suggestive evidence (0.7) that the identity of the organism is staphylococcus.

  This is the form in which the rules are stated to the user. They are actually represented internally in an easy-to manipulate LISP list structure.



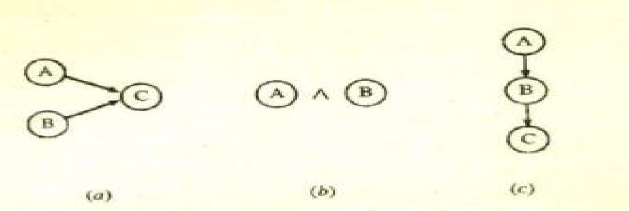
```
PREMISE:  ($AND  (SAME  CNTXT  GRAM  GRAMPOS)
                 (SAME  CNTXT  MORPH  COCCUS)
                 (SAME  CNTXT  CONFORM  CLUMPS))
ACTION:   (CONCLUDE  CNTXT  IDENT  STAPHYLOCOCCUS  TALLY  0.7)
```

- ➢ MYCIN uses these rules to reason backward to the clinical data available from its goal of finding significant disease-causing organisms. Once it finds the identities of such organisms, it then attempts to select a therapy by which the disease(s) may be treated.
- ➢ In order to understand how MYCIN exploits uncertain information, we need answers to two questions:
  1. "What do certainty factors mean?"
  2. “How does MYCN combine the estimates of certainty in each of its rules to produce a final estimate of the certainty of its conclusions?"
- ➢ A certainty factor (CF[h, e]) is defined in terms of two components:
  - **MB[h, e]: a measure (between 0 and 1) of belief** in hypothesis h given the evidence e. MB measures the extent to which the evidence supports the hypothesis. It is zero if the evidence fails to support the hypothesis.
  - **MD[h, e]: a measure (between 0 and 1) of disbelief** in hypothesis h given the evidence e. MD measures the extent to which the evidence supports the negation of the hypothesis. It is zero if the evidence supports the hypothesis.
  - we can define the certainty factor as:

CF[h, el = MB[h, e] – MD[h, e]

- ➢ Since any particular piece of evidence either supports or denies a hypothesis (but not both), and since each MYCIN rule corresponds to one piece of evidence, a single number suffices for each rule to define both the MB and MD and thus the CF.
- ➢ The CF's of MYCIN's rules are provided by the experts who write the rules. They reflect the expert’s assessments of the strength of the evidence in support of the hypothesis. As MYCIN reasons, these CF's

need to be combined to reflect the operation of multiple pieces of evidence and multiple rules applied to a problem.



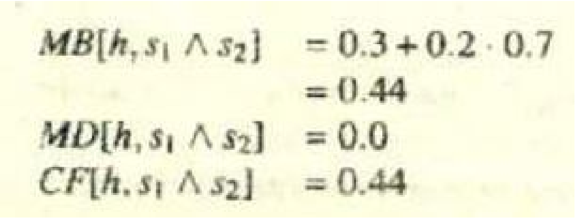
- The above figure illustrates three combination scenarios that we need to consider. In Figure (a), several rules provide evidence that relates to a single hypothesis. In Figure (b), we need to consider our belief in a collection of several propositions taken together. In Figure (c), the output of one rule provides the input to another.
- The following are the properties that the combining functions should satisfy:
  - Since the order in which evidence is collected is arbitrary, the combining functions should be commutative and associative.
  - Until certainty is reached, additional confirming evidence should increase MB (and similarly for disconfirming evidence and MD).
  - It uncertain inferences are chained together, then the result should be less certain than either of the inferences alone.
- Consider **case (a)**, where several pieces of evidence are combined to determine the CF of one hypothesis, the measures of belief and disbelief of a hypothesis given two observations sl and s2 are computed from:

$$MB[h, s_1 \wedge s_2] = \begin{cases} 0 & \text{if } MD[h, s_1 \wedge s_2] = 1 \\ MB[h, s_1] + MB[h, s_2] \cdot (1 - MB[h, s_1]) & \text{otherwise} \end{cases}$$

$$MD[h, s_1 \wedge s_2] = \begin{cases} 0 & \text{if } MB[h, s_1 \wedge s_2] = 1 \\ MD[h, s_1] + MD[h, s_2] \cdot (1 - MD[h, s_1]) & \text{otherwise} \end{cases}$$

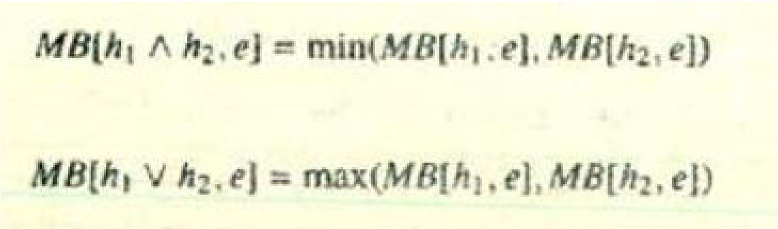
The measure of belief in h is 0 if h, is disbelieved with certainity. Otherwise, the measure of belief in h, given two observations is the measure of belief given only one observation plus some increment for the second observation. This increment is computed by first taking the difference between 1 (certainity) and the belief given only the first observation. This difference is the most that can be added by the second observation.

Example: Suppose we make an initial observation that confirms our belief in h with MB = 0.3. then MD[h,s1] = 0 and CF[h,s1] = 0.3. Now we make a second observation, which also confirms h, with MB[h, s2] = 0.2. Now:



$$\begin{aligned} MB[h, s_1 \wedge s_2] &= 0.3 + 0.2 \cdot 0.7 \\ &= 0.44 \\ MD[h, s_1 \wedge s_2] &= 0.0 \\ CF[h, s_1 \wedge s_2] &= 0.44 \end{aligned}$$

- Consider **case (b)**, in which we need to compute the certainty factor of a combination of hypotheses. This is necessary when we need to know the certainty factor of a rule antecedent that contains several clauses. The combination certainty factor can be computed from its MB and MD. The formulas MYCIN uses for the MB of the conjunction and the disjunction of two hypotheses are:



$$MB[h_1 \wedge h_2, e] = \min(MB[h_1, e], MB[h_2, e])$$

$$MB[h_1 \vee h_2, e] = \max(MB[h_1, e], MB[h_2, e])$$

- Consider **case (c),** in which rules are chained together with the result that the uncertain outcome of one rule must provide the input to another. Our solution to this problem will also handle the case in which we must assign a measure of uncertainty to initial inputs. This could easily happen in situations where the evidence is the outcome of an experiment or a laboratory test whose results are not completely accurate. In such a case, the certainty factor of the hypothesis must take into account both the strength with which the evidence suggests the hypothesis and the level of confidence in the evidence. MYCIN provides a chaining rule that is defined as follows:

Let MB^1[h,s] be the measure of belief in h given that we are absolutely sure of the validity of s. Let e be the evidence that led us to believe in s.

$$MB[h,s] = MB'[h,s] \cdot \max(0, CF[s,e])$$

- Since initial CF's in MYCIN are estimates that are given by experts who write the rules, they define MB as:

$$MB[h,e] = \begin{cases} 1 & \text{if } P(h) = 1 \\ \dfrac{\max[P(h|e), P(h)] - P(h)}{1 - P(h)} & \text{otherwise} \end{cases}$$

MD is the proportionate decrease in belief in h as a result of e:

$$MD[h,e] = \begin{cases} 1 & \text{if } P(h) = 0 \\ \dfrac{\min[P(h|e), P(h)] - P(h)}{-P(h)} & \text{otherwise} \end{cases}$$

- It turns out that these definitions are incompatible with a Bayesian view of conditional probability, we can redefine MB as:

$$MB[h,e] = \begin{cases} 1 & \text{if } P(h) = 1 \\ \dfrac{\max[P(h|e), P(h)] - P(h)}{(1 - P(h)) \cdot P(h|e)} & \text{otherwise} \end{cases}$$

MYCIN violates Bayesian system:

- Each CF in a MYCIN rule represents the contribution of in individual rule to MYCIN's belief in a hypothesis. It represents a conditional probability, P(H/E).
- But in a pure Bayesian system, P(H/E) describes the conditional probability of H given that the only relevant evidence is E. It there is other evidence, joint probabilities need to be considered.
- This is where MYCIN diverges from a pure Bayesian system, with the result that it is easier to write and more efficient to execute, but with the corresponding risk that its behaviour will be counterintuitive.
- In particular, the MYCtN formulas for the entire three combination scenarios make the assumption that all rules are independent. The burden of guaranteeing independence (at least to the extent that it matters) is on the rule writer. Each of the combination scenarios is vulnerable when this independence assumption is violated.

Bayesian Networks

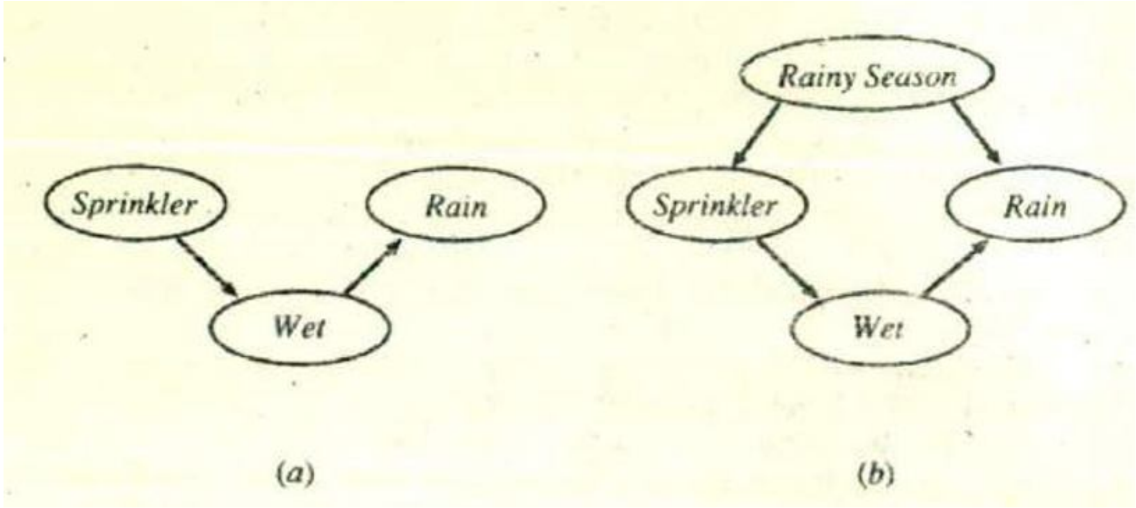
- In Bayesian Networks, we preserve the formalism and rely instead on the modularity of the world we are trying to model.
- The main idea is that to describe the real world, it is not necessary to use a huge joint probability table in which we list the probabilities of all combinations of events.
- Most events are conditionally independent of most other ones, so their interactions need not be considered. Instead, we can use a more local representation in which we will describe clusters of events that interact.
- Let's consider a example:

  S: sprinkler was on last night

  W: grass is wet

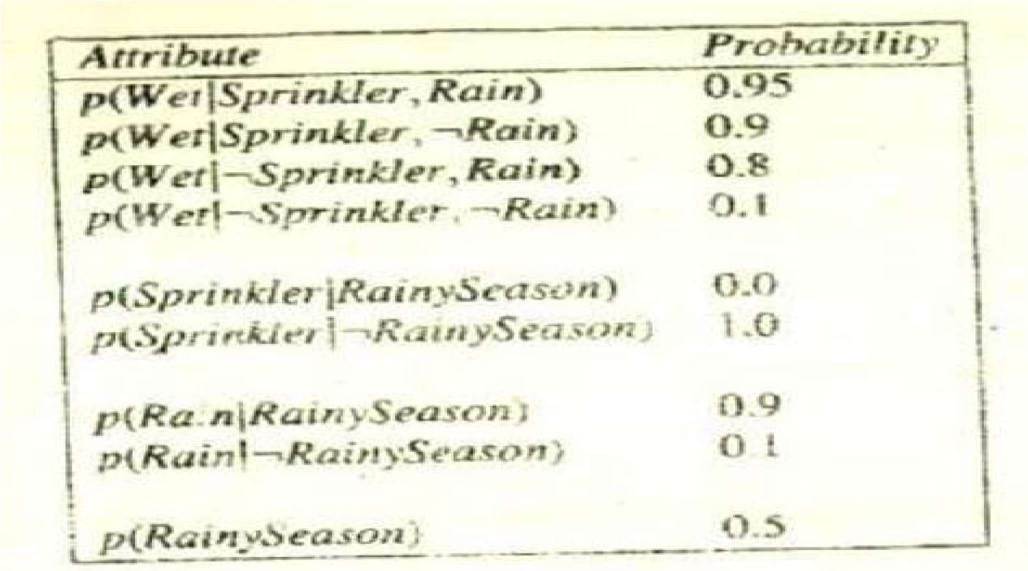
  R: it rained last night

This scenario can be shown by MYCIN and Bayesian network as follows:



- There are two different ways that propositions can influence the likelihood of each other.
    - The first is that causes influence the likelihood of their symptoms.
    - The second is that observing a symptom affects the likelihood of all of its possible causes.
- The idea behind the Bayesian network structure is to make a clear distinction between these two kinds of influence. We construct a **directed acyclic graph (DAG)** that represents causality relationships (cause and effect) among variables. The idea of a **causality graph** (or network) has proved to be very useful in several systems, particularly medical diagnosis systems.
- In the above example, in addition to the three nodes we have been talking about, the graph contains a new node corresponding to the propositional variable that tells us whether it is currently the rainy season.

- A DAG illustrates the causality relationships that occur among the nodes it contains. In order to use it as a basis for probabilistic reasoning, we need more information. In particular, we need to know, for each value of a parent node, what evidence is provided about the values that the child node can take on. We can state this in a table in which the conditional probabilities are provided.



Attribute	Probability
$p(Wet \mid Sprinkler, Rain)$	0.95
$p(Wet \mid Sprinkler, \neg Rain)$	0.9
$p(Wet \mid \neg Sprinkler, Rain)$	0.8
$p(Wet \mid \neg Sprinkler, \neg Rain)$	0.1
$p(Sprinkler \mid RainySeason)$	0.0
$p(Sprinkler \mid \neg RainySeason)$	1.0
$p(Rain \mid RainySeason)$	0.9
$p(Rain \mid \neg RainySeason)$	0.1
$p(RainySeason)$	0.5

- From the table we see that the prior probability of the rainy season is 0.5. Then, if it is the rainy season, the probability of rain on a given night is 0.9; if it is not, the probability is only 0.1.
- To be useful as a basis for problem solving, we need a mechanism for computing the influence of any arbitrary node on any other.
- Suppose that we have observed that it rained last night. 'What does that tell us about the probability that it is the rainy season?
- To answer this question we require that the initial DAG be converted to an **undirected graph** in which the arcs can he used to transmit probabilities in either direction depending on where the evidence has coming from.
- We require a mechanism for using the graph that guarantees that probabilities are transmitted correctly.
- For example, while it is true that observing wet grass may be evidence for rain, and observing rain is evidence for wet grass, we must

guarantee that no cycle is ever traversed in such a way that wet grass is evidence for rain, which is then taken as evidence for wet grass, and so forth.

- There are three broad classes of algorithms for doing these computations:
  - a message passing method
  - a clique triangulation method,
  - a variety of stochastic algorithms.
- The message-passing approach is based on the observation that to compute the probability of a node A given what is known about other nodes in the network, it is necessary to know three things:
  - π- the total support arriving at A from its parent nodes (which represent its causes).
  - λ - the total support arriving at A from its children (which represent its symptoms)
  - The entry in the fixed conditional probability matrix that relates A to its causes.
- Several methods for propagating π and λ messages and updating the probabilities at the nodes have been developed. The structure of the network determines what approach can be used.

Dempster-Shafer Theory

- We have described several techniques, all of which consider individual propositions and assign to each of them a point estimate (i.e., a single number) of the degree of belief that is warranted given the evidence.
- This new approach considers sets of propositions and assigns to each of them an interval: **[Belief, Plausibility]** in which the degree of belief must lie.
- **Belief (usually denoted Bel),** measures the strength of the evidence in favour of a set of propositions. It ranges from 0(indicating no evidence) to 1 (denoting certainty).

- **Plausibility (P1)** is defined to be: **Pl (s) = I – Bel (¬s).** It also ranges from 0 to 1 and measures the extent to which evidence in favour of ¬s leaves room for belief in s.
- If we have certain evidence in favour of ¬s, then Bel (¬s) will be 1 and Pl(s) will be 0. This tells us that the only possible value for Bel(s) is also 0.
- The belief-plausibility interval we have just defined measures not only our level of belief in some propositions, but also the amount of information we have.
- Suppose that we are currently considering three competing hypotheses: A. B, and C .If we have no information, we represent that, for each of them, that the true likelihood is in the range [1, 0]. As evidence is accumulated, this interval can be expected to shrink, representing increased confidence that we know how likely each hypothesis is.
- This contrasts with a pure Bayesian approach, in which we would probably begin by distributing the prior probability equally among the hypotheses and thus assert to each that P(h) = 0.33. The interval approach makes it clear that we have no information when we start.

Unit- VI
Assignment-Cum-Tutorial Questions

Objective Questions

1. **At any given point, a particular fact is believed to be ________, believed to be ________, or not considered one way or the other.**
2. **The fundamental notion of Bayesian statistics is ______________.**
3. **The probability of hypothesis H given that we have observed evidence E is _________. [ ]**
   (a) p(H / E) (b) P(E / H) (c) P(HE) (d) P(H)
4. **___________ probability states that hypothesis i is true in the absence of any specific evidence. [ ]**
   (a) conditional (b) Prior (c) statistical (d) Bayesian
5. **State Bayes theorem.**
6. __________ provide an attractive basis for an uncertain reasoning system. [ ]

   (a) Bayesian statistics (b) probability (c) Reasoning (d) none
7. A certainty factor (CF[h, e]) is defined in terms of two components:__________ and __________.
8. __________ measures the extent to which the evidence supports the hypothesis. [ ]
   (a) MB (b) MD (c) CF (d) none
9. __________ measures the extent to which the evidence supports the negation of the hypothesis. [ ]
   (a) MB (b) MD (c) CF (d) none
10. In ____________, we preserve the formalism and rely on the modularity of the world we are trying to model. [ ]
    (a) Bayesian Networks (b) Fuzzy logic (c) Bayes theorem (d) none
11. In Bayesian networks, we construct a ________ that represents causality relationships among variables. [ ]
    (a) Graph (b) DAG (c) Tree (d) nodes
12. DAG illustrates the ________ relationships that occur among the nodes it contains. [ ]

(a) influence(b) causality (c) parent-node (d) none

13. In message-passing approach, ________ represents the total support arriving at node A from its parent nodes. [ ]

    (a) π (b) λ (c) μ (d) α

14. In message-passing approach, ________ represents the total support arriving at A from its children. [ ]

    (a) π (b) λ (c) μ (d) α

15. ____________ measures the strength of the evidence in favour of a set of propositions. [ ]

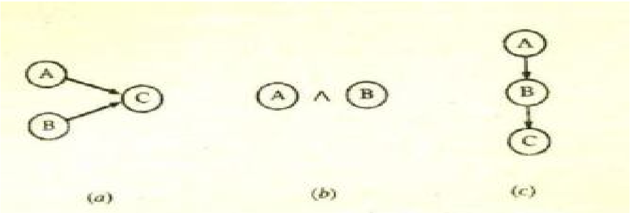
    (a) Plausibility (b) Belief (c) Point Estimate (d) none

16. If we have certain evidence in favour of ¬s, then Bel (¬s) will be ______ and Pl(s) will be ________. [ ]

    (a) 1, 0 (b) 0, 0 (c) 0, 1 (d) 1,1

Descriptive Questions

1. Explain how probability and Bayes theorem helps in reasoning uncertainty.
2. Illustrate how MYCIN exploits uncertain information?
3. Calculate the certainty factor in the following cases?



4. Analyze the cases in which MYCIN violates Bayesian System.
5. Explain about Bayesian Networks?
6. Represent the above scenario using MYCIN and Bayesian Network.

   S: sprinkler was on last night
   W: grass is wet
   R: it rained last night
7. Briefly explain with an example how Directed acyclic graph (DAG) that represents causality relationships (cause and effect) among variables?
8. Explain briefly about Dempster- Shafer theory?