Principles of Artificial Intelligence

Prepared by

Dr. Chetan K R
Professor and Head
Dept. of AIML
JNNCE

MODULE-1

Overview of Artificial Intelligence:

- Artificial Intelligence is concerned with the design of intelligence in an artificial device. The term was coined by John McCarthy in 1956.
- Intelligence is the ability to acquire, understand and apply the knowledge to achieve goals in the world.
- AI is the study of the mental faculties through the use of computational models
- AI is the study of intellectual/mental processes as computational processes.
- AI program will demonstrate a high level of intelligence to a degree that equals or exceeds the intelligence required of a human in performing some task.
- AI is unique, sharing borders with Mathematics, Computer Science, Philosophy, Psychology, Biology, Cognitive Science and many others.
- Although there is no clear definition of AI or even Intelligence, it can be described as an attempt to build machines that like humans can think and act, able to learn and use knowledge to solve problems on their own.

Definitions of AI

- **Thinking humanly**-Machine with minds. It is an effort to make computers think. It involves automation of human activities such as decision making, problem solving and learning
- **Thinking rationally**- Machine with models. It is The study of the computations that make it possible to perceive, reason, and act.
- **Acting humanly** -Create machines working as people. The study of how to make computers do things at which, at the moment, people are better
- **Acting rationally** - Computational Intelligence is the study of the design of intelligent agents.

Acting humanly:

- A machine passes Turing Test if human interrogator cannot differentiate whether responses came from human or machine. To pass Turing test, machine requires:
- Natutal Langauge processing: To enable it to communicate successfully in English; Knowledge representation to store what it knows or hears;
- Automated reasoning to use the stored information to answer questions and to draw new conclusions;
- machine learning to adapt to new circumstances and to detect and extrapolate pattern

Total Turing Test

In the Total Turing Test, verbal behaviors are not the sole standard for intelligence: other behaviors are examined too. "“The candidate must be able to do, in the real world of objects and people, everything that real people can do”. Total Turing Test can only be applied to a robot, or some other agent that is situated and embodied in the physical world. To pass the total Turing Test, the computer will need:

- computer vision to perceive objects, and
- robotics to manipulate objects and move about

Thinking humanly-Cognitive modeling approach:

Thinking humanly is to make a system or program to think like a human. But to achieve that, we need to know how does a human thinks. Suppose if we ask a person to explain how his brain connects different things during the thinking process, he/she will probably close both eyes and will start to check how he/she thinks but he/she cannot explain or interpret the process.

For example – If we want to model the thinking of Roger Federer and make the model system to compete with someone or against him to play in a tennis game, it may not be possible to replicate the exact thinking as Roger Federer, however, a good build of Intelligence systems (Robot) can play and win the game against him.

To understand the exact process of how we think, we need to go inside the human mind to see how this giant machine works. We can interpret how the human mind thinks in theory, in three ways as follows:

1. Introspection method – Catch our thoughts and see how it flows.
2. Psychological Inspections method – Observe a person on the action.
3. Brain Imaging method (MRI (Magnetic resonance imaging) or fMRI (Functional Magnetic resonance imaging) scanning) – Observe a person's brain in action.

Thinking rationally-Laws of thought approach:

Aristotle's syllogisms provided patterns for argument structures that always provide correct premises.A famous example, "Socrates is a man; all men are mortal; therefore, Socrates is mortal". Another example – All TVs use energy; Energy always generates heat; therefore, all TVs generate heat".

These arguments initiated the field called logic. Notations for statements for all kinds of objects were developed and interrelated between them to show logic. By 1965, programs existed that could solve problems that were described in logical notation and provides a solution.

<u>There are two limitations to this approach:</u>

1. First, it is not easy to take informal knowledge to use logical notation when there is not enough certainty on the knowledge.
2. Solving in principle and solving in practice varies hugely.

Acting rationally-Rational Agent approach:

A traditional computer program blindly executes the code that we write. Neither it acts on its own nor it adapts to change itself based on the outcome. The so-called agent program that we refer to here is expected to do more than the traditional computer program. It is expected to create and pursue the goal, change state, and operate autonomously.

A rational agent is an agent that acts to achieve its best performance for a given task. The "Logical Approach" to AI emphasizes correct inferences and achieving a correct inference is a part of the rational agent. Being able to give a logical reason is one way of acting rationally. But all correct inferences cannot be called rationality, because there are situations that don't always have a correct thing to do. It is also

possible to act rationally without involving inferences. Our reflex actions are considered as best examples of acting rationally without inferences.
The rational agent approach to AI has a couple of advantage over other approaches:

1. A correct inference is considered a possible way to achieve rationality but is not always required to achieve rationality.
2. It is a more manageable scientific approach to define rationality than others that are based on human behavior or human thought.

Foundations of Artificial Intelligence:

Philosophy

- Idealism is the belief that the mind and ideas is the primary structure of reality and that physical or material reality is secondary.
- Materialism is the opposite of Idealism and sees matter as the primary reality and all other things including thoughts as the product of interactions of matter.
- Rationalism is the belief that the rational mind is the best way to know something. If you are a rationalist you believe that your mind is more trustworthy than your sense. A stick in the water might look bent, but you know rationally that it only looks that way because it is in the water.
- Empiricism is the opposite of rationalism and it is the belief that the senses are the best way to know something. You might think something is true, but you only know it is true if your senses confirm it.
- The principle of induction, as applied to causation, says that, if *A* has been found very often accompanied or followed by *B*, then it is probable that on the next occasion on which *A* is observed, it will be accompanied or followed by *B*.
- Logical positivism is a combination of the two approaches upheld by positivism and symbolic logic. Positivism is a particular school of knowledge which advocates that valid knowledge must be based on sense knowledge. Any knowledge which is not based on senses is meaningless. It could be noted here that positivism is the extreme form of empiricism as the empiricists do not claim that knowledge not based on senses is invalid or meaningless, though they too advocate that knowledge should begin with sense experience

Mathematics

- Philosophers staked out some of the fundamental ideas of AI, but the leap to a formal science required a level of mathematical formalization in three fundamental areas: logic, computation, and probability
- In mathematics and computer science, an algorithm is a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing
- Incompleteness theorem states that in any reasonable mathematical system there will always be true statements that cannot be proved.
- Computable functions are the formalized analogue of the intuitive notion of algorithms, in the sense that a function is computable if there exists an algorithm that can do the job of the function, i.e. given an input of the function domain it can return the corresponding output.
- Tractable problems can be solved by computer algorithms that run in

polynomial time; i.e., for a problem of size n, the time or number of steps needed to find the solution is a polynomial function of n. Algorithms for solving hard, or intractable, problems, on... In P versus NP problem.

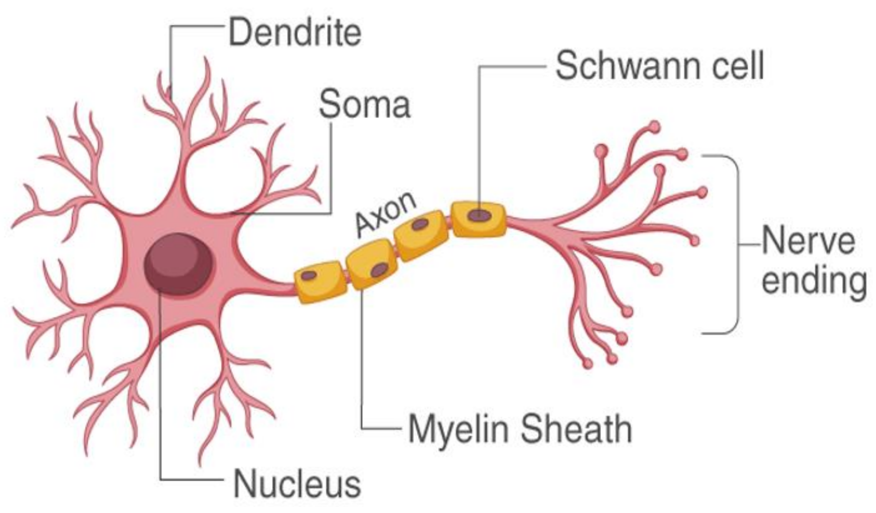
Economics

- In economics, utility is a term used to determine the worth or value of a good or service. More specifically, utility is the total satisfaction or benefit derived from consuming a good or service. Economic theories based on rational choice usually assume that consumers will strive to maximize their utility.
- Both decision and game theory concern the reasoning process underlying people's choices, that is, how their desires, beliefs, and other attitudes combine in a way that make people choose one option over another.
- Whereas decision theory is concerned with an individual decision-maker who tries to make the best decision based on their understanding of the world, game theory is concerned with the interaction between different decision-makers each of whom is trying to make the best decision based on their beliefs about what others will choose.
- A Markov decision process (MDP) refers to a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system. It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time.
- Satisficing is a decision-making strategy that aims for a satisfactory or adequate result, rather than the optimal solution. Instead of putting maximum exertion toward attaining the ideal outcome, satisficing focuses on pragmatic effort when confronted with tasks.

Neuroscience

Neuroscience is the study of the nervous system – from structure to function, development to degeneration, in health and in disease. It covers the whole nervous system, with a primary focus on the brain. Incredibly complex, our brains define who we are and what we do. They store our memories and allow us to learn from them. Our brain cells and their circuits create new thoughts, ideas and movements and reinforce old ones. Their individual connections (synapses) are responsible for a baby's first steps and every record-breaking athletic performance, with each thought and movement requiring exquisitely precise timing and connections.

STRUCTURE OF NEURON



Neurons (also called neurones or nerve cells) are the fundamental units of the brain and nervous system, the cells responsible for receiving sensory input from the external world, for sending motor commands to our muscles, and for transforming and relaying the electrical signals at every step in between. More than that, their interactions define who we are as people. A neuron has three main parts: dendrites, an axon, and a cell body or soma (see image below), which can be represented as the branches, roots and trunk of a tree, respectively. A dendrite (tree branch) is where a neuron receives input from other cells. Dendrites branch as they move towards their tips, just like tree branches do, and they even have leaf-like structures on them called spines.

A crude comparison of the raw computational resources available to the IBM BLUE GENE supercomputer, a typical personal computer of 2008, and the human brain.

	Supercomputer	Personal Computer	Human Brain
Computational units	10^4 CPUs, 10^{12} transistors	4 CPUs, 10^9 transistors	10^{11} neurons
Storage units	10^{14} bits RAM	10^{11} bits RAM	10^{11} neurons
	10^{15} bits disk	10^{13} bits disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{15}	10^{10}	10^{17}
Memory updates/sec	10^{14}	10^{10}	10^{14}

Psychology

- Psychology is the scientific study of the mind and behavior. Psychologists are actively involved in studying and understanding mental processes, brain functions, and behavior. The field of psychology is considered a "Hub Science" with strong connections to the medical sciences, social sciences, and education
- Cognitive psychology is the branch of psychology dedicated to studying how people think. The cognitive perspective in psychology focuses on how the interactions of thinking, emotion, creativity, and problem-solving abilities affect how and why you think the way you do. Cognitive psychology attempts to measure different types of intelligence, determine how you organize your thoughts, and compare different components of cognition.
- three key steps of a knowledge-based agent: (1) the stimulus must be translated into an internal representation, (2) the representation is manipulated by cognitive processes to derive new internal representations, and (3) these are in turn retranslated back into action

Computer Engineering

- Computer engineering is defined as the discipline that embodies the science and technology of design, construction, implementation, and maintenance of software and hardware components of modern computing systems and computer-controlled equipment.
- Computer engineering has traditionally been viewed as a combination of both computer science (CS) and electrical engineering (EE).
- It has evolved over the past three decades as a separate, although intimately related, discipline.
- Computer engineering is solidly grounded in the theories and principles of computing, mathematics, science, and engineering and it applies these theories and principles to solve technical problems through the design of computing hardware, software, networks, and processes.
- Increasingly, computer engineers are involved in the design of computer-based systems to address highly specialized and specific application needs.
- Computer engineers work in most industries, including the computer, aerospace, telecommunications, power production, manufacturing, defense, and electronics industries.

Control Theory and Cybernetics

- Control theory (Goodwin et al., 2001) deals with influencing the behavior of dynamical systems. Although a major application of control theory involves control system engineering, which deals with the design of process control systems for industry, other applications range far beyond this. This section presents the basic principles for modeling a dependable system as a control system.
- Cybernetics (Wiener, 1948), control theory, is a transdisciplinary branch of engineering and computational mathematics. It deals with the behavior of dynamical systems with inputs and how their behavior is modified by feedback.
- The objective of control theory is to control a system so that the system's output follows a desired control signal, called the reference. To do this, a (normally

feedback) controller is designed that determines what output needs to be monitored, how to compare it with the reference, which system behaviors need to be adjusted, and how to adjust them. The difference between actual and desired output, called the error signal, is applied as feedback to the system input, to bring the actual output closer to the reference.

Linguistics

- Linguistics is the scientific study of language, and its focus is the systematic investigation of the properties of particular languages as well as the characteristics of language in general.
- It encompasses not only the study of sound, grammar and meaning, but also the history of language families, how languages are acquired by children and adults, and how language use is processed in the mind and how it is connected to race and gender.
- With close connections to the humanities, social sciences and the natural sciences, linguistics complements a diverse range of other disciplines such as anthropology, philosophy, psychology, sociology, biology, computer science, health sciences, education and literature.
- The subfield of Applied Linguistics emphasizes the use of linguistic concepts in the classroom to help students improve their ability to communicate in their native language or a second language.

HISTORY OF AI

The gestation of artificial intelligence (1943–1955)

- Knowledge of basic psychology, functions of neurons in the brain and propositional logic
- A **neural network** is a neural circuit of biological neurons, sometimes also called a biological neural network, or a network of artificial neurons or nodes in the case of an artificial neural network.[1]
- Artificial neural networks are used for solving artificial intelligence (AI) problems; they model connections of biological neurons as weights between nodes. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed.
- Neuron on or off based on stimulation by neighboring neurons
- Simple neural network with logical connectives like AND, OR, NOT etc
- Hebbian Learning attempts to connect the psychological and neurological underpinnings of learning. he basis of the theory is when our brains learn something new, neurons are activated and connected with other neurons, forming a neural network. These connections start off weak, but each time the stimulus is repeated, the connections grow stronger and stronger, and the action becomes more intuitive.

The birth of artificial intelligence (1956)

- he field of Artificial Intelligence (AI) was officially born and christened at a workshop organized by John McCarthy in 1956 at the Dartmouth Summer Research Project on Artificial Intelligence. The goal was to investigate ways in which machines could be made to simulate aspects of intelligence.

- Although the Dartmouth workshop created a unified identity for the field and a dedicated research community, many of the technical ideas that have come to characterize AI existed much earlier.
- In the eighteenth century, Thomas Bayes provided a framework for reasoning about the **probability** of events.
- George Boole showed that **logical reasoning**—dating back to Aristotle—could be performed systematically in the same manner as solving a system of equations.
- Emergence of the field of **statistics**, which enables inferences to be drawn rigorously from data.
- The idea of physically engineering a machine to execute sequences of instructions, which had captured the imagination of pioneers such as Charles Babbage, had matured by the 1950s, and resulted in the construction of the first **electronic computers**.
- Primitive **robots**, which could sense and act autonomously, had also been built by that time.

Why AI as a separate branch

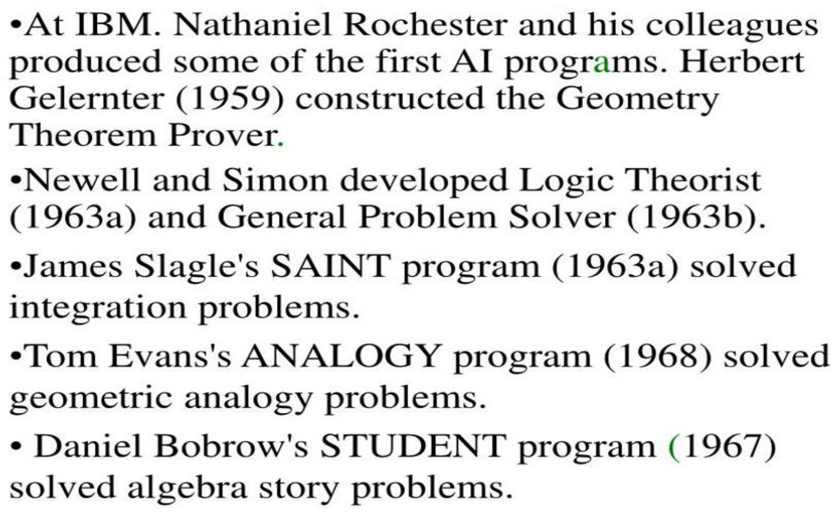
- AI to duplicate human facilities such as creativity, self-improvement and language use
- Uses computer science methodologies
- Design autonomous systems

Early enthusiasm, great expectations (1952–1969)

A physical symbol system takes physical patterns (symbols), combining them into structures (expressions) and manipulating them (using processes) to produce new expressions.

Eg:

- Formal logic
- Algebra
- Chess
- Computer program



•At IBM. Nathaniel Rochester and his colleagues produced some of the first AI programs. Herbert Gelernter (1959) constructed the Geometry Theorem Prover.

•Newell and Simon developed Logic Theorist (1963a) and General Problem Solver (1963b).

•James Slagle's SAINT program (1963a) solved integration problems.

•Tom Evans's ANALOGY program (1968) solved geometric analogy problems.

• Daniel Bobrow's STUDENT program (1967) solved algebra story problems.

Lisp, an acronym for *list processing*, is a functional programming language that was designed for easy manipulation of data strings. As one of the oldest programming languages still in use, Lisp offers several different dialects and has influenced the development of other languages.

A unique feature of early Lisp versions compared to most other programming languages is that the code could be directly interpreted without a compiler. The source code itself could be parsed and interpreted directly on a system. Today, however, most Lisp versions require that code be compiled and then loaded into an image to run. This offers faster program execution speeds compared to direct interpretation.

To cope with the bewildering complexity of the real world, scientists often ignore less relevant details; for instance, physicists often ignore friction and elasticity in their models. In 1970 Marvin Minsky and Seymour Papert of the MIT AI Laboratory proposed that, likewise, AI research should focus on developing programs capable of intelligent behaviour in simpler artificial environments known as microworlds. Much research has focused on the so-called blocks world, which consists of coloured blocks of various shapes and sizes arrayed on a flat surface.

A dose of reality (1966–1973)

- 1960-70s
- Too early for wide applications
- Intractability issues: From a computational complexity stance, intractable problems are problems for which there exist no efficient algorithms to solve them. Most intractable problems have an algorithm – the same algorithm – that provides a solution, and that algorithm is the brute-force search.
- The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. Genetic algorithms also faced intractability
- When training a neural network, there's going to be some data that the neural network **trains on,** and there's going to be some data reserved for checking the performance of the neural network. If the neural network performs well on the data which it **has not** trained on, we can say it has **generalized well** on the given data. Generalization of neural networks did not work

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

- **DENDRAL**, an early expert system, developed beginning in 1965 by the artificial intelligence (AI) researcher Edward Feigenbaum and the geneticist Joshua Lederberg, both of Stanford University in California. Heuristic DENDRAL (later shortened to DENDRAL) was a chemical-analysis expert system.
- The substance to be analyzed might, for example, be a complicated compound of carbon, hydrogen, and nitrogen. Starting from spectrographic data obtained from the substance, DENDRAL would hypothesize the substance's molecular structure.
- DENDRAL's performance rivaled that of chemists expert at this task, and the program was used in industry and in academia.
- An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

- The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for compsex problems using **both facts and heuristics like a human expert**.
- It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as **medicine, science,** etc.
- MYCIN was an early backward chaining expert system that used AI to identify microorganisms causing severe diseases like bacteremia and meningitis and propose antibiotics based on patient weight.

AI becomes an industry (1980–present)

- It has taken more than 25 years to gain recognition from the industry. Xcon was the very first industrial application of AI. Xcon is an expert system that configures mini-computers to meet user requirements. That process was normally taken 8 hours but xcon did it within 8 minutes.
- The neural network was reborn with a backpropagation training algorithm. In the late 1980s AI-based weapons were demonstrated. The best example of that is the DART expert system which is used in the gulf war.

AI adopts the scientific method (1987–present)

- The field of speech recognition illustrates the pattern. In recent years, approaches based on hidden Markov models (HMMs) have come to dominate the area. Two aspects of HMMs are relevant. First, they are based on a rigorous mathematical theory. Second, they are generated by a process of training on a large corpus of real speech data. This ensures that the performance is robust, and in rigorous blind tests the HMMs have been improving their scores steadily.
- Speech technology and the related field of handwritten character recognition are already making the transition to widespread industrial and consumer applications.
- Machine translation follows the same course as speech recognition. Much of the work on neural nets in the 1980s was done in an attempt to scope out what could be done and to learn how neural nets differ from "traditional" techniques. Using improved methodology and theoretical frameworks, the field arrived at an understanding in which neural nets can now be compared with corresponding techniques from statistics, pattern recognition, and machine learning, and the most promising technique can be applied to each application.
- Probabilistic Reasoning in Intelligent Systems led to a new acceptance of probability and decision theory in AI. The Bayesian network formalism was invented to allow efficient representation of, and rigorous reasoning with, uncertain knowledge.
- Similar gentle revolutions have occurred in robotics, computer vision, and knowledge representation. A better understanding of the problems and their complexity properties, combined with increased mathematical sophistication, has led to workable research agendas and robust methods.

Agents and Environments

An agent is anything that can be viewed as perceiving its environment through sensors and SENSOR acting upon that environment through actuators. A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators. A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen. Percept refers to the agent's perceptual inputs at any given instant. An agent's percept sequence is the complete history of everything the agent has ever perceived.

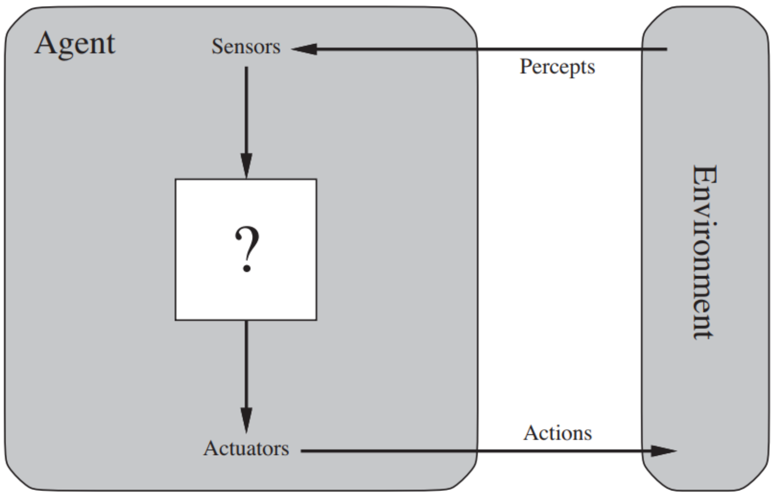
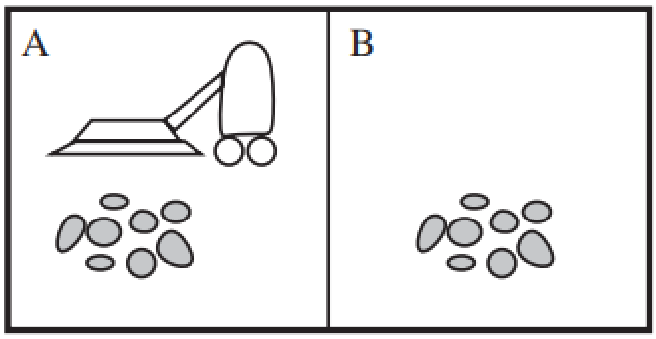


Figure 2.1 Agents interact with environments through sensors and actuators.

Agent function maps any given percept sequence to action. Agent program implements agent function in a physical system.

Eg: Vacuum cleaner for 2 locations



Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$

Rational Agents

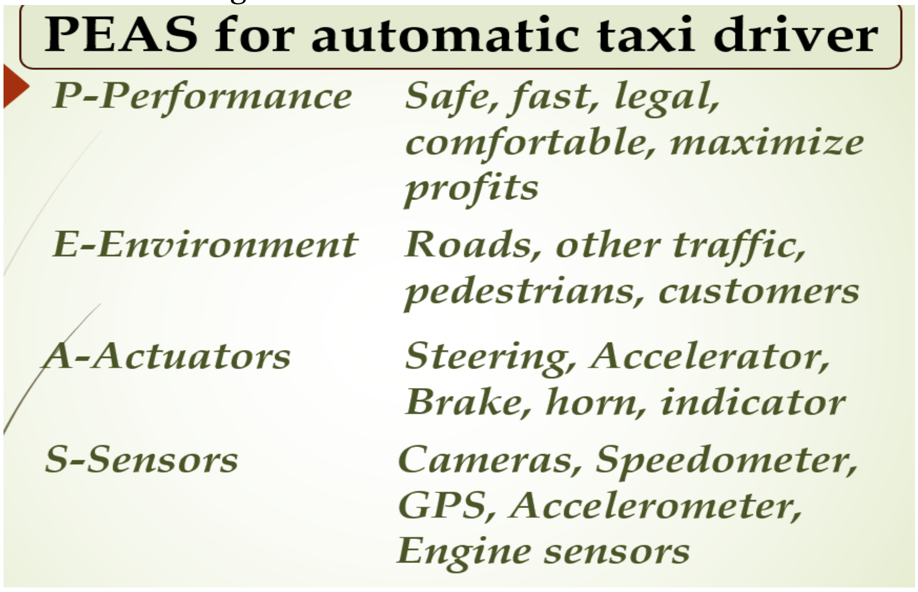
A rational agent is a computer program that uses logical reasoning and the ability to make decisions to determine its following action. An excellent example of a rational agent is a chess player. A chess player can analyze the board and determine which moves will result in the most advantageous outcome for itself. It can move one piece over another, move its king out of danger, or attack an opponent's piece.
A rational agent has four primary characteristics:

- Perception: The ability to perceive the current state of the environment and gather relevant information.
- Actuators: The ability to take actions within the environment to achieve its goals.
- Performance measure: A way to evaluate the success or failure of the agent's actions.
- Rationality: The ability to make decisions based on logical reasoning and optimize behavior to achieve its goals, considering its perception of the environment and the performance measure.

PEAS

- PEAS in AI is an acronym representing the foundational components that define an artificial intelligence agent's behavior. It stands for Performance Measure, Environment, Actuators, and Sensors.
- Performance Measure refers to the criterion an AI agent uses to evaluate its actions; the environment encompasses the external context it operates within, Actuators are the mechanisms enabling the agent to interact with the environment, and Sensors provide the agent with the means to perceive and gather information.
- PEAS serves as a structured approach to designing and understanding AI systems, aiding in conceptualizing objectives, interactions, and constraints. Whether applied to self-driving cars, virtual assistants, or medical diagnosis,

the PEAS framework offers a systematic lens to dissect and model AI's role within its surroundings.



PEAS for automatic taxi driver

P-Performance	*Safe, fast, legal, comfortable, maximize profits*
E-Environment	*Roads, other traffic, pedestrians, customers*
A-Actuators	*Steering, Accelerator, Brake, horn, indicator*
S-Sensors	*Cameras, Speedometer, GPS, Accelerometer, Engine sensors*

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

Types of Environments in AI

An environment in artificial intelligence is the surrounding of the agent. The agent takes input from the environment through sensors and delivers the output to the environment through actuators. There are several types of environments:

- Fully Observable vs Partially Observable
- Deterministic vs Stochastic
- Competitive vs Collaborative
- Single-agent vs Multi-agent
- Static vs Dynamic
- Discrete vs Continuous
- Episodic vs Sequential
- Known vs Unknown

1. Fully Observable vs Partially Observable

- When an agent sensor is capable to sense or access the complete state of an agent at each point in time, it is said to be a fully observable environment else it is partially observable.
- Maintaining a fully observable environment is easy as there is no need to keep track of the history of the surrounding.
- An environment is called unobservable when the agent has no sensors in all environments.
- Examples:
    - Chess – the board is fully observable, and so are the opponent's moves.
    - Driving – the environment is partially observable because what's around the corner is not known.

2. Deterministic vs Stochastic

- When a uniqueness in the agent's current state completely determines the next state of the agent, the environment is said to be deterministic.
- The stochastic environment is random in nature which is not unique and cannot be completely determined by the agent.
- Examples:
    - Chess – there would be only a few possible moves for a coin at the current state and these moves can be determined.
    - Self-Driving Cars- the actions of a self-driving car are not unique, it varies time to time.

3. Competitive vs Collaborative

- An agent is said to be in a competitive environment when it competes against another agent to optimize the output.
- The game of chess is competitive as the agents compete with each other to win the game which is the output.
- An agent is said to be in a collaborative environment when multiple agents cooperate to produce the desired output.

- When multiple self-driving cars are found on the roads, they cooperate with each other to avoid collisions and reach their destination which is the output desired.

4. Single-agent vs Multi-agent

- An environment consisting of only one agent is said to be a single-agent environment.
- A person left alone in a maze is an example of the single-agent system.
- An environment involving more than one agent is a multi-agent environment.
- The game of football is multi-agent as it involves 11 players in each team.

5. Dynamic vs Static

- An environment that keeps constantly changing itself when the agent is up with some action is said to be dynamic.
- A roller coaster ride is dynamic as it is set in motion and the environment keeps changing every instant.
- An idle environment with no change in its state is called a static environment.
- An empty house is static as there's no change in the surroundings when an agent enters.

6. Discrete vs Continuous

- If an environment consists of a finite number of actions that can be deliberated in the environment to obtain the output, it is said to be a discrete environment.
- The game of chess is discrete as it has only a finite number of moves. The number of moves might vary with every game, but still, it's finite.
- The environment in which the actions are performed cannot be numbered i.e. is not discrete, is said to be continuous.
- Self-driving cars are an example of continuous environments as their actions are driving, parking, etc. which cannot be numbered.

7.Episodic vs Sequential

- In an Episodic task environment, each of the agent's actions is divided into atomic incidents or episodes. There is no dependency between current and previous incidents. In each incident, an agent receives input from the environment and then performs the corresponding action.
- Example: Consider an example of Pick and Place robot, which is used to detect defective parts from the conveyor belts. Here, every time robot(agent) will make the decision on the current part i.e. there is no dependency between current and previous decisions.
- In a Sequential environment, the previous decisions can affect all future decisions. The next action of the agent depends on what action he has taken previously and what action he is supposed to take in the future.
- Example:
  Checkers- Where the previous move can affect all the following moves.

8. Known vs Unknown

- In a known environment, the output for all probable actions is given. Obviously, in case of unknown environment, for an agent to make a decision, it has to gain knowledge about how the environment works.

Agent program

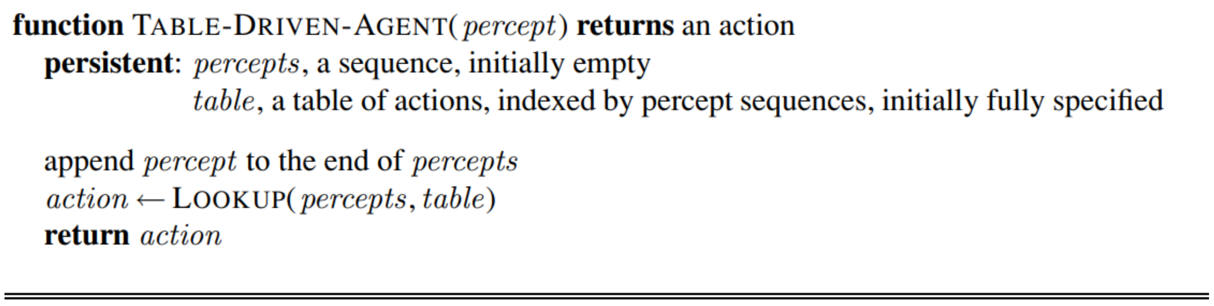
The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.



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

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

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

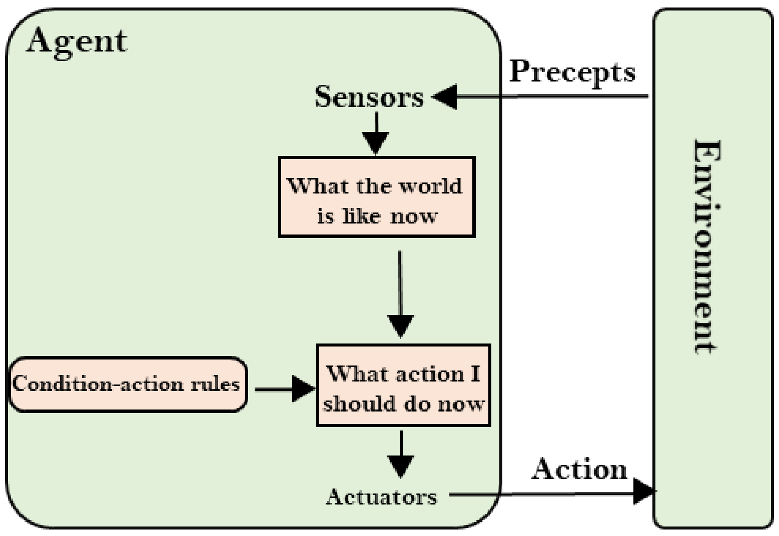
Types of Agents in AI

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

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

1. Simple Reflex agent:

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



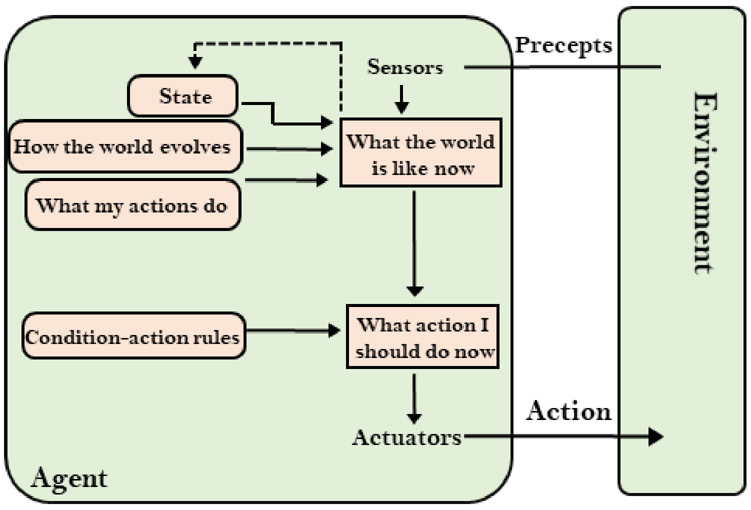
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

2. Model-based reflex agent

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
  - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
  - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
  - How the world evolves
  - How the agent's action affects the world.



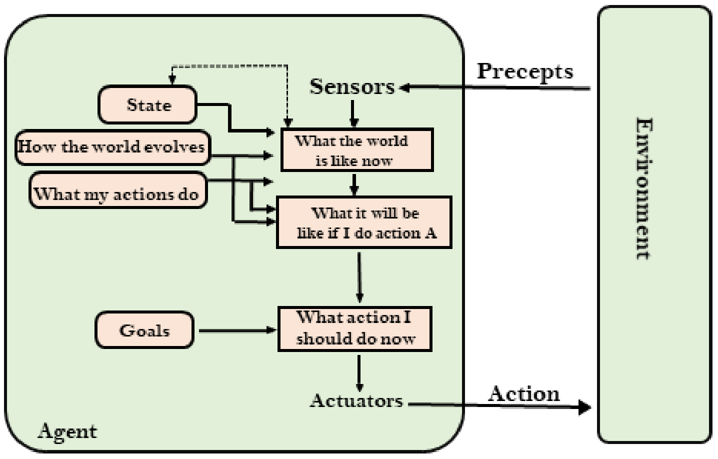
```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action,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

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

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

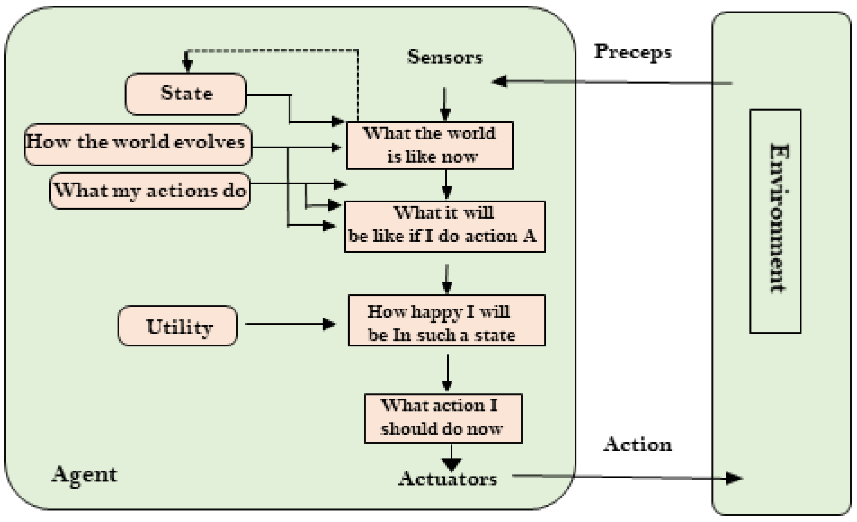
3. Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.



4. Utility-based agents

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

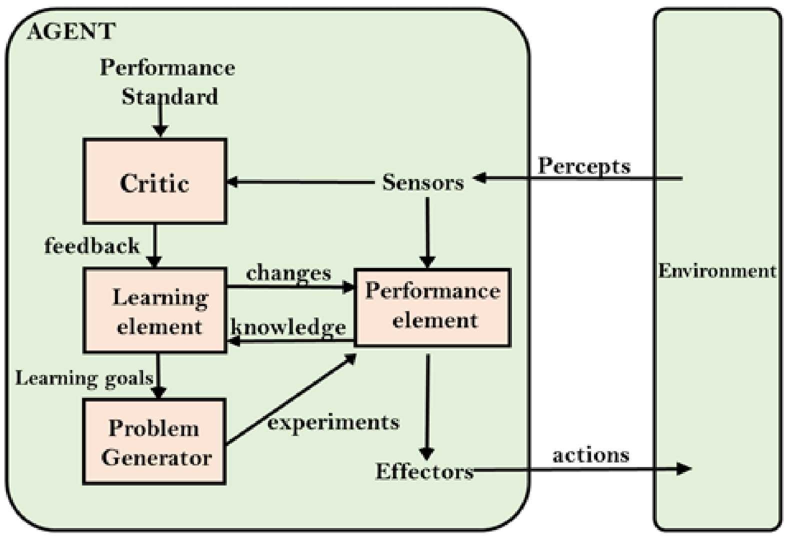

Sensors
Preceps
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be In such a state
What action I should do now
Agent
Actuators
Action
Environment

5. Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:

1. **Learning element:** It is responsible for making improvements by learning from environment
2. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
3. **Performance element:** It is responsible for selecting external action
4. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

- Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.



Agent components

Agents can be represented in three ways:

- Atomic/factored/structured is a qualitative measure of how much "internal structure" those models have, from least to most. Atomic models have no internal structure; the state either does or does not match what you're looking for. In a sliding tile puzzle, for instance, you either have the correct alignment of tiles or you do not.

- Factored models have more internal structure, although exactly what will depend on the problem. Typically, you're looking at variables or performance metrics of interest; in a sliding puzzle, this might be a simple heuristic like "number of tiles out of place," or "sum of manhatten distances."

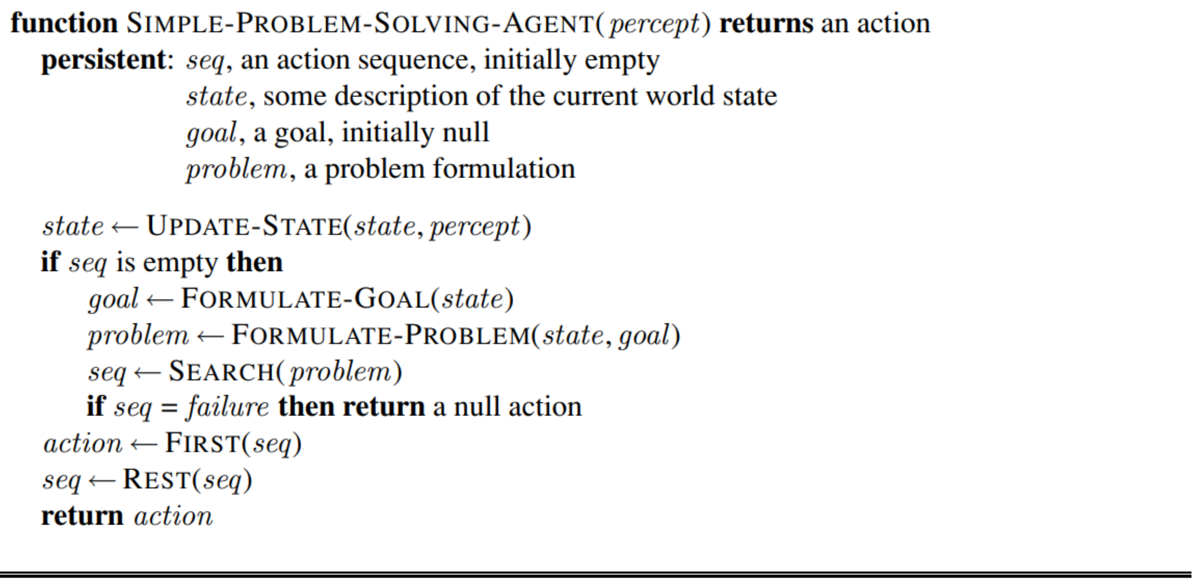
- Structured models have still more; again, exactly what depends on the problem, but they're often relations either of components of the model to itself, or components of the model to components of the environment.

MODULE-2

Problem Solving Agents

- In goal formulation, we decide which aspects we are interested in and which aspects can be ignored.
- In the goal formulation process, the goal is to be set and we should assess those states in which the goal is satisfied.
- In problem formulation, we decide how to manipulate the important aspects, and ignore the others. So, without doing goal formulation, if we do the problem formulation, we would not know what to include in our problem and what to leave, and what should be achieved.
- So problem formulation must follow goal formulation. That means problem formulation must be done only after the goal formation is done.
- The process of looking for a sequence of actions that reaches the goal is called search. A search algorithm takes a problem as input and returns a solution in the form of an action sequence.
- Once a solution is found, the actions it recommends can be carried out. This is called the execution phase. Thus, we have a simple "formulate, search, execute" design for the agent.

Simple problem solving Agent



```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

Steps performed by Problem-solving agent

- Goal Formulation: It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal.

- Problem Formulation: It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal.
- There are following five components involved in problem formulation:
- Initial State: It is the starting state or initial step of the agent towards its goal.
- Actions: It is the description of the possible actions available to the agent.
- Transition Model: It describes what each action does.
- Goal Test: It determines if the given state is a goal state.
- Path cost: It assigns a numeric cost to each path that follows the goal. The problem solving agent selects a cost function, which reflects its performance measure.
    - State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions. The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.
- Search: It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.
- Solution: It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- Execution: It executes the best optimal solution from the searching algorithms to reach the goal state from the current state n goal formulation.

Example problems

Basically, there are two types of problem approaches:

- Toy Problem: It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.
- Real-world Problem: It is real-world based problems which require solutions. Unlike a toy problem, it does not depend on descriptions, but we can have a general formulation of the problem.

Some Toy Problems

Vaccuum cleaner problem

This can be formulated as a problem as follows:

- States: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states.
- Initial state: Any state can be designated as the initial state.
- Actions: In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments might also include Up and Down.
- Transition model: The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect.
- Goal test: This checks whether all the squares are clean.
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

State space of Vacuum cleaner problem

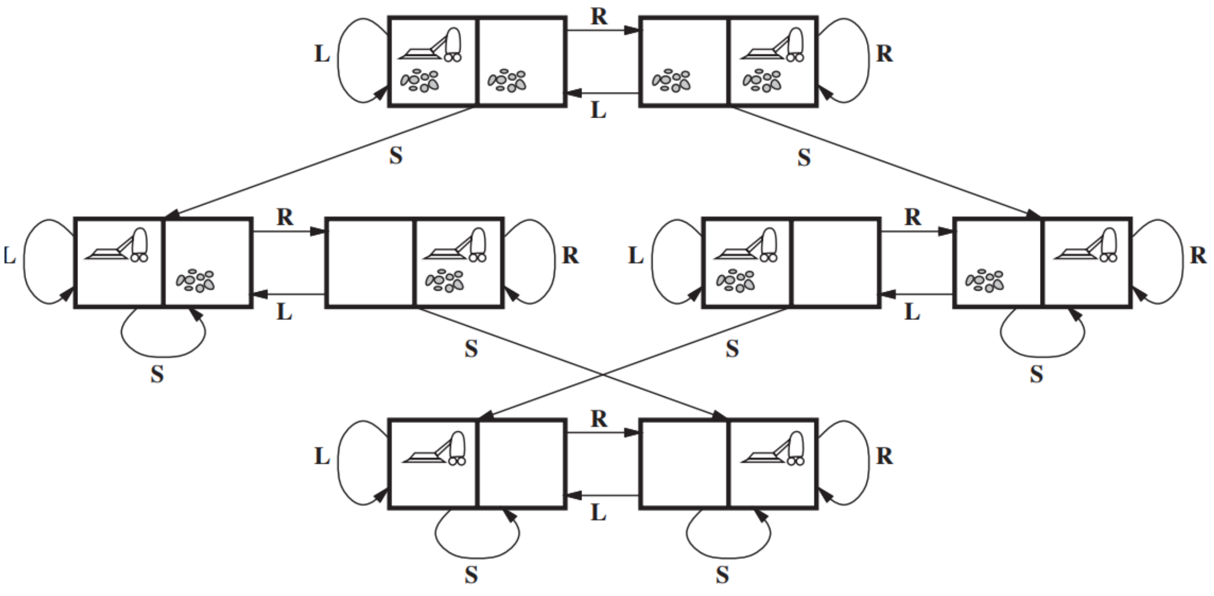
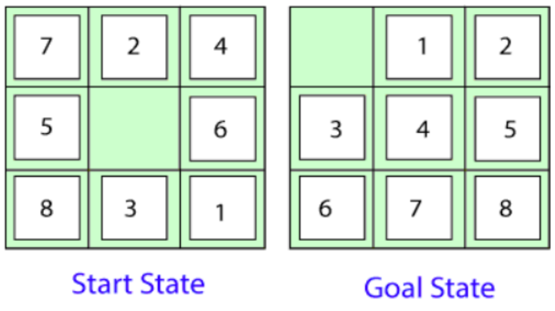


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

8 Puzzle Problem:

Here, we have a 3×3 matrix with movable tiles numbered from 1 to 8 with a blank space. The tile adjacent to the blank space can slide into that space. The objective is to reach a specified goal state similar to the goal state, as shown in the below figure. The task is to convert the current state into goal state by sliding digits into the blank space.



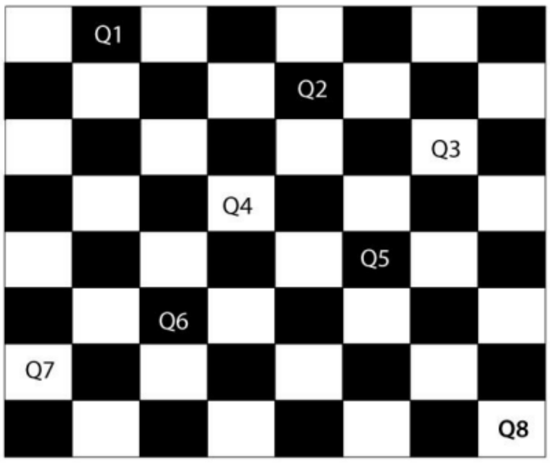
Here the task is to convert the current(Start) state into goal state by sliding digits into the blank space. The problem formulation is as follows:

- States: It describes the location of each numbered tiles and the blank tile.
- Initial State: We can start from any state as the initial state.
- Actions: Here, actions of the blank space is defined, i.e., either left, right, up or down
- Transition Model: It returns the resulting state as per the given state and actions.
- Goal test: It identifies whether we have reached the correct goal-state.
- Path cost: The path cost is the number of steps in the path where the cost of each step is 1.

Note: The 8-puzzle problem is a type of sliding-block problem which is used for testing new search algorithms in artificial intelligence.

8-queens problem:

The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either diagonally or in same row and column. From the following figure, we can understand the problem as well as its correct solution



It is noticed from the above figure that each queen is set into the chessboard in a position where no other queen is placed diagonally, in same row or column. Therefore, it is one right approach to the 8-queens problem. For this problem, there are two main kinds of formulation: 1. Incremental formulation: It starts from an empty state where the operator augments a queen at each step. Following steps are involved in this formulation:

- States: Arrangement of any 0 to 8 queens on the chessboard
- .• Initial State: An empty chessboard
- Actions: Add a queen to any empty box.
- Transition model: Returns the chessboard with the queen added in a box.
- Goal test: Checks whether 8-queens are placed on the chessboard without any attack. • Path cost: There is no need for path cost because only final states are counted. In this formulation, there is approximately 1.8 x 10^{14} possible sequence to investigate.

Complete-state formulation: It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks. Following steps are involved in this formulation

- States: Arrangement of all the 8 queens one per column with no queen attacking the other queen.
- Actions: Move the queen at the location where it is safe from the attacks.

This formulation is better than the incremental formulation as it reduces the state space from 1.8 x 10^{14} to 2057, and it is easy to find the solutions.

Number problem

Knuth conjectured that, starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5\,.$$

The problem definition is very simple:
- States: Positive numbers.
- Initial state: 4.
- Actions: Apply factorial, square root, or floor operation (factorial for integers only).
- Transition model: As given by the mathematical definitions of the operations.
- Goal test: State is the desired positive integer

Real-world problems

Route-finding algorithms

They are used in a variety of applications. Some, such as Web sites and in-car systems that provide driving directions, are relatively straightforward extensions of the Romania example. . Consider the airline travel problems that must be solved by a travel-planning Web site:

- States: Each state obviously includes a location (e.g., an airport) and the current time.
- Initial state: This is specified by the user's query.
- Actions: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- Transition model: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
- Goal test: Are we at the final destination specified by the user?
- Path cost: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on

Touring problems

They are closely related to route-finding problems, but with an important difference. As with route finding, the actions correspond to trips between adjacent cities. The state space, however, is quite different. Each state must include not just the current location but also the set of cities the agent has visited. The goal test would check whether the agent is in a place and all cities have been visited

The travelling salesman problem

It is a graph computational problem where the salesman needs to visit all cities (represented using nodes in a graph) in a list just once and the distances (represented using edges in the graph) between all these cities are known. The solution that is needed to be found for this problem is the shortest possible route in which the salesman visits all the cities and returns to the origin city.

A VLSI layout problem

This requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield. The layout problem comes after the logical design phase and is usually split into two parts: cell layout and channel routing.

In cell layout, the primitive components of the circuit are grouped into cells, each of which performs some recognized function. Each cell has a fixed footprint (size and shape) and requires a certain number of connections to each of the other cells. The aim is to place the cells on the chip so that they do not overlap and so that there is room for the connecting wires to be placed between the cells.

Channel routing finds a specific route for each wire through the gaps between the cells. These search problems are extremely complex, but definitely worth solving.

Robot navigation

Here a robot can move in a continuous space with (in principle) an infinite set of possible actions and states. For a circular robot moving on a flat surface, the space is essentially two-dimensional. When the robot has arms and legs or wheels that must also be controlled, the search space becomes many-dimensional. Advanced techniques are required just to make the search space finite.

Protein design problem

- The goal in rational protein design is to predict amino acid sequences that will fold to a specific protein structure. Although the number of possible protein sequences is vast, growing exponentially with the size of the protein chain, only a subset of them will fold reliably and quickly to one native state.
- Protein design involves identifying novel sequences within this subset. The native state of a protein is the conformational free energy minimum for the chain.
- In design, a tertiary structure is specified, and a sequence that will fold to it is identified. Hence, it is also termed *inverse folding*. Protein design is then an optimization problem: using some scoring criteria, an optimized sequence that will fold to the desired structure is chosen.

Searching for Solutions

- Search tree is a tree generated as the search space is traversed. The search space itself is not necessarily a tree, frequently it is a graph. This tree specifies possible paths through the search space.
- Expansion of nodes occurs as states are explored, the corresponding nodes are expanded by applying the successor function which generates a new set of (child) nodes.
- Fringe (frontier) is the set of nodes not yet visited and newly generated nodes are added to the fringe.
- Search strategy determines the selection of the next node to be expanded. It can be achieved by ordering the nodes in the fringe – e.g. queue (FIFO), stack (LIFO), "best" node w.r.t. some measure (cost).
- A loopy path is a special case of redundant paths, where there are more than one paths from one state to another (for example, Arad — Sibiu and Arad — Zerind — Oradea — Sibiu).

- The redundant path situation occurs in almost every problem, and often makes the solution algorithm less efficient, worsening the performance of the searching agent. One way to eliminate the redundancy is to utilize the advantage given by the problem definition itself. For example, in the case of traveling from Arad to Bucharest, since the path costs are additive and step costs are non-negative, only one path among the various redundant paths has the least cost (and it is the shortest distance between the two states), and loopy paths are never better than the same path with loops removed.

Graph Search

Tree search can be used is the state space is a tree, otherwise graph search must be used. All search algorithms (BFS, DFS, uniform-cost, A*, etc) are variations of one of these (usually graph search). The only difference between tree search and graph search is that tree search does not need to store the explored set, because we are guaranteed never to attempt to visit the same state twice.

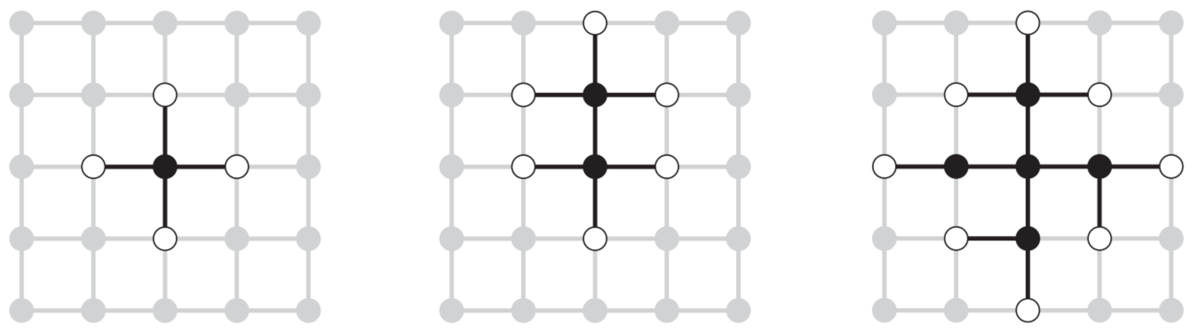
Tree Search Algorithm:

```
function TREE-SEARCH(problem) returns a solution, or failure
   initialize the frontier using the initial state of problem
   loop do
       if the frontier is empty then return failure
       choose a leaf node and remove it from the frontier
       if the node contains a goal state then return the corresponding solution
       expand the chosen node, adding the resulting nodes to the frontier
```

Graph Search Algorithm:

```
function GRAPH-SEARCH(problem) returns a solution, or failure
   initialize the frontier using the initial state of problem
   initialize the explored set to be empty
   loop do
       if the frontier is empty then return failure
       choose a leaf node and remove it from the frontier
       if the node contains a goal state then return the corresponding solution
       add the node to the explored set
       expand the chosen node, adding the resulting nodes to the frontier
          only if not in the frontier or explored set
```

The separation property of GRAPH-SEARCH is illustrated on a rectangular-grid problem in the following figure. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

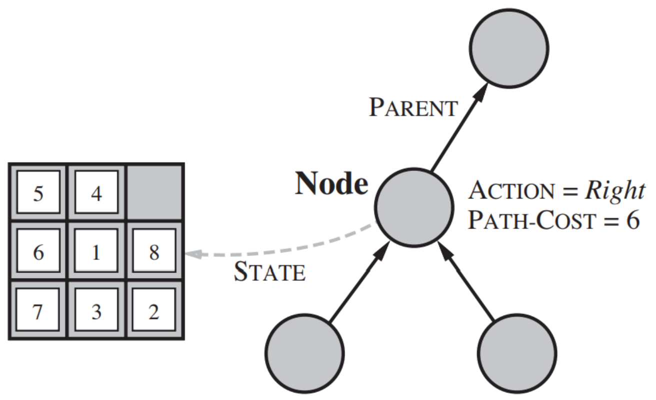


Infrastructure for Search Algorithms

For each node n of the tree, we have a structure that contains five components:

- n.STATE: the state in the state space to which the node corresponds.
- n.PARENT: the node in the search tree that generated this node.
- n.ACTION: the action that was applied to the parent to generate the node.
- n.DEPTH: the depth of tho node n, i.e., number of nodes in the path from the rood to this node n.
- n.PATH-COST: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.

Eg:



Algorithm for maintenance of the infrastructure

```
function CHILD-NODE(problem, parent, action) returns a node
   return a node with
      STATE = problem.RESULT(parent.STATE, action),
      PARENT = parent, ACTION = action,
      PATH-COST = parent.PATH-COST + problem.STEP-COST(parent.STATE, action)
```

Measuring Performance of Search algorithms:

Strategies are evaluated based on:

- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded space

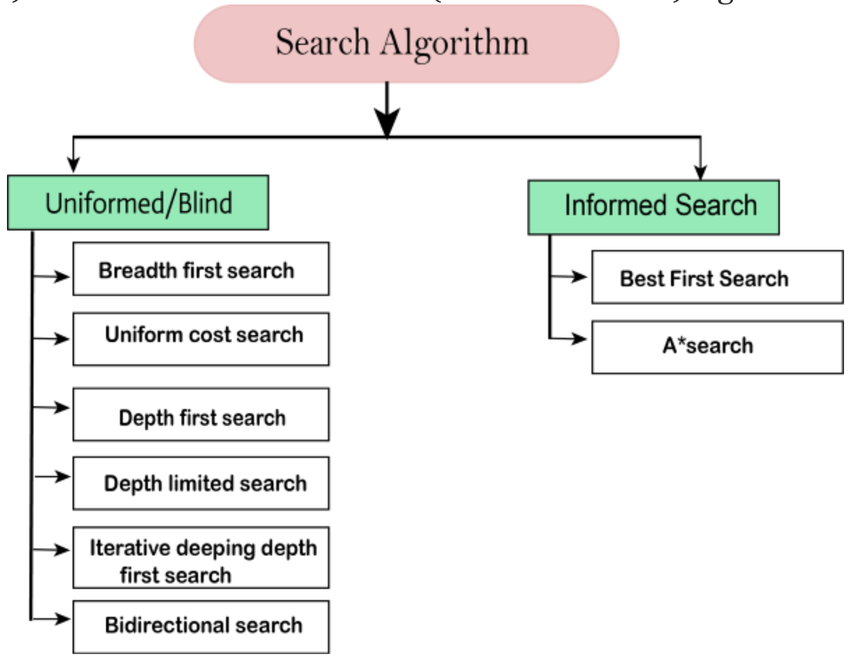
- complexity—maximum number of nodes in memory
- optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of :

- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be ∞)

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

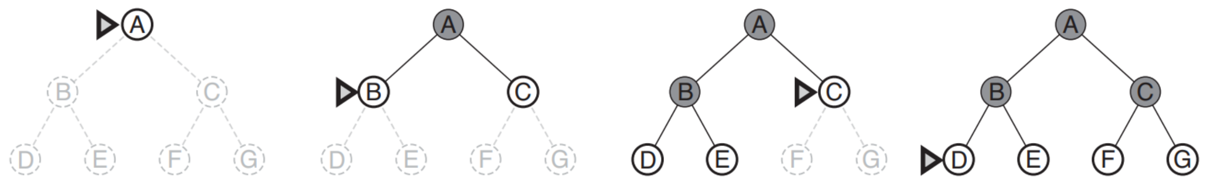
- BFS will provide a solution if any solution exists.

- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:



Algorithm:

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

Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.......+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Uniform-cost Search Algorithm:

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their

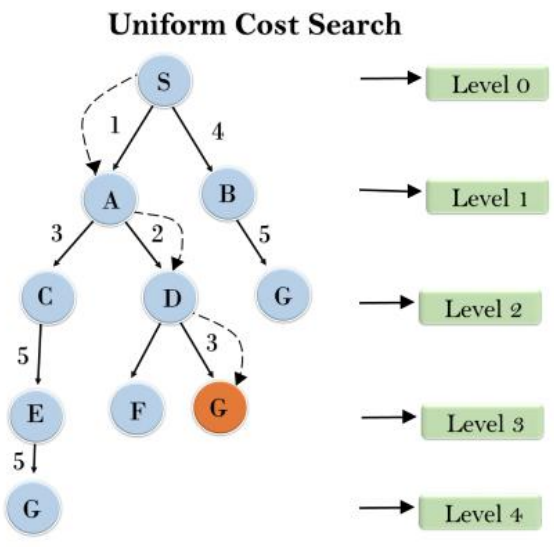
path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Eg:



Hence at each level, minimum cost is considered and other paths are discarded.
Completeness:
Uniform-cost search is complete, such as if there is a solution, UCS will find it.
Time Complexity:
Let C* is Cost of the optimal solution, and ε is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε. Hence, the worst-case time complexity of Uniform-cost search is $O(b^{1 + [C^*/\varepsilon]})/$.
Space Complexity:
The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $O(b^{1 + [C^*/\varepsilon]})$.
Optimal:
Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Algorithm:

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

Depth-first Search

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Advantages:

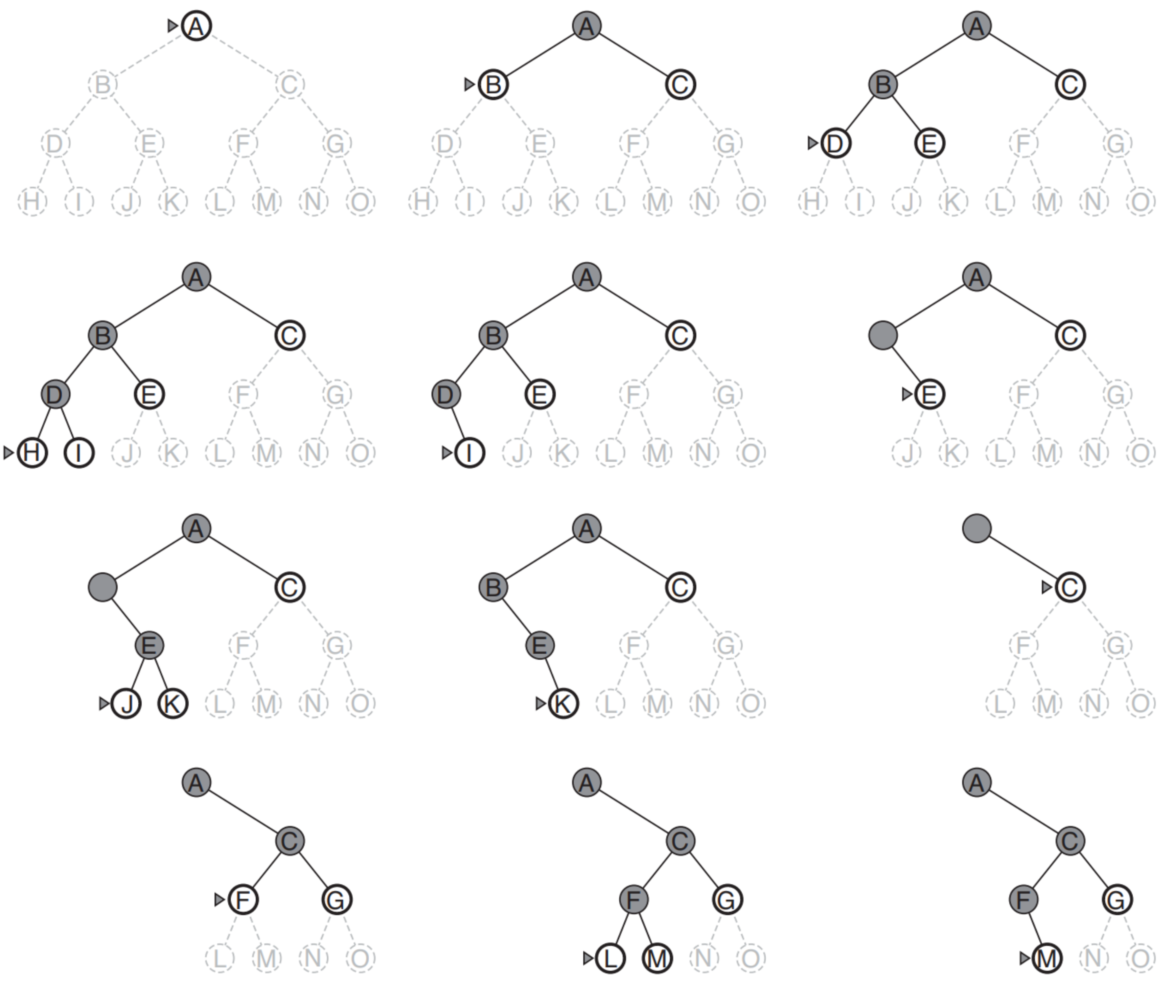
- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantages:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.
- Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.
- Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:
- $T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$
- Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)
- Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is O(bm).
- Optimal: DFS search algorithm is non-optimal, as it may generate a large

number of steps or high cost to reach to the goal node.

Example:



Once a wrong path is chosen, then pruning of entire subtree is done.

Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

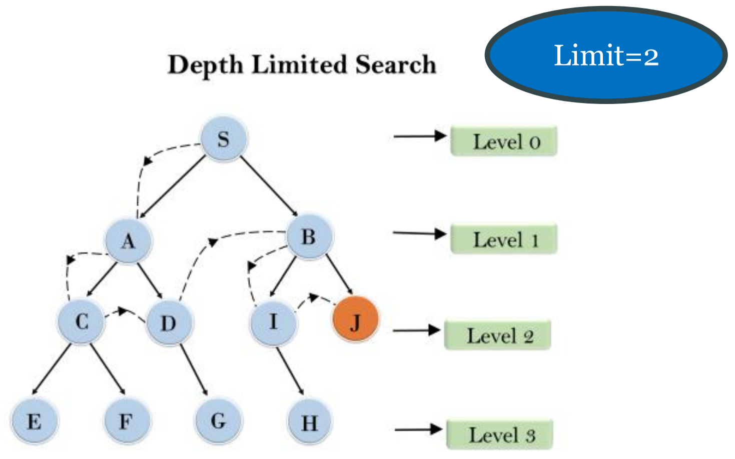
- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Algorithm:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
     cutoff_occurred? ← false
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        result ← RECURSIVE-DLS(child, problem, limit − 1)
        if result = cutoff then cutoff_occurred? ← true
        else if result ≠ failure then return result
     if cutoff_occurred? then return cutoff else return failure
```

Eg:



Iterative deepening Depth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found. This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found. This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

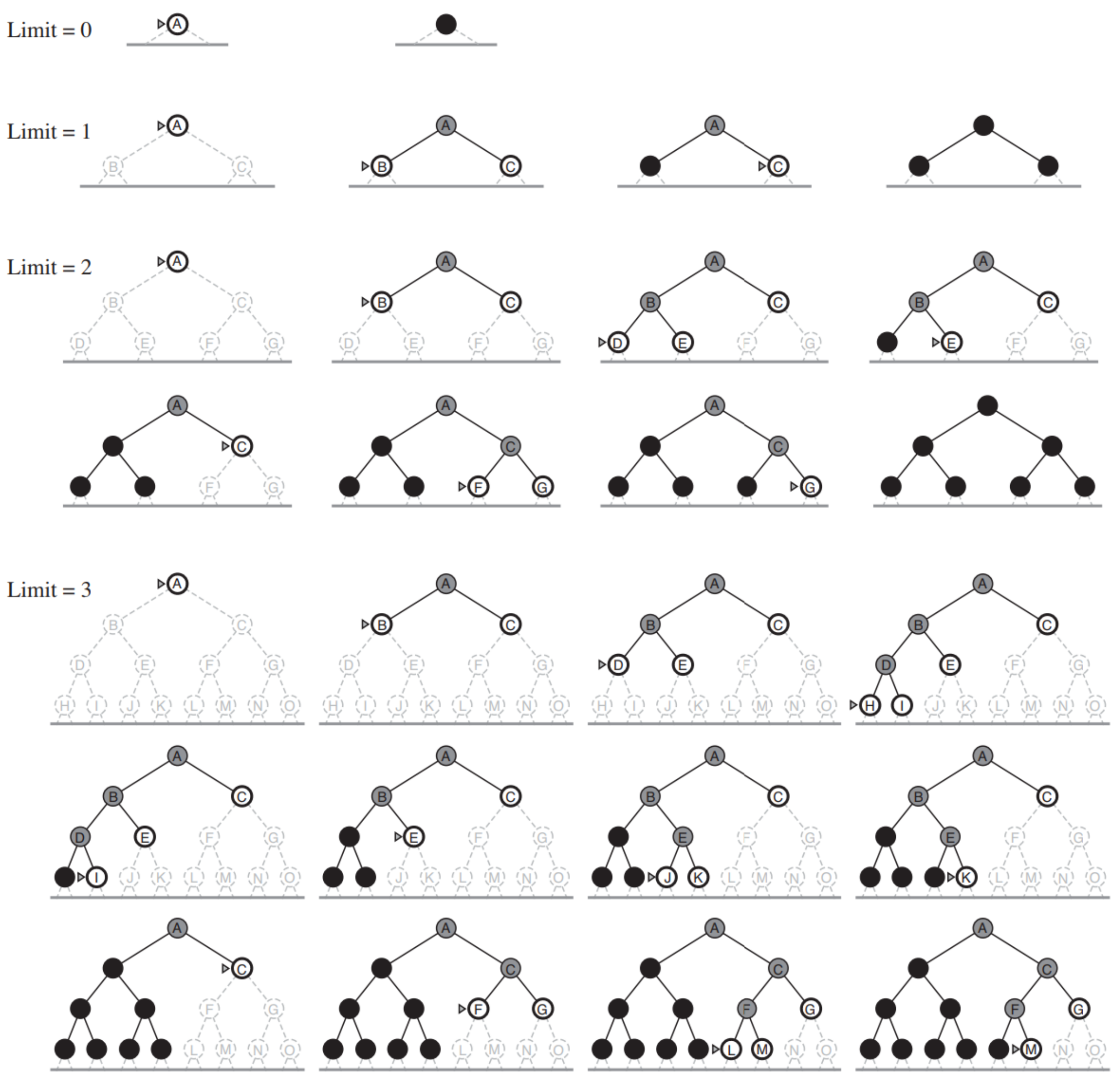
Advantages:

- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Eg:



Different iterations of Iterative Deepening Search

Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other. Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

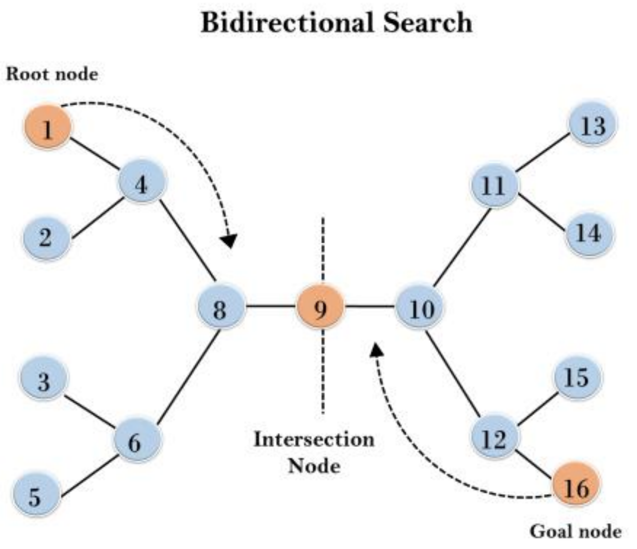
- Bidirectional search is fast.
- Bidirectional search requires less memory

Disadvantages:

- Implementation of the bidirectional search tree is difficult.

- In bidirectional search, one should know the goal state in advance.

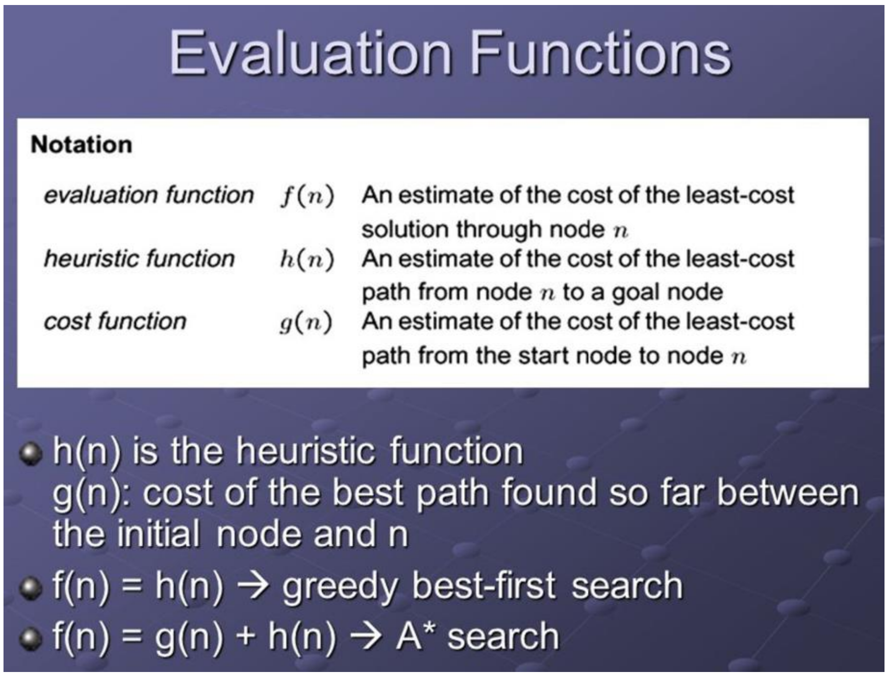
Example:
In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.
The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.
Time Complexity: Time complexity of bidirectional search using BFS is $O(b^d)$.
Space Complexity: Space complexity of bidirectional search is $O(b^d)$.
Optimal: Bidirectional search is Optimal.

Informed (Heuristic) Search Strategies

- Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.
- Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.



Evaluation Functions

Notation

evaluation function	$f(n)$	An estimate of the cost of the least-cost solution through node n
heuristic function	$h(n)$	An estimate of the cost of the least-cost path from node n to a goal node
cost function	$g(n)$	An estimate of the cost of the least-cost path from the start node to node n

- h(n) is the heuristic function
  g(n): cost of the best path found so far between the initial node and n
- f(n) = h(n) → greedy best-first search
- f(n) = g(n) + h(n) → A* search

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found. In the informed search there are two main algorithms:

- Best First Search Algorithm(Greedy search)
- A* Search Algorithm

Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

f(n)= h(n).

Were, h(n)= estimated cost from node n to the goal.

Advantages:

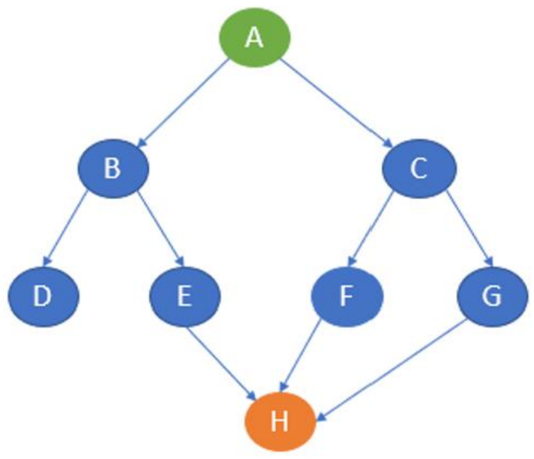
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.

- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

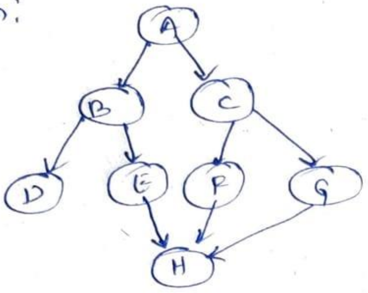
Problem



NODES	HEURISTICS
A	13
B	12
C	4
D	7
E	3
F	8
G	2
H	0

Solution:

Problem:



Heuristic Table

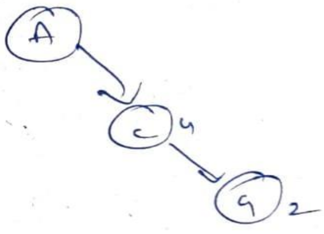
Nodes	h(n)
A	13
B	12
C	4
D	7
E	3
F	8.
G	2
H	0

Given,

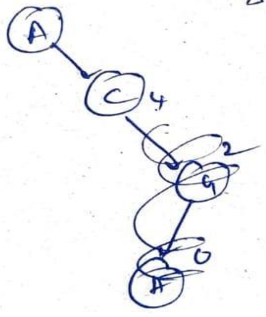
Source Node = A

Goal Node = H

Step2:

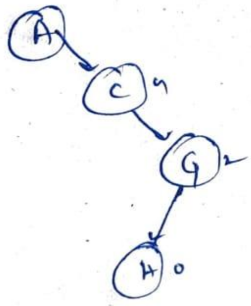


Step2:



~~Step2~~:

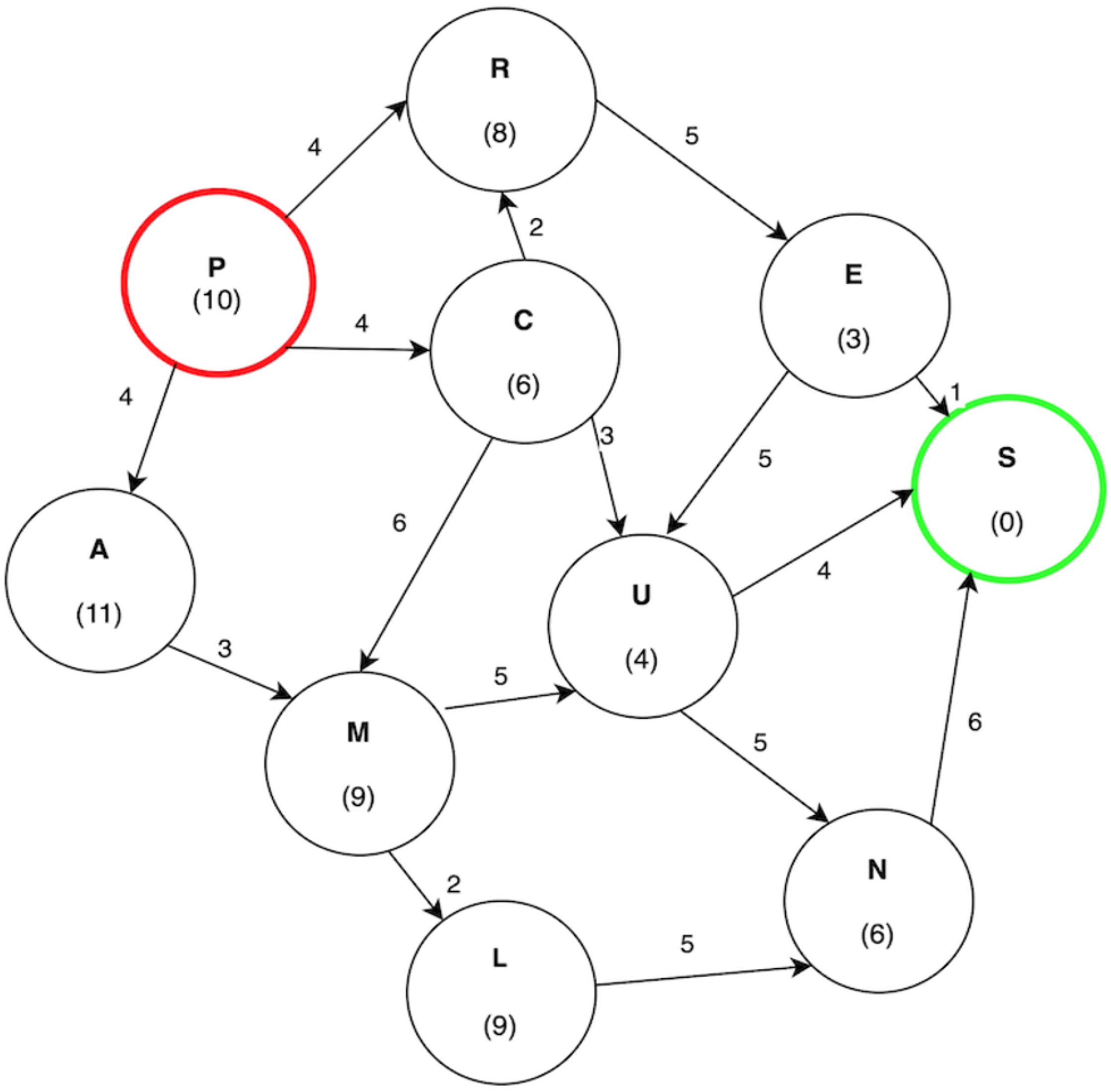
Step3:



	Open List	Closed List
1.	[A]	[A]
2.	[B, C]	[A, C]
3.	[B, F, G]	[A, C, G]
4.	[B, F, H]	

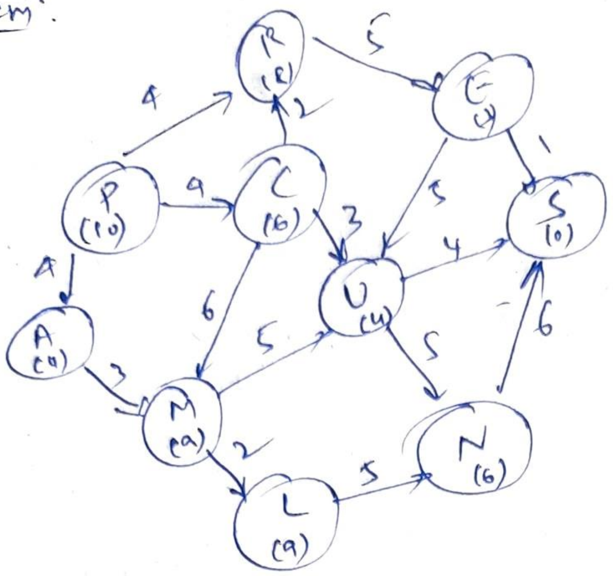
Hence solution A → C → G → H

Problem:



Solution:

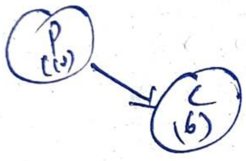
Problem:



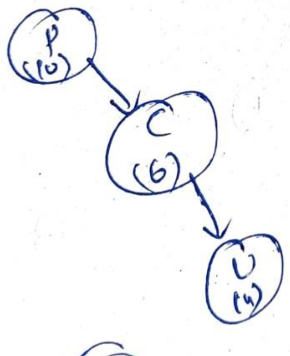
Solution:

Given Source Node = P, Goal Node = S

Step 1:

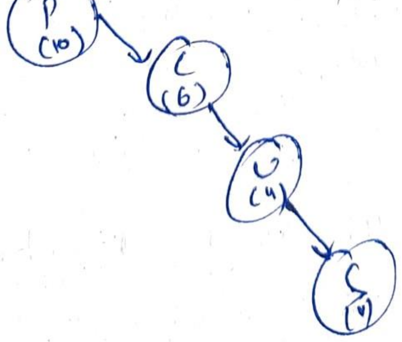


Step 2:



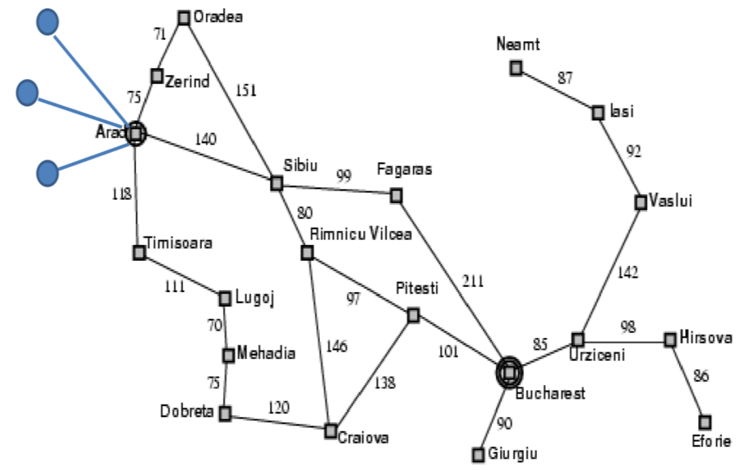
	Open List	Closed List
1.	[P]	φ
2.	[A, C, R]	[P]
3.	[A, R, M, U]	[P, C]
4.	[A, R, M, S, N]	[P, C, U]
5.	[A, R, M, N]	[P, C, U, S]

Step 3:



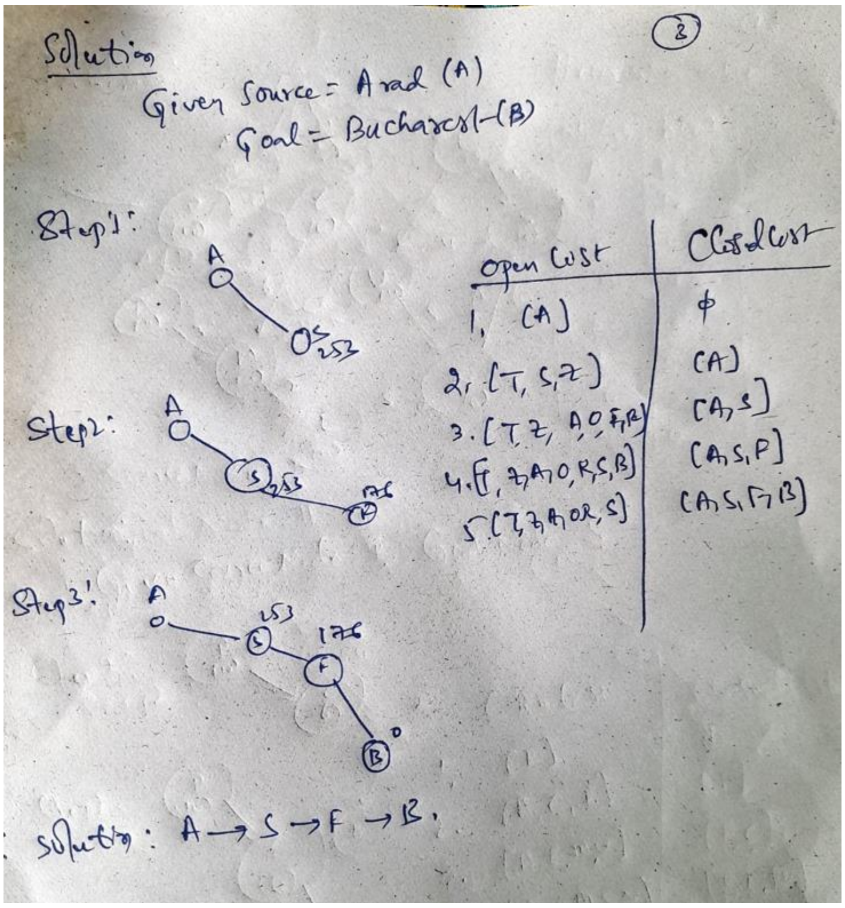
Hence, solution : P → C → U → S

Problem:



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Solution:



Solution (8)

Given Source = Arad (A)
Goal = Bucharest (B)

Step 1:

Step 2:

Step 3:

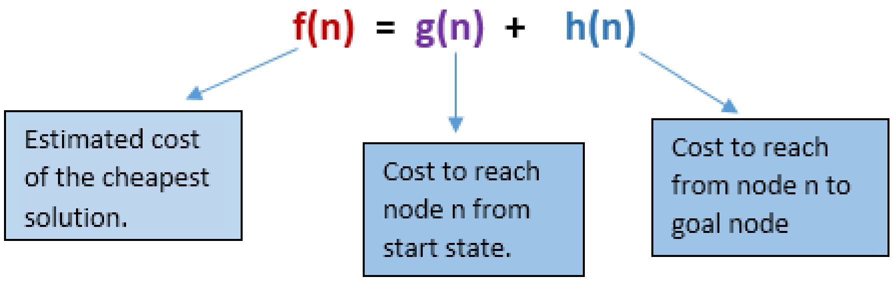
Open Cost	Closed Cost
1. [A]	φ
2. [T, S, Z]	[A]
3. [T, Z, A, O, F, R]	[A, S]
4. [T, Z, A, O, R, S, B]	[A, S, F]
5. [T, Z, A, O, R, S]	[A, S, F, B]

Solution : A → S → F → B.

A
S 253
F 176
B 0

A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n). In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence, we can combine both costs as following, and this sum is called as a fitness number.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

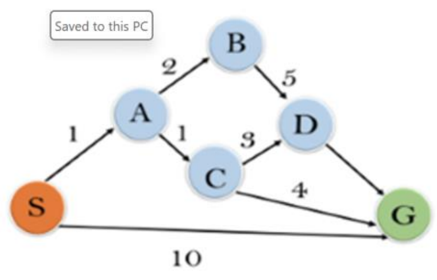
Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

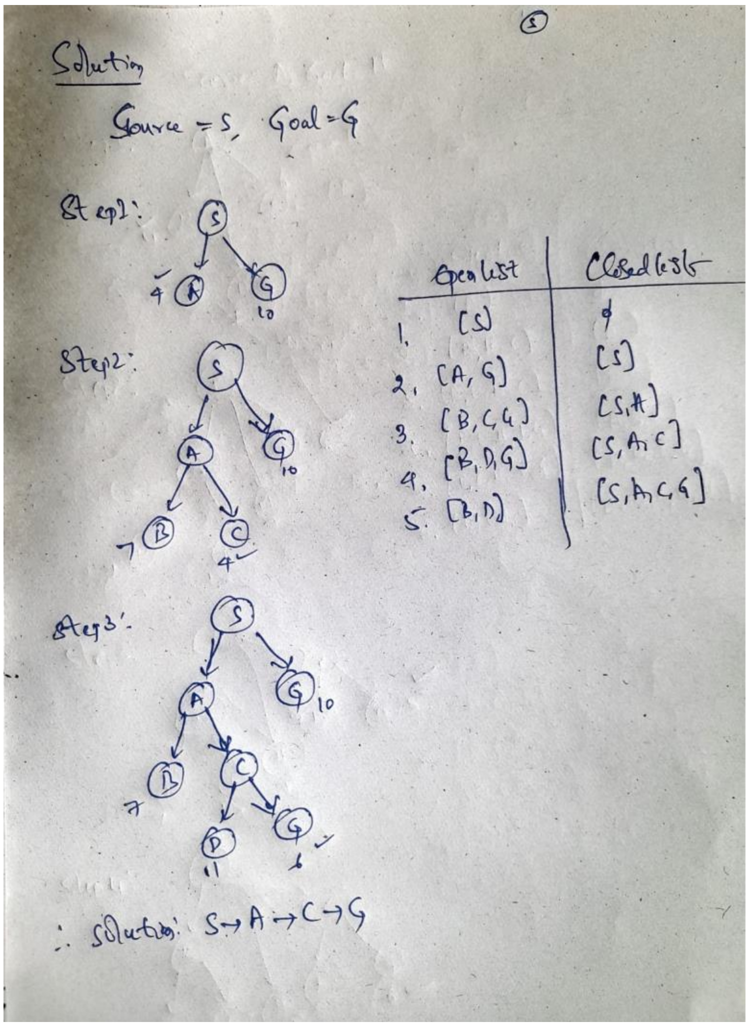
- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Problem:



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



5

Solution

Source = S, Goal = G

Step1:

S
A
4
G
10

Step2:

S
A
G
10
B
7
C
4

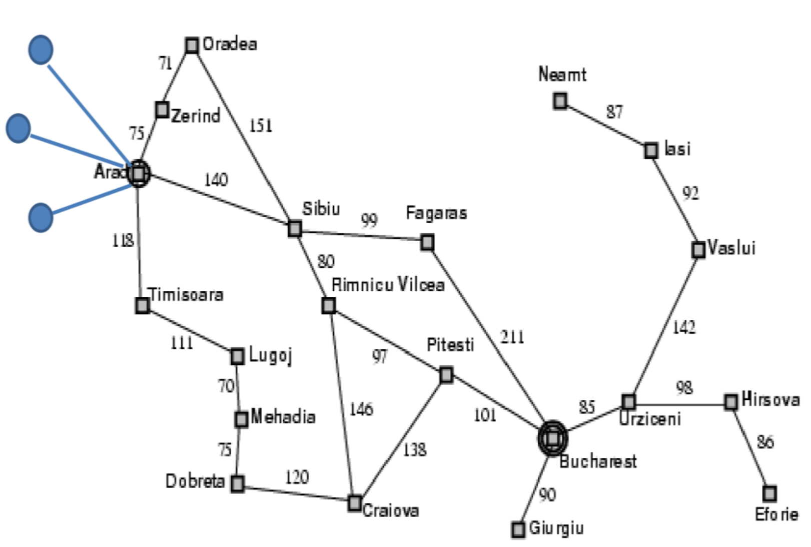
Step3:

S
A
G
10
B
7
C
D
11
G
6

	Open list	Closed list
1.	[S]	ϕ
2.	[A, G]	[S]
3.	[B, C, G]	[S, A]
4.	[B, D, G]	[S, A, C]
5.	[B, D]	[S, A, C, G]

∴ Solution: S → A → C → G

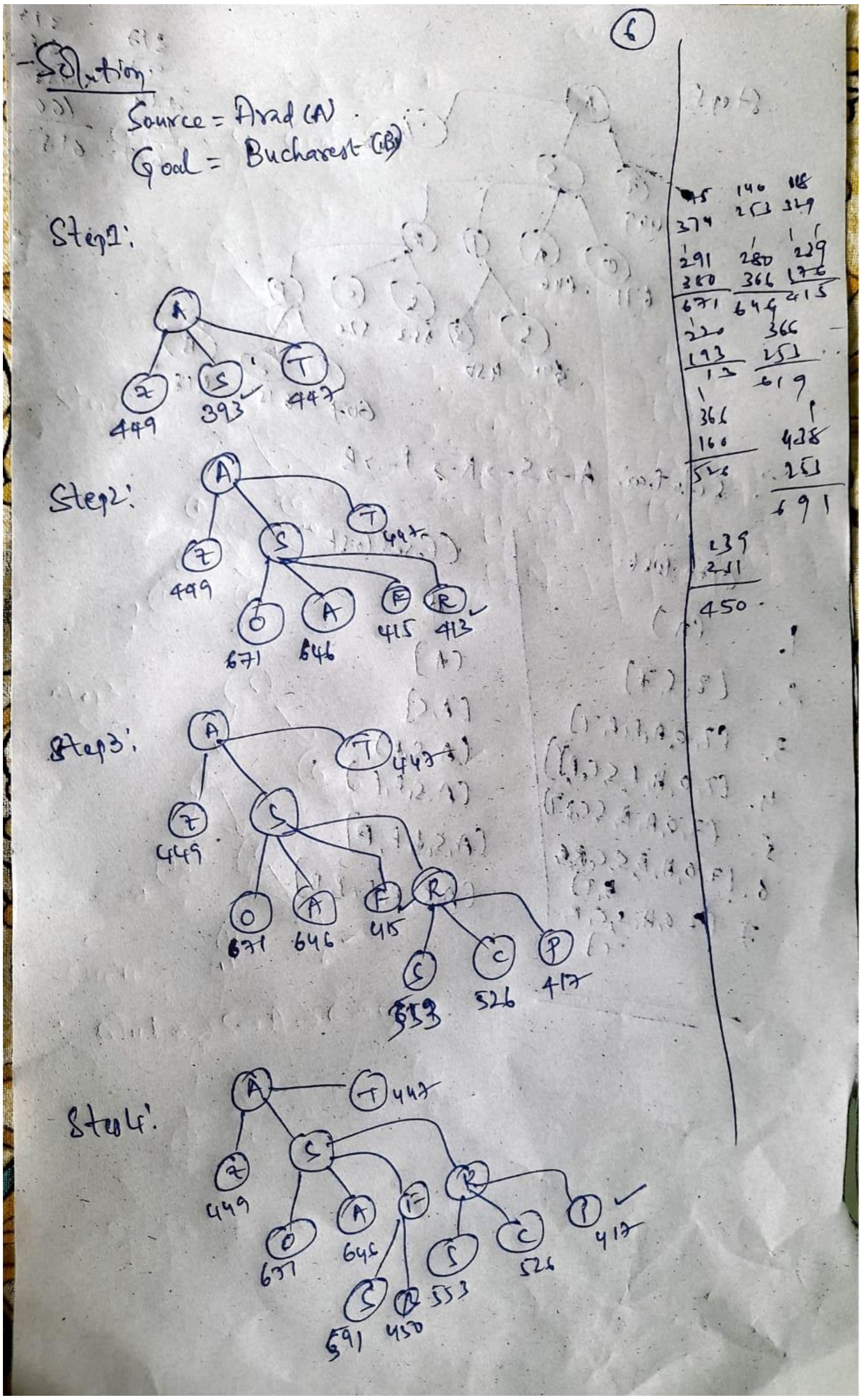
Problem:



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Solution:



Solution

Source = Arad (A)

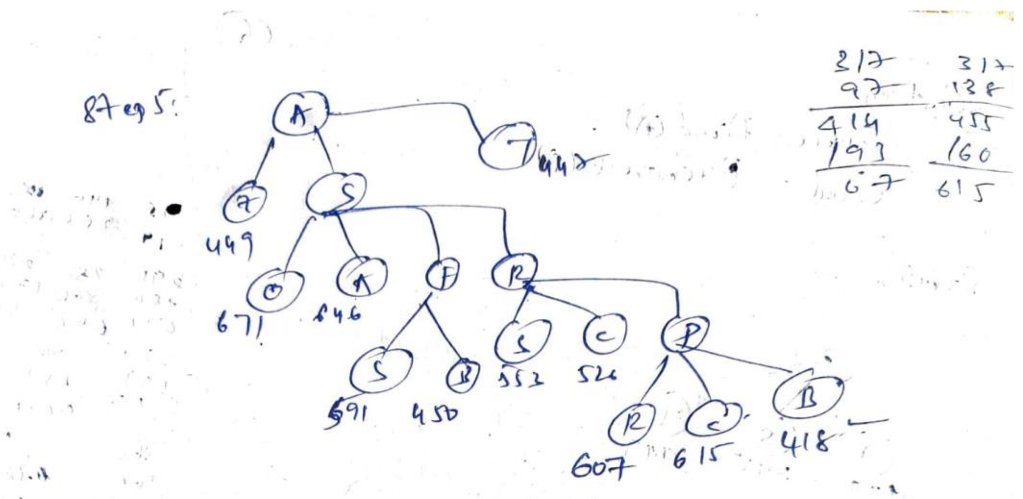
Goal = Bucharest (B)

Step1:

Step2:

Step3:

Step4:

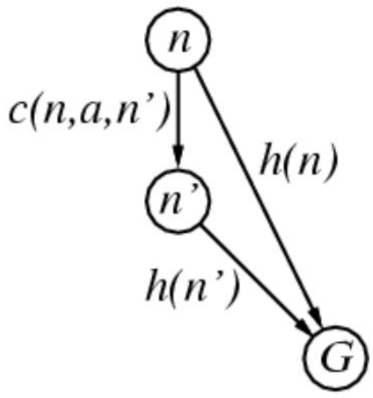


∴ Solution: $A \rightarrow S \rightarrow R \rightarrow P \rightarrow B$

	Open list	Closed Cost
1.	(A)	φ
2.	[Z, S, T]	(A]
3.	(Z, O, A, F, R, T]	[A, S]
4.	(Z, O, A, F, S, C, P, T]	(A, S, R)
5.	(Z, O, A, B, S, C, P, T]	(A, S, R, F)
6.	[Z, O, A, B, S, C, R, ~~B~~, T]	(A, S, R, F, P]
7.	[Z, O, A, S, C, R, T]	(A, S, R, F, P, B]

Admissibility of A* search:

- The heuristic function h(n) is called admissible if h(n) is never larger than h*(n), namely h(n) is always less or equal to true cheapest cost from n to the goal.
- A* is admissible if it uses an admissible heuristic, and h(goal) = 0
- If the heuristic function, h always underestimates the true cost (h(n) is smaller than h*(n)), then A* is guaranteed to find an optimal solution.
- Consistency of A* search:
- A heuristic is consistent if for every node n, every successor n' of n generated by any action a, h(n) ≤ c(n,a,n') + h(n')
- If h is consistent, we have f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') ≥ g(n) + h(n) = f(n), f(n) is non-decreasing along any path.



Recursive Best First Search

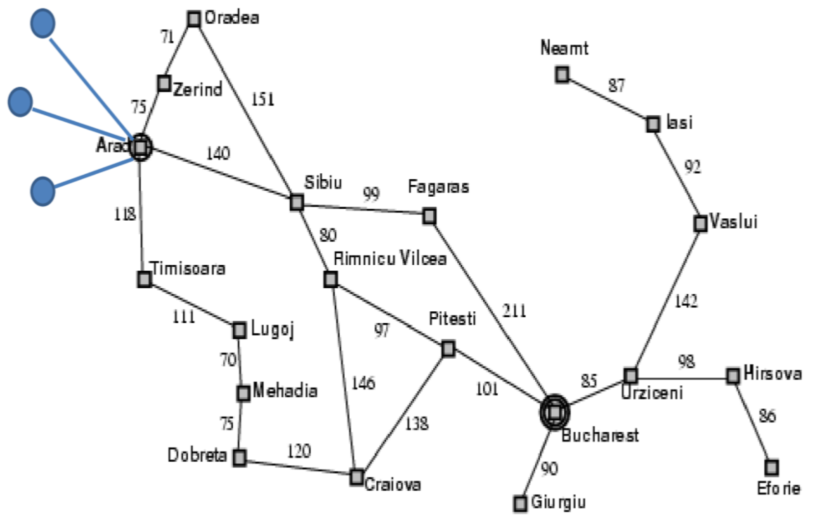
It is simple recursive algorithm that resembles the operation of standard best first search but uses only linear space. It is similar to recursive DFS and differs from Recursive DFS as follows: It keeps track of the f value of the best alternative path available from any ancestor of the current node. Instead of continuing indefinitely down the current path.

Algorithm:

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), $\infty$)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new $f$-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors $\leftarrow$ [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, $\infty$
  for each s in successors do /* update $f$ with value from previous search, if any */
      $s.f \leftarrow \max(s.g + s.h,\ node.f)$
  loop do
      best $\leftarrow$ the lowest $f$-value node in successors
      if $best.f > f\_limit$ then return failure, $best.f$
      alternative $\leftarrow$ the second-lowest $f$-value among successors
      result, $best.f \leftarrow$ RBFS(problem, best, min(f_limit, alternative))
      if result $\neq$ failure then return result
```

Problem:



Straight–line distance
to Bucharest

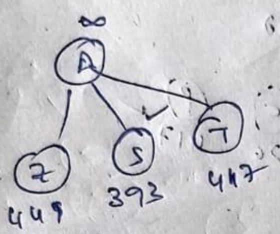
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Solution:

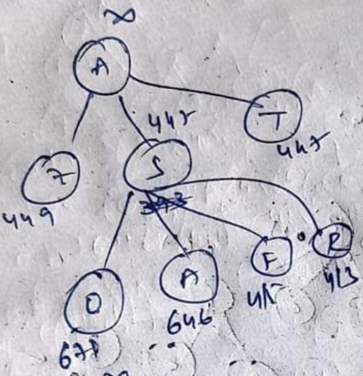
Solution: Source ~~Goal~~ = Arad (A)

Goal = Bucharest (B)

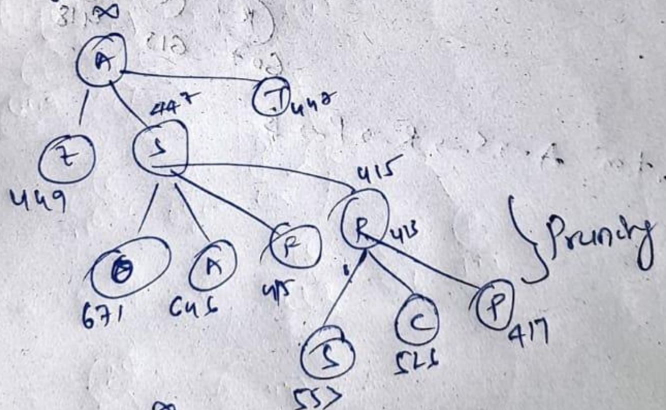
Step 1:



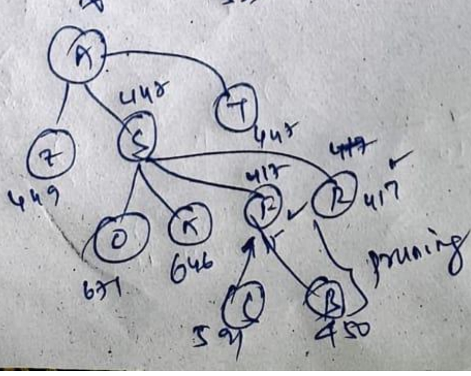
Step 2:

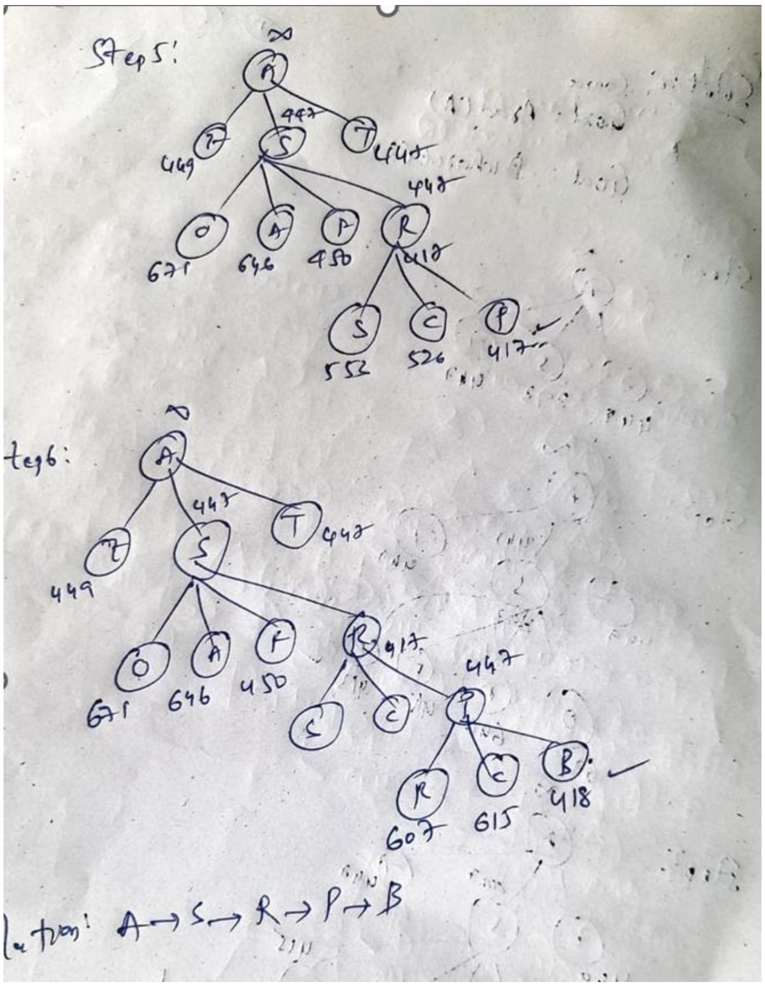


Step 3:



Step 4:





Memory Bound Search Algorithms:

IDA* (Iterative-Deepening A*)

IDA* algorithm - IDA* uses the same idea of DFID algorithm and uses the admissibility of the heuristic function by limiting the f-cost of the nodes examined by the DFS search, rather than the depth. IDA* (memory-bounded) algorithm does not keep any previously explored path (the same as A*). It needs to re-expand path if it is necessary and this will be a costly operation

Memory Bounded A* (MA*)

MA* is near-identical to A* aside from key differences:
When the number of nodes in OPEN and CLOSED reaches some preset limit, MA* prunes the OPEN list by removing the leaf-node with highest f-cost.
For each new successor the f-cost is propagated back up the tree.
This keeps the tree very “informed” allowing the search to make better decisions, at the cost of some overhead.

Simplified MA* (SMA*)

SMA* proceeds just like A*, expanding the best leaf until memory is full. At this point, it cannot add a new node to the search tree without dropping an old one. SMA* always drops the *worst* leaf node—the one with the highest f-value. Like RBFS, SMA* then backs up the value of the forgotten node to its parent. In this way, the ancestor of a forgotten subtree knows the quality of the best path in that subtree. With this information, SMA* regenerates the subtree only when all other paths have been shown to look worse than the path it has forgotten. Another way of saying this is that, if all the descendants of a node n are forgotten, then we will not know which way to go from n, but we will still have an idea of how worthwhile it is to go anywhere from n

Heuristic Functions:

- A heuristic function in artificial intelligence, also known as a heuristic or simply a heuristic, is an evaluation function used to estimate the cost or potential of reaching a goal state from a given state in a problem-solving domain.
- Heuristics are typically rules of thumb or approximate strategies that guide the search for a solution. They provide a way to assess the desirability of different options without exhaustively exploring every possibility.
- Heuristics are used to make informed decisions in situations where it's computationally expensive to search through all possible states or actions. They help prioritize the exploration of more promising paths.

Key Properties of Heuristic Functions

- Heuristic functions are essential components of AI problem-solving. To ensure their effectiveness, it's crucial to understand their key properties. Let's dive into two critical properties: admissibility and consistency.
- Admissibility is a fundamental property of heuristic functions that profoundly influences their use in AI problem-solving.
- Admissibility refers to a property of heuristic functions that ensures they never overestimate the true cost to reach a goal state. In other words, an admissible heuristic provides a lower bound on the actual cost.
- Imagine planning a road trip using a GPS navigation system. If the GPS estimates the travel time to be 4 hours, it's admissible if you reach your destination in less than or exactly 4 hours, but it's inadmissible if it takes longer. Admissible heuristics set an upper limit on the estimated cost.
- Admissible heuristics guarantee that search algorithms exploring the state space won't overlook optimal solutions. They create a balance, ensuring that the algorithm doesn't prematurely discard paths that might lead to the best outcome.
- Consistency, also known as the monotonicity property, is another critical aspect of heuristic functions.
- Consistency defines a heuristic's behavior by considering the estimated cost from the current state to a successor state along with the heuristic value of the successor state. If this combined value is always greater than or equal to the heuristic value of the current state, the heuristic is considered consistent.
- Let's return to our travel analogy. Suppose we're assessing travel times between cities. If the estimated travel time from City A to City B (the current state) plus the estimated travel time from City B to City C (the successor state) is greater than or equal to the estimated travel time from City A to City C (the goal state), then the heuristic is consistent.
- Consistent heuristics are particularly advantageous in informed search algorithms like A*. They ensure that as the search algorithm progresses, it doesn't encounter situations where a more promising path is overlooked due to heuristic inconsistencies.

Methods for Designing and Developing Heuristic Functions:

When it comes to designing heuristic functions, several methods and strategies can be employed. These methods are instrumental in developing heuristics that provide valuable guidance to search algorithms.

Approaches for Heuristic Design:

1. Domain Knowledge: Leveraging expert knowledge about the problem domain to construct heuristics.
2. Relaxation: Creating a simplified version of the problem where heuristics are more easily derived and then transferring these heuristics back to the original problem.
3. Pattern Databases: Storing precomputed heuristic values for specific problem subgoals, enabling efficient lookup during search.

Knowledge based Agents

- Knowledge-based agents are those agents who have the capability of maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.
- Knowledge-based agents are composed of two main parts:
  - Knowledge-base and
  - Inference system.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

Knowledgebase (KB)

Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world. Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Operations Performed by KBA

Following are three operations which are performed by KBA in order to show the intelligent behavior:

1. TELL: This operation tells the knowledge base what it perceives from the environment.
2. ASK: This operation asks the knowledge base what action it should perform.
3. Perform: It performs the selected action.

Following is the structure outline of a generic knowledge-based agents program:

```
function KB-AGENT(percept):
persistent: KB, a knowledge base
        t, a counter, initially 0, indicating time
TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
```

```
Action = ASK(KB, MAKE-ACTION-QUERY(t))
TELL(KB, MAKE-ACTION-SENTENCE(action, t))
 t = t + 1
 return action
```

- The knowledge-based agent takes percept as input and returns an action as output. The agent maintains the knowledge base, KB, and it initially has some background knowledge of the real world. It also has a counter to indicate the time for the whole process, and this counter is initialized with zero.
- The MAKE-PERCEPT-SENTENCE generates a sentence as setting that the agent perceived the given percept at the given time.
- The MAKE-ACTION-QUERY generates a sentence to ask which action should be done at the current time.
- MAKE-ACTION-SENTENCE generates a sentence which asserts that the chosen action was executed.

Various levels of knowledge-based agent:

A knowledge-based agent can be viewed at different levels which are given below:

1. Knowledge level

Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Logical level:

At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs. At the logical level we can expect to the automated taxi agent to reach to the destination B.

3. Implementation level:

This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implements his knowledge and logic so that he can reach to the destination.

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

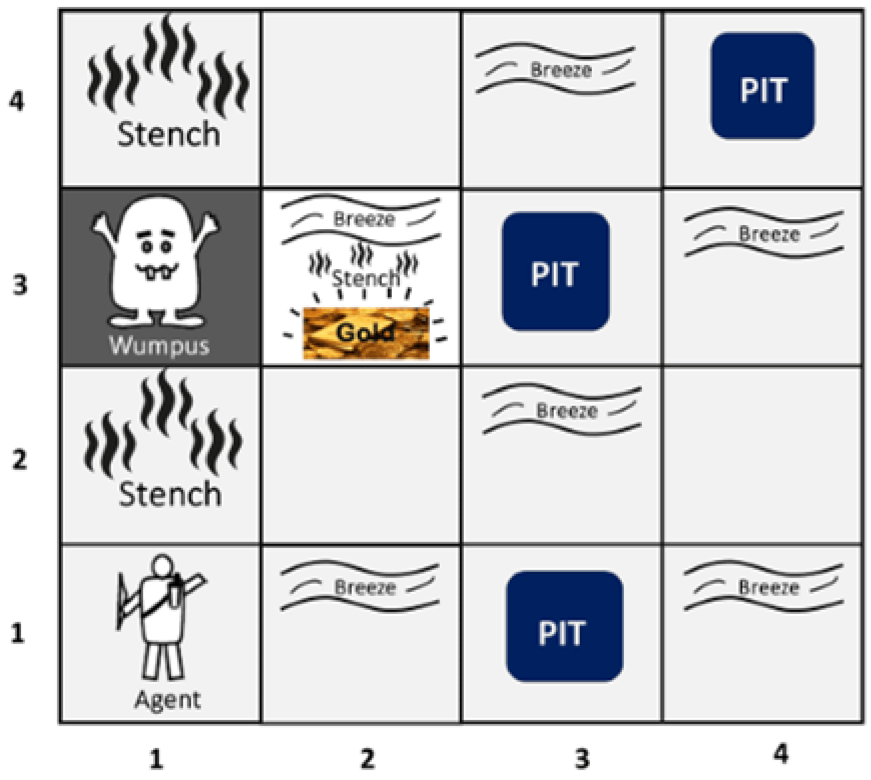
1. Declarative approach: We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
2. Procedural approach: In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

However, in the real world, a successful agent can be built by combining both declarative and procedural approaches, and declarative knowledge can often be compiled into more efficient procedural code.

Wumpus world:

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game Hunt the Wumpus by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.



- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

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

Environment:

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

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

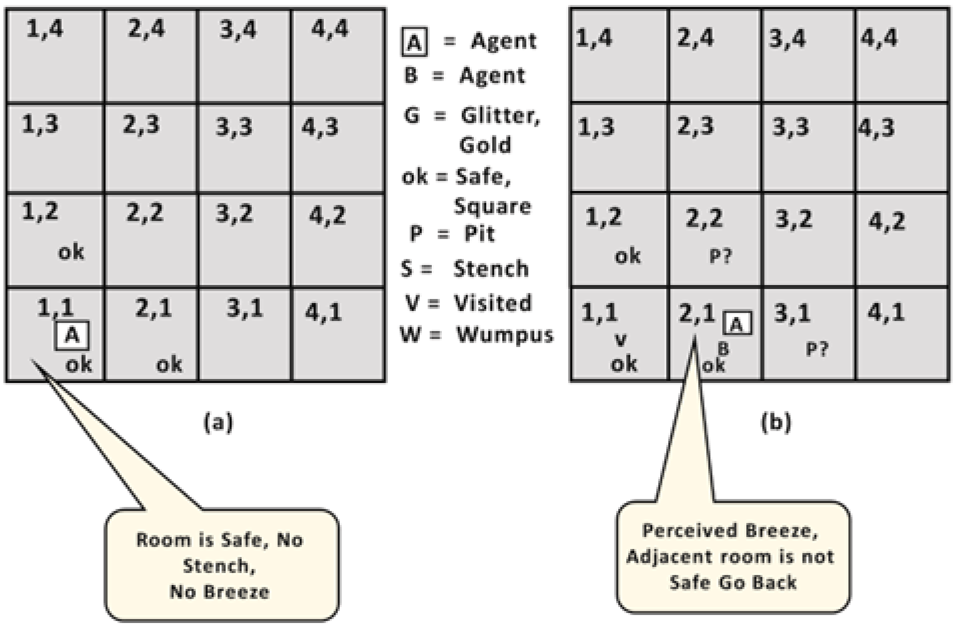
Sensors:

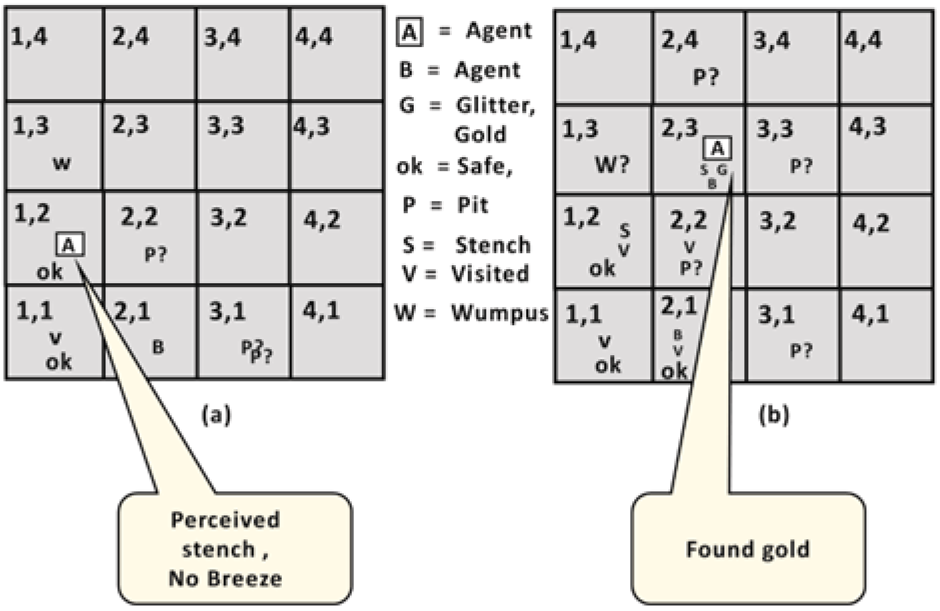
- The agent will perceive the stench if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive breeze if he is in the room directly adjacent to the Pit.
- The agent will perceive the glitter in the room where the gold is present.
- The agent will perceive the bump if he walks into a wall.
- When the Wumpus is shot, it emits a horrible scream which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as: [Stench, Breeze, None, None, None].

The Wumpus world Properties:

- Partially observable: The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- Deterministic: It is deterministic, as the result and outcome of the world are already known.
- Sequential: The order is important, so it is sequential.
- Static: It is static as Wumpus and Pits are not moving.
- Discrete: The environment is discrete.
- One agent: The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

Exploring the Wumpus world:





At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Logic

- Logic is a formal language to express real world sentences
- Syntax – notation to express sentences
- Semantics – meaning of the sentences
- Truth value – sentence truth value (True or False)
- Model – World for sentences

Entailment

If I was to tell you that either I was going to be promoted or get a big bonus because I just landed a huge account for the company I work for, then you would think a few things:

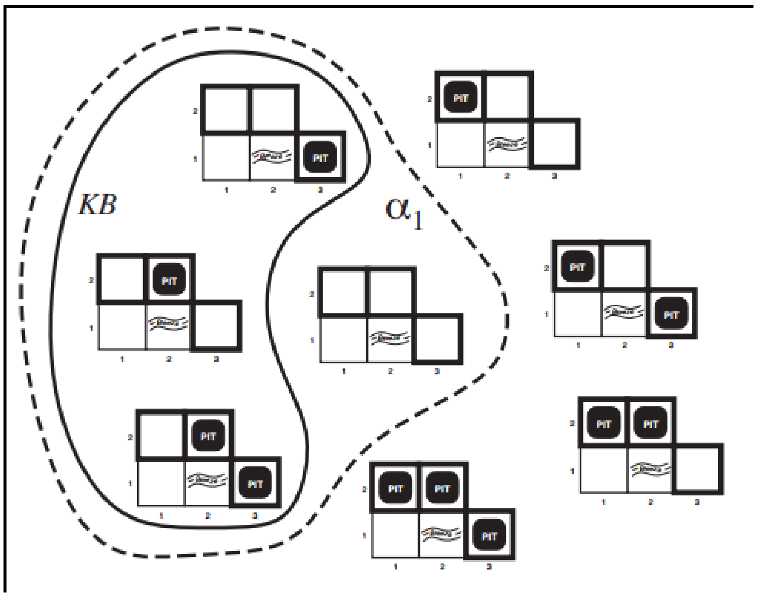
- If he doesn't get promoted, then at least he'll get compensated with a bonus
- If he doesn't get a bonus, then at least he'll get compensated with a promotion
- His company won't both *not* promote him *and not* give him a bonus.

All of these claims follow from the original claim. They "follow" in the sense that if the original claim is in fact true, then this conclusion *must* be true. There's some sense in which they "mean the same thing": they describe the same world or claim the same thing to be true about the world. A lot of logic consequence is similar: it's a relationship of "following" or "entailment" between statements which mean essentially the same thing. Other logical consequences are cases where one statement entails another statement (if the first is true, the second *must* be true), but not because they essentially mean the same thing. Instead, because the first statement is making a "stronger" claim than the other.

Logical entailment is denoted by: $\boldsymbol{\alpha| = \beta}$

Which means β follows from α

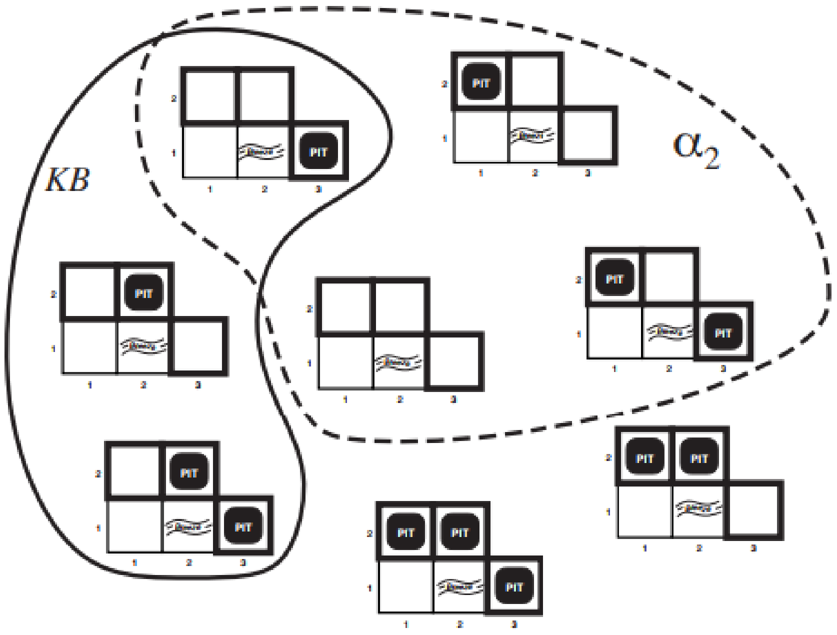
Eg:



α_1-No pit in [1,2]

Then, $\boldsymbol{KB| = \alpha_1}$

Another example:



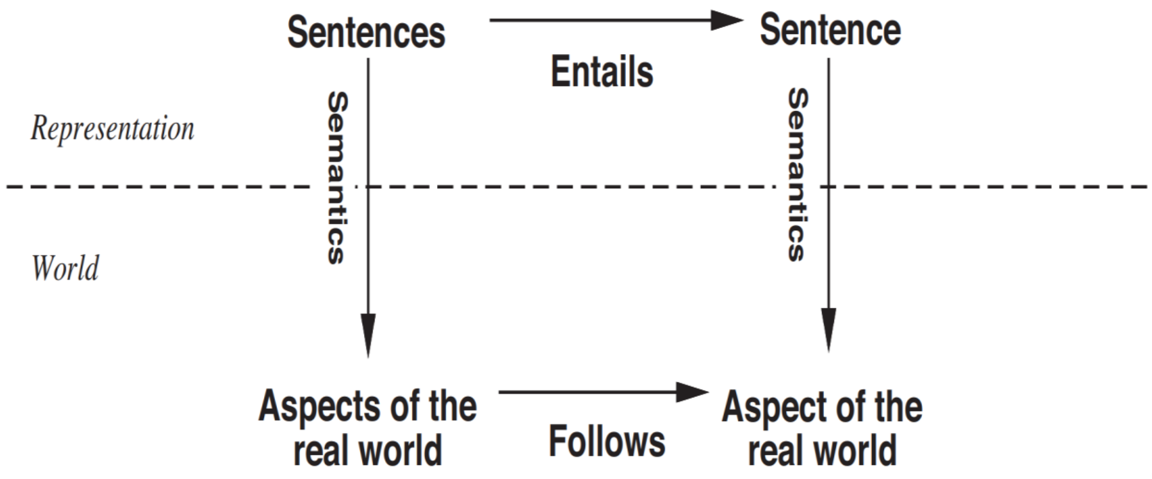
α_2*-No pit in [2,2]*

$KB| \neq \alpha_2$

Entailment is like the needle being in the haystack; inference is like finding it. This distinction is embodied in some formal notation: if an inference algorithm i can derive α from KB, we write
KB ⊢i α , which is pronounced "α is derived from KB by i" or "i derives α from KB."

- An inference algorithm that derives only entailment is called "sound"
- An inference algorithm that derives all possible entailment is called "complete"

Logical Reasoning with Entailment



Grounding

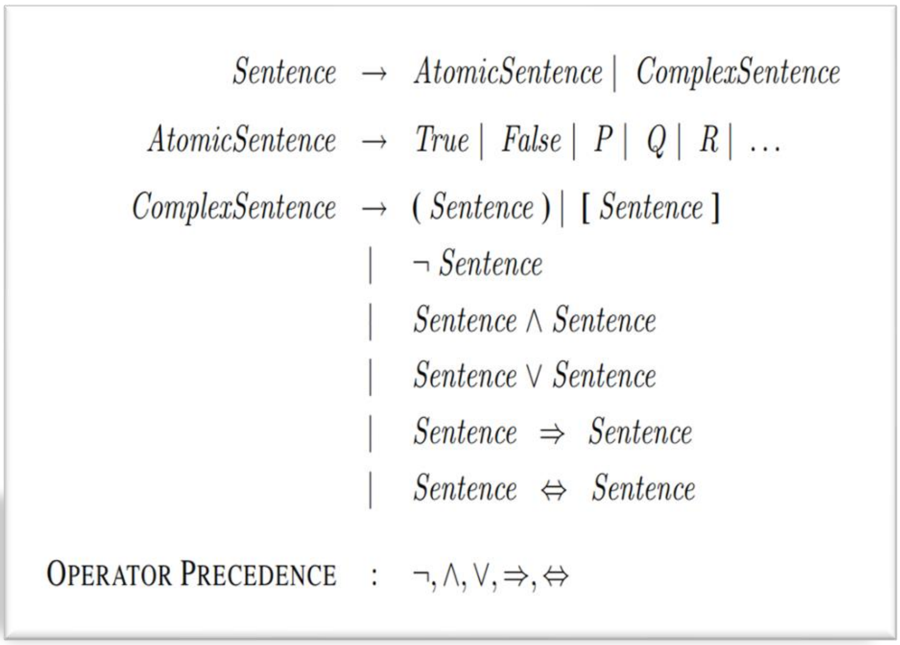
It is the connection between logical reasoning processes and the real environment in which the agent exists. In particular, *how do we know that* KB *is true in the real* *world?*. A simple answer is that the agent's sensors create the connection. For example, the wumpus-world agent has a smell sensor. The agent program creates a suitable sentence whenever there is a smell. Then, whenever that sentence is in the knowledge base, it is true in the real world.

Propositional logic in Artificial intelligence

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.
Example:

- It is Sunday.
- The Sun rises from West (False proposition)
- 3+3= 7(False proposition)
- 5 is a prime number.

Propositional Logic syntax:



$$\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence & \rightarrow & (\, Sentence \,) \mid [\, Sentence \,] \\
 & \mid & \neg\, Sentence \\
 & \mid & Sentence \land Sentence \\
 & \mid & Sentence \lor Sentence \\
 & \mid & Sentence \Rightarrow Sentence \\
 & \mid & Sentence \Leftrightarrow Sentence \\
 & & \\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}$$

Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.
Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.
Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. Negation: A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. Conjunction: A sentence which has ∧ connective such as, P ∧ Q is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as, P= Rohan is intelligent,Q= Rohan is hardworking. → P∧ Q.

3. Disjunction: A sentence which has ∨ connective, such as P ∨ Q. is called disjunction, where P and Q are the propositions. Example: "Ritika is a doctor or Engineer", Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as P ∨ Q.
4. Implication: A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as: If it is raining, then the street is wet. Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. Biconditional: A sentence such as P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive, P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called Truth table. Following are the truth table for all logical connectives:

For Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

Write the following English sentences in symbolic form

1. If it rains, then I will stay at home

    P: It rains

    Q: I will stay at home

    Representation: P->Q

2. He is poor but honest.
P: He is poor, Q: He is honest, Representation: P ∧ Q
3. Birds fly if and only if sky is clear.
P: Birds fly, Q: Sky is clear, Representation: P⇔ Q
4. I will go only if he stays.
I will go if he stays.
P: I will go, Q: He stays, Representation: P->Q
5. It is hot or else it is both cold and cloudy
P: It is hot, Q: It is cold R: it is cloudy, Representation: P v ($Q \Lambda$R)
6. Either today is Sunday or Monday
P: Today is Sunday, Q: Today is Monday, Representation: P v Q

Consider,

$P_{x,y}$ is true if there is a pit in $[x, y]$.
$W_{x,y}$ is true if there is a wumpus in $[x, y]$, dead or alive.
$B_{x,y}$ is true if the agent perceives a breeze in $[x, y]$.
$S_{x,y}$ is true if the agent perceives a stench in $[x, y]$.

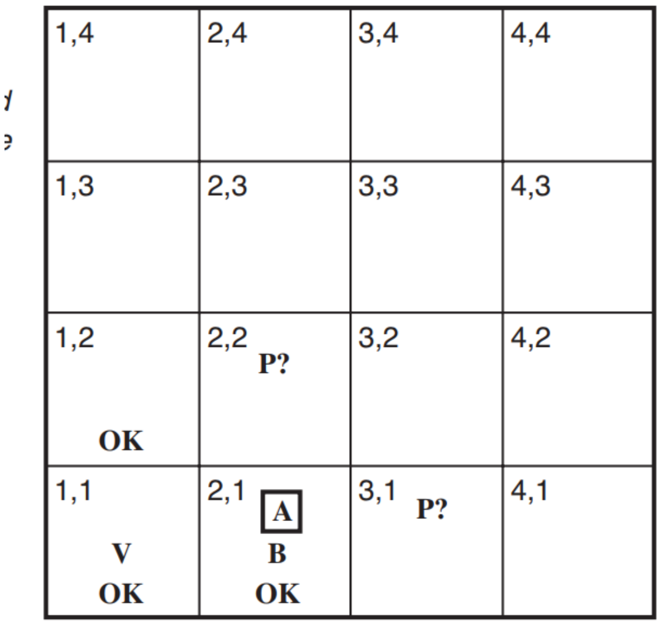
There is no pit in [1,1]

$$R_1 : \quad \neg P_{1,1}$$

A square is breezy if and only if there is a pit in a neighboring square.

$$R_2 : \quad B_{1,1} \quad \Leftrightarrow \quad (P_{1,2} \vee P_{2,1}) \; .$$
$$R_3 : \quad B_{2,1} \quad \Leftrightarrow \quad (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

For:



Write breeze for squares visited

$R_4: \quad \neg B_{1,1}$.
$R_5: \quad B_{2,1}$.

Standard Logical Equivalences

Logical equivalence is the condition of equality that exists between two statements or sentences in propositional logic. The relationship between the two statements translates verbally into "if and only if." In mathematics, logical equivalence is typically symbolized by a double arrow (⇔ or ⟷) or triple lines (≡).
This expression provides an example of logical equivalence between two simple statements:
$A \lor B \Leftrightarrow B \lor A$
The expression includes the statements $A \lor B$ and $B \lor A$, which are connected together by the IIF function. Each statement uses the OR Boolean function (∨) to indicate an inclusive disjunction between variables A and B. This means that the statement returns a true value if either variable is true or if both variables are true, but it returns a false value if both variables are false. The expression in its entirety is effectively stating that the statement "variable A or variable B" is logically equivalent to the statement "variable B or variable A."
Some of the standard logical equivalences:

$$\begin{aligned}
(\alpha \land \beta) &\equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) &\equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) &\equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) &\equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) &\equiv (\neg\alpha \lor \neg\beta) \quad \text{De Morgan} \\
\neg(\alpha \lor \beta) &\equiv (\neg\alpha \land \neg\beta) \quad \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) &\equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) &\equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{aligned}$$

Inference rules:

Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal. In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

- Implication: It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.
- Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.
- Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.
- Inverse: The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

Proof:

P	**Q**	**P → Q**	**Q→ P**	**¬ Q → ¬ P**	**¬ P → ¬ Q.**
T	**T**	T	T	T	T
T	**F**	F	T	F	T
F	**T**	T	F	T	F
F	**F**	T	T	T	T

Types of Inference rules:

Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\dfrac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:
Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I am sleepy" ==> P
Conclusion: "I go to bed." ==> Q.
Hence, we can say that, if P→ Q is true and P is true then Q will be true.

AND Elimination:

This rule states that if P∧ Q is true, then Q or P will also be true. It can be represented as:

Notation of Simplification rule: $\dfrac{P \land Q}{Q}$ Or $\dfrac{P \land Q}{P}$

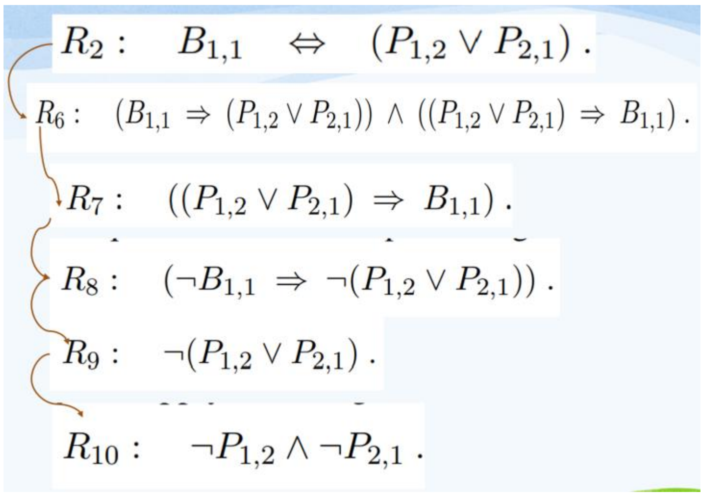
Prove logically that there is no pit in [1,2]
Knowledgebase:

$R_2: \quad B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})\,.$

$(P_{1,1} \lor P_{2,2} \lor P_{3,1})$

$R_4: \quad \neg B_{1,1}\,.$

$R_5: \quad B_{2,1}\,.$



$R_2: \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.

$R_6: \quad (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.

$R_7: \quad ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.

$R_8: \quad (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$.

$R_9: \quad \neg(P_{1,2} \vee P_{2,1})$.

$R_{10}: \quad \neg P_{1,2} \wedge \neg P_{2,1}$.

Proof by resolution

The idea of resolution is simple: if we know that

- p is true or q is true
- and we also know that p is false or r is true
- then it must be the case that q is true or r is true.

This line of reasoning is formalized in the
Resolution Tautology:
(p V q) Λ (¬ p V r) -> q V r
Eg: Given the following hypotheses:

1. If it rains, Joe brings his umbrella (r -> u)
2. If Joe has an umbrella, he doesn't get wet (u -> NOT w)
3. If it doesn't rain, Joe doesn't get wet (NOT r -> NOT w)

prove that Joes doesn't get wet (NOT w)
We first put each hypothesis in CNF:

1. r -> u == (NOT r OR u)
2. u -> NOT w == (NOT u OR NOT w)
3. NOT r -> NOT w == (r OR NOT w)

We then use resolution on the hypotheses to derive the conclusion (NOT w):

1. NOT r OR u	Premise
2. NOT u OR NOT w	Premise
3. r OR NOT w	Premise
4. NOT r OR NOT w	L1, L2, resolution
5. NOT w OR NOT w	L3, L4, resolution
6. NOT w	L5, idempotence

Prove logically that there is a pit in [3,1]

$$B_{2,1} \leftrightarrow P_{1,1} \vee P_{3,1} \vee P_{2,2}$$

$$(B_{2,1} \rightarrow P_{1,1} \vee P_{3,1} \vee P_{2,2}) \wedge (P_{1,1} \vee P_{3,1} \vee P_{2,2} \rightarrow B_{2,1})$$

$$B_{2,1}$$

$$(P_{1.1} \vee P_{3.1} \vee P_{2.2})$$

$$\neg P_{1,1}$$

$$P_{3,1} \vee P_{2,2}$$

$$\neg P_{2,2}$$

$$Hence, P_{3,1}$$

Conjunctive Normal Form

Resolution works best when the formula is of the special form: it is an ∧ of ∨s of (possibly negated, ¬) variables (called literals).
Eg:
(y∨¬z) ∧ (¬y) ∧ (y ∨ z)   CNF
(x ∨ ¬y ∧ z) Not CNF

To convert a formula into a CNF.

- Convert double implication to single
- Open up the implications to get ORs.
- Get rid of double negations.
- Use Demorgans Law
- Use distributivity
- Eg: F ∨ (G ∧ H) can be written as
    - (F ∨ G) ∧ (F ∨ H)

Eg: Convert to CNF: A→ (B ∧ C)

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

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

Horn Clause in AI

The term "horn clause" refers to a disjunction of literals in which, at most, one literal is not negated. A horn clause is a clause that has exactly one literal that is not negated.The logician Alfred Horn first recognized the importance of Horn clauses in 1951. Horn clauses are a type of logical formula used in logic programming, formal specification, universal algebra, and model theory due to their helpful qualities in these areas and others.

Types of Horn Clauses :

- Definite clause / Strict Horn clause – It has precisely one positive literal.
- Unit clause - Definite clause containing no negative literals.

- Goal clause – Horn clause lacking a literal positive.

Horn clauses perform a fundamental role in both constructive and computational logic.

Syntax:

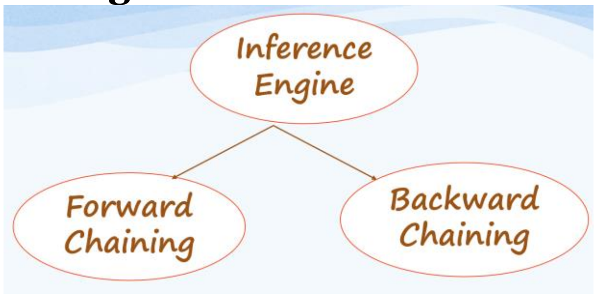
$$\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots \\
HornClauseForm &\rightarrow DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{aligned}$$

Inference Engines:

An inference engine is a component of an AI system that is responsible for drawing conclusions based on evidence and information that is provided to it. In other words, it is responsible for making deductions and inferences based on what it knows.
The inference engine is often compared to the human brain, as it is responsible for making the same kinds of deductions and inferences that we do. However, the inference engine is not limited by the same constraints as the human brain. It can process information much faster and is not subject to the same biases and errors that we are.

The inference engine is a critical component of AI systems because it is responsible for making the decisions that the system needs to make in order to function. Without an inference engine, an AI system would be little more than a collection of data.

Types of Inference engines:

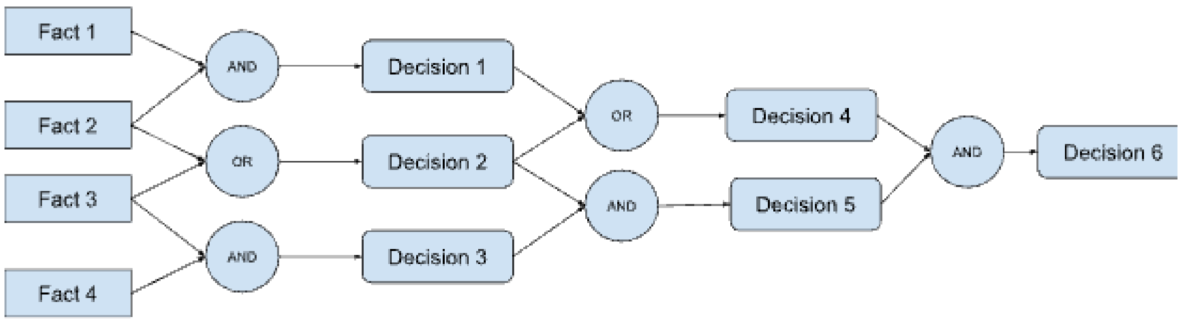


- **Forward chaining:** Forward chaining is a form of reasoning for an AI expert system that starts with simple facts and applies inference rules to extract more data until the goal is reached.

- **Backward chaining:** Backward chaining is another strategy used to shape an AI expert system that starts with the end goal and works backward through the AI's rules to find facts that support the goal.

Forward Chaining:

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. The forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied and adds their conclusion to the known facts. This process repeats until the problem is solved. In this type of chaining, the inference engine starts by evaluating existing facts, derivations, and conditions before deducing new information. An endpoint, or goal, is achieved through the manipulation of knowledge that exists in the knowledge base.



Forward Chaining Properties

- Forward chaining follows a down-up strategy, going from bottom to top.
- It uses known facts to start from the initial state (facts) and works toward the goal state, or conclusion.
- The forward chaining method is also known as data-driven because we achieve our objective by employing available data.
- The forward chaining method is widely used in expert systems such as CLIPS, business rule systems and manufacturing rule systems.
- It uses a breadth-first search as it has to go through all the facts first.
- It can be used to draw multiple conclusions.

Eg:

Knowledgebase:

1. John's credit score is 780.
2. A person with a credit score greater than 700 has never defaulted on their loan.
3. John has an annual income of $100,000.
4. A person with a credit score greater than 750 is a low-risk borrower.
5. A person with a credit score between 600 to 750 is a medium-risk borrower.
6. A person with a credit score less than 600 is a high-risk borrower.
7. A low-risk borrower can be given a loan amount up to 4X of his annual income at a 10 percent interest rate.
8. A medium-risk borrower can be given a loan amount of up to 3X of his annual income at a 12 percent interest rate.

9. A high-risk borrower can be given a loan amount of up to 1X of his annual income at a 16 percent interest rate.
10.

Question:

1. What max loan amount can be sanctioned for John?
2. What will the interest rate be?

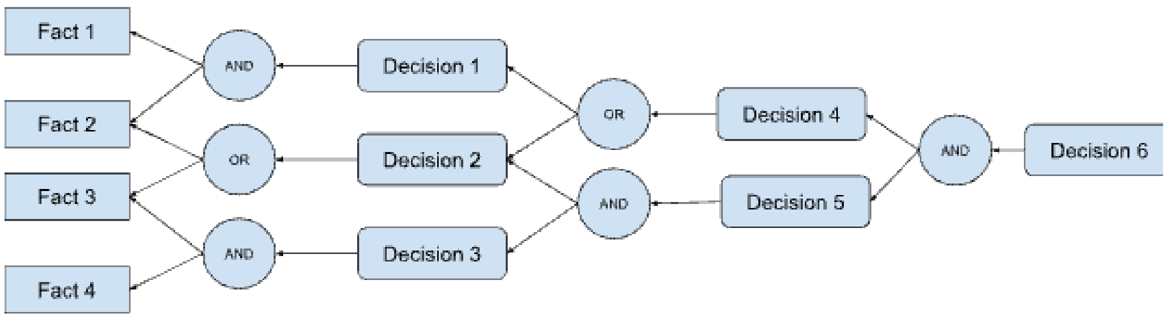
Solution:
To deduce the conclusion, we apply forward chaining on the knowledge base. We start from the facts which are given in the knowledge base and go through each one of them to deduce intermediate conclusions until we are able to reach the final conclusion or have sufficient evidence to negate the same.
John' CS = 780 AND CS > 750 are Low Risk Borrower → John is a Low Risk Borrower

Loan Amount for Low Risk Borrower is 4X annual income AND John's annual income is $100k

→ Max loan amount that can be sanctioned is $400k at a 10% interest rate.

Backward Chaining

Backward chaining is also known as a backward deduction or backward reasoning method when using an inference engine. In this, the inference engine knows the final decision or goal. The system starts from the goal and works backward to determine what facts must be asserted so that the goal can be achieved.
For example, it starts directly with the conclusion (hypothesis) and validates it by backtracking through a sequence of facts. Backward chaining can be used in debugging, diagnostics and prescription applications.



Properties of Backward Chaining

- Backward chaining uses an up-down strategy going from top to bottom.
- The modus ponens inference rule is used as the basis for the backward chaining process. This rule states that if both the conditional statement (p->q) and the antecedent (p) are true, then we can infer the subsequent (q).

- In backward chaining, the goal is broken into sub-goals to prove the facts are true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- The backward chaining algorithm is used in game theory, automated theorem-proving tools, inference engines, proof assistants and various AI applications.
- The backward-chaining method mostly used a depth-first search strategy for proof.

Eg:
Knowledgebase:

- John is taller than Kim
- John is a boy
- Kim is a girl
- John and Kim study in the same class
- Everyone else other than John in the class is shorter than Kim

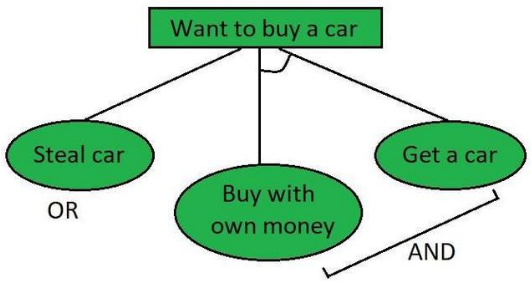
Question:

- Is John the tallest boy in class?

Now, to apply backward chaining, we start from the goal and assume that John is the tallest boy in class. From there, we go backward through the knowledge base comparing that assumption to each known fact to determine whether it is true that John is the tallest boy in class or not.
Height (John) > Height (anyone in the class)
AND
John and Kim both are in the same class
AND
Height (Kim) > Height (anyone in the class except John)
AND
John is boy
SO
Height (John) > Hight(Kim)
Which aligns with the knowledge base fact. Hence the goal is proved true.

AO* algorithm – Artificial intelligence

Best-first search is what the AO* algorithm does. The AO* method divides any given difficult problem into a smaller group of problems that are then resolved using the AND-OR graph concept. AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems. The AND side of the graph represents a set of tasks that must be completed to achieve the main goal, while the OR side of the graph represents different methods for accomplishing the same main goal.



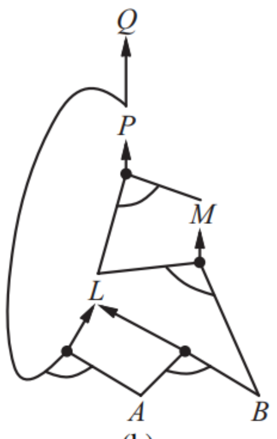
In this figure, the buying of a car may be broken down into smaller problems or tasks that can be accomplished to achieve the main goal in the above figure, which is an example of a simple AND-OR graph. The other task is to either steal a car that will help us accomplish the main goal or use your own money to purchase a car that will accomplish the main goal. The AND symbol is used to indicate the AND part of the graphs, which refers to the need that all subproblems containing the AND to be resolved before the preceding node or issue may be finished.

Representation of Horn clauses using AND-OR graph for forward chaining

Consider the example inference rules and facts:

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

The corresponding AND-OR graph is:



(b)

MODULE-4

Limitations of Propositional logic:

- Propositional logic is a puny mechanism to express knowledge and inferencing
- We cannot represent relations like ALL, some, or none with propositional logic. Example:
    a. All the girls are intelligent.
    b. Some apples are sweet.
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Alternate Solutions:

Programming languauges:

The syntax and semantics of programming languages can be used to express the reasoning. In programming language procedures/methods can be the logic and data structures containing data will be the facts. However programming languages lack following:

- One cannot derive facts from other facts
- Inferencing not automatic and needs to programmed for each case
- They lack expressiveness required to handle partial information
- Composionality is missing.
-

Natural Language libraries:

Natural language processing (NLP) is a branch of artificial intelligence (AI) that enables computers to comprehend, generate, and manipulate human language. Natural language processing has the ability to interrogate the data with natural language text or voice. This is also called "language in." Most consumers have probably interacted with NLP without realizing it. For instance, NLP is the core technology behind virtual assistants, such as the Oracle Digital Assistant (ODA), Siri, Cortana, or Alexa. Ambiguity, generally used in natural language processing, can be referred as the ability of being understood in more than one way. In simple terms, we can say that ambiguity is the capability of being understood in more than one way. Natural language is very ambiguous. NLP has the following types of ambiguities –

Lexical Ambiguity

The ambiguity of a single word is called lexical ambiguity. For example, treating the word silver as a noun, an adjective, or a verb.

Syntactic Ambiguity

This kind of ambiguity occurs when a sentence is parsed in different ways. For example, the sentence "The man saw the girl with the telescope". It is ambiguous whether the man saw the girl carrying a telescope or he saw her through his telescope.

Semantic Ambiguity

This kind of ambiguity occurs when the meaning of the words themselves can be misinterpreted. In other words, semantic ambiguity happens when a sentence contains an ambiguous word or phrase. For example, the sentence "The car hit the pole while it was moving" is having semantic ambiguity because the interpretations can be "The car, while moving, hit the pole" and "The car hit the pole while the pole was moving".

Anaphoric Ambiguity

This kind of ambiguity arises due to the use of anaphora entities in discourse. For example, the horse ran up the hill. It was very steep. It soon got tired. Here, the anaphoric reference of "it" in two situations cause ambiguity.

Pragmatic ambiguity

Such kind of ambiguity refers to the situation where the context of a phrase gives it multiple interpretations. In simple words, we can say that pragmatic ambiguity arises when the statement is not specific. For example, the sentence "I like you too" can have multiple interpretations like I like you (just like you like me), I like you (just like someone else dose).

Combining the best of formal and natural languages

Formal languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemists use a formal language to represent the chemical structure of molecules.

Elements of a natural language: Nouns,, and noun phrases that refer to objects (squares, pits, wumpuses) and verbs and verb phrases that refer to relations among objects (is breezy, is adjacent to, shoots). Some of these relations are functions—relations in which there is only one "value" for a given "input." It is easy to start listing examples of objects, relations, and functions:

• Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries

• Relations: these can be unary relations or properties such as red, round, bogus, prime,

• Functions: father of, best friend, third inning of, one more than, beginning of

Eg: Identify the objects, relations, functions and properties if any in the following examples:

"one plus two equals three"

Objects: one, two, three; Relations: equals; Function: plus

"Squares neighboring wumpus are smelly"

Objects: squares,Wumpus; Relations:neighbouring; Property:smelly;

"Evil King John ruled England in 1200"

Objects: King John, England, 1200; Relation: ruled; Property:Evil

Formal languages and their ontological and epistemological commitments.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

- Propositional logic is the logic that deals with a collection of declarative statements which have a truth value, true or false. Propositions are combined with Logical Operators or Logical Connectives like Negation(¬), Disjunction(∨), Conjunction(∧), Exclusive OR(⊕), Implication(⇒), Bi-Conditional or Double Implication(⇔). It cannot deal with sets of entities.

- First order logic is an expression consisting of variables with a specified domain. It consists of objects, relations and functions between the objects. It helps analyze the scope of the subject over the predicate. There are three quantifiers : Universal Quantifier (∀) depicts for all, Existential Quantifier (∃) depicting there exists some and Uniqueness Quantifier (∃!) depicting exactly one. It can deal with set of entities with the help of quantifiers.
- Temporal logic is a subfield of mathematical logic that deals with reasoning about time and the temporal relationships between events. In artificial intelligence, temporal logic is used as a formal language to describe and reason about the temporal behavior of systems and processes. Temporal logic extends classical propositional and first-order logic with constructs for specifying temporal relationships, such as "before," "after," "during," and "until." This allows for the expression of temporal constraints and the modeling of temporal aspects of a system, such as its evolution over time and the relationships between events.
- Probability is defined as the chance of happening or occurrences of an event. Generally, the possibility of analyzing the occurrence of any event with respect to previous data is called probability. For example, if a fair coin is tossed, what is the chance that it lands on the head? These types of questions are answered under probability. Probability theory uses the concept of random variables and probability distribution to find the outcome of any situation. Probability theory is an advanced branch of mathematics that deals with the odds and statistics of happening an event.
- The term fuzzy refers to things that are not clear or are vague. In the real world many times we encounter a situation when we can't determine whether the state is true or false, their fuzzy logic provides very valuable flexibility for reasoning. In this way, we can consider the inaccuracies and uncertainties of any situation. Fuzzy Logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1, instead of just the traditional values of true or false. It is used to deal with imprecise or uncertain information and is a mathematical method for representing vagueness and uncertainty in decision-making. Fuzzy Logic is based on the idea that in many cases, the concept of true or false is too restrictive, and that there are many shades of gray in between. It allows for partial truths, where a statement can be partially true or false, rather than fully true or false.
- Ontology: the study of what there is in the world that we should know about, and Epistemology: the study of how we should get to know the things in the world. Obviously, this was long before AI was a thing, and they were merely concerned with the structure of knowledge and its acquisition by humans. when it comes to creating AI we should take an epistemological approach: how does the only working truly intelligent system (the human brain) models the world, innately or by learning. This is in contrast with the ontological approach: focusing on organizing what we know in data ontologies and then trying to instill those in computers.

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.
Basic Elements of First-order logic:
Syntax of First Order Logic
The syntax of First Order Logic is written using Backnaus Normal Form.

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{Predicate} \mid \mathit{Predicate}(\mathit{Term},\ldots) \mid \mathit{Term} = \mathit{Term} \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Rightarrow\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Leftrightarrow\ \mathit{Sentence} \\
 & \mid & \mathit{Quantifier}\ \mathit{Variable},\ldots\ \mathit{Sentence} \\
 & & \\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term},\ldots) \\
 & \mid & \mathit{Constant} \\
 & \mid & \mathit{Variable} \\
 & & \\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & A \mid X_1 \mid \mathit{John} \mid \cdots \\
\mathit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\mathit{Predicate} & \rightarrow & \mathit{True} \mid \mathit{False} \mid \mathit{After} \mid \mathit{Loves} \mid \mathit{Raining} \mid \cdots \\
\mathit{Function} & \rightarrow & \mathit{Mother} \mid \mathit{LeftLeg} \mid \cdots \\
\textsc{Operator Precedence} & : & \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as
  Predicate (term1, term2, ..., term n).

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

Complex Sentences:

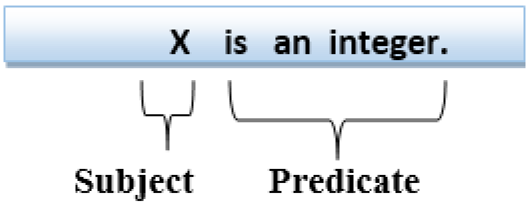
- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- Subject: Subject is the main part of the statement.

- Predicate: A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
  a. Universal Quantifier, (for all, everyone, everything)
  b. Existential quantifier, (for some, at least one).

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

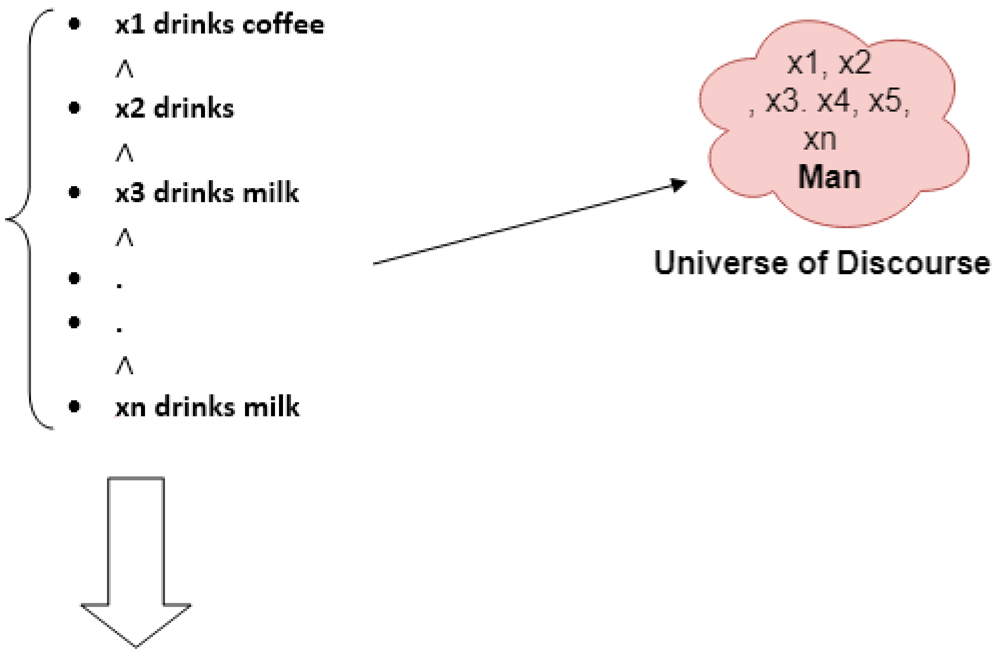
If x is a variable, then ∀x is read as:

- For all x
- For each x
- For every x.

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



So in shorthand notation, we can write it as :

∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

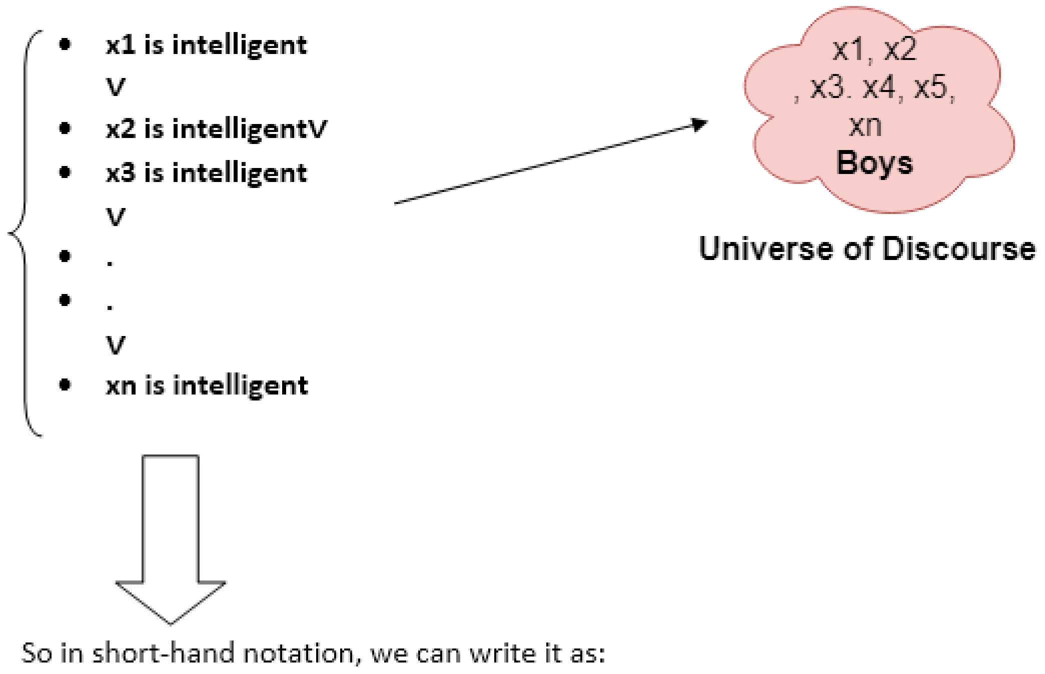
Note: In Existential quantifier we always use AND or Conjunction symbol (∧).

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- There exists a 'x.'
- For some 'x.'
- For at least one 'x.'

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Some Examples of FOL using quantifier:

1. All birds fly.

In this question the predicate is "fly(bird)."

And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.

In this question, the predicate is "respect(x, y)," where x=man, and y= parent.

Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.

In this question, the predicate is "play(x, y)," where x= boys, and y= game. Since there are some boys so we will use ∃, and it will be represented as:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "like(x, y)," where x= student, and y= subject.
Since there are not all students, so we will use ∀ with negation, so following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this question, the predicate is "failed(x, y)," where x= student, and y= subject.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Kinship domain

Objects- Richard, John, 2 left legs and crown

Binary relations:

brotherhood: {<Richard>,<John>}

on head: {<crown>,<John>}

Unary relations (properties):

person: {<Richard>}

person: {<John>}

king: {<John>}

crown: {<John>}

Functions: Relations with mappings

<Richard>->Richards left-leg

<John>->Johns left-leg

Symbols and Interpretations

The symbols of the language 0,1, add, prime and so on—have very suggestive names. When we interpret sentences of this language over the domain N, for example, it is clear for which elements of the domain prime "should" be true, and for which it "should" be false. But let us consider a first order language that has only two unary predicate symbols, fancy and tall, then it depends on the domain to define the value. It is undefined for the domain of natural numbers.
Symbols stand for objects, relations and functions. There are 3 types of symbols:

- ❑ Constant symbols representing objects. Eg: John, Richard
- ❑ Predicate symbols representing relations. Eg: Brother, OnHead
- ❑ Function symbols representing functions. Eg: LeftLeg

Each model includes an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols. Logician calls it as Intended Interpretation

Nested Quantifiers

Nested quantifiers are quantifiers that occur within the scope of other quantifiers. Example: ∀x∃yP(x, y). Quantifier order matters! ∀x∃yP(x, y) ≠ ∃y∀xP(x, y)
Let L(x, y) be the statement "x loves y," where the domain for both x and y consists of all people in the world. Use quantifiers to express each of these statements.
a) Everybody loves Jerry. ∀x L(x, Jerry)
b) Everybody loves somebody. ∀x∃yL(x, y)
c) There is somebody whom everybody loves. ∃y∀xL(x, y)
d) Nobody loves everybody. ∀x∃y¬L(x, y) or ¬∃x∀yL(x, y)
e) Everyone loves himself or herself ∀xL(x, x)

Confusion created when same variable name used in nested quantifiers

$$\forall \boldsymbol{x}\ (\boldsymbol{crown}(\boldsymbol{x}) \vee (\exists\ \boldsymbol{x}\ \boldsymbol{Brother}(\boldsymbol{Richard}, \boldsymbol{x})))$$

Solution: use different variable names

$$\forall \boldsymbol{x}\ (\boldsymbol{crown}(\boldsymbol{x}) \vee (\exists\ \boldsymbol{z}\ \boldsymbol{Brother}(\boldsymbol{Richard}, \boldsymbol{z})))$$

Connections between ∀ and ∃

The two quantifiers are actually intimately connected with each other, through negation. Asserting that everyone dislikes parsnips is the same as asserting there does not exist someone who likes them, and vice versa:
∀ x ¬Likes(x, Parsnips) is equivalent to ¬∃ x Likes(x, Parsnips) . We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream: ∀ x Likes(x, IceCream) is equivalent to ¬∃ x ¬Likes(x, IceCream) .
Equality
We can use the equality symbol to signify that two terms refer to the same object. For example, Father (John)=Henry says that the object referred to by Father (John) and the object referred to by Henry are the same. The equality symbol can be used to state facts about a given function, as we just did for the Father symbol. It can also be used with negation to insist that two terms are not the same object. To say that Richard has at least two brothers, we would write ∃ x, y Brother (x,Richard) ∧ Brother (y,Richard) ∧¬(x=y) .

USING FIRST ORDER LOGIC

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called assertions. For example, we can assert that John is a king, Richard is a person, and all kings are persons:

TELL(KB, King(John))
TELL(KB, Person(Richard))

TELL(KB, $\forall$ x King(x) $\Rightarrow$ Person(x)) We can ask questions of the knowledge base using ASK.

For example, ASK(KB, King(John)) returns true. Questions asked with ASK are called queries or goals If we want to know what value of x makes the sentence true, we will need a different function, ASKVARS, which we call with ASKVARS(KB, Person(x)) and which yields a stream of answers. In this case there will be two answers: {x/John} and {x/Richard}. Such an answer is called a substitution or binding list.

The kinship domain

The first example we consider is the domain of family relationships, or kinship. This domain includes facts such as "Elizabeth is the mother of Charles" and "Charles is the father of William" and rules such as "One's grandmother is the mother of one's parent." Clearly, the objects in our domain are people. We have two unary predicates, Male and Female. Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: Parent, Sibling, Brother , Sister , Child , Daughter, Son, Spouse, Wife, Husband, Grandparent , Grandchild , Cousin, Aunt, and Uncle. We use functions for Mother and Father , because every person has exactly one of each of these.

For example, one's mother is one's female parent:

$\forall$ m, c Mother (c)=m $\Leftrightarrow$ Female(m) $\wedge$ Parent(m, c) .

One's husband is one's male spouse:

$\forall$ w, h Husband(h,w) $\Leftrightarrow$ Male(h) $\wedge$ Spouse(h,w) .

Male and female are disjoint categories:

$\forall$ x Male(x) $\Leftrightarrow$ $\neg$Female(x) . Parent and child are inverse relations: $\forall$ p, c Parent(p, c) $\Leftrightarrow$ Child (c, p) .

A grandparent is a parent of one's parent:

$\forall$ g, c Grandparent (g, c) $\Leftrightarrow$ $\exists$p Parent(g, p) $\wedge$ Parent(p, c) . A sibling is another child of one's parents: $\forall$ x, y

Sibling(x, y) $\Leftrightarrow$ x _= y $\wedge$ $\exists$p Parent(p, x) $\wedge$ Parent(p, y) . Each of these sentences can be viewed as an axiom of the kinship domain. Axioms are commonly associated with purely mathematical domains. Our kinship axioms are also definitions; they have the form $\forall$ x, y P(x, y) $\Leftrightarrow$ ..... The axioms define the Mother function and the Husband, Male, Parent, Grandparent, and Sibling predicates in terms of other predicates. For example, consider the assertion that siblinghood is symmetric: $\forall$ x, y Sibling(x, y) $\Leftrightarrow$ Sibling(y, x) .

Natural numbers in first-order logic

The natural numbers can be described in first-order logic. The language of natural numbers has

- a single constant 0, defined by predicate NatNum(0)
- a function Successor, S(n) which defines next number after n in the series of natural number
- The successor is expressed as a quantifier:
- $\forall n,\ NatNum(n) \Rightarrow NatNum(S(n))$

$$\forall n,\ 0 \neq S(n)$$

- Two natural numbers cannot have same successor
- $\forall m, n \quad m \neq n \Rightarrow S(m) \neq S(n)$
- + is a function defined on two natural numbers and equality is defined between 2 natural numbers using FOL:

- $\forall m,\ n\ \ \mathrm{NatNum}(m) \wedge \mathrm{NatNum}(n) \Rightarrow +(S(m), n) = S(+(m, n))$

  - 0 is an arithmetic identity as
  - $\forall m,\ \mathrm{NatNum}(m) \Rightarrow +(0, m) = m$

A set is a collection of objects; any one of the objects in a set is called a member or an element of the set.
The basic statement in set theory is element inclusion: an element a is included in some set S. Formally written as:
$a \in S$
If an element is not included, we write:
$a \notin S$
Statements are either true or false, depending on the context. For example, given the above sets, the first statement is true, whereas the second is false. If a statement S is true in a given context C, we say the statement is valid in C. Formally, we write this as:
$C \models S$
If the statement is not valid in that context, we write:
$C \not\models S$
The operators to compose new sets out of existing ones are:

1. A special set is the empty set, which contains no elements at all: $\emptyset$
2. Union: create a set S containing all elements from A, from B, or from both. Formally: $S = A \cup B$
3. Intersection: create a set S containing all elements that are both in A and in B. Formally: $S = A \cap B$
4. Exclusion: create a set S from the elements of A that are not in B. Formally: $S = A \setminus B$

These sets can be interpreted as quantified statements:

$$\begin{array}{lcl} A \cup B & \iff & \forall a \in (A \cup B) : ((a \in A) \vee (a \in B)) \\ A \cap B & \iff & \forall a \in (A \cap B) : ((a \in B) \wedge (a \in B)) \\ A \setminus B & \iff & \forall a \in (A \setminus B) : ((a \in A) \wedge (a \notin B)) \end{array}$$

Subsets:
∀ s1, s2 s1 ⊆ s2 ⇔ (∀ x x ∈ s1 ⇒ x ∈ s2) .

Equality of two sets:
∀ s1, s2 (s1 = s2) ⇔ (s1 ⊆ s2 ∧ s2 ⊆ s1) .

List vs Sets
Lists are similar to sets. The differences are that lists are ordered and the same element can appear more than once in a list. We can use the vocabulary of Lisp for lists:
Nil is the constant list with no elements;
Cons, Append, First, and Rest are functions; and
Find is the predicate that does for lists what Member does for sets.
List? is a predicate that is true only of lists. elements,

First Order Logic with Wumpus World

- The wumpus agent receives a percept vector with five elements. The corresponding firstorder sentence stored in the knowledge base must include both the percept and the time at which it occurred; otherwise, the agent will get confused about when it saw what.
- We use integers for time steps. A typical percept sentence would be Percept ([Stench, Breeze, Glitter , None, None], 5) . Here, Percept is a binary predicate, and Stench and so on are constants placed in a list.
- The actions in the wumpus world can be represented by logical terms: Turn(Right), Turn(Left), Forward , Shoot , Grab, Climb .
- To determine which is best, the agent program executes the query ASKVARS(∃ a BestAction(a, 5)) , which returns a binding list such as {a/Grab}.
- The agent program can then return Grab as the action to take.
- The raw percept data implies certain facts about the current state. For example: ∀t,s,g,m,c Percept ([s,Breeze,g,m,c],t) ⇒ Breeze(t) , ∀t,s,b,m,c Percept ([s,b,Glitter,m,c],t) ⇒ Glitter (t) These rules exhibit a trivial form of the reasoning process called perception.
- Simple "reflex" behavior can also be implemented by quantified implication sentences. For example, we have ∀ t Glitter (t) ⇒ BestAction(Grab, t) .
- Given the percept and rules from the preceding paragraphs, this would yield the desired conclusion BestAction(Grab, 5)—that is, Grab is the right thing to do.
- For example, if the agent is at a square and perceives a breeze, then that square is breezy: ∀ s, t At(Agent, s, t) ∧ Breeze(t) ⇒ Breezy(s) . It is useful to know that a square is breezy because we know that the pits cannot move about. Notice that Breezy has no time argument.
- Having discovered which places are breezy (or smelly) and, very important, not breezy (or not smelly), the agent can deduce where the pits are (and where the wumpus is). first-order logic just needs one axiom: ∀ s Breezy(s) ⇔ ∃r Adjacent (r, s) ∧ Pit(r) .

Inference in First-Order Logic

- Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.
- Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write F[a/x], so it refers to substitute a constant "a" in place of variable "x".
- Equality-First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use equality symbols which specify that the two terms refer to the same object.
- Example: Brother (John) = Smith.
- As in the above example, the object referred by the Brother (John) is similar to the object referred by Smith. The equality symbol can also be used with negation to represent that two terms are not the same objects.
- Example: ¬(x=y) which is equivalent to x ≠y.

- FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- Universal Instantiation
- Existential Instantiation

Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, we can infer any sentence obtained by substituting a ground term for the variable.
- The UI rule state that we can infer any sentence by substituting a ground term v with g in the universe of discourse.
- $$\frac{\forall v \;\; \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$
  Example:1.
- IF "Every person like ice-cream"=> we can infer "John likes ice-cream" => P(c)
  Example: 2.
- "All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:
- ∀x king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

- King(John) ∧ Greedy (John) → Evil (John),
- King(Richard) ∧ Greedy (Richard) → Evil (Richard),
- King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),

Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- Represented as:

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

Example:
From the given sentence: ∃x Crown(x) ∧ OnHead(x, John),

So we can infer: Crown(K) ∧ OnHead(K, John), as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is called Skolem constant.
- The Existential instantiation is a special case of Skolemization process.

Generalized Modus Ponens Rule:

For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.
Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."
According to Modus Ponens, for atomic sentences pi, pi', q. Where there is a substitution θ such that SUBST (θ, pi',) = SUBST(θ, pi), it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \land p2 \land \ldots \land pn \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Example:
We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

1.	p1' is king(John)	p1 is king(x)
2.	p2' is Greedy(y)	p2 is Greedy(x)
3.	θ is {x/John, y/John}	q is evil(x)
4.	SUBST(θ,q).	

Unification

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\Psi_1\sigma = \Psi_2\sigma$, then it can be expressed as UNIFY(Ψ_1, Ψ_2).
- Example: Find the MGU for Unify{King(x), King(John)}

Let Ψ_1 = King(x), Ψ_2 = King(John),
Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

Conditions for Unification:
Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Unification Algorithm:

```
Algorithm: Unify($\Psi_1$, $\Psi_2$)
  Step. 1: If $\Psi_1$ or $\Psi_2$ is a variable or constant, then:
          a) If $\Psi_1$ or $\Psi_2$ are identical, then return NIL.
          b) Else if $\Psi_1$is a variable,
                  a. then if $\Psi_1$ occurs in $\Psi_2$, then return FAILURE
                  b. Else return { ($\Psi_2$/ $\Psi_1$)}.
```

```
        c) Else if $\Psi_2$ is a variable,
                a. If $\Psi_2$ occurs in $\Psi_1$ then return FAILURE,
                b. Else return {( $\Psi_1$/ $\Psi_2$)}.
        d) Else return FAILURE.
Step.2: If the initial Predicate symbol in $\Psi_1$ and $\Psi_2$ are not same, then return
FAILURE.
Step. 3: IF $\Psi_1$ and $\Psi_2$ have a different number of arguments, then return FAILURE.
Step. 4: Set Substitution set(SUBST) to NIL.
Step. 5: For i=1 to the number of elements in $\Psi_1$.
        a) Call Unify function with the ith element of $\Psi_1$ and ith element of $\Psi_2$, and
put the result into S.
        b) If S = failure then returns Failure
        c) If S $\neq$ NIL then do,
                a. Apply S to the remainder of both L1 and L2.
                b. SUBST= APPEND(S, SUBST).
Step.6: Return SUBST.
```

For each pair of the following atomic sentences find the most general unifier (If exist).

1. Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)

SUBST θ= {f(a) / X}

S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))

SUBST θ= {f(a) / g(y)}, Unification failed.

Unification is not possible for these expressions.

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))

S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}

SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}

SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}

SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} Unified Successfully.

And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

3. Find the MGU of {p (X, X), and p (Z, f(Z))}

Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))

S_0 => {p (X, X), p (Z, f(Z))}

SUBST θ= {X/Z}

S1 => {p (Z, Z), p (Z, f(Z))}

SUBST θ= {f(Z) / Z}, Unification Failed.

4. UNIFY(knows(Richard, x), knows(Richard, John))

Here, Ψ_1 = knows(Richard, x), and Ψ_2 = knows(Richard, John)

S_0 => { knows(Richard, x); knows(Richard, John)}

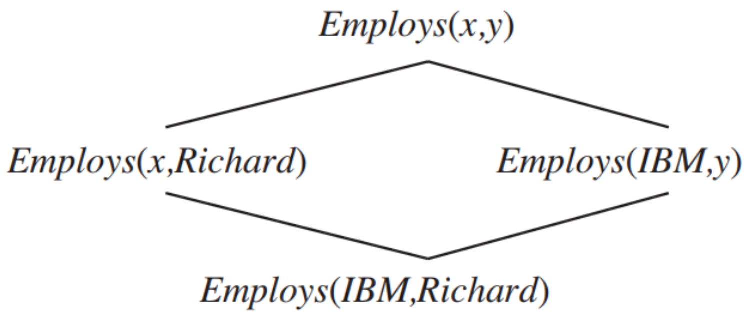
SUBST θ= {John/x}

S_1 => { knows(Richard, John); knows(Richard, John)}, Successfully Unified.

Unifier: {John/x}.

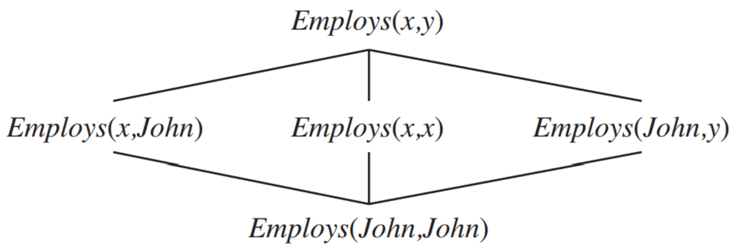
Subsumption Lattice

It is a structure created with most generic unifiers at the top and specific at the bottom. It helps to answer queries efficiently. Each bottom level is derived by applying single substitution on the top level. Eg:



Subsitute y/Richard to get left child, x/IBM to get right child. Later applying the same yields common child.

Eg:



Inference Engine using FOL

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

a. Forward chaining

b. Backward chaining

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the first-order definite clause.

Definite clause: A clause which is a disjunction of literals with exactly one positive literal is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with at most one positive literal is known as horn clause. Hence all the definite clauses are horn clauses.

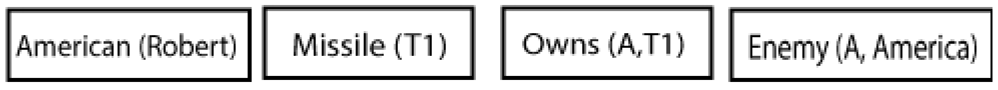
Example: $(\neg p \lor \neg q \lor k)$. It has only one positive literal k.

It is equivalent to $p \land q \rightarrow k$.

Forward Chaining
Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.
Example:
"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."
Prove that "Robert is criminal."
To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.
Facts Conversion into FOL:

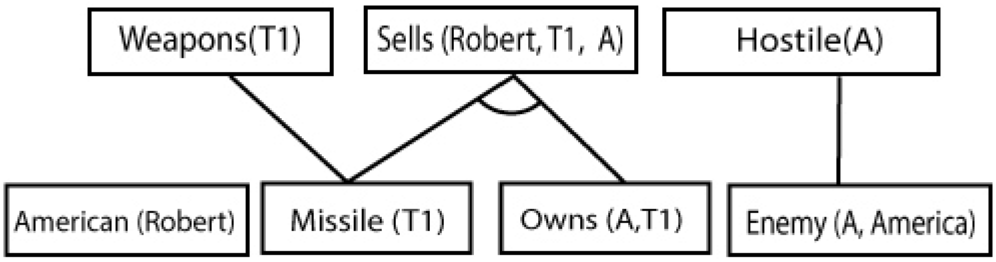
- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)
  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. ?p Owns(A, p) ∧ Missile(p). It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.
  Owns(A, T1) ......(2)
  Missile(T1) ....(3)
- All of the missiles were sold to country A by Robert.
  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
- Missiles are weapons.
  Missile(p) → Weapons (p) ....(5)
- Enemy of America is known as hostile.
  Enemy(p, America) →Hostile(p) ....(6)
- Country A is an enemy of America.
  Enemy (A, America) ....(7)
- Robert is American
  American(Robert). ....(8)

Forward chaining proof:
Step-1:
In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1). All these facts will be represented as below.

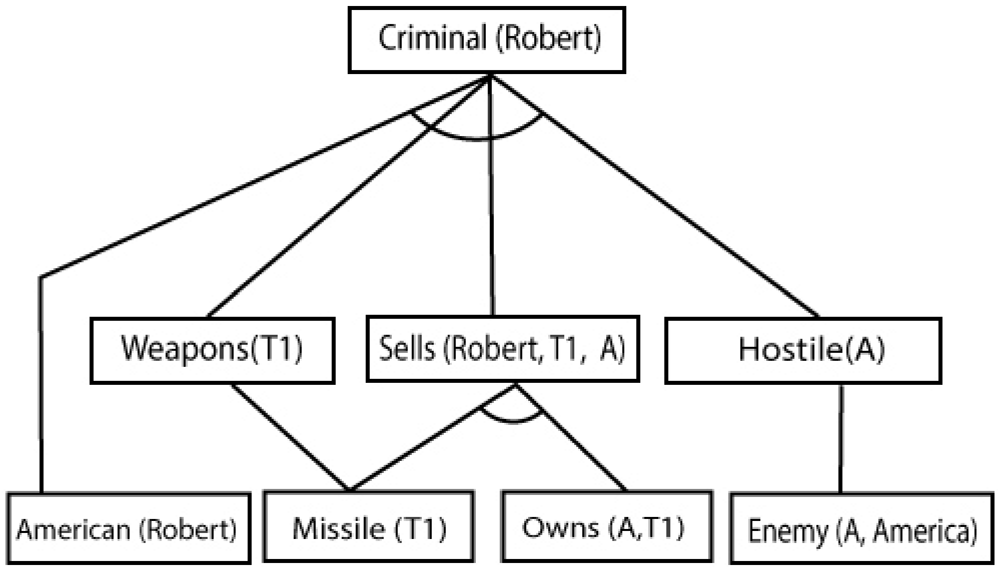


Step-2:
At the second step, we will see those facts which infer from available facts and with satisfied premises.
Rule-(1) does not satisfy premises, so it will not be added in the first iteration.
Rule-(2) and (3) are already added.
Rule-(4) satisfy with the substitution {p/T1}, so Sells (Robert, T1, A) is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:
At step-3, as we can check Rule-(1) is satisfied with the substitution {p/Robert, q/T1, r/A}, so we can add Criminal(Robert) which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

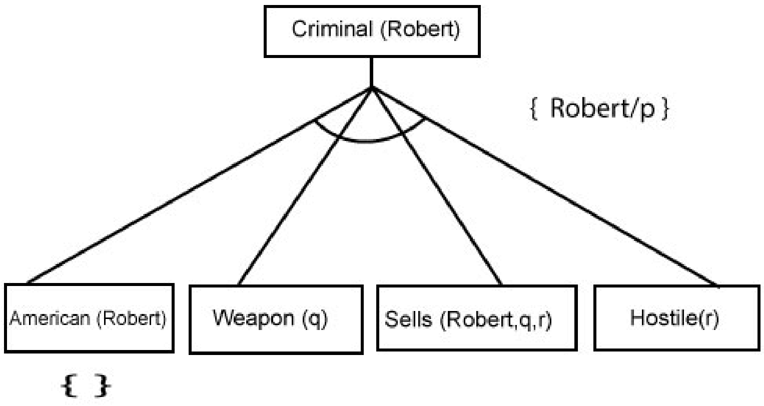
Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.
Example:
In backward-chaining, we will use the same above example, and will rewrite all the rules.

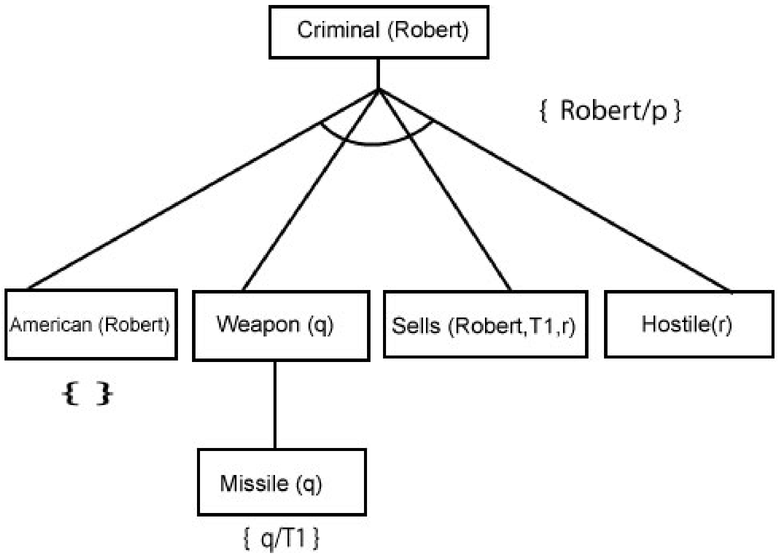
- American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
  Owns(A, T1) ....(2)
- Missile(T1)
- ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ......(4)
- Missile(p) → Weapons (p) ....(5)
- Enemy(p, America) →Hostile(p) ....(6)
- Enemy (A, America) ....(7)
- American(Robert). ....(8)

Backward-Chaining proof:
In Backward chaining, we will start with our goal predicate, which is Criminal(Robert), and then infer further rules.
Step-1:
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Step-2:
At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.
Here we can see American (Robert) is a fact, so it is proved here.

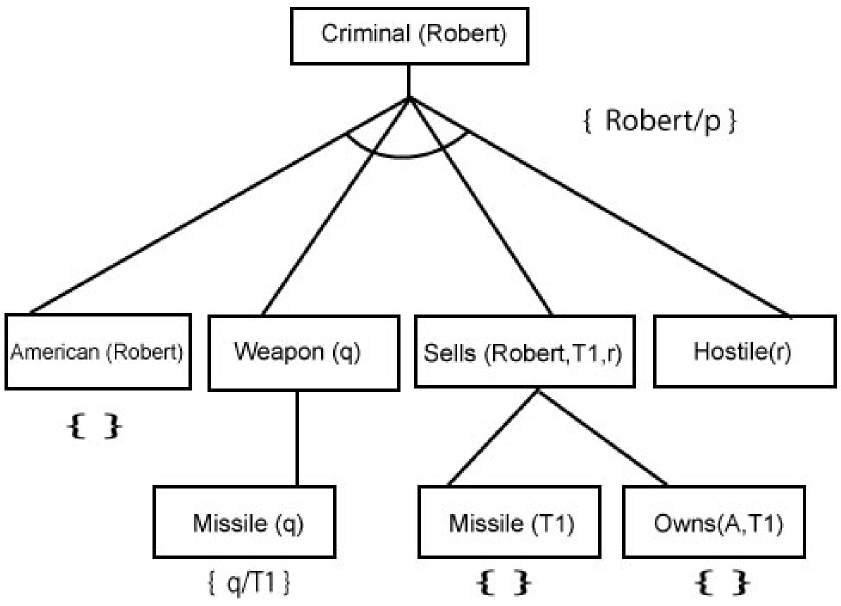


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.

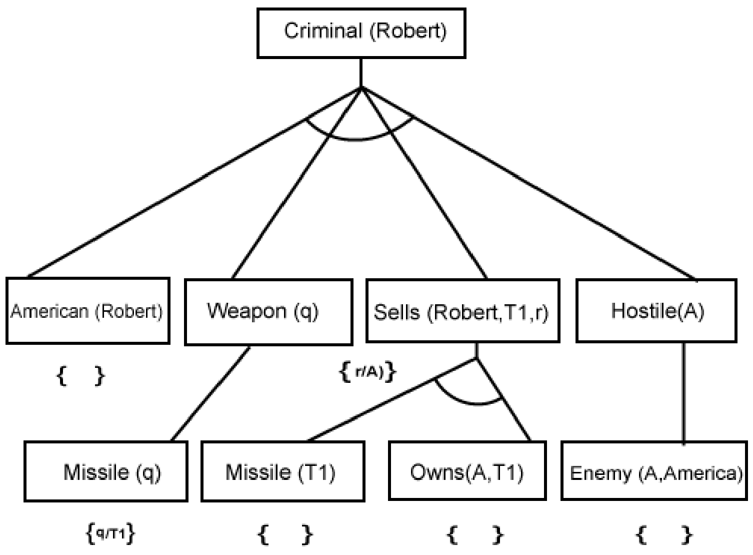


Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the Rule- 4, with the substitution of A in place of r. So these two statements are proved here.



Step-5:
At step-5, we can infer the fact Enemy(A, America) from Hostile(A) which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Resolution in FOL

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965. Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

Clause: Disjunction of literals (an atomic sentence) is called a clause. It is also known as a unit clause.
Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be conjunctive normal form or CNF.

The resolution inference rule:
The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

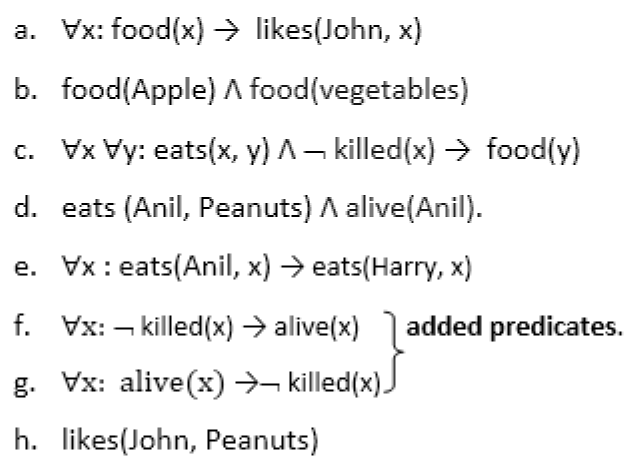
$$\frac{l_1 \vee \ldots\ldots \vee l_k, \qquad m_1 \vee \ldots\ldots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \ldots\ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots\ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots\ldots \vee m_n)}$$

Where l_i and m_j are complementary literals.
This rule is also called the binary resolution rule because it only resolves exactly two literals.
Example:
We can resolve two clauses which are given below:
[Animal (g(x) V Loves (f(x), x)] and [¬ Loves(a, b) V ¬Kills(a, b)]
Where two complimentary literals are: Loves (f(x), x) and ¬ Loves (a, b) These literals can be unified with unifier θ= [a/f(x), and b/x] , and it will generate a resolvent clause:
[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:
1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.
Example:
a. John likes all kind of food.
b. Apple and vegetable are food
c. Anything anyone eats and not killed is food.
d. Anil eats peanuts and still alive
e. Harry eats everything that Anil eats.
Prove by resolution that:
f. John likes peanuts.

Step-1: Conversion of Facts into FOL
In the first step we will convert all the given statements into its first order logic.



a. ∀x: food(x) → likes(John, x)
b. food(Apple) Λ food(vegetables)
c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) Λ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } added predicates.
g. ∀x: alive(x) →¬ killed(x)
h. likes(John, Peanuts)

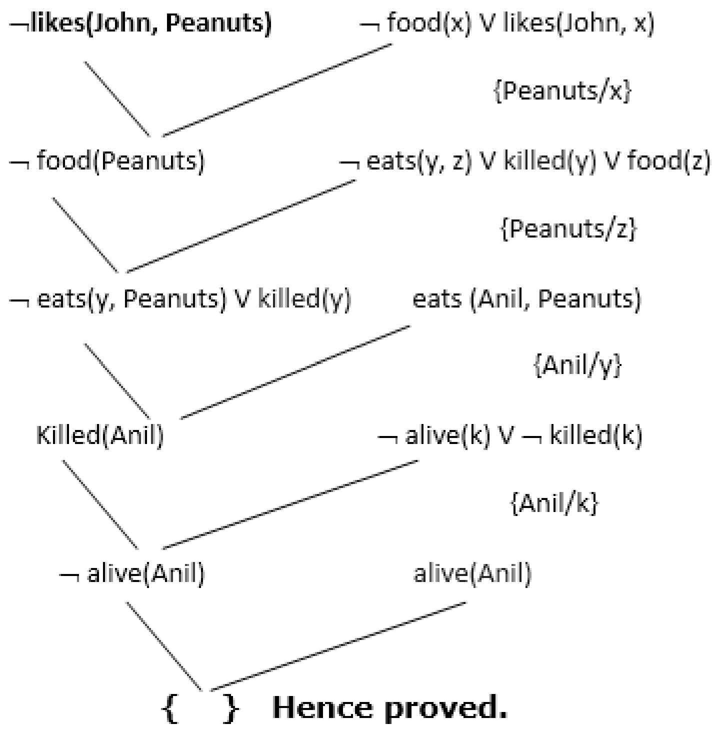
Step-2: Conversion of FOL into CNF
In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- Eliminate all implication (→) and rewrite
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x¬ [¬ killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).
- Move negation (¬)inwards and rewrite
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  f. ∀x ¬killed(x)] V alive(x)
  g. ∀x ¬ alive(x) V ¬ killed(x)
  h. likes(John, Peanuts).
- Rename variables or standardize variables
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀w¬ eats(Anil, w) V eats(Harry, w)
  f. ∀g ¬killed(g)] V alive(g)
  g. ∀k ¬ alive(k) V ¬ killed(k)
  h. likes(John, Peanuts).
- Eliminate existential instantiation quantifier by elimination. In this step, we will eliminate existential quantifier ∃, and this process is known as Skolemization. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

- Drop Universal quantifiers.
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.
  a. ¬ food(x) V likes(John, x)
  b. food(Apple)
  c. food(vegetables)
  d. ¬ eats(y, z) V killed(y) V food(z)
  e. eats (Anil, Peanuts)
  f. alive(Anil)
  g. ¬ eats(Anil, w) V eats(Harry, w)
  h. killed(g) V alive(g)
  i. ¬ alive(k) V ¬ killed(k)
  j. likes(John, Peanuts).
- Distribute conjunction ∧ over disjunction ¬.
  This step will not make any change in this problem.
-

Step-3: Negate the statement to be proved
In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph: Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, ¬likes(John, Peanuts) , and likes(John, x) get resolved(canceled) by substitution of {Peanuts/x}, and we are left with ¬ food(Peanuts)

- In the second step of the resolution graph, ¬ food(Peanuts) , and food(z) get resolved (canceled) by substitution of { Peanuts/z}, and we are left with ¬ eats(y, Peanuts) V killed(y) .
- In the third step of the resolution graph, ¬ eats(y, Peanuts) and eats (Anil, Peanuts) get resolved by substitution {Anil/y}, and we are left with Killed(Anil) .
- In the fourth step of the resolution graph, Killed(Anil) and ¬ killed(k) get resolve by substitution {Anil/k}, and we are left with ¬ alive(Anil) .
- In the last step of the resolution graph ¬ alive(Anil) and alive(Anil) get resolved.

MODULE-5

Acting Under uncertainty

- A logical agent believes a sentence to be either true or false. Probabilistic agents are those that have a degree of belief about the validity of a given sentence. And the belief could be ranging from 0 to 1.
- Uncertainty can arise because of incompleteness and incorrectness in the agents understanding of the properties of the environment. when we are talking of handling uncertainty, one needs to recall that if I am using first order logic to cope with complex domains, like medical diagnosis or criminal investigation or some form of methods and techniques to figure out false in a system.
- Example: diagnosing a toothache – Diagnosis: classic example of a problem with inherent uncertainty – Attempt 1: Toothache ⇒ HasCavity • But: not all toothaches are caused by cavities. Not true! – Attempt 2: Toothache ⇒ Cavity ∨ GumDisease ∨Abscess ∨ etc ∨ etc • To be true: would need nearly unlimited list of options...some unknown. – Attempt 3: Try make causal: Cavity ⇒ Toothache • Nope: not all cavities cause toothaches!
- There would be 3 main reasons why such first order logic systems would fail. One, we call it laziness, this is about too much work involved to create the complete set of antecedents or consequence, needed to ensure an exception less rule. And it is too hard to use the enormous rules that result out of this. So, if you are looking for completely covering one of these domains, then it would be too much of task either to list the complete set of incidents or consequence for a given rule or it would be even hard to really get all the rules in the system. Number 2 is about certain ignorance, which is referred to as the theoretical ignorance, it is about the expertise of the area, which may not be sufficient to have complete theory for the domain that is being worked with. And finally we have what is called the practical ignorance. Suppose we know all the rules yet we may be uncertain about particular cases, because all the necessary tests or evaluations may have not been or possibly cannot be done for that particular case.
- So agent's knowledge under such a situation can at best provide only a degree of belief in the relevant sentences. And that is why every sentence in under such a scenario cannot evaluate to being just true or false. We have to associate to each sentence that the knowledge of the agent comprises of to a degree of belief. This is not only true for the medical domain that I have been emphasizing. It is also true for most other judgmental domains like law, business, design, automobile repair, gardening, so on and so forth. So dealing with degrees of belief is what is done through the probability theory, which assigns a numerical degree of belief between 0 and 1 to the sentences.
- Probability provides a way of summarizing the uncertainty. And one needs to realize that this uncertainty comes from our laziness and ignorance. Laziness here refers to our inability to completely quantify the domain and ignorance their reference to either our theoretical ignorance of the domain or the practical ignorance when I am working with certain cases. So to make choices that is to make rational decisions, an agent must first have preferences between the possible outcomes of the plans and this is where we use what is called the utility theory.
- The utility theory is used to represent and reason with preferences. Here preference refers to options choices, and other alternatives of what is more preferred, outcomes are completely specified state and the utility theory is about figuring out which is more useful, or in terms we say the quality of being useful.

```
function DT-AGENT(percept) returns an action
  persistent: belief_state, probabilistic beliefs about the current state of the world
              action, the agent's action

  update belief_state based on action and percept
  calculate outcome probabilities for actions,
     given action descriptions and current belief_state
  select action with highest expected utility
     given probabilities of outcomes and utility information
  return action
```

Basic Notations of Probability

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.
$0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.
P(A) = 0, indicates total uncertainty in an event A.
P(A) =1, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

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

- P(¬A) = probability of a not happening event.
- P(¬A) + P(A) = 1.

Event: Each possible outcome of a variable is called an event. An event A is any subset of Ω – Allows us to group possible worlds, e.g., "doubles rolled with dice" – P(A) = Σ{ω∈A} P(ω) – e.g., P(doubles rolled) = P (1,1) + P (2,2) + ... + P (6,6)
Sample space: The collection of all possible events is called sample space. set Ω = all possible worlds that might exist – e.g., after two dice roll: 36 possible worlds (assuming distinguishable dice) – Possible worlds are exclusive and mutually exhaustive. Only one can be true (the actual world); at least one must be true – ω ∈ Ω is a sample point (possible world)
probability space or probability model is a sample space with an assignment P(ω) for every ω∈ Ω such that: – 0 ≤ P(ω) ≤ 1 – Σω P(ω) = 1 – e.g. for die roll: P(1,1) = P(1,2) = P(1,3) =... = P(6,6) = 1/36
Random variables: Random variables are used to represent the events and objects in the real world.
A proposition in the probabilistic world is then simply an assertion that some event (describing a set of possible worlds) is true. – θ="doubles rolled" à asserts event "doubles" is true à asserts {[1,1] ∨ [2,2] ∨...∨ [6,6]} is true. – Propositions can be compound: θ=(doubles ∧(total>4)) – P(θ) = Σω∈θ P(ω) à probability of proposition is sum of its parts
Prior probability: The prior probability of an event is probability computed before observing new information.
Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

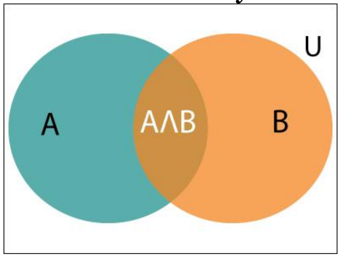
Conditional probability: Conditional probability is a probability of occurring an event when another event has already happened.
Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

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

Where $P(A \wedge B)$= Joint probability of a and B
P(B)= Marginal probability of B.
If the probability of A is given and we need to find the probability of B, then it will be given as:

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

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of $P(A \wedge B)$ by P(B).



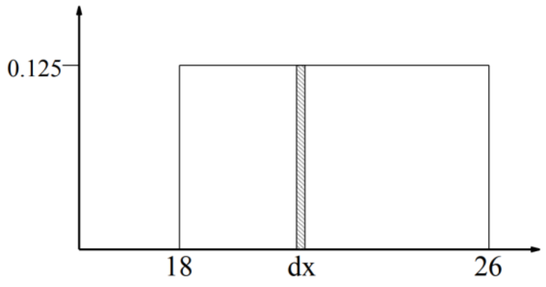
Probability Distributions

- One can express the probability of a proposition: –P(Weather=sunny) = 0.6 ; P(Cavity=false) = P(¬cavity)=0.1
- Probability Distribution expresses all possible probabilities for some event – So for: P(Weather=sunny) = 0.6; P(Weather=rain) = 0.1; P(Weather) = {0.72, 0.1, 0.29, 0.01} for Weather={sun, rain, clouds, snow}
- Hence probability distribution can be seen as total function that returns probabilities for all values of Weather. It is normalized, i.e., sum of all probabilities adds up to 1.
- Joint Probability Distribution: for a set of random variables, gives probability for every combination of values of every variable. – Gives probability for every event within the sample space – P(Weather, Cavity) = a 4x2 matrix of values:

Weather =	*sunny*	*rain*	*cloudy*	*snow*
Cavity = true	0.144	0.02	0.016	0.02
Cavity = false	0.576	0.08	0.064	0.08

- Full Joint Probability Distribution = joint distribution for all random variables in domain. Every probability question about a domain can be answered by full joint distribution, because every event is a sum of sample points (variable/value pairs). if the variables are Cavity, Toothache, and Weather, then the full joint distribution is given by P(Cavity, Toothache, Weather). This joint distribution can be represented as a 2 × 2 × 4 table with 16 entries. Because every proposition's probability is a sum over possible worlds, a full joint distribution suffices, in principle, for calculating the probability of any proposition.

- Some variables are continuous, e.g. P(Temp=82.3) = 0.23; P(Temp=82.5)= 0.24; etc. • Also could assert ranges: P(Temp<85) or P(40 < Temp <67)
- We can express distributions as a parameterized function of value: • P (X = x) = U [18, 26](x) = uniform density between 18 and 26. It is known as a probability density function (pdf). P is a really a density distribution; the whole range integrates to 1. The distribution pictorially looks like:



Kolomogorovs Axioms:

1. P(¬a) = 1- P(a)
Proof:
P(¬a) = Pω∈¬a P(ω)   (Definition of probability of any event)
= Pω∈¬a P(ω) + Pω∈a P(ω) - Pω∈a P(ω)
= Pω∈Ω P(ω) - Pω∈a P(ω) (grouping the first two terms)
= 1 - P(a)

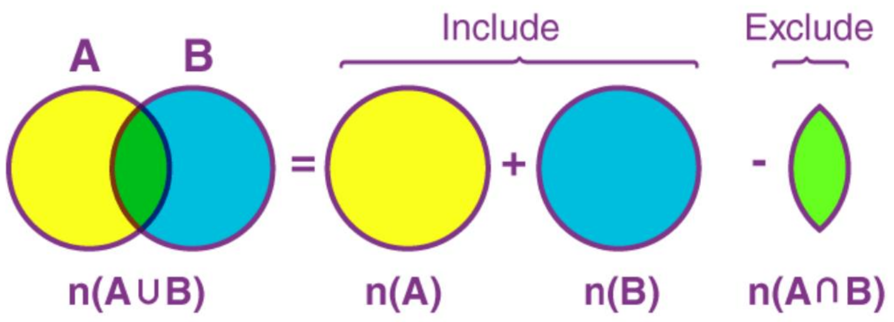
2. Probability distribution on a discrete random variable is 1
Proof:Let X be a discrete random variables of n events
Then, X={X1,X2,........,Xn}
P(X)=P(X1,X2...,Xn)
=P(X=X1)+P(X=X2)+....+P(X=Xn)
=$\sum_{x=1}^{n} P(x)$
= Pω∈Ω P(ω)   (All sample spaces)
=1

3. P(A|B)=1-P(A'|B)
Proof:
P(B)=P((A∩B)∪(A′∩B))
=P(A∩B)+P(A′∩B)
P(A∩B)=P(B)−P(A′∩B)
Divide each term by P(B)
P(A∩B)/P(B)=1- P(A′∩B)/P(B)
P(A|B)=1-P(A'|B) (By definition of conditional probability)

4. Inclusion-Exclusion principle

$$P(A \wedge B) = P(A) + P(B) - P(A \vee B)$$

Proof:



P(AUB)=P(AU(B-A))
=P(A)+P(B-A)
=P(A)+P(B-A)+P(A∩B)- P(A∩B)
=P(A)+P((B-A)U(A∩B))- P(A∩B)
=P(A)+P(B)- P(A∩B)

Eg: Given that bus arriving late=0.3 and a student oversleeping probability is 0.4., find the probability that student gets late

Let A=event that bus arrives late
B=event that student oversleeps
Given, P(A)=0.3 and P(B)=0.4
As per inclusion-exclusion principle
P(student gets late)=P(bus arrives late or student oversleeps)
=P(AUB)
= P(A)+P(B)- P(A∩B)
=0.3+0.4-P(A).P(B) [as A and B can both occur independently]
=0.3+0.4-0.3*0.4=0.7-0.12=0.58

Expectations and probability

A gambling theory where a money is attached to the outcome of the probability is called expectations. Think of it as a game between two agents: Agent 1 states, “my degree of belief in event a is 0.4.” Agent 2 is then free to choose whether to wager for or against a at stakes that are consistent with the stated degree of belief. That is, Agent 2 could choose to accept Agent 1’s bet that a will occur, offering $6 against Agent 1’s $4. Or Agent 2 could accept Agent 1’s bet that ¬a will occur, offering $4 against Agent 1’s $6. Then we observe the outcome of a, and whoever is right collects the money. If an agent’s degrees of belief do not accurately reflect the world, then you would expect that it would tend to lose money over the long run to an opposing agent whose beliefs more accurately reflect the state of the world

Agent 1		Agent 2		Outcomes and payoffs to Agent 1			
Proposition	Belief	Bet	Stakes	a, b	$a, \neg b$	$\neg a, b$	$\neg a, \neg b$
a	0.4	a	4 to 6	–6	–6	4	4
b	0.3	b	3 to 7	–7	3	–7	3
$a \vee b$	0.8	$\neg(a \vee b)$	2 to 8	2	2	2	–8
				–11	–1	–1	–1

Thus in all cases agent1 loses money due to the belief values it has attached.

Inference using Full Joint Distributions

Logical Inference is asking whether something is true (entailed), given the KB. However, Probabilistic Inference is about asking how likely something is, given the KB . The process is to just compute the posterior probability for query proposition, given KB!. We use the full joint probability distribution as the KB! Which contains the probability of all possible worlds!. The mechanism of the inference is to look up the probability of a query proposition. Further, extract and sum up the appropriate “slice” of the joint distribution.

Example: Consider a world with just three boolean variables – Toothache (has one or not) – Cavity (has or not) – Catch (dentists tool catches or not).

Start with the full joint distribution for this world:

	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬cavity	**.016**	**.064**	**.144**	**.576**

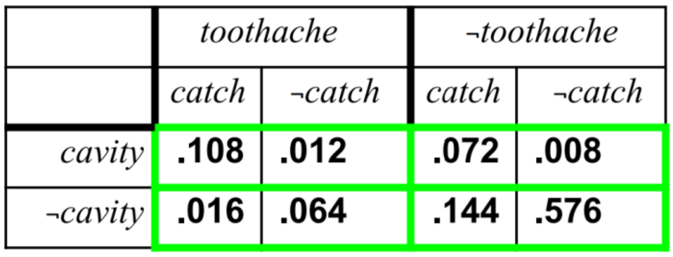
Marginalization:

	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬cavity	**.016**	**.064**	**.144**	**.576**

Sum up probabilities across values of other (non-specified) variables. In this case: Cavity and Catch.

Generally: P(Y) = Σz∈Z P(Y,z) ,or also, by product rule: P(Y)=Σz∈Z P(Y|z) P(z)

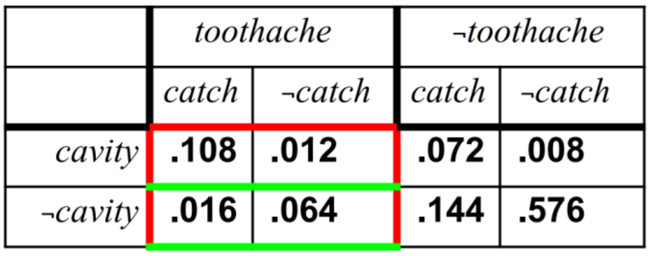
Mutual Exclusion



	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬cavity	**.016**	**.064**	**.144**	**.576**

P (cavity∨toothache) = 0.108+0.012+0.072+0.008+0.016+0.064 = 0.28
For any proposition φ, the P(φ) = sum the atomic events where it is true:
P (φ) = Σω:ω|=φP (ω)

Conditional probabilities



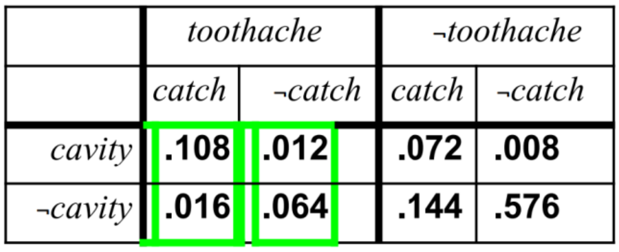
	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬cavity	**.016**	**.064**	**.144**	**.576**

$$P(\neg cavity \mid toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)} \qquad \text{(Product rule)}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064}$$

$$= 0.4$$

Normalization



	toothache		*¬toothache*	
	catch	*¬catch*	*catch*	*¬catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬cavity	**.016**	**.064**	**.144**	**.576**

$$P(cavity|toothache) = \frac{P(cavity \wedge toothache)}{P(toothache)}$$

Denominator can be viewed as a normalization constant α for the distribution P(Cavity| toothache) ensures that the probability of the distribution adds up to 1.
P(Cavity|toothache) = α P(Cavity, toothache)
= α [P(Cavity, toothache, catch) + P(Cavity, toothache, ¬catch)]
= α [(0.108, 0.016) + (0.012, 0.064)]
= α (0.12, 0.08) = (0.6, 0.4)

Note that proportions between (0.12, 0.08) and (0.6, 0.4) are same. The latter are just normalized by application of α to add up to 1. So if α just normalizes, one could also normalize "manually" à divide by sum of two.

Problems

1. In a shipment of 20 apples, 3 are rotten. 3 apples are randomly selected. What is the probability that all three are rotten if the first and second are not replaced?

Solution:
P(first apple is rotten)=3/20
P(second apple is rotten)=2/19 (as first apple is not replaced)
P(third apple is rotten)=1/18 (as second apple is also not replaced)
P(all three are rotten)=3/20 * 2/19 * 1/18=1/1140

2. A die is cast twice and a coin is tossed twice. What is the probability that the die will turn a 6 each time and the coin will turn a tail every time?

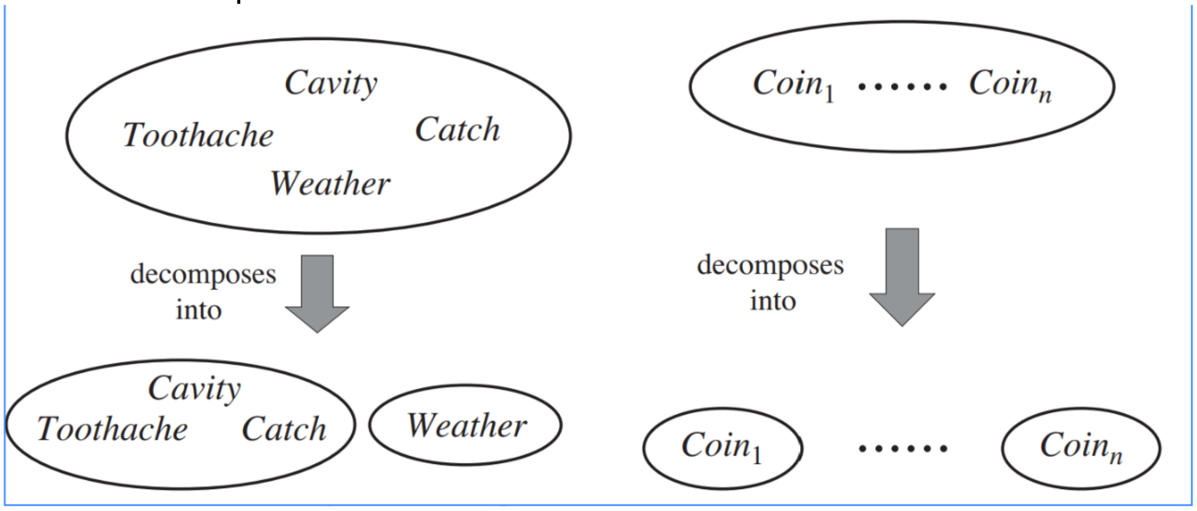
Solution:
P(die turn up 6 first time)=1/6
P(die turn up 6 second time)=1/6
P(coin turn up tail first time)=1/2
P(coin turn up tail second time)=1/2
P(die turn up 6 and coin turn up tail each time)=1/6*1/6*1/2*1/2=1/144

3. An instructor has a question bank with 300 Easy T/F, 200 Difficult T/F, 500 Easy MCQ, and 400 Difficult MCQ. If a question is selected randomly from the question bank, What is the probability that it is an easy question given that it is an MCQ?
Solution
P(Easy)=800/1400
P(MCQ)=900/1400
P(Easy $\wedge MCQ$)=500/1400
P(Easy | MCQ)=P(Easy $\wedge MCQ$)/P(MCQ)=500/900=5/9

Independence of variables

The problem of full joint distribution get huge fast with the cross-product of all variables, all values in their range. Different probability for every variables...conditional on all values of all other variables is required. But are all of these variables really related? Is every variable really related to all others?
Consider P(toothache, catch, cavity, cloudy) à 2 x 2 x 2 x 4 joint distr. = 32 entries – By product rule:
P(toothache, catch, cavity, cloudy) = P(cloudy|toothache,catch,cavity) P(touchache,catch,cavity).
But it the weather really conditional on toothaches, cavities and dentist's tools? No! – So realistically:
P(cloudy|toothache,catch,cavity) = P(cloudy).
So then actually:
P(toothache, catch, cavity, cloudy) = P(cloudy) P(touchache,catch,cavity).

We say that cloudy and dental variables are independent (also absolute independence). Effectively: the 32-element joint distribution table becomes one 8-element table + 4-element table



$$\mathbf{P}(X \mid Y) = \mathbf{P}(X) \quad \text{or} \quad \mathbf{P}(Y \mid X) = \mathbf{P}(Y) \quad \text{or} \quad \mathbf{P}(X, Y) = \mathbf{P}(X)\mathbf{P}(Y) .$$

Independence assertions based on judgment, specific knowledge of domain. It can dramatically reduce information needed for full joint distribution

Bayes' theorem:

Bayes' theorem is also known as Bayes' rule, Bayes' law, or Bayesian reasoning, which determines the probability of an event with uncertain knowledge. In probability theory, it relates the conditional probability and marginal probabilities of two random events. Bayes' theorem was named after the British mathematician Thomas Bayes. The Bayesian inference is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).
Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:
As from product rule we can write:
P(A ∧ B)= P(A|B) P(B) or
Similarly, the probability of event B with known event A:
P(A ∧ B)= P(B|A) P(A)
Equating right hand side of both the equations, we will get:

$$\boxed{P(A|B) = \frac{P(B|A)\, P(A)}{P(B)}} \quad \text{....(a)}$$

The above equation (a) is called as Bayes' rule or Bayes' theorem. This equation is basic of most modern AI systems for probabilistic inference.
It shows the simple relationship between joint and conditional probabilities. Here,
P(A|B) is known as posterior, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.
P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.
P(A) is called the prior probability, probability of hypothesis before considering the evidence

P(B) is called marginal probability, pure probability of an evidence.
In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

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

Where $A_1, A_2, A_3, \ldots\ldots, A_n$ is a set of mutually exclusive and exhaustive events.
Applying Bayes' rule:
Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})\,P(\text{cause})}{P(\text{effect})}$$

Eg: a doctor knows that the disease meningitis causes the patient to have a stiff neck,say, 70% of the time. probability that a patient has meningitis is 1/50,000, and probability that any patient has a stiff neck is 1%. Find probability of meningitis due to stiff neck

- The Known probability that a patient has meningitis disease is 1/50,000.
- The Known probability that a patient has a stiff neck is 1%.

Let s be the proposition that patient has stiff neck and m be the proposition that patient has meningitis. , so we can calculate the following as:
P(m|s) = 0.7
P(m) = 1/50000
P(s)= .02

$$\begin{aligned} P(s\,|\,m) &= 0.7 \\ P(m) &= 1/50000 \\ P(s) &= 0.01 \\ P(m\,|\,s) &= \frac{P(s\,|\,m)P(m)}{P(s)} = \frac{0.7 \times 1/50000}{0.01} = 0.0014\,. \end{aligned}$$

Problems on Bayes Theorem:

In a neighbourhood, 90% children were falling sick due flu and 10% due to measles and no other disease. The probability of observing rashes for measles is 0.95 and for flu is 0.08. If a child develops rashes, find the child's probability of having flu.

Solution:
Let,
F: children with flu
M: children with measles
R: children showing the symptom of rash
P(F) = 90% = 0.9
P(M) = 10% = 0.1
P(R|F) = 0.08

P(R|M) = 0.95

$$P(F|R) = \frac{P(R|F)P(F)}{P(R|M)P(M)+P(R|F)P(F)}$$

$$P(F|R) = \frac{0.08\times0.9}{0.95\times0.1+0.08\times0.9}$$

= 0.072/(0.095 + 0.072) = 0.072/0.167 ≈ 0.43

⇒ P(F|R) = 0.43

Bayes Theorem with combined evidence

$$\mathbf{P}(Cavity \mid toothache \wedge catch)$$
$$= \alpha\, \mathbf{P}(toothache \wedge catch \mid Cavity)\, \mathbf{P}(Cavity)\,.$$

$$\mathbf{P}(toothache \wedge catch \mid Cavity) = \mathbf{P}(toothache \mid Cavity)\mathbf{P}(catch \mid Cavity)\,.$$

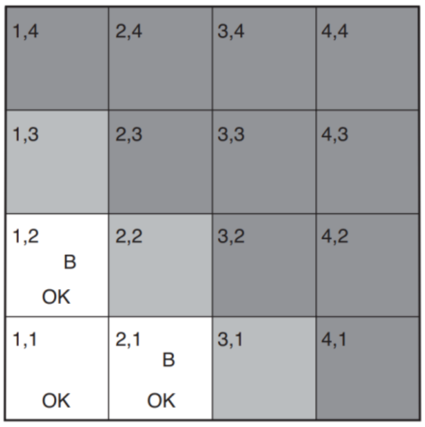
$$\mathbf{P}(Cavity \mid toothache \wedge catch)$$
$$= \alpha\, \mathbf{P}(toothache \mid Cavity)\, \mathbf{P}(catch \mid Cavity)\, \mathbf{P}(Cavity)\,.$$

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.One of the most simple and effective classification algorithms, the Naïve Bayes classifier aids in the rapid development of machine learning models with rapid prediction capabilities. Naïve Bayes algorithm is used for classification problems. It is highly used in text classification. In text classification tasks, data contains high dimension (as each word represent one feature in the data). It is used in spam filtering, sentiment detection, rating classification etc. The advantage of using naïve Bayes is its speed. It is fast and making prediction is easy with high dimension of data.

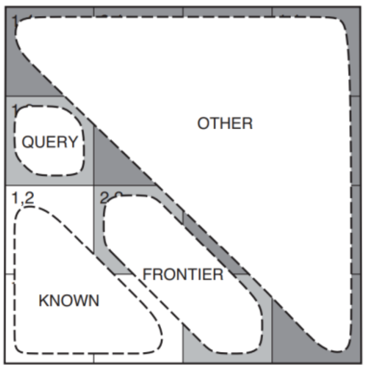
The "Naive" part of the name indicates the simplifying assumption made by the Naïve Bayes classifier. The classifier assumes that the features used to describe an observation are conditionally independent, given the class label. The "Bayes" part of the name refers to Reverend Thomas Bayes, an 18th-century statistician and theologian who formulated Bayes' theorem.

Probability and Wumpus world

Uncertainty arises in the wumpus world because the agent's sensors give only partial information about the world. For example, Figure below shows a situation in which each of the three reachable squares—[1,3], [2,2], and [3,1]—might contain a pit. Pure logical inference can conclude nothing about which square is most likely to be safe, so a logical agent might have to choose randomly. Probabilistic agent performs better than logic agent.

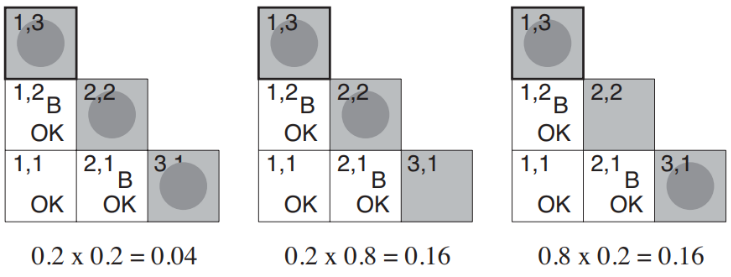


The aim is to calculate the probability that each of the three squares contains a pit. The relevant properties of the wumpus world are that (1) a pit causes breezes in all neighboring squares, and (2) each square other than [1,1] contains a pit with probability 0.2. Wumpus world is divided into three regions: Query node whose probability is seeked. Known nodes whose probability is known. Frontier nodes are nodes adjacent to known nodes except query node. Remaining nodes in the Wumpus world form other nodes.

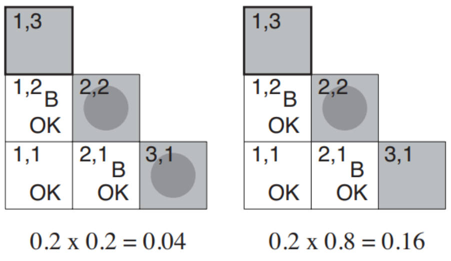


Different probability calculations are shown.

Case-1: [1,3] as a pit and various combinations of frontiers are computed.



Case-2: [1,3] do not have put and various combinations of frontiers are computed.


1,3
1,2 B OK
2,2
1,1 OK
2,1 B OK
3,1
0.2 x 0.2 = 0.04
0.2 x 0.8 = 0.16