

Unit-1

Introduction to Artificial Intelligence

Assessment Component	ICA (100 Marks) (Marks scaled to 50)				TEE (100 marks) (Marks scaled to 50)	
	Lab Performance	Assignment/ Mini Project	Presentation of a research paper (Group activity)	Class Test1 and Class Test 2	Class Participation	
Weightage	10%	10%	5%	20%	5%	50%
Marks	20	20	10	20+20	10	100

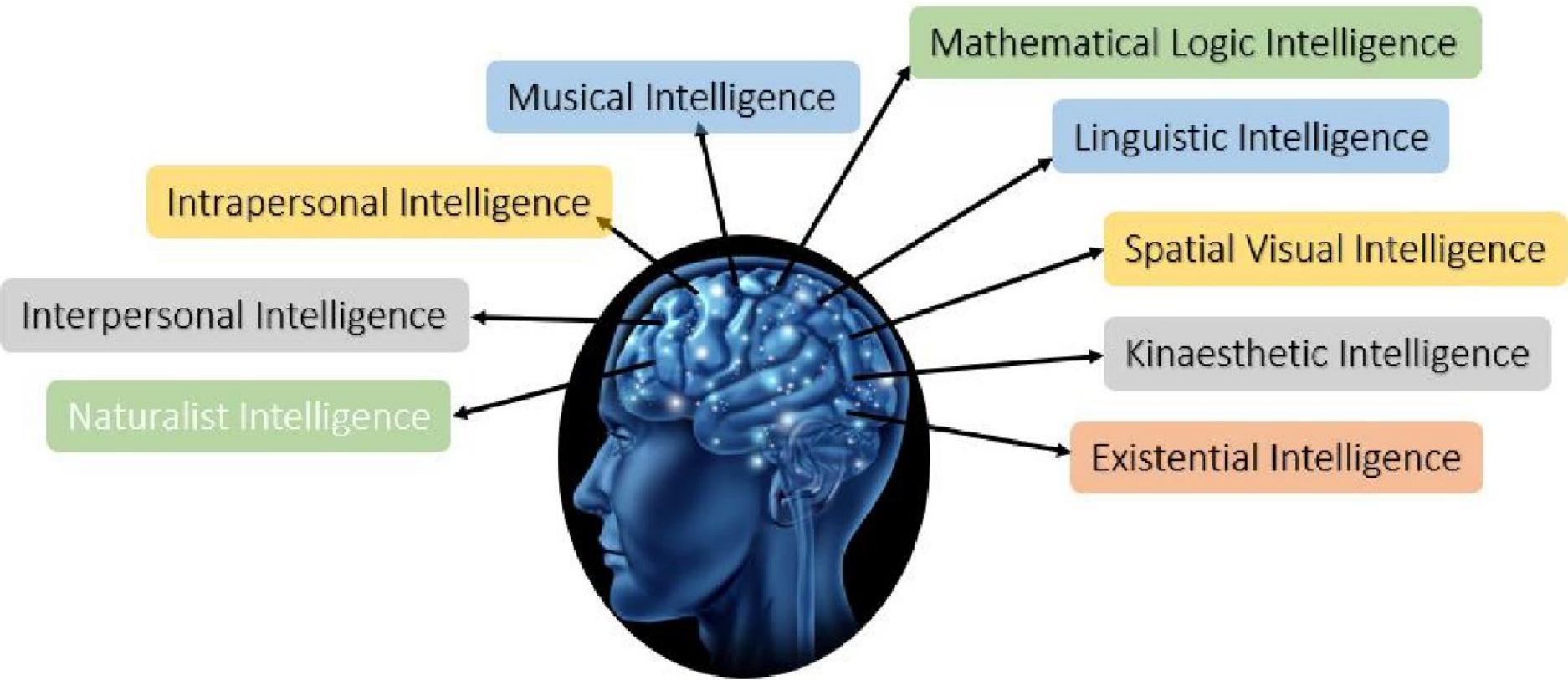
Intelligence vs Artificial Intelligence

- Intelligence is a property/ability attributed to people, such as to know, to think, to talk, to learn.

Intelligence = Knowledge + ability to perceive, feel, comprehend, process, communicate, judge, learn.

- Artificial Intelligence is an interdisciplinary field aiming at developing techniques and tools for solving problems that people are good at.

Define Intelligence with the Help of its Traits



Mathematical Logical Reasoning

- A person's ability to regulate, measure, and understand numerical symbols, abstraction and logic.

Linguistic Intelligence

- Language processing skills both in terms of understanding or implementation in writing or verbally.

Spatial Visual Intelligence

- It is defined as the ability to perceive the visual world and the relationship of one object to another.

Kineesthetic Intelligence

- Ability that is related to how a person uses his limbs in a skilled manner.

Musical Intelligence

- As the name suggests, this intelligence is about a person's ability to recognize and create sounds, rhythms, and sound patterns.

Intrapersonal Intelligence

- Describes how high the level of self-awareness someone has is. Starting from realizing weakness, strength, to his own feelings.

Existential Intelligence

- An additional category of intelligence relating to religious and spiritual awareness.

Naturalist Intelligence

- An additional category of intelligence relating to the ability to process information on the environment around us.

Interpersonal intelligence

- Interpersonal intelligence is the ability to communicate with others by understanding other people's feelings & influence of the person.

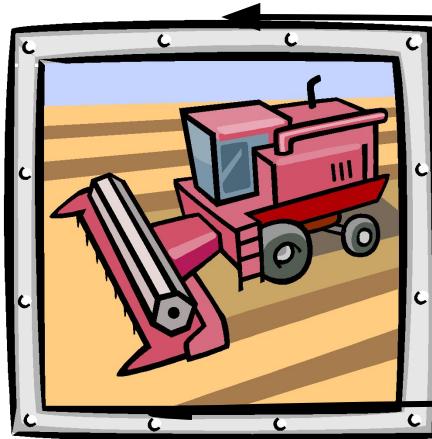
You are locked inside a room with 3 doors to move out of the locked room and you need to find a safe door to get your way out. Behind the 1st door is a lake with a deadly shark. The 2nd door has a mad psychopath ready to kill with a weapon and the third one has a lion that has not eaten since the last 2 months.



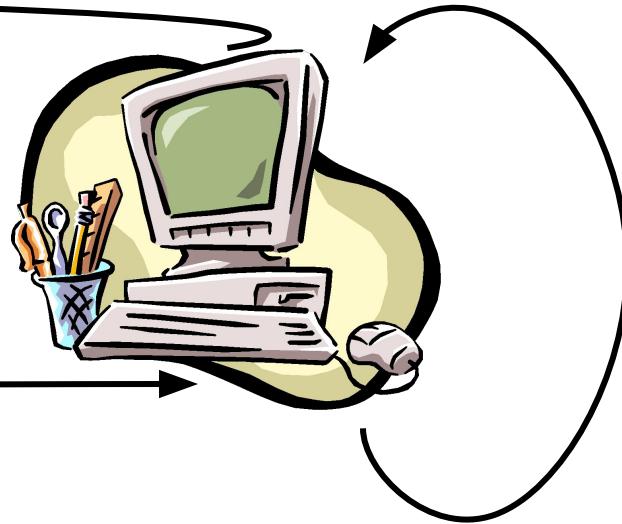
In other words, we may define intelligence as:

- **Ability to interact with the real world**
 - To perceive, understand and act
 - Example: Speech Recognition
 - Example: Image Recognition
 - Example: Ability to take action: to have an effect
- **Reasoning and planning**
 - Modelling the external world, given input
 - Solving new problems, planning and making decisions
 - Ability to deal with unexpected problems, uncertainties
- **Learning and adaptation**
 - Continuous learning and adapting graph
 - Our internal models are always being updated
 - Example: Baby learning to categorize and recognise animals

Apply



Solve



Formulate

Definitions of AI

Existing definitions advocate everything from replicating human intelligence to simply solving knowledge-intensive tasks.

Examples:

“Artificial Intelligence is a study of complex information processing problems that often have their roots in some aspect of biological information processing. The goal of the subject is to identify solvable and interesting information processing problems, and solve them.” -- **David Marr**

“Artificial Intelligence is the design, study and construction of computer programs that behave intelligently.” -- **Tom Dean**

“Artificial Intelligence is the enterprise of constructing physical symbol system that can reliably pass the Turing test.” -- **Matt Ginsberg.**

“AI is defined as an experimental discipline utilizing the ideas and the methods of computation.”

- As per first reference book, “**The study of how to make computers do things at which, at the moment, people are better.**”

Rich & Knight, 1991

Goals of Artificial Intelligence

- **Scientific goal:** understand the mechanism behind human intelligence.
- **Engineering goal:** develop concepts and tools for building intelligent agents capable of solving real world problems. Examples:
 - **Knowledge-based systems:** capture expert knowledge and apply them to solve problems in a limited domain.
 - **Common sense reasoning systems:** capture and process knowledge that people commonly hold which is not explicitly communicated.
 - **Learning systems:** posses the ability to expand their knowledge based on the accumulated experience.
 - **Natural language understanding systems.**
 - **Intelligent robots.**
 - **Speech and vision recognition systems.**
 - **Game playing (IBM's Deep Blue)**

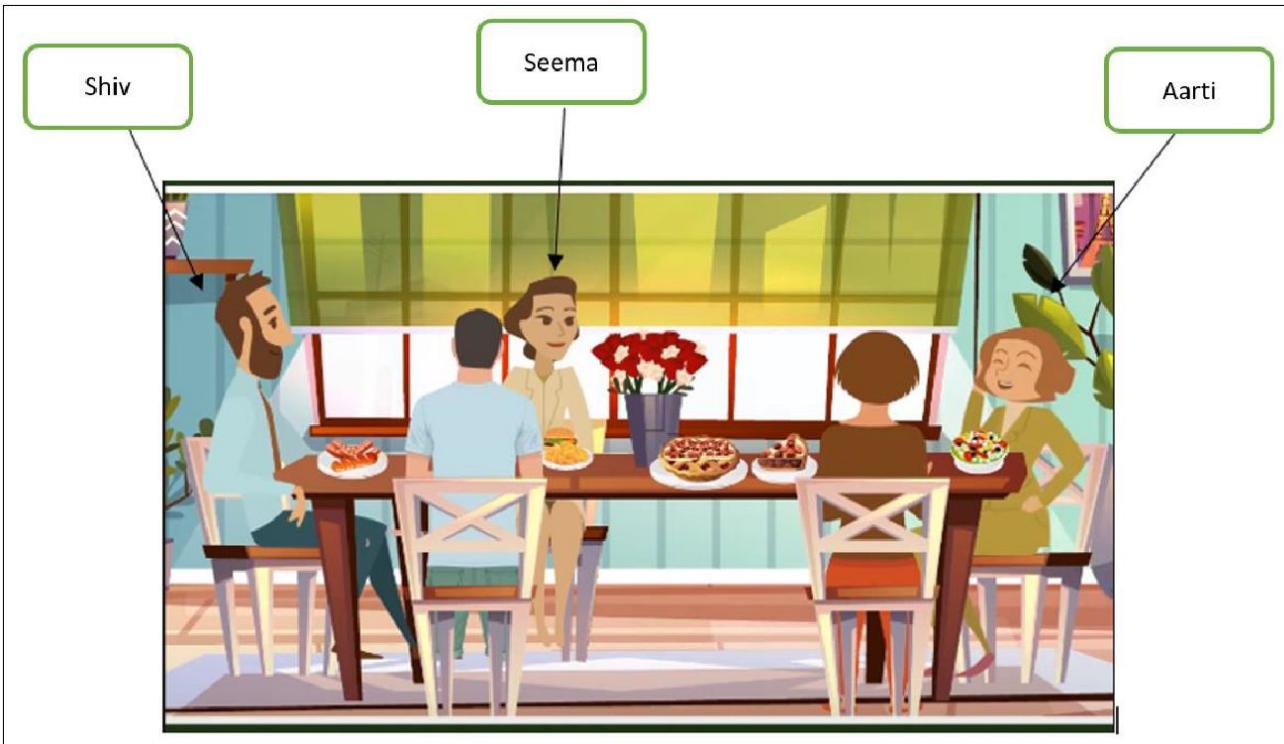
Goals of Artificial Intelligence

- To make computers more useful by letting them take over dangerous or tedious tasks from human
- Understand principles of human intelligence

Applications of AI

- **Gaming** – AI plays important role for machine to think of large number of possible positions based on deep knowledge in strategic games. for example, chess, river crossing, N-queens problems and etc.
- **Natural Language Processing** – Interact with the computer that understands natural language spoken by humans.
- **Expert Systems** – Machine or software provide explanation and advice to the users.
- **Vision Systems** – Systems understand, explain, and describe visual input on the computer.
- **Speech Recognition** – There are some AI based speech recognition systems have ability to hear and express as sentences and understand their meanings while a person talks to it. For example Siri and Google assistant.
- **Handwriting Recognition** – The handwriting recognition software reads the text written on paper and recognize the shapes of the letters and convert it into editable text

Aarti invited four of her friends to her House.. They hadn't seen each other in a long time, so they chatted all night long and had a good time. In the morning, two of the friends Aarti had invited, died. The police arrived at the house and found that both the friends were poisoned and that the poison was in the strawberry pie. The three surviving friends told the police that they hadn't eaten the pie. The police asked," Why didn't you eat the pie ?". Shiv said, " I am allergic to strawberries.". Seema said, " I am on a diet." And Aarti said, "I ate too many strawberries while cooking the pie, I just didn't want anymore."



who is the murderer?



What is Artificial Intelligence ?

- making computers that think?
- the automation of activities we associate with human thinking, like decision making, learning ... ?
- the art of creating machines that perform functions that require intelligence when performed by people ?
- the study of mental faculties through the use of computational models ?

What is Artificial Intelligence ?

- the study of computations that make it possible to perceive, reason and act ?
- a field of study that seeks to explain and emulate intelligent behaviour in terms of computational processes ?
- a branch of computer science that is concerned with the automation of intelligent behaviour ?
- anything in Computing Science that we don't yet know how to do properly ? (!)

What is Artificial Intelligence ? (A rough classification)

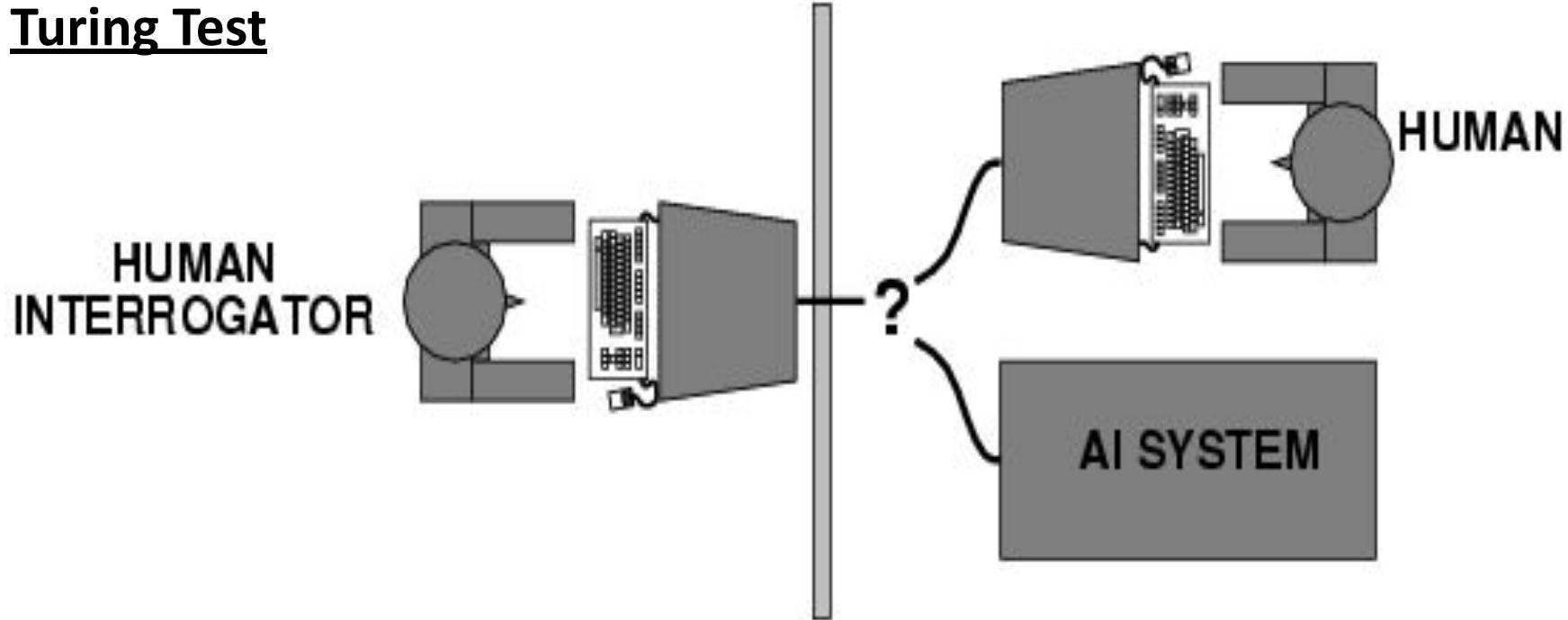
	THOUGHT	Systems that think like humans	Systems that think rationally
BEHAVIOUR		Systems that act like humans	Systems that act rationally

HUMAN **RATIONAL**

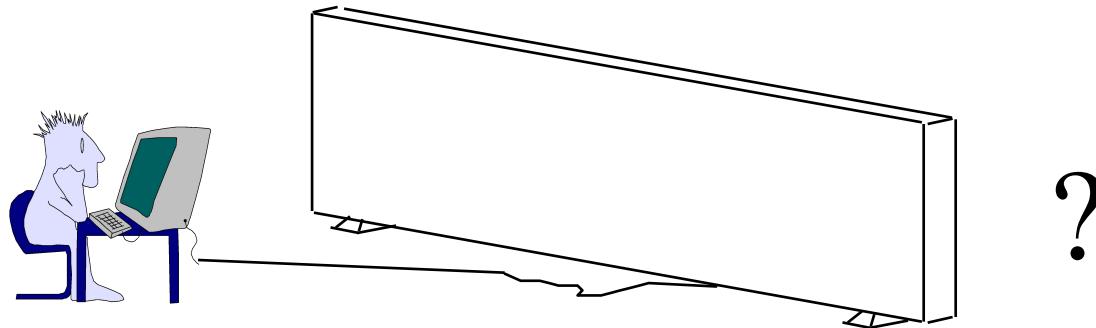
Systems that act like humans: Turing Test

- “The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil)
- “The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight)

Turing Test



Systems that act like humans



- You enter a room which has a computer terminal. You have a fixed period of time to type what you want into the terminal, and study the replies. At the other end of the line is either a human being or a computer system.
- If it is a computer system, and at the end of the period you cannot reliably determine whether it is a system or a human, then the system is deemed to be intelligent.

Systems that act like humans

- The Turing Test approach
 - a human questioner cannot tell if
 - there is a computer or a human answering his question, via teletype (remote communication)
 - The computer must behave intelligently
- Intelligent behavior
 - to achieve human-level performance in all cognitive tasks

Systems that act like humans

- These cognitive tasks include:
 - *Natural language processing*
 - for communication with human
 - *Knowledge representation*
 - to store information effectively & efficiently
 - *Automated reasoning*
 - to retrieve & answer questions using the stored information
 - *Machine learning*
 - to adapt to new circumstances

The total Turing Test

- Includes two more issues:
 - *Computer vision*
 - to perceive objects (seeing)
 - *Robotics*
 - to move objects (acting)

What is Artificial Intelligence ?

	Humanly	Rationally
Thinking	Thinking humanly — cognitive modeling. Systems should solve problems the same way humans do.	Thinking rationally — the use of logic. Need to worry about modeling uncertainty and dealing with complexity.
Acting	Acting humanly — the Turing Test approach.	Acting rationally — the study of rational agents: agents that maximize the expected value of their performance measure given what they currently know.

Systems that think like humans: cognitive modeling

- Humans as observed from ‘inside’
- How do we know how humans think?
 - Introspection vs. psychological experiments
- Cognitive Science
- “The exciting new effort to make computers think ... machines with *minds* in the full and literal sense” (Haugeland)
- “[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ...” (Bellman)

What is Artificial Intelligence ?

	THOUGHT	Systems that think like humans	Systems that think rationally
BEHAVIOUR		Systems that act like humans	Systems that act rationally
HUMAN		RATIONAL	

Systems that think ‘rationally’

"laws of thought"

- Humans are not always ‘rational’
- Rational - defined in terms of logic?
- Logic can’t express everything (e.g. uncertainty)
- Logical approach is often not feasible in terms of computation time (needs ‘guidance’)
- “The study of mental facilities through the use of computational models” (Charniak and McDermott)
- “The study of the computations that make it possible to perceive, reason, and act” (Winston)

What is Artificial Intelligence ?

	THOUGHT	Systems that think like humans	Systems that think rationally
BEHAVIOUR		Systems that act like humans	Systems that act rationally
HUMAN		RATIONAL	

Systems that act rationally: “Rational agent”

- Rational behavior: doing the right thing
- The right thing: that which is expected to maximize goal achievement, given the available information
- Giving answers to questions is ‘acting’.
- I don't care whether a system:
 - replicates human thought processes
 - makes the same decisions as humans
 - uses purely logical reasoning

Systems that act rationally

- Logic \square only *part* of a rational agent, not *all* of rationality
 - Sometimes logic cannot reason a correct conclusion
 - At that time, some specific (in domain) human knowledge or information is used
- Thus, it covers more generally different situations of problems
 - Compensate the incorrectly reasoned conclusion

Systems that act rationally

- Study AI as rational agent –
2 advantages:
 - It is more general than using logic only
 - Because: LOGIC + Domain knowledge
 - It allows extension of the approach with more scientific methodologies

Rational agents

- An **agent** is an entity that perceives and acts
- **This course is about designing rational agents**
- Abstractly, an agent is a function from percept histories to actions:

$$[f: P^* \rightarrow A]$$

- For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance
 - computational limitations make perfect rationality unachievable
 - design best program for given machine resources

- Artificial
 - Produced by human art or effort, rather than originating naturally.
- Intelligence is the ability to acquire knowledge and use it" [Pigford and Baur]
- **So AI was defined as:**
 - **AI** is the study of ideas that enable computers to be intelligent.
 - **AI** is the part of computer science concerned with design of computer systems that exhibit human intelligence(From the Concise Oxford Dictionary)

From the above two definitions, we can see that AI has two major roles:

- Study the intelligent part concerned with humans.
- Represent those actions using computers.

AI Problems (Task Domains)

Mundane Tasks

- Perception
 - Vision
 - Speech
- Natural language
 - Understanding
 - Generation
 - Translation
- Commonsense reasoning
- Robot control

Formal Tasks

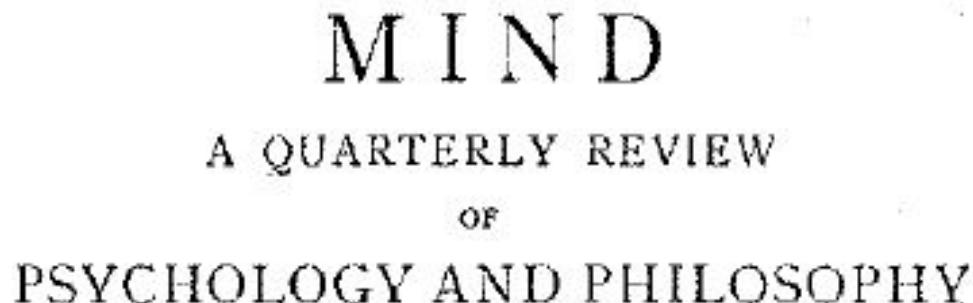
- Games
 - Chess
 - Backgammon
 - Checkers
 - Go
- Mathematics
 - Geometry
 - Logic
 - Integral calculus
 - Proving properties of programs

Expert Tasks

- Engineering
 - Design
 - Fault finding
 - Manufacturing planning
- Scientific analysis
- Medical diagnosis
- Financial analysis

Brief history of AI

- The history of AI begins with the following articles:
 - Turing, A.M. (1950), Computing machinery and intelligence, Mind, Vol. 59, pp. 433-460.



I.—COMPUTING MACHINERY AND
INTELLIGENCE

By A. M. TURING

I propose to consider the question, ‘Can machines think?’ ...

Alan Turing - Father of AI

Alan Turing

- Born 23 June 1912, Maida Vale, London, England
- Died 7 June 1954 (aged 41), Wilmslow, Cheshire, England
- Fields: Mathematician, logician, cryptanalyst, computer scientist
- Institutions:
 - University of Manchester
 - National Physical Laboratory
 - Government Code and Cypher School (Britain's codebreaking centre)
 - University of Cambridge



Alan Turing memorial
statue in Sackville Park,
Manchester

Brief history of AI - The Birth of AI

- **The birth of artificial intelligence**

- 1950: Turing's landmark paper "Computing machinery and intelligence" and Turing Test
- 1951: AI programs were developed at Manchester:
 - A draughts-playing program by Christopher Strachey
 - A chess-playing program by Dietrich Prinz
 - These ran on the Ferranti Mark I in 1951.
- 1955: Symbolic reasoning and the Logic Theorist
 - Allen Newell and (future Nobel Laureate) Herbert Simon created the "Logic Theorist". The program would eventually prove 38 of the first 52 theorems in Russell and Whitehead's Principia Mathematica
- 1956: Dartmouth Conference - "Artificial Intelligence" adopted

Brief history of AI - The Birth of AI

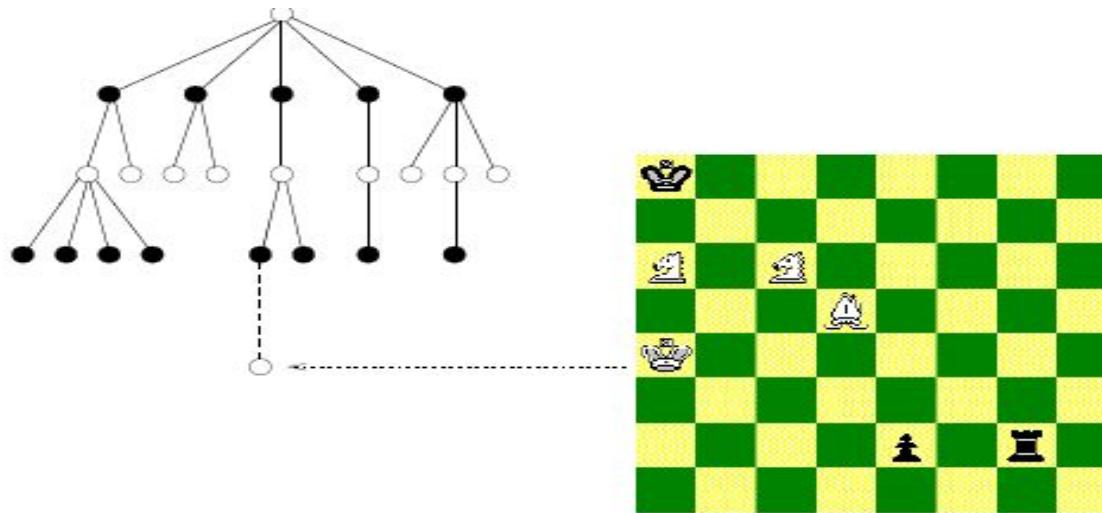
- **The birth of artificial intelligence**
 - 1956: Dartmouth Conference - "Artificial Intelligence" adopted
 - The term 'Artificial Intelligence' was coined in a proposal for the conference at Dartmouth College in 1956



- Check PDF

Brief history of AI – The Birth of AI

- One of the early research in AI is search problem such as for game-playing. Game-playing can be usefully viewed as a search problem in a space defined by a fixed set of rules



- Nodes are either white or black corresponding to reflect the adversaries' turns.
- The tree of possible moves can be searched for favourable positions.

Brief history of AI – The Birth of AI

- The real success of AI in game-playing was achieved much later after many years' effort.
- It has been shown that this search based approach works extremely well.
- In 1996 IBM Deep Blue beat Gary Kasparov for the first time. and in 1997 an upgraded version won an entire match against the same opponent.



Brief history of AI – The Birth of AI

- Another of the early research in AI was applied the similar idea to **deductive logic**:

$$\begin{array}{c} \text{All men are mortal} \\ \text{Socrates is a man} \\ \hline \text{Socrates is mortal} \end{array} \qquad \begin{array}{c} x \ (\underset{\nabla}{\text{man}}(x) \rightarrow \text{mortal}(x) \) \\ \text{man}(\text{Socrates}) \\ \hline \text{mortal}(\text{Socrates}) \end{array}$$

- The discipline of developing programs to perform such logical inferences is known as (automated) **theorem-proving**
- Today, theorem-provers are highly-developed . . .

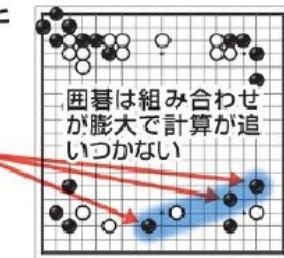
Current status of AI

- In March 2016, Alpha-Go of DeepMind defeated Lee Sedol, who was the strongest human GO player at that time.
- This is a big news that may have profound meaning in the human history.

人工知能の従来の方式と
「アルファ碁」の違い

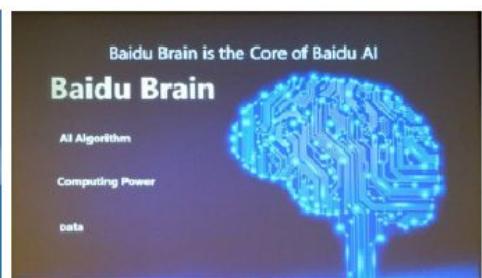
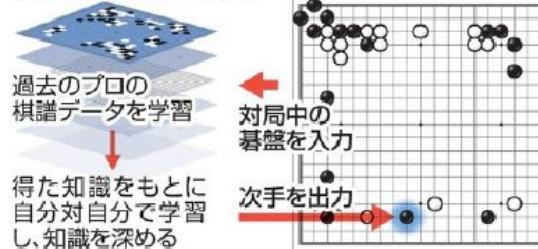
従来の方式

終局までランダムに
碁石を置くシミュレ
ーションを繰り返し、
勝率が高い次手を
求める

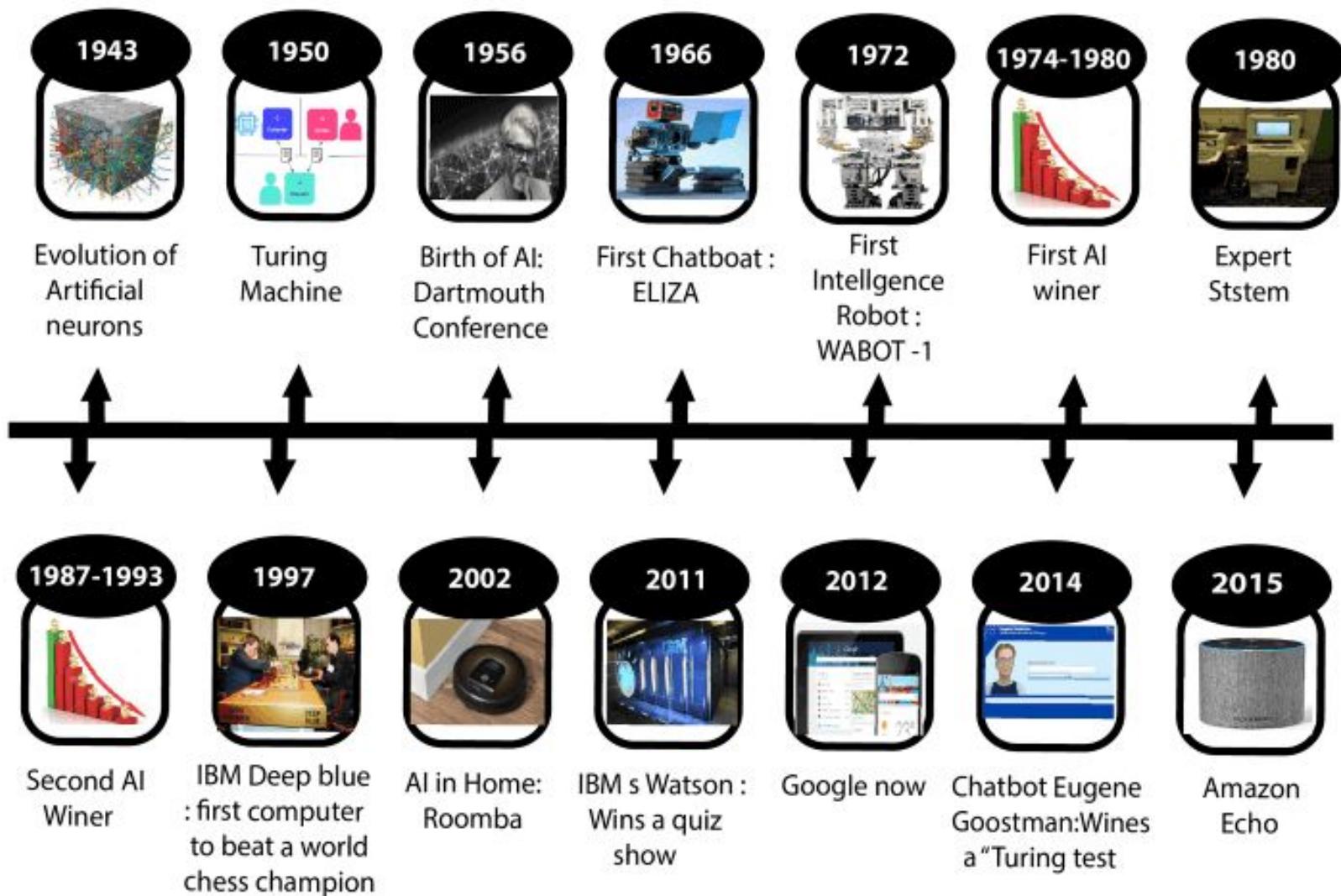


アルファ碁

まず囲碁の打ち方を学習…



History of AI



Related research fields

- Search and optimization
- Knowledge representation
- Reasoning and automatic proving
- Learning and understanding
- Pattern classification / recognition
- Planning
- Problem solving

Artificial vs. Human Intelligence

- Today's computers can do many well-defined tasks (for example, arithmetic operations), much faster and more accurate than human beings.
- However, the computers' interaction with their environment is not very sophisticated yet.
- How can we test whether a computer has reached the general intelligence level of a human being?
- **Turing Test:** Can a computer convince a human interrogator that it is a human?
- But before thinking of such advanced kinds of machines, we will start developing our own extremely simple “intelligent” machines.

The main topics in AI

Artificial intelligence can be considered under a number of headings:

- Search (includes Game Playing).
- Representing Knowledge and Reasoning with it.
- Planning.
- Learning.
- Natural language processing.
- Expert Systems.
- Interacting with the Environment
 - (e.g. Vision, Speech recognition, Robotics)

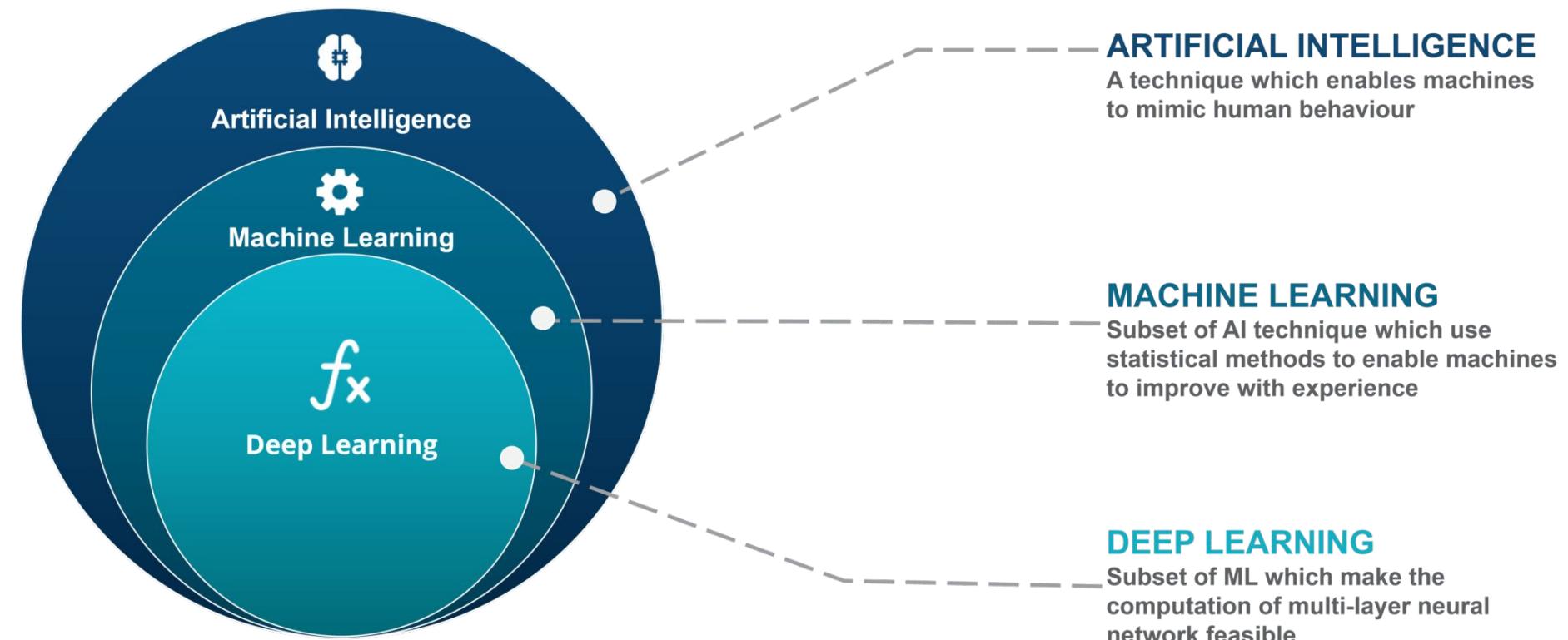
Some Advantages of Artificial Intelligence

- more powerful and more useful computers
- new and improved interfaces
- solving new problems
- better handling of information
- relieves information overload
- conversion of information into knowledge

Constraints

- increased costs
- difficulty with software development - slow and expensive
- few experienced programmers
- few practical products have reached the market as yet.

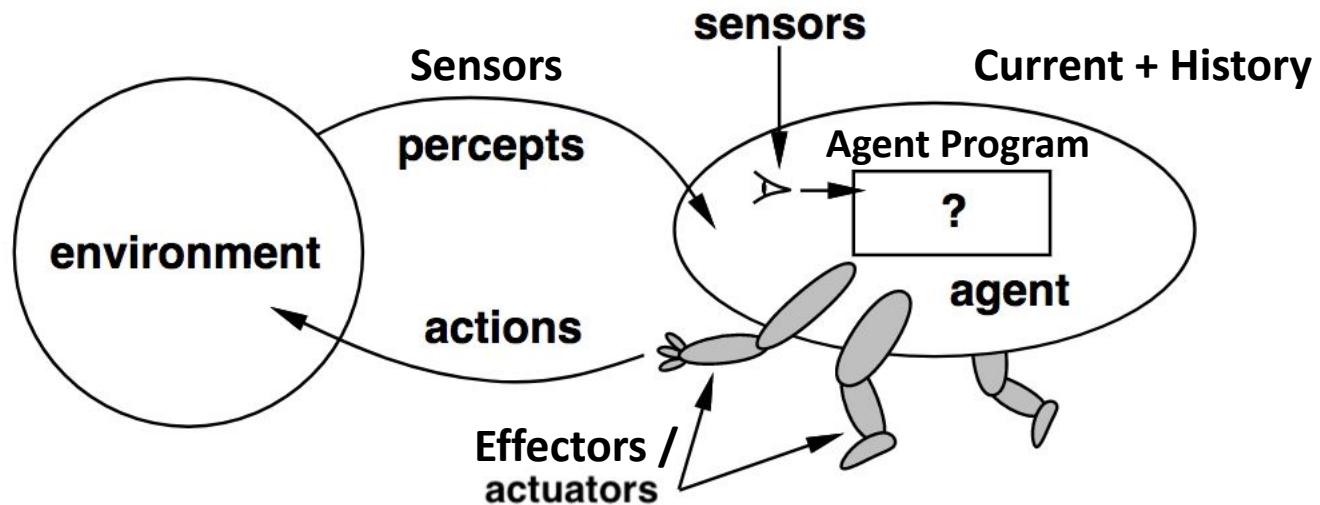
Artificial Intelligence, Machine Learning & Deep Learning



10 most impressive Research Papers around Artificial Intelligence

<https://analyticsindiamag.com/10-impressive-research-papers-around-artificial-intelligence/>

Intelligent Agents



Goals of Agent

High Performance

Optimized Result

Rational (right) Action

Design Issues

Performance

Environment

Actions

Sensors

What is an Intelligent Agent (IA)?

- This agent has some level of autonomy that allows it to perform specific, predictable, and repetitive tasks for users or applications.
- It's also termed as 'intelligent' because of its ability to learn during the process of performing tasks.
- The two main functions of intelligent agents include perception and action. Perception is done through sensors while actions are initiated through actuators.
- Intelligent agents consist of sub-agents that form a hierarchical structure. Lower-level tasks are performed by these sub-agents.
- The higher-level agents and lower-level agents form a complete system that can solve difficult problems through intelligent behaviors or responses.

Characteristics of intelligent agents

- They have some level of autonomy that allows them to perform certain tasks on their own.
- They have a learning ability that enables them to learn even as tasks are carried out.
- They can interact with other entities such as agents, humans, and systems.
- New rules can be accommodated by intelligent agents incrementally.
- They exhibit goal-oriented habits.
- They are knowledge-based. They use knowledge regarding communications, processes, and entities.

The structure of intelligent agents

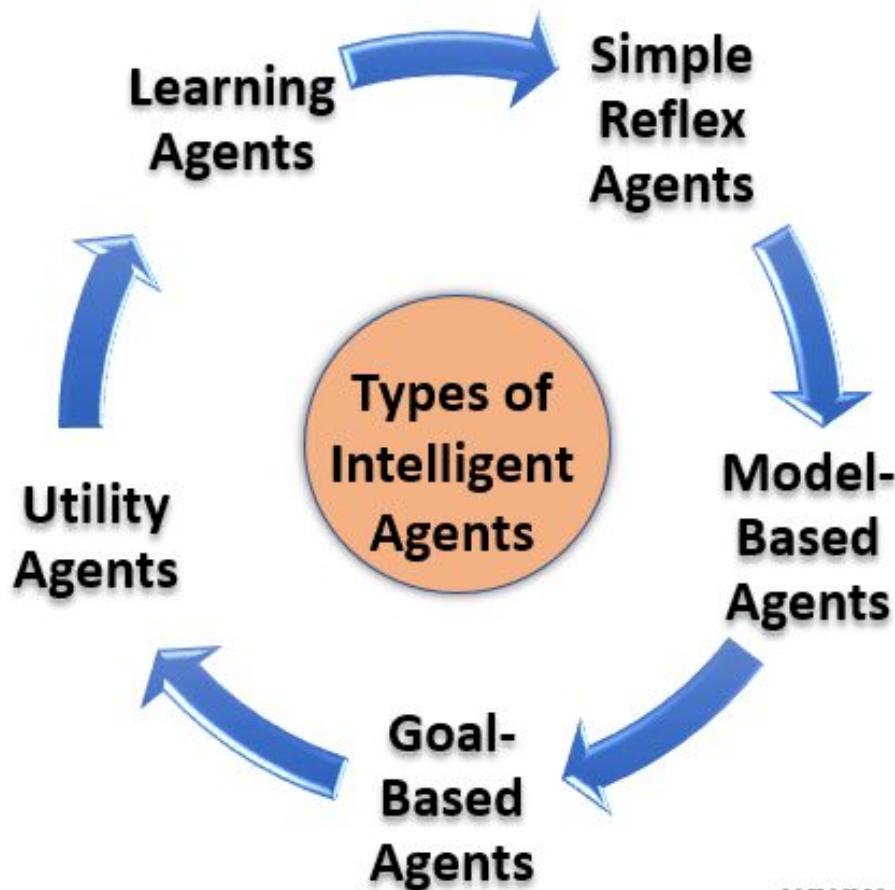
The IA structure consists of three main parts: architecture, agent function, and agent program.

- **Architecture:** This refers to machinery or devices that consists of actuators and sensors. The intelligent agent executes on this machinery. Examples include a personal computer, a car, or a camera.
- **Agent function:** This is a function in which actions are mapped from a certain percept sequence. Percept sequence refers to a history of what the intelligent agent has perceived.
- **Agent program:** This is an implementation or execution of the agent function. The agent function is produced through the agent program's execution on the physical architecture.

Applications of intelligent agents

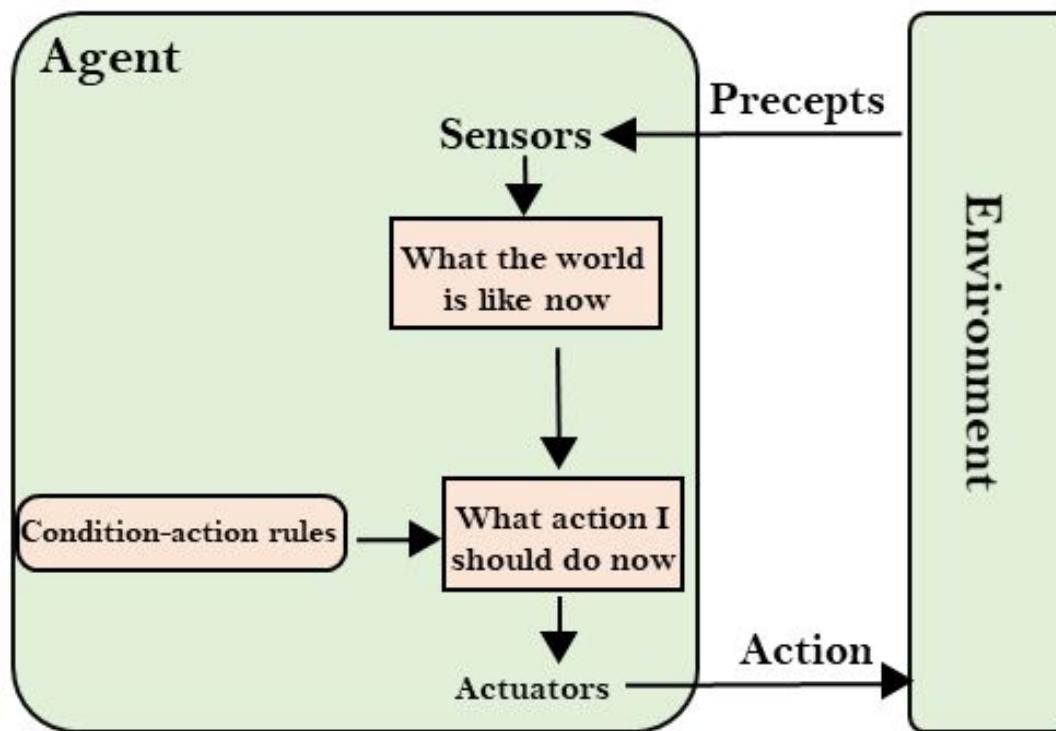
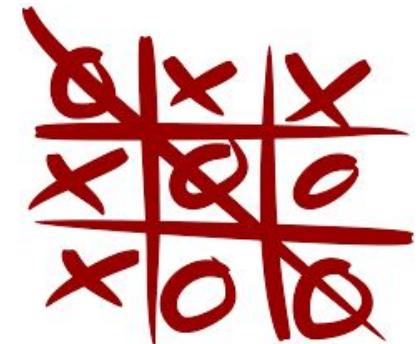
- Information search, retrieval, and navigation
- Repetitive office activities
- Medical diagnosis
- Vacuum cleaning
- Autonomous driving

Types of Intelligent Agents



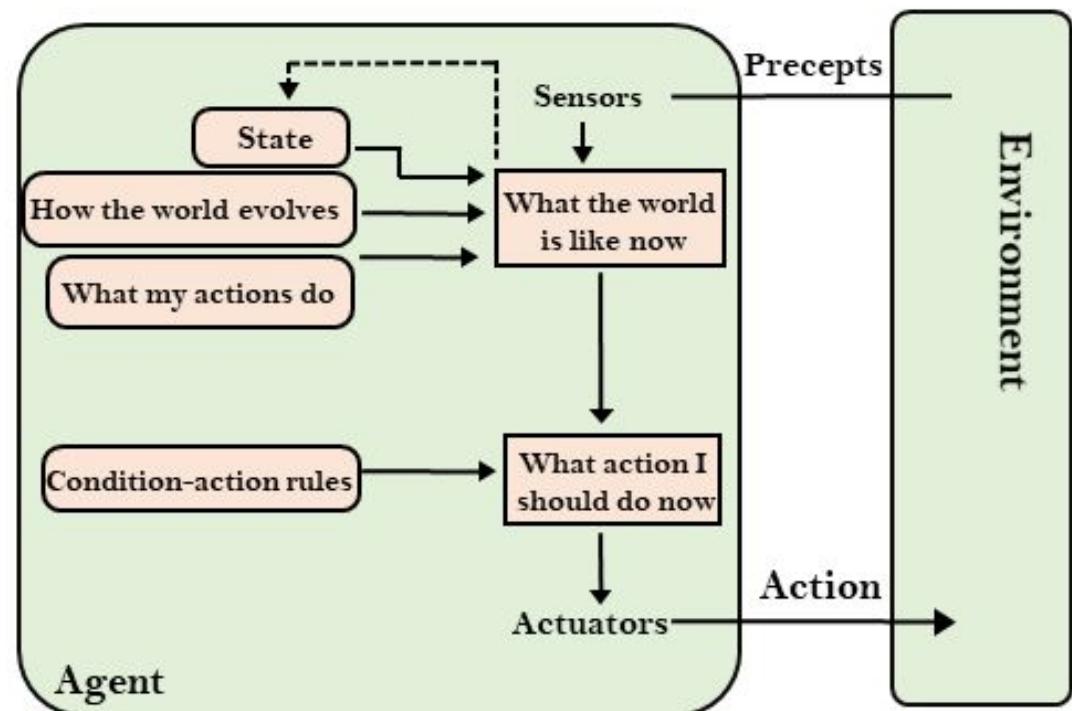
Simple Reflex Agent

- Act only on bases of current perception
- Ignore the rest of percept history
- Based on If-Else rules
- Environment should be fully observable (e.g. tic-tac-toe)



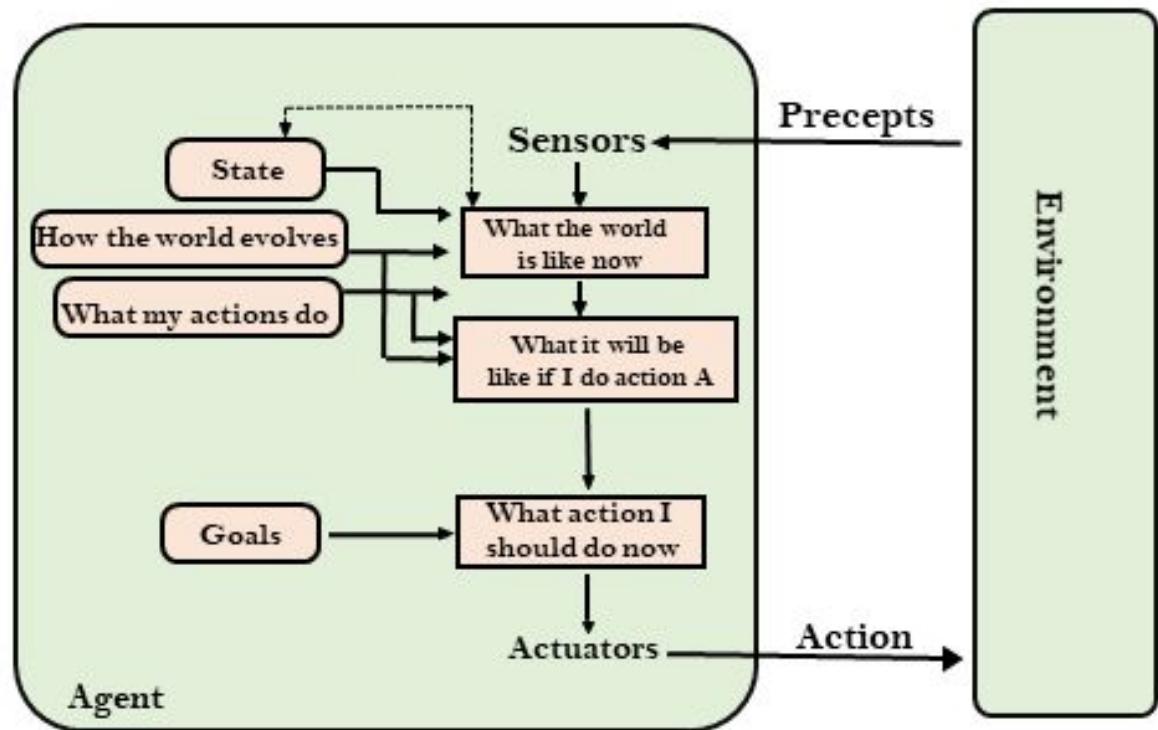
Model based agent

- e.g. Self driving Car
- Partially observable environment
- Store percept history



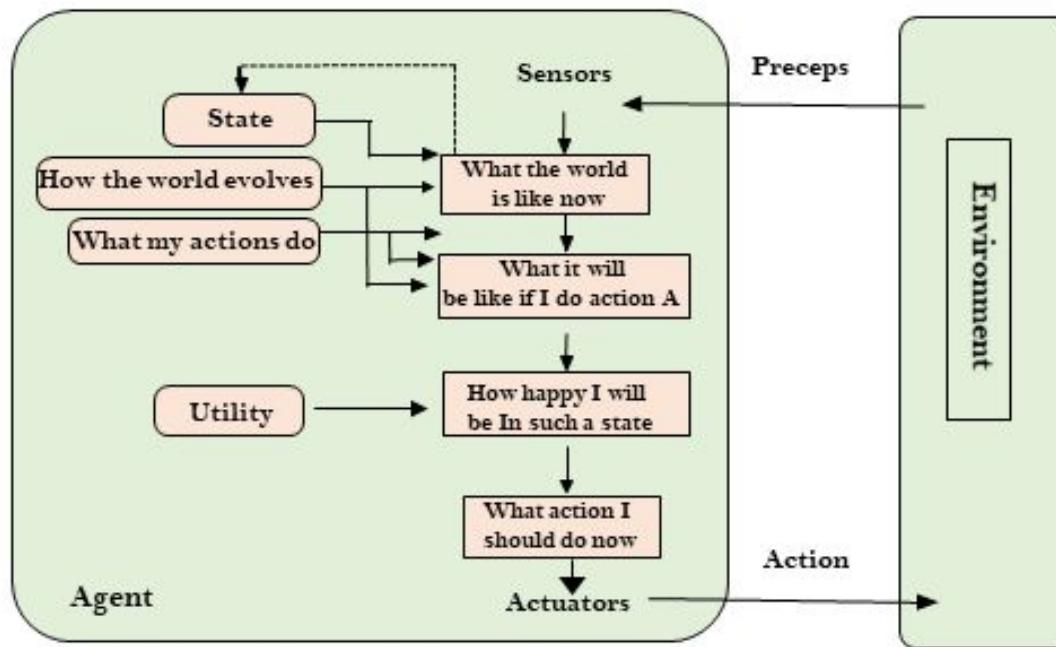
Goal based agent

- Expansion of model based agent
- Desirable situation (goal)
- Searching and planning



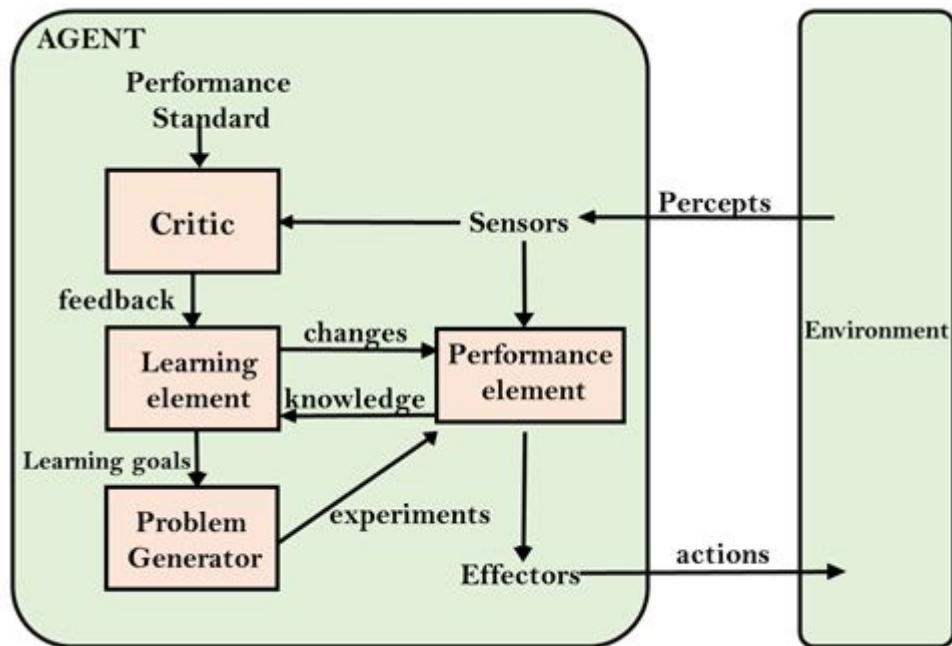
Utility agent

- 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.



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:
 - **Learning element:** It is responsible for making improvements by learning from environment
 - **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
 - **Performance element:** It is responsible for selecting external action
 - **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.



Research Paper (Group Activity)

- **Presentation of a research paper (10 marks)**
 - Make group of 2-3 students
 - Identify a reputed journal/conference research paper based on some topic relevant to the course
 - Get the topic approval
 - Read and understand the paper
- **Assessment:** will be based on how well students are able to understand and explain the algorithm and findings used to solve the problem using AI.

Unit-2

Problem Solving, Problems,
Problem Space & Search

Introduction to Problem and its solution

- To build a system to solve a particular problem, we need to do four things:
 1. Define the problem precisely.
 2. Analyze the problem.
 3. Isolate and represent the task knowledge which is necessary to solve the problem.
 4. Choose the best problem-solving technique and apply it to the particular problem.

Examples:

- Chess :-

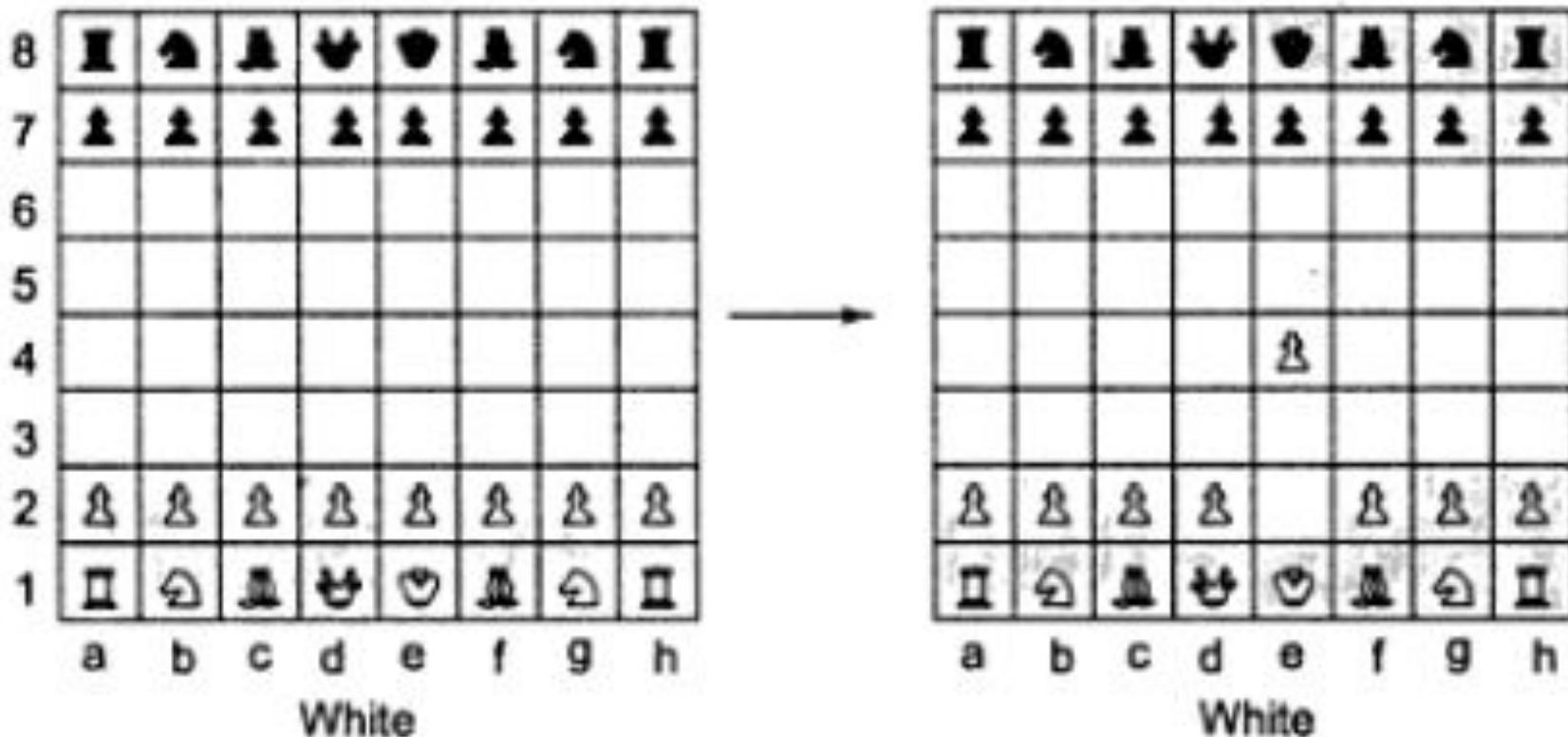


Fig. 2.1 One Legal Chess Move

- For the problem “Play Chess”, it is fairly easy to provide a formal and complete problem description.
- 8 x 8 chessboard, allowing only legal moves and can’t be back.
- Around 10^{120} possible board positions
 - No person could ever supply a complete set of such rules.
 - No program could easily handle all those rules.

- The state space representation forms the basis of most of the AI methods and its structure of problem solving in two important ways:
 - It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
 - It permits us to define the process of solving a particular problem as a combination of known techniques and search.

- Water Jug Problem:-

“You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?”



4-gallon water jug



3-gallon water jug

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

Fig. 2.3 Production Rules for the Water Jug Problem

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

Fig. 2.4 One Solution to the Water Jug Problem

Production Systems

- A production system consists of :
 - **A set of rules**, each consisting of a left side (a pattern) which determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
 - One or more **knowledge/databases** that contain whatever information is appropriate for the particular task.
 - A **control strategy** that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
 - A **rule applier**.

Problem Characteristics

- In order to choose the most appropriate method or combination of methods for a particular problem, it is necessary to analyze the problem along several key dimensions:

Is the problem decomposable into a set of (nearly) independent smaller or easier subproblems?

Can solution steps be ignored or at least undone if they prove unwise?

Is the problem's universe predictable?

Is a good solution to the problem obvious without comparison to all other possible solutions?

Is the desired solution a state of the world or a path to a state?

Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain the search?

Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

Is the problem decomposable into a set of (nearly) independent smaller or easier subproblems?

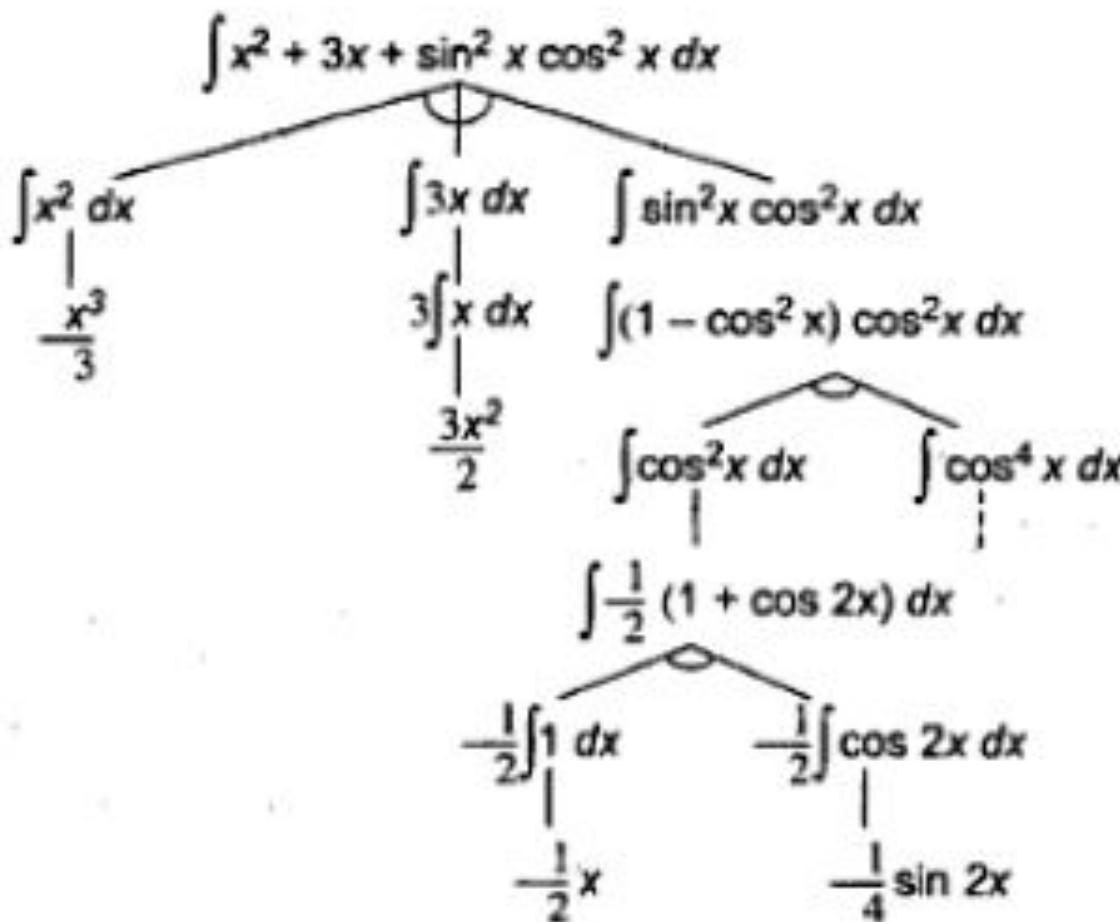
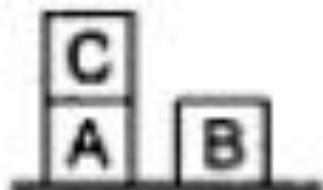


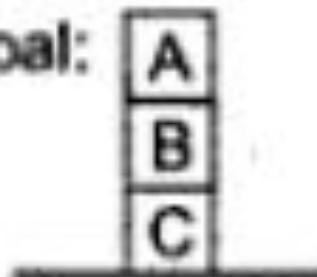
Fig. 2.9 A Decomposable Problem

Start:



ON(C,A)

Goal:



ON(B,C) and ON(A,B)

Fig. 2.10 *A Simple Blocks World Problem*

Can solution steps be ignored or at least undone if they prove unwise?

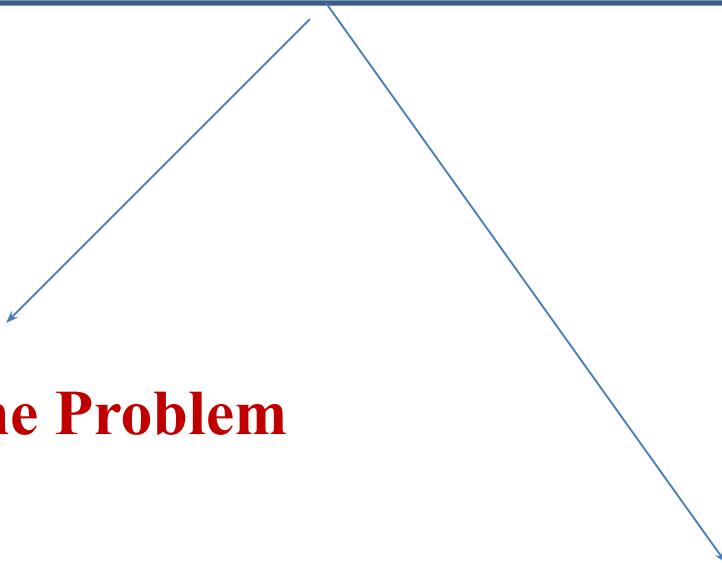
Start	Goal
2 8 3	1 2 3
1 6 4	8 4
7 5	7 6 5

Fig. 2.12 *An Example of the 8-Puzzle*

These three problems – theorem proving, 8-puzzle and chess – illustrate the differences between three important classes of problems:

- **Ignorable** (eg theorem proving), in which solution steps can be ignored.
- **Recoverable** (eg. 8-puzzle), in which solution steps can be undone.
- **Irrecoverable** (eg. Chess), in which solution steps cannot be undone.

Is the problem's universe predictable?



Certain Outcome Problem

Uncertain Outcome Problem

Is a good solution to the problem obvious without comparison to all other possible solutions?

- This means – absolute or relative solution?

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

Is Marcus alive? ----- answer this question?

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

OR

7. It is now 1991 A.D.	axiom 7
5. All Pompeians died in 79 A.D.	axiom 5
11. All Pompeians are dead now.	7, 5
2. Marcus was a Pompeian.	axiom 2
12. Marcus is dead.	11, 2

Fig. 2.13 Two Ways of Deciding That Marcus Is Dead

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

Fig. 2.14 An Instance of the Traveling Salesman Problem



Fig. 2.15 One Path among the Cities

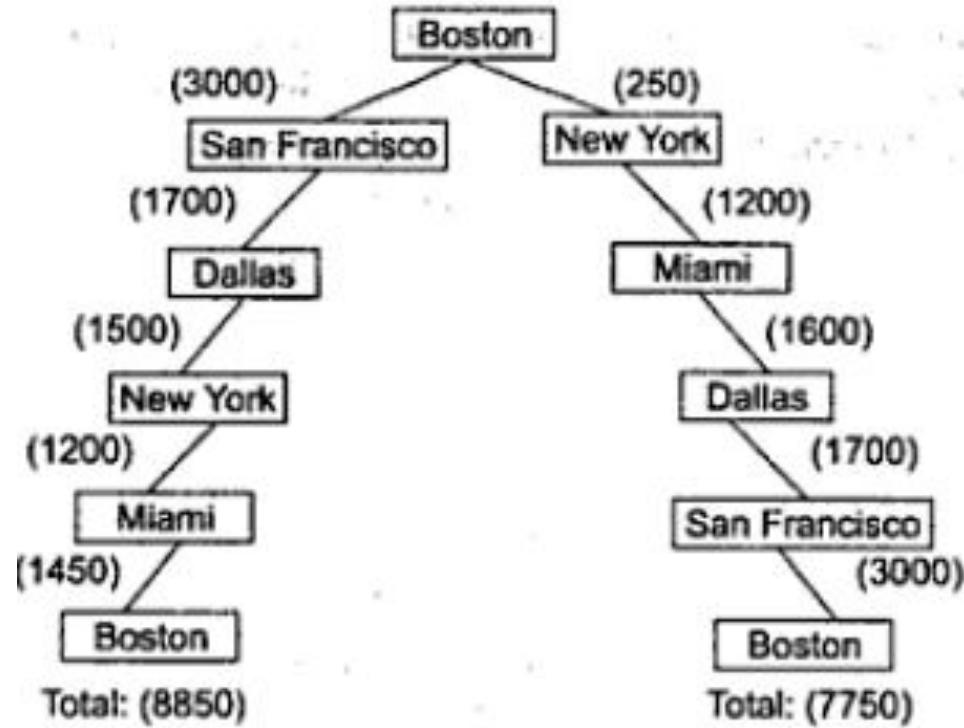


Fig. 2.16 Two Paths Among the Cities

Is the desired solution a state of the world or a path to a state?

- Natural Language Understanding:

“The **bank president** ate a **dish** of **pasta salad** with the **fork**. ”

There are several components of this sentence, each of which, in isolation, may have more than one meaning or interpretation. But the components must form a coherent whole, and so they constrain each other’s interpretations.

- Water Jug Problem:

Here it is not sufficient to report that we have solved the problem and that the final state is (0,2).

For this kind of problem, what we really must report is not the final state but the path that we found to that state.

Thus a statement of a solution to this problem must be a sequence of operations that produces the final state.

Is a large amount of knowledge absolutely required to solve the problem or is knowledge important only to constrain the search?

- Playing a chess
 - knowledge important only to constrain the search
- Will the person vote for republican or democrats?
 - large amount of knowledge absolutely required

Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

- Solitary, in which the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation of the reasoning process.
- Conversational, in which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both.

Production System Characteristics (later)

	Monotonic	Nonmonotonic
Partially commutative	Theorem proving	Robot navigation
Not partially commutative	Chemical synthesis	Bridge

Fig. 2.17 *The Four Categories of Production Systems*

Issues in the Design of Search Programs

- The direction in which to conduct the search (*forward versus backward reasoning*). We can search forward thru the state space from the start state to a goal state or we can search backward from the goal.
- How to select applicable rules (*matching*). Production systems typically spend most of their time looking for rules to apply, so it is critical to have efficient procedures for matching rules against states.
- How to represent each node of the search process (the *knowledge representation problem* and the frame problem).

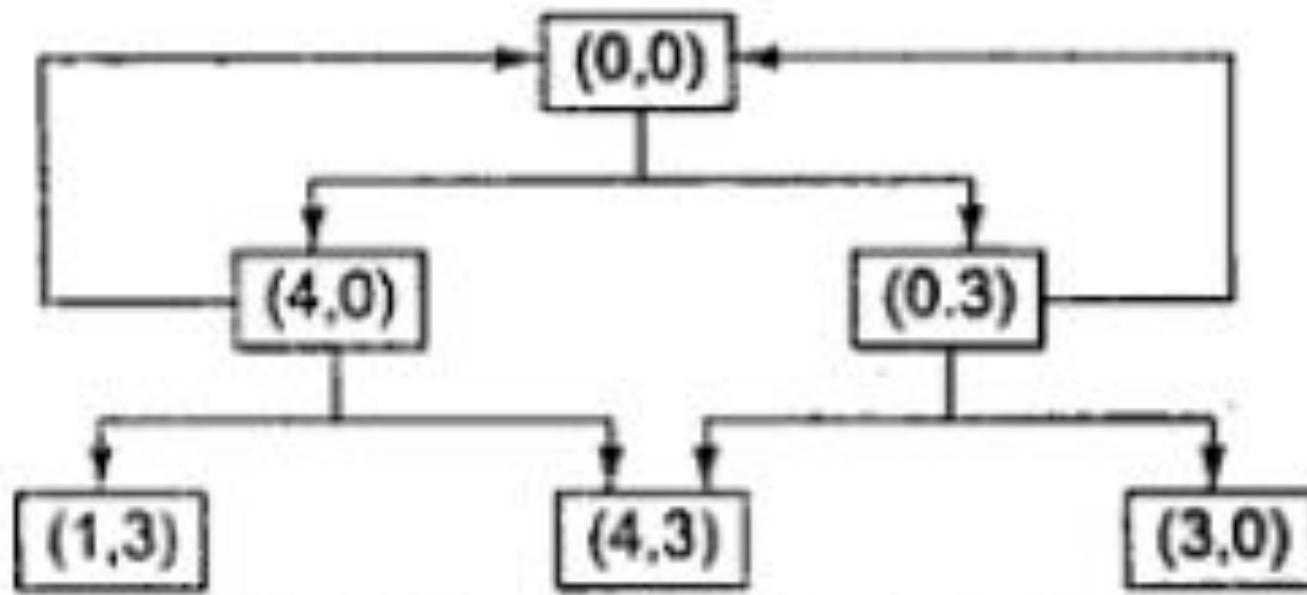


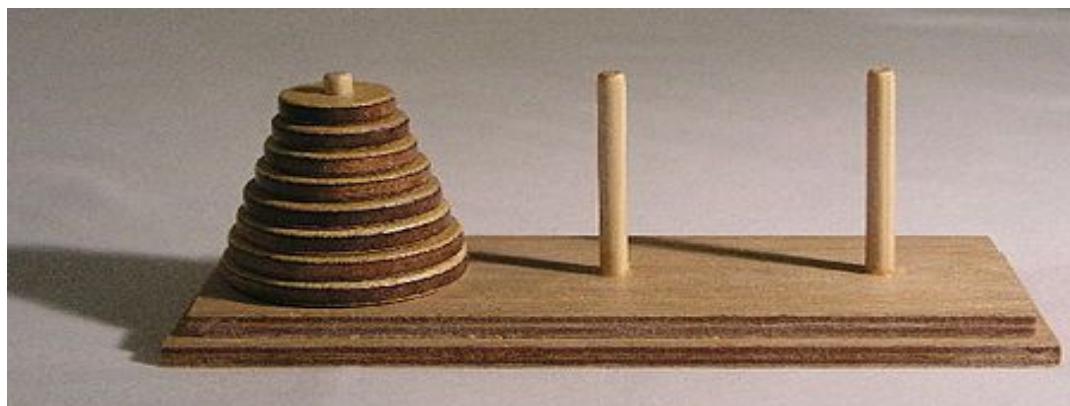
Fig. 2.19 A Search Graph for the Water Jug Problem

Additional Problems

- Missionaries and Cannibals problem



- Tower of Hanoi Problem



Additional Problems

- Monkey and Banana Problem



- Crypt arithmetic Problem

$$\begin{array}{r} \text{BASE} \\ + \text{BALL} \\ \hline \text{GAMES} \end{array} \quad \longrightarrow \quad \begin{array}{|c|c|} \hline \text{B} & 7 \\ \hline \text{A} & 4 \\ \hline \text{S} & 8 \\ \hline \text{E} & 3 \\ \hline \text{L} & 5 \\ \hline \text{G} & 1 \\ \hline \text{M} & 9 \\ \hline \end{array}$$

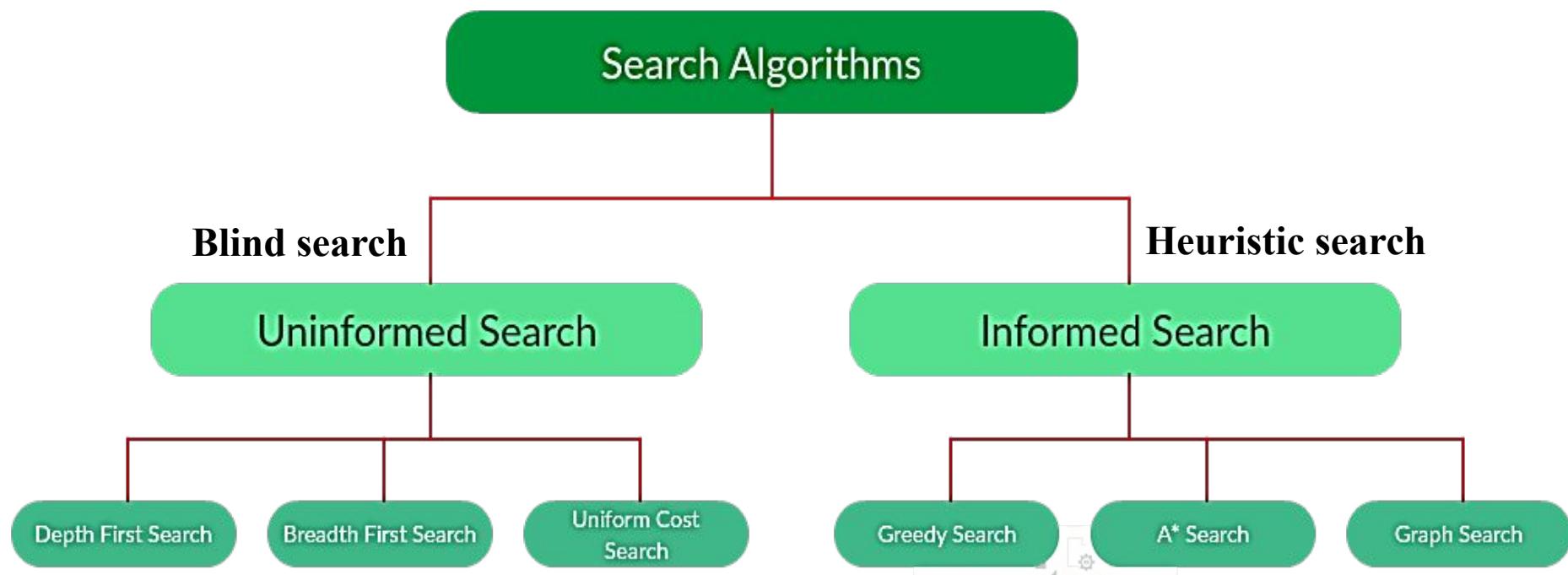
Unit-3

Search Techniques

Search Algorithms in AI

- Artificial Intelligence is the study of building agents that act rationally.
 - Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.
- A search problem consists of:
 - A **State Space**. Set of all possible states where you can be.
 - A **Start State**. The state from where the search begins.
 - A **Goal Test**. A function that looks at the current state returns whether or not it is the goal state.
- The Solution to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state.
- This plan is achieved through search algorithms.

Types of Search Algorithms



Uninformed vs. informed search

- **Uninformed search strategies**
 - Also known as “blind search,” uninformed search strategies use no information about the likely “direction” of the goal node(s)
 - Uninformed search methods: Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- **Informed search strategies**
 - Also known as “heuristic search,” informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
 - Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*

Uninformed/Blind search strategies

- Topics to be covered...
 - Breadth First Search
 - Depth First Search
 - Uniform Cost Search
 - Depth limited search
 - Iterative Deepening Depth First Search

Algorithm: Breadth-First Search

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

Algorithm: Depth-First Search

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



A, B, C, D, E, F

BFS



A, B, D, C, E, F

DFS



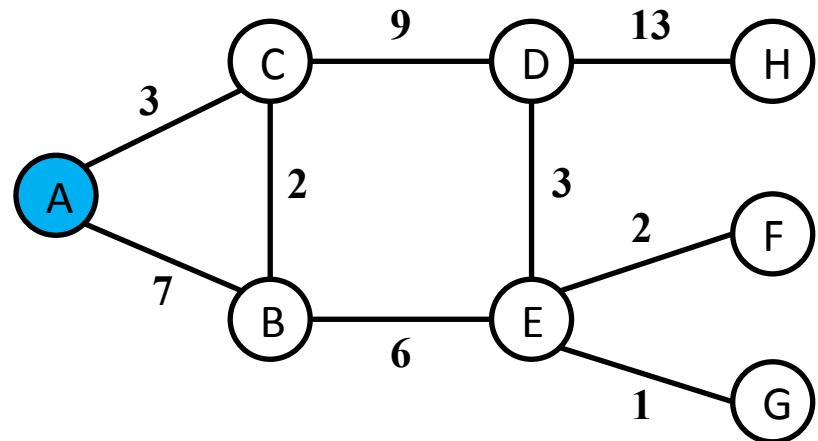
- Advantages of Breadth-First Search:
 - BFS will not get trapped exploring a blind path. This contrasts with DFS, which may follow a single, unfruitful path for a long time.
 - If there is a solution, then BFS is guaranteed to find it.
- Advantages of Depth-First Search:
 - DFS requires less memory since only the nodes of current path are stored. This contrasts with BFS, where all of the tree that has so far been generated must be stored.
 - DFS may find a solution without examining much of the search space at all. DFS can stop when one of them is found.

Example : TSP (Traveling Salesperson Problem)

Uniform Cost Search

(uninformed search, backtracking)

- Used for **weighted** graph/tree
- Objective: find path to goal state with **lowest cumulative cost**
- Node expansion is based on path cost
- Priority queue** is used for implementation (high priority to minimum cost)
- Advantage:** optimal solution
- Disadvantage:** may have infinite loop



If **G** is goal state:

A -> C -> B -> E -> G
TOTAL COST: $3 + 2$

$$+ 6 + 1 = 12$$

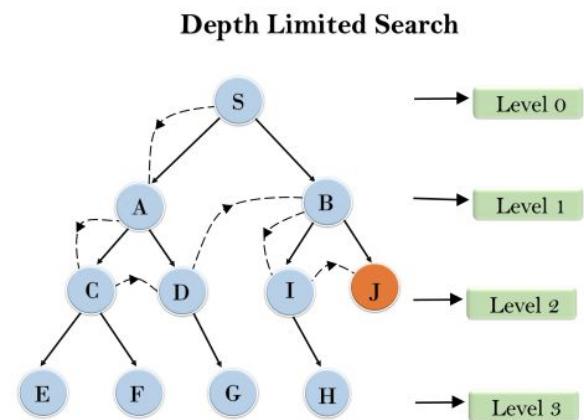
If **H** is goal state:

A -> C -> B -> E -> G -> E -> F -> E -> D -> H

$$\text{TOTAL COST: } 3 + 2 + 6 + 1 + 1 + 2 + 2 + 3 + 13$$

Depth-Limited Search

- 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.



Bidirectional Search

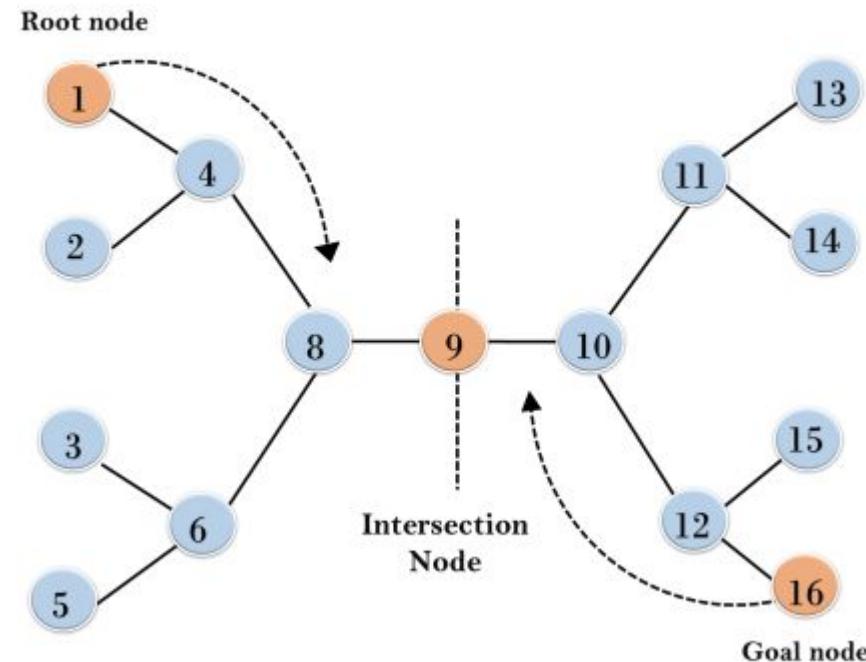
- Bidirectional search algorithm runs two simultaneous searches, one from initial state called as forward-search and other from goal node called as backward-search, to find the goal node.
- Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from 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

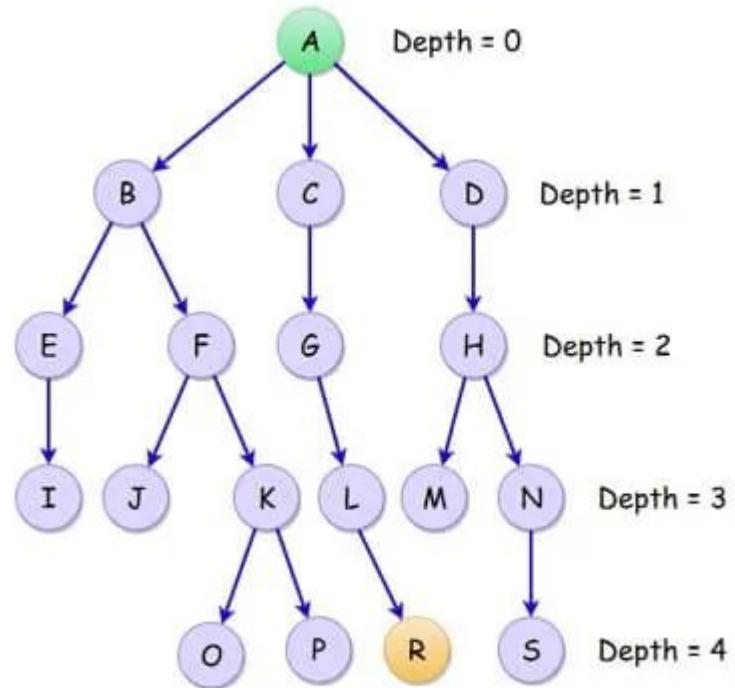
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- In bidirectional search, one should know the goal state in advance.



Iterative Deepening Depth-First Search

- The algorithm do a limited depth-first search up to a fixed **“limited depth”**.
- Then we keep on incrementing the depth (**iteratively**) limit by iterating the procedure unless we have found the **goal node** or have traversed the whole tree whichever is earlier.



DEPTH LIMITS

0

1

2

3

4

IDDFS

A

A B C D

A B E F C G D H

A B E I F J K C G L D H M N

A B E I F J K O P C G L R D H M N S

Informed/Heuristic search strategies

Topics to be covered...

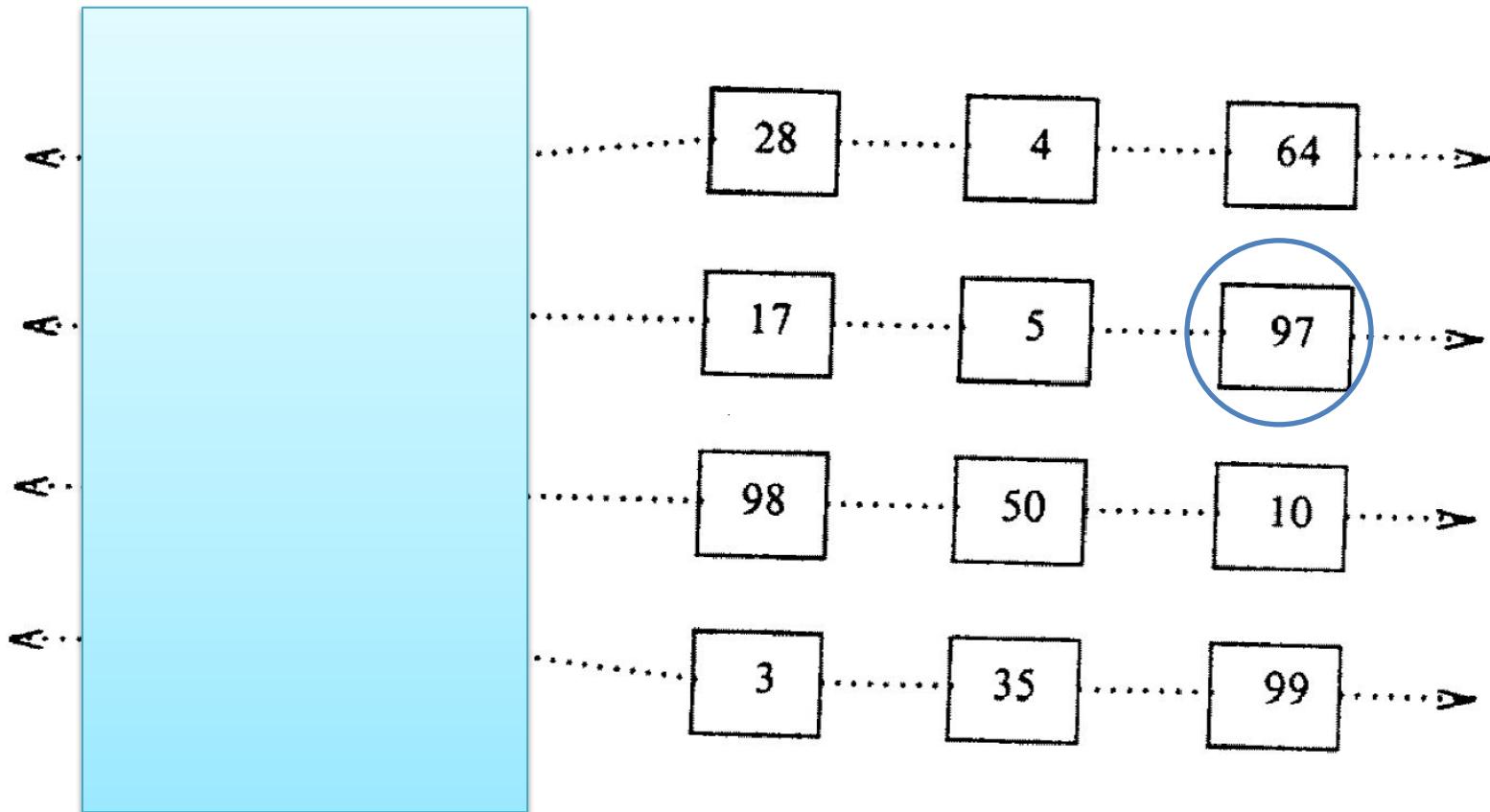
- Introduction to heuristic functions
- Generate-and-test
- Best-first search
 - Greedy Best First Search
 - A* Algorithm
- Problem reduction
 - General Approach
 - AO* Algorithm
- Hill Climbing
 - Simple Hill Climbing
 - Steepest-Ascent Hill Climbing
 - Simulated Annealing
- Local Beam Search

Heuristic Search

- A **heuristic function** or simply **heuristic** is a technique which improves the efficiency of a search process, possibly by sacrificing claims of completeness.

OR
- A **heuristic** is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.
- Heuristics are tour guides.

Example (heuristic?)



- Search for block number $97 = 97 \bmod 4 = 1$.
- So block number 97 is available in bl no. 1

Example: Heuristic Function

- Consider the following 8-puzzle problem where we have a start state and a goal state.
- Our task is to slide the tiles of the current/start state and place it in an order followed in the goal state.
- There can be four moves either left, right, up, or down.
- There can be several ways to convert the current/start state to the goal state, but, we can use a **heuristic function $h(n)$** to solve the problem more efficiently.

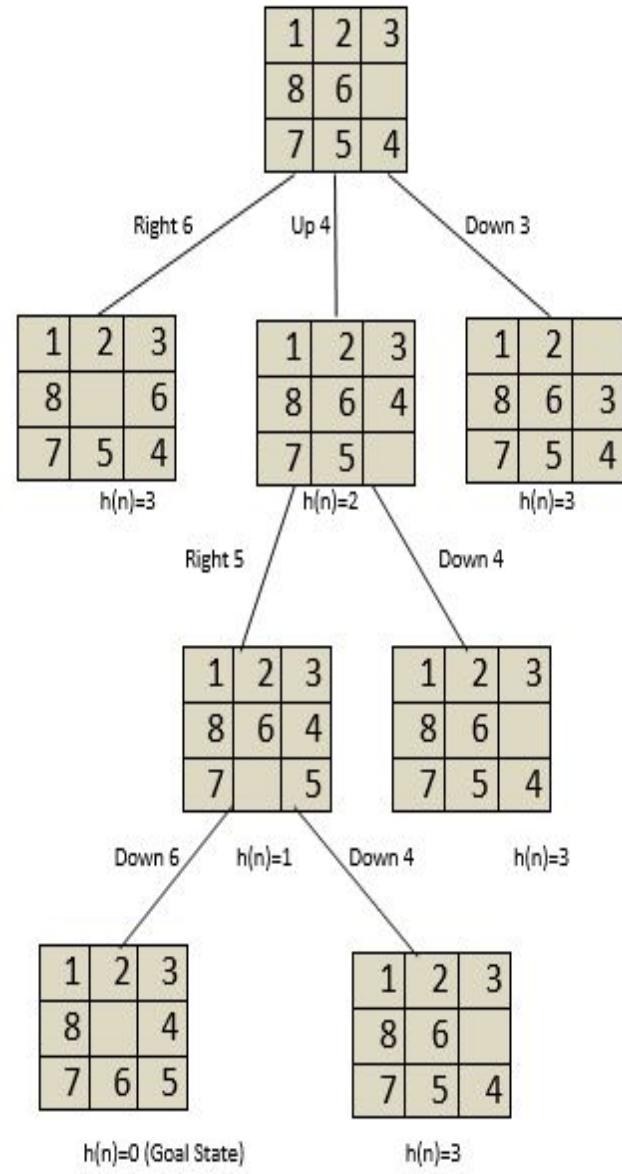
1	2	3
8	6	
7	5	4

Start State

1	2	3
8		4
7	6	5

Goal State

- Two classes of local search algorithms exist.
- The first one is that of greedy or **non-randomized** algorithms. These algorithms proceed by changing the current assignment by always trying to decrease (or at least, non-increase) its cost.
- The main problem of these algorithms is the possible presence of **plateaus**, which are regions of the space of assignments where no local move decreases cost.
- The second class of local search algorithm have been invented to solve this problem. They escape these plateaus by doing random moves, and are called **randomized** local search algorithms.



- A heuristic function, or simply a heuristic, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow.
- A heuristic function, is a function that calculates an approximate cost to a problem (or ranks alternatives).
- For example the problem might be finding the shortest driving distance to a point. A heuristic cost would be the straight line distance to the point. It is simple and quick to calculate, an important property of most heuristics. The true distance would likely be higher as we have to stick to roads and is much harder to calculate.
- Well designed heuristic functions can play an important part in efficiently guiding a search process toward a solution.

- Examples of Heuristic Functions:
 - *Chess* : the material advantage of our side over the opponent
 - *Traveling Salesperson* : the sum of the distances so far
 - *Tic-tac-toe* : 1 for each row in which we could win and in which we already have one piece plus 2 for each such row in which we have two pieces.

Generate-and-test

(Heuristic Search Techniques)

Generate-and-test

(Heuristic technique, DFS with backtracing)

- This is the simplest of all approaches. It consists of the following steps:

Algorithm: Generate-and-test

1. Generate a possible solution.
2. Test to see if this is actually a solution.
3. Quit if a solution has been found.
Otherwise, return to step 1.

- Acceptable for simple problems.
- Inefficient for problems with large space.
- **Exhaustive** generate-and-test.
- **Heuristic** generate-and-test: not consider paths that seem unlikely to lead to a solution.

Two digit – 3 parts PIN number

e.g. **135379**

00 00 00

00 00 01

.....

.....

EXAMPLE:

Exhaustive: Will take $(100)^3$

Find good heuristic ???
(domain knowledge available)

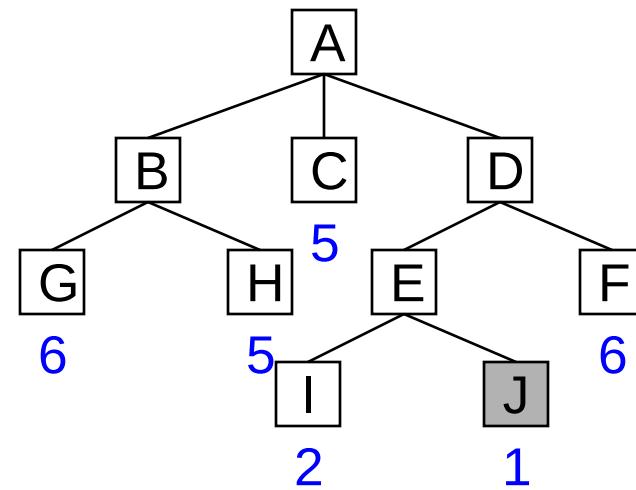
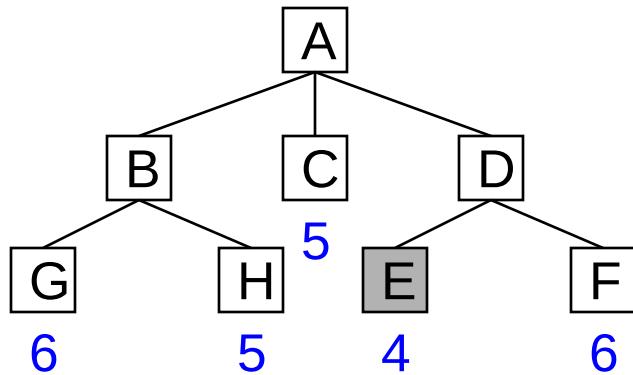
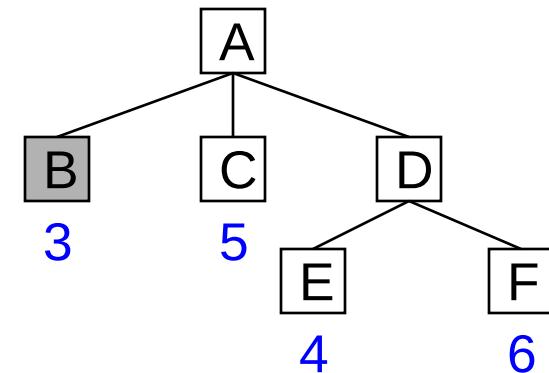
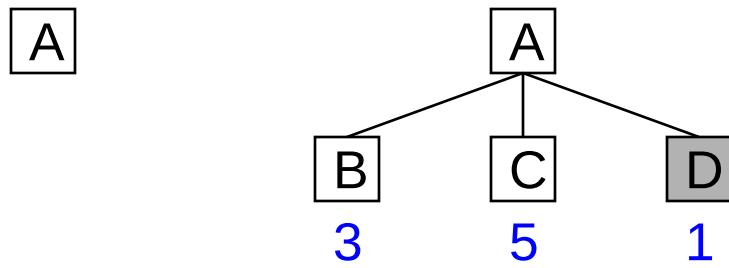
For example, the PIN number uses only **prime** numbers
e.g. $(25)^3$

Greedy Best-First Search

(BFS+DFS, Heuristic Search Techniques)

Best-first search

- Best-first search – way of combining the advantages of BFS and DFS into a single method.
- Depth-first search: not all competing branches having to be expanded.
- Breadth-first search: not getting trapped on dead-end paths.
 - ⇒ Combining the two is to follow a single path at a time, but switch paths whenever some competing path look more promising than the current one.



- **OPEN**: nodes that have been generated, but have not examined.
 - This is organized as a **priority queue**.
- **CLOSED**: nodes that have already been examined.
 - Whenever a new node is generated, **check** whether it has been **generated before**.

Algorithm: Best-First Search

1. $\text{OPEN} = \{\text{initial state}\}.$
2. Loop until a goal is found or there are no nodes left in $\text{OPEN}:$
 - (a) Pick the best node in OPEN
 - (b) Generate its successors
 - (c) For each successor do:
 - (i) If it has not been generated before, evaluate it, add it to OPEN , and record its parent.
 - (ii) If it has **been generated** before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

- We are in need of a heuristic function f' that estimates the merits of each node we generate.

$$f' = g + h'$$

- where function g is a measure the cost of getting from the initial state to the current node, and
- Function h' is an estimate of the additional cost of getting the current node to goal state.

Algorithm: Best-First Search

OPEN = {initial state}.

Loop until a goal is found or there are no nodes left in OPEN:

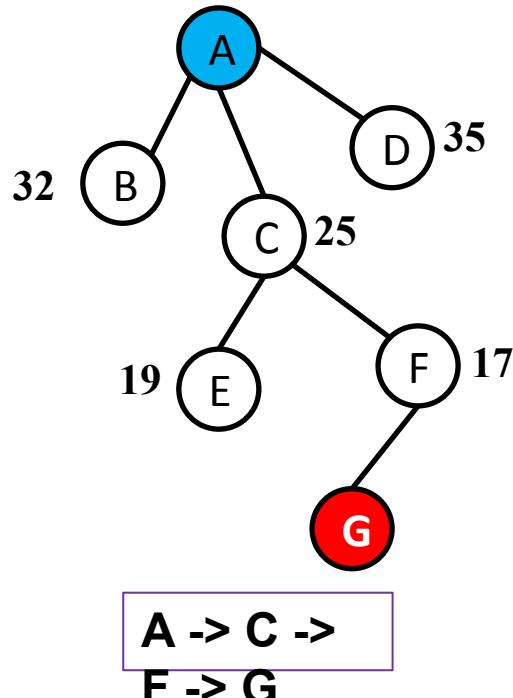
(a) Pick the best node in OPEN

(b) Generate its successors

(c) For each successor do:

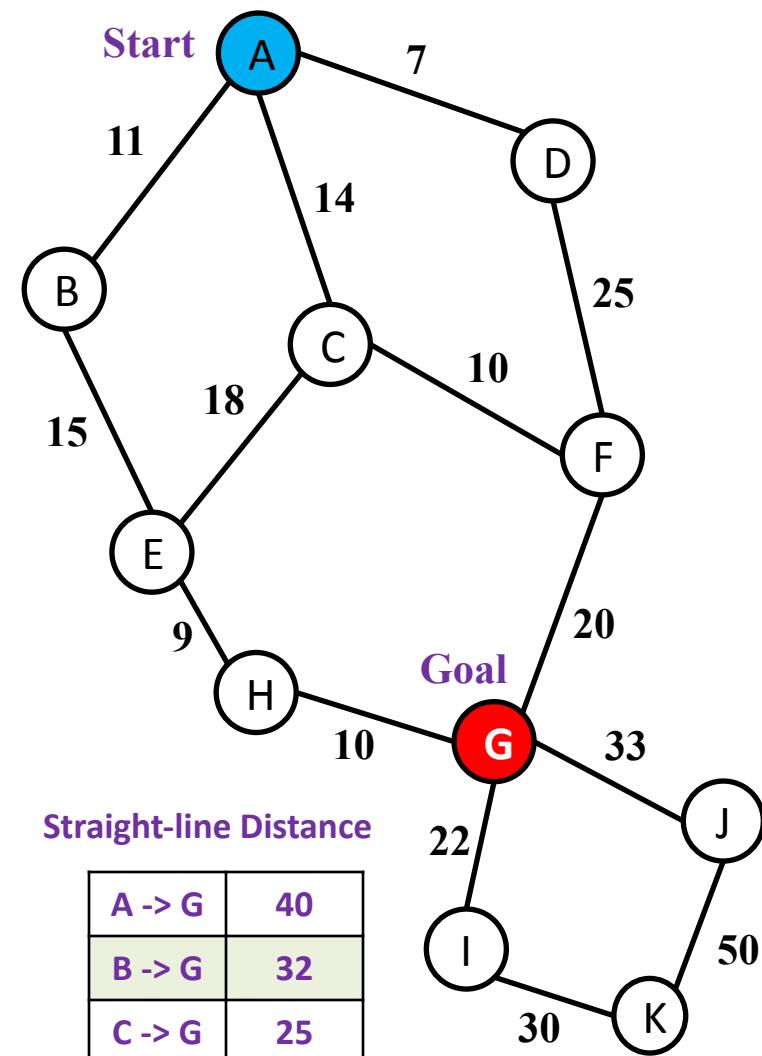
(i) If it has **not been generated** before, evaluate it, add it to OPEN, and record its parent.

(ii) If it has **been generated** before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.



Distance between two points $P(x_1, y_1)$ and $Q(x_2, y_2)$ is given by:

$$d(P, Q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad \{\text{Distance formula}\}$$



A* Algorithm

(Heuristic Search Techniques)

A* Algorithm

- The best-first search algorithm is a simplification of A* algorithm, which was first presented by Hart [1968, 1972].
- This algorithm uses the same f' , g and h' functions as well as the lists OPEN and CLOSED.

- This algorithm was given by **Hart ,Nilsson & Rafael** in 1968.
- A* is a best first search $f(n) = g(n) + h(n)$
where , $g(n)$ = *sum of edge costs from start to n.*
 $h(n)$ = *estimate of lowest cost path from n to goal*
- $f(n)$ = *actual distance so far + estimated distance remaining.*

OPEN $\square(S)$; **Parent** (S) \square NILL; **CLOSED** \square NILL

While $\text{OPEN} \neq \text{NILL}$

pick best(lowest f value where $f = g + h$) node n and add it to **CLOSED**.

If n is a goal node,

return success (*path p*)

else

Generate the successors of node n .

for each m in successors

case 1: m is not in OPEN and not in CLOSED.

compute $h(m)$

$\text{parent}(m) \square n$

$g(m) = g(n) + k(n, m)$ (where k is the cost from n to m)

$f(m) = g(m) + h(m)$

Add m to OPEN.

case 2: m is in OPEN

if $g(n) + k(n, m) < g(m)$

$\text{parent}(m) \square n$

$g(m) = g(n) + k(n, m)$

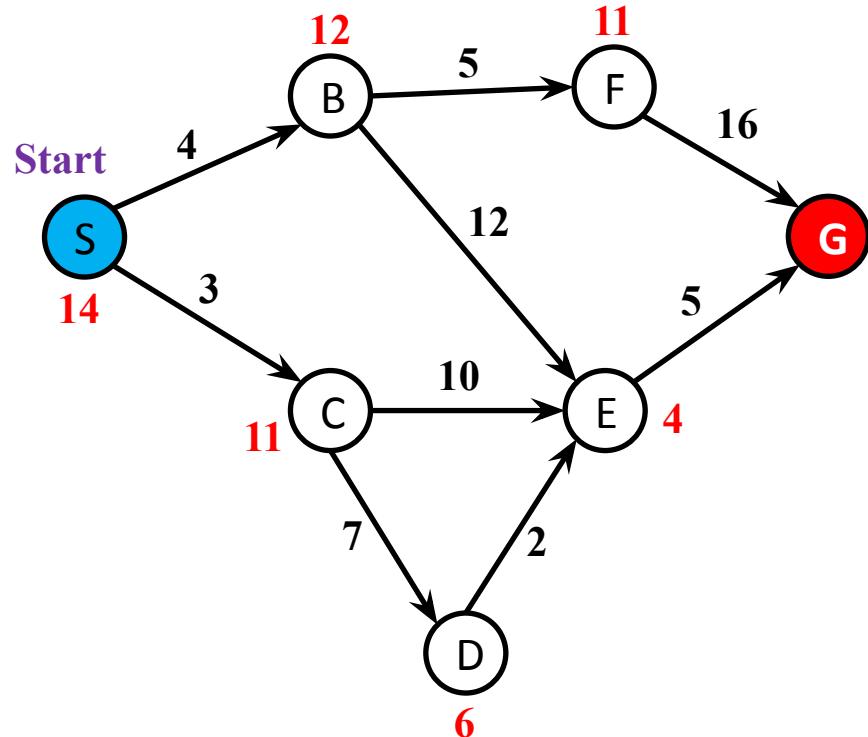
$f(m) = g(m) + h(m)$

case 3: m is in CLOSED

if better path found then like **case 2**,

and propagate improved cost to subtree below m .

Working of A* Algorithm



$$f(N) = g(N) + h(N)$$

Actual cost from
start node to n

estimation cost from
start node to n

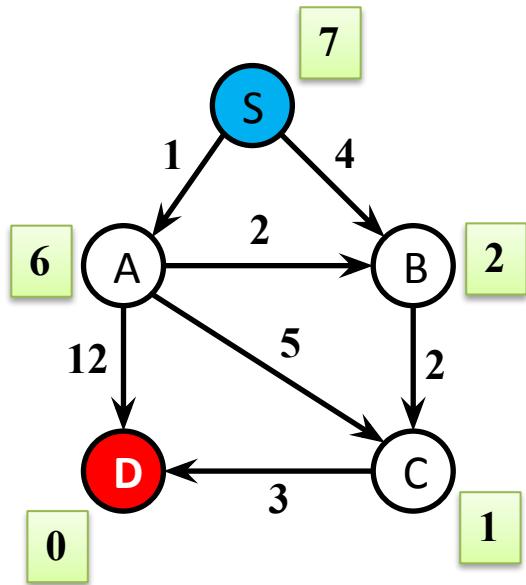
Start:	$S \square B$	$S \square C$
	$4 + 12$	$3 + 11$
	<u>16</u>	<u>14</u>

$SC \square E$	$SC \square D$
$3+10+4$	$3+7+6$
<u>17</u>	<u>16</u>

$SB \square F$	$SB \square E$
$5+4+11$	$4+12+4$
<u>20</u>	<u>20</u>

$SCD \square E$	$SCDE \square G$
$3+7+2+4$	$3+7+2+5+0$
<u>16</u>	<u>17</u>

A* Example



$$1. S \square A = 1 + 6 = 7$$

$$5. S \square A \square B = 1 + 2 + 2 = 5$$

$$6. S \square A \square C = 1 + 5 + 1 = 7$$

$$7. S \square A \square D = 1 + 12 + 0 = 13$$

$$2. S \square B = 4 + 2 = 6$$

$$3. S \square B \square C = 4 + 2 + 1 = 7$$

$$4. S \square B \square C \square D = 4 + 2 + 3 + 0 = 9$$

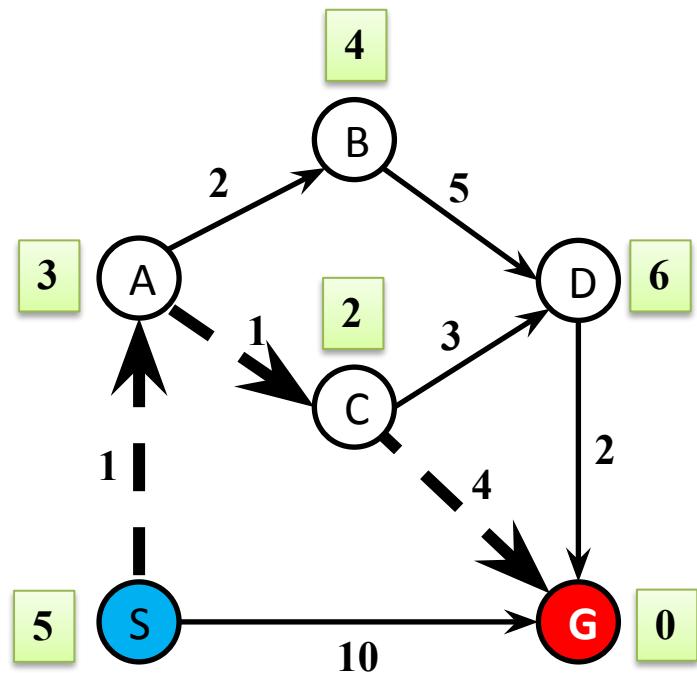
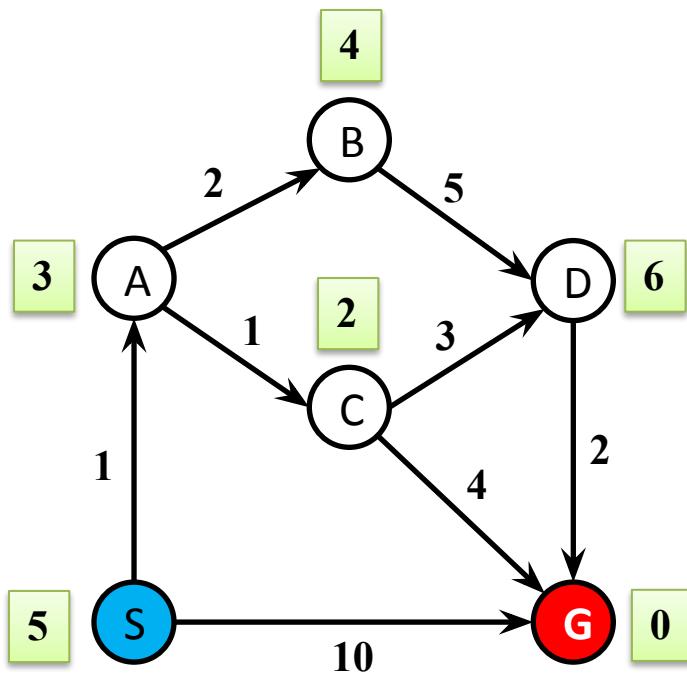
$$8. S \square A \square B \square C = 1 + 2 + 2 + 1 = 6$$

$$9. S \square A \square B \square C \square D = 1 + 2 + 2 + 3 + 0 = 8$$

$$10. S \square A \square C \square D = 1 + 5 + 3 + 0 = 9$$

Best Path: $S \square A \square B \square C \square D = 1 + 2 + 2 + 3 + 0 = 8$

Another A* Example



$$S \square A \square C \square G = 1 + 1 + 4 + 0 = 6$$

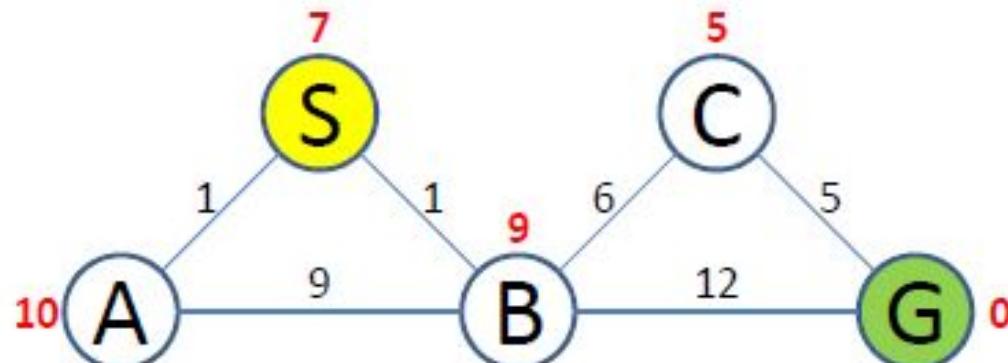
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$

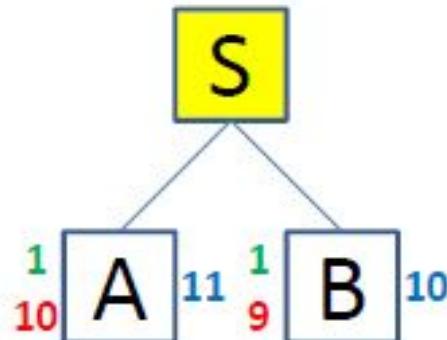
0
7 S 7

QUEUE = *path containing root*

QUEUE: <S>



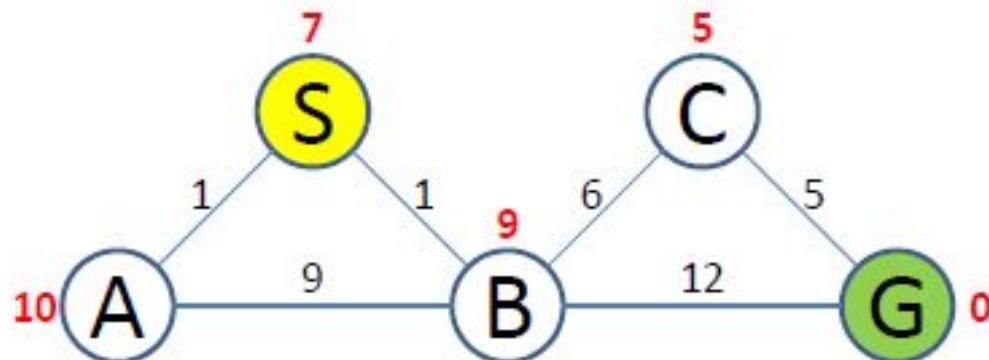
A* algorithm by Example



$f = \text{accumulated path cost} + \text{heuristic}$

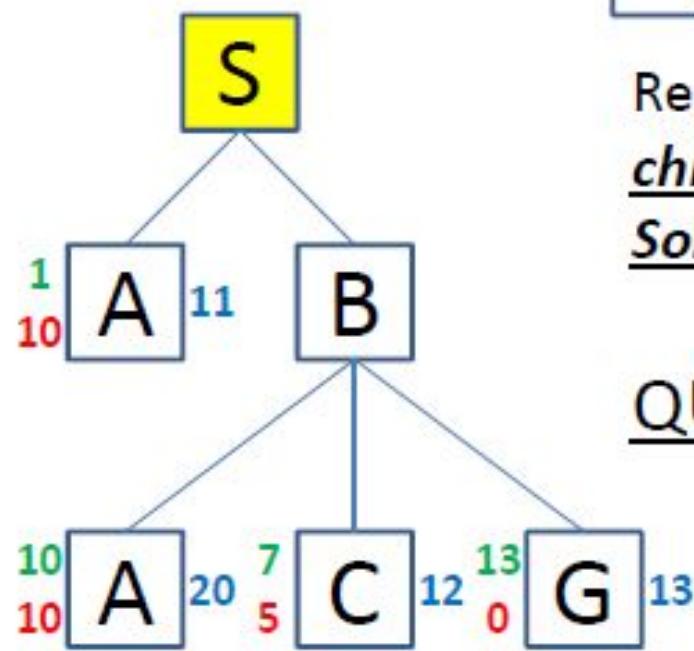
Remove first path, Create paths to all children, Reject loops and Add paths.
Sort QUEUE by f

QUEUE: <SB,SA>



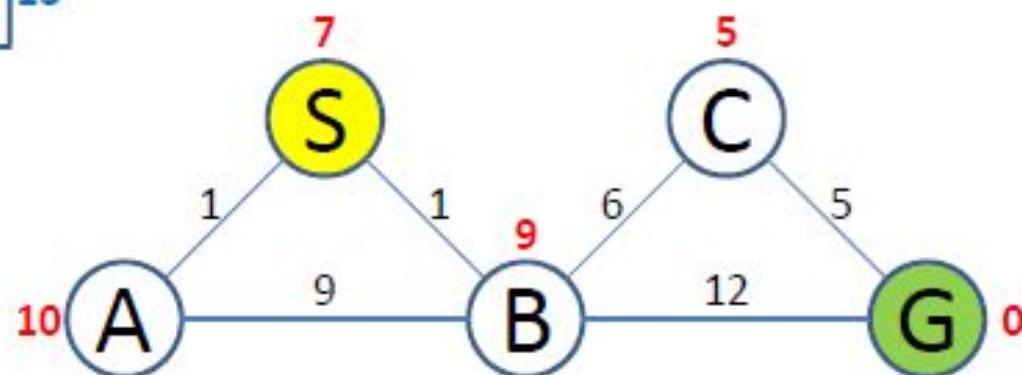
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



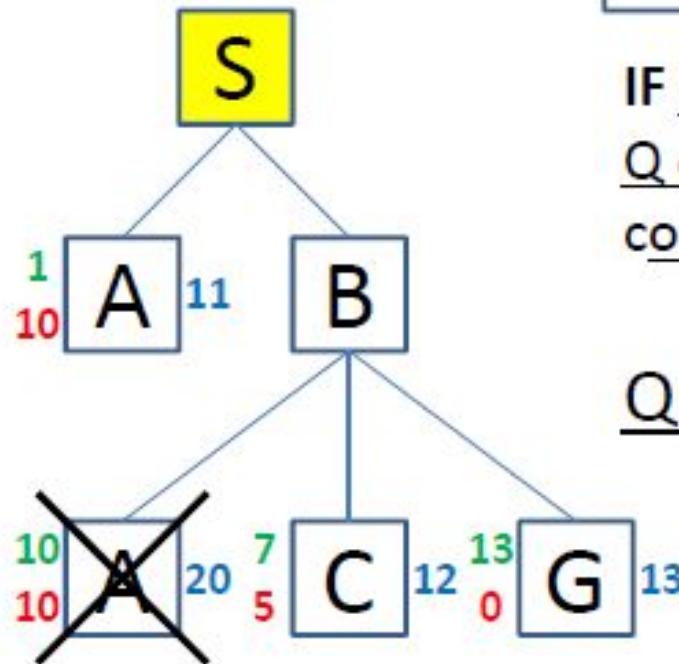
Remove first path, Create paths to all children, Reject loops and Add paths.
Sort QUEUE by f

QUEUE: <SA, SBC, SBG, SBA>



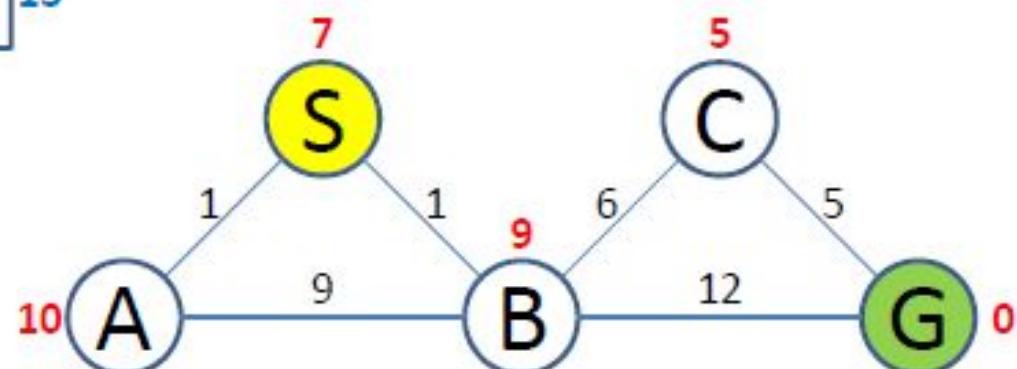
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



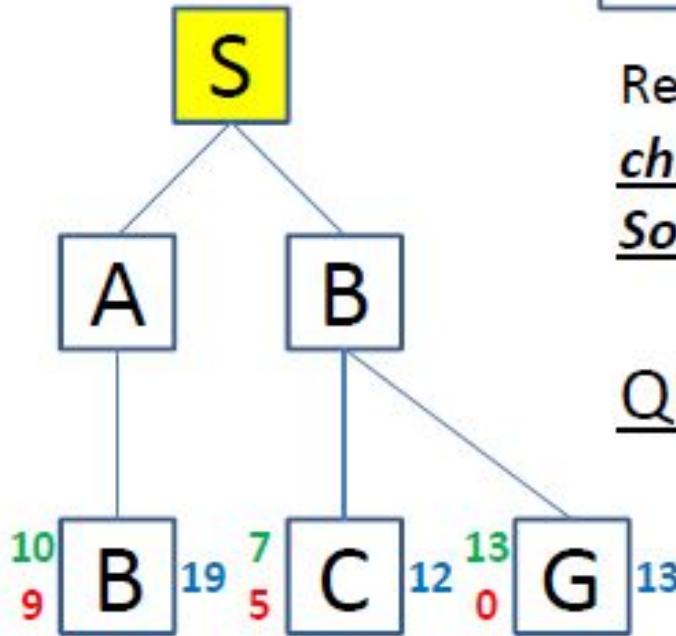
IF P terminating in I with cost_P &&
Q containing I with cost_Q AND
cost_P ≥ cost_Q THEN remove P

QUEUE: <SA,SBC,SBG,**SBA**>



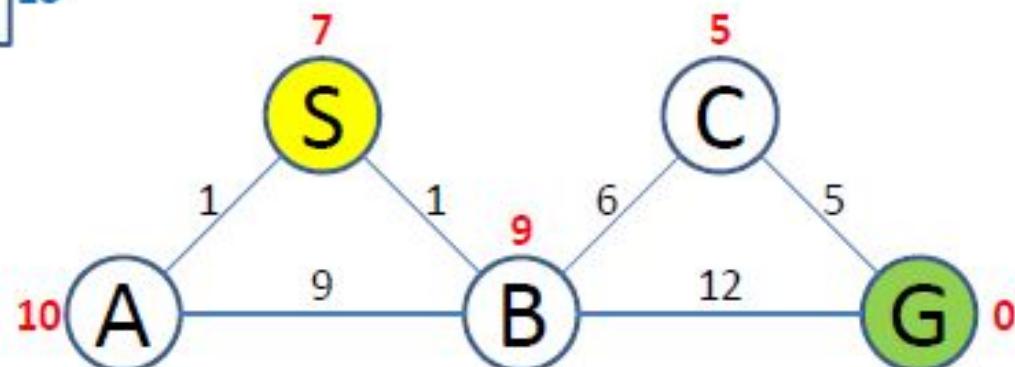
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



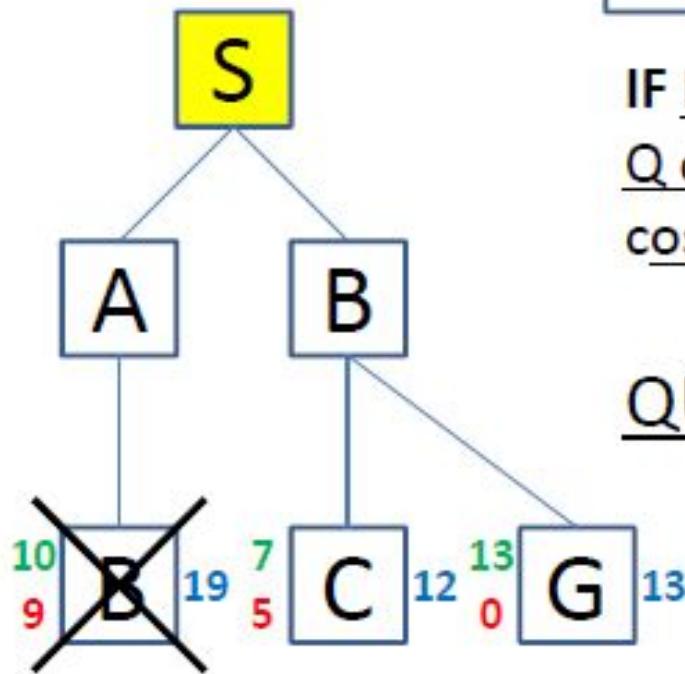
Remove first path, Create paths to all children, Reject loops and Add paths.
Sort QUEUE by f

QUEUE: <SBC,SBG,SAB>



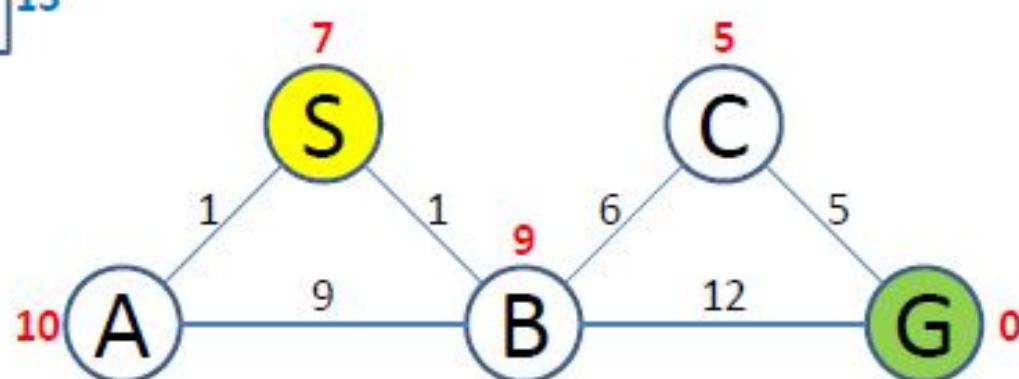
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



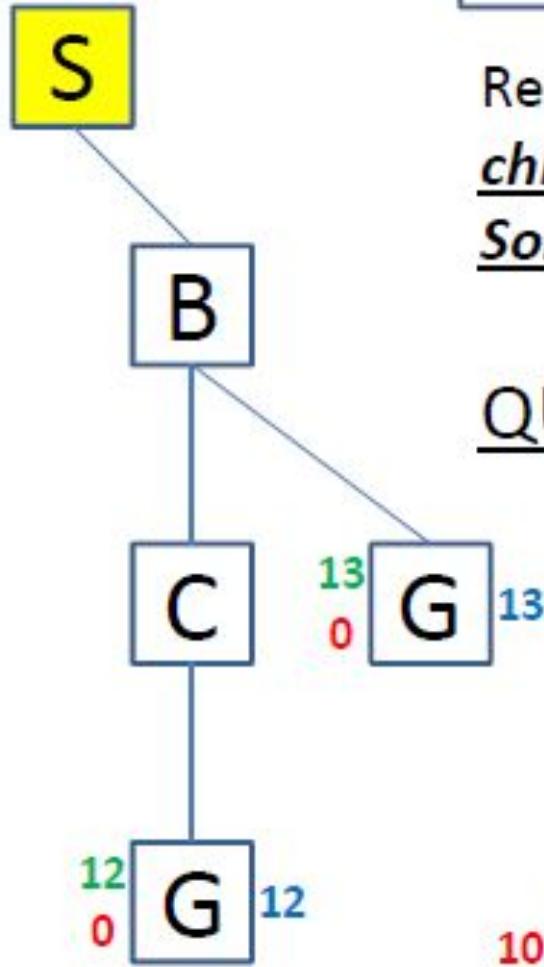
IF P terminating in I with cost \underline{P} &&
Q containing I with cost \underline{Q} AND
 $\underline{cost}_P \geq \underline{cost}_Q$ THEN remove P

QUEUE: <SBC,SBG,**SAB**>



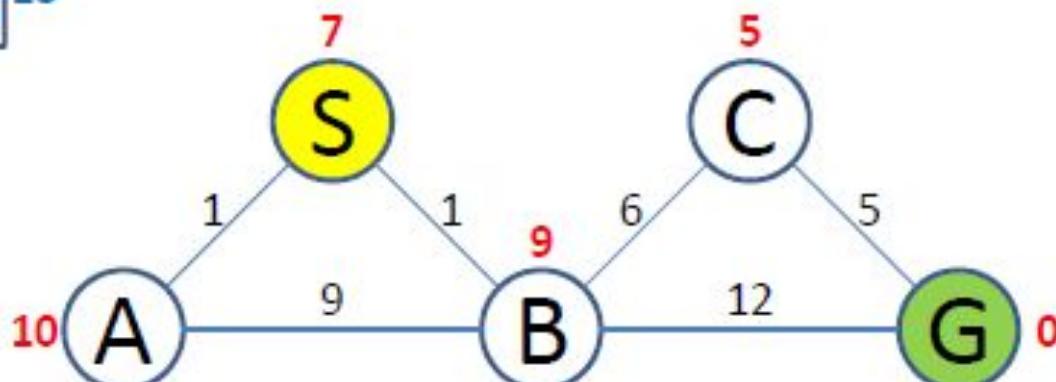
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



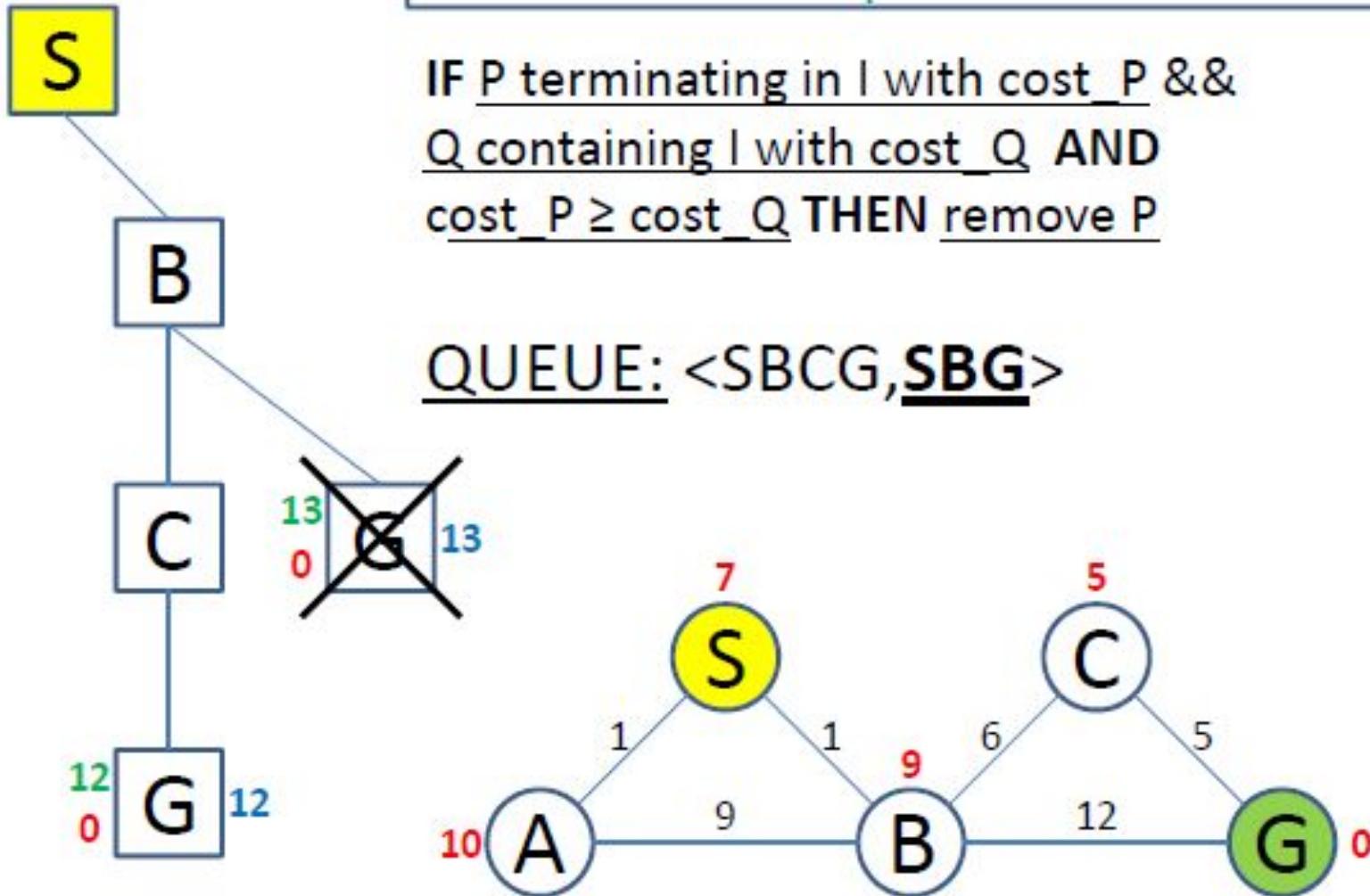
Remove first path, Create paths to all children, Reject loops and Add paths.
Sort QUEUE by f

QUEUE: <SBCG,SBG>



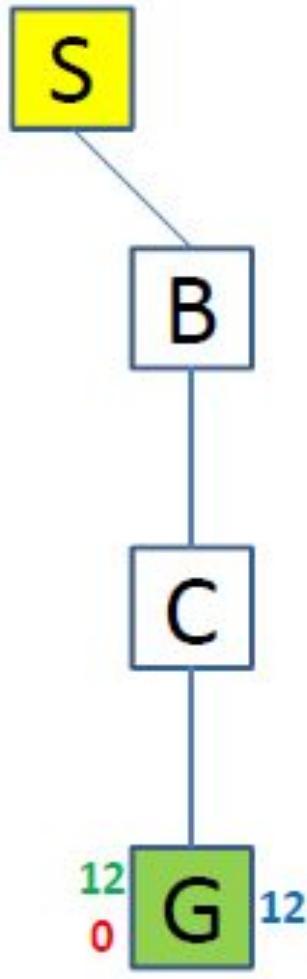
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



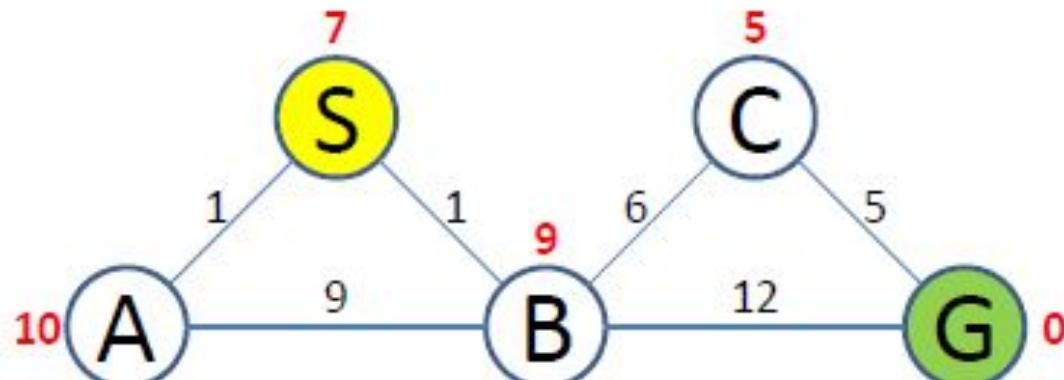
A* algorithm by Example

$f = \text{accumulated path cost} + \text{heuristic}$



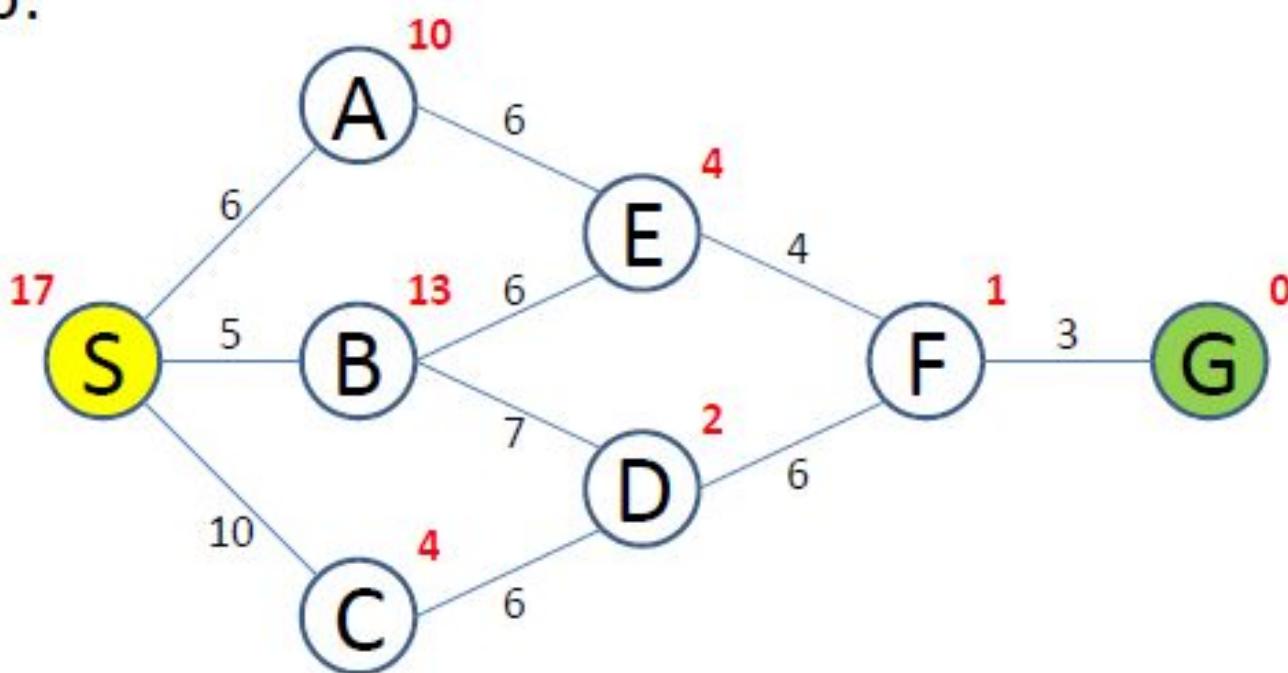
SUCCESS

QUEUE: <SBCG>



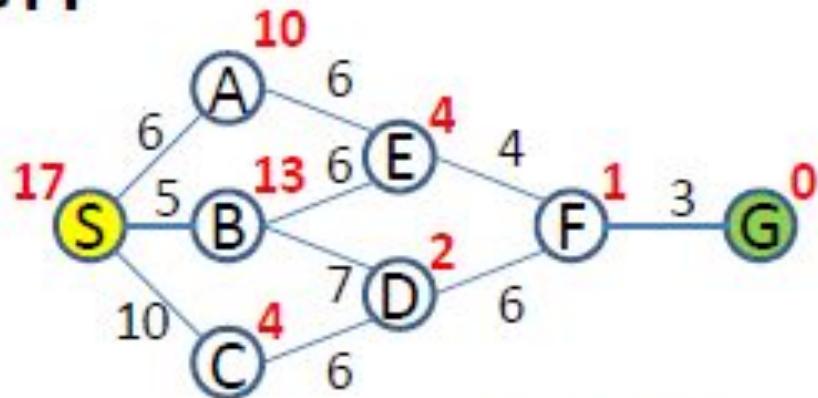
Problem

- Perform the A* Algorithm on the following figure. Explicitly write down the queue at each step.



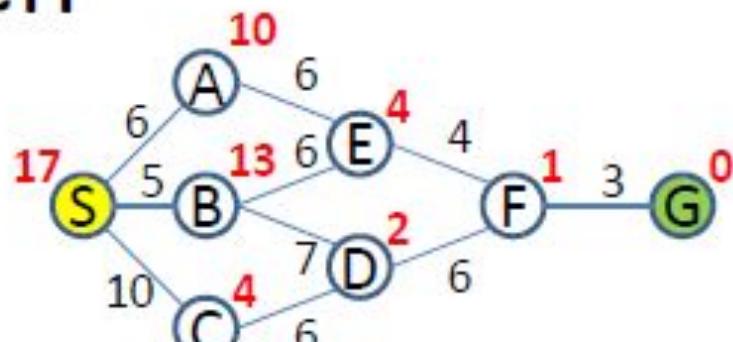
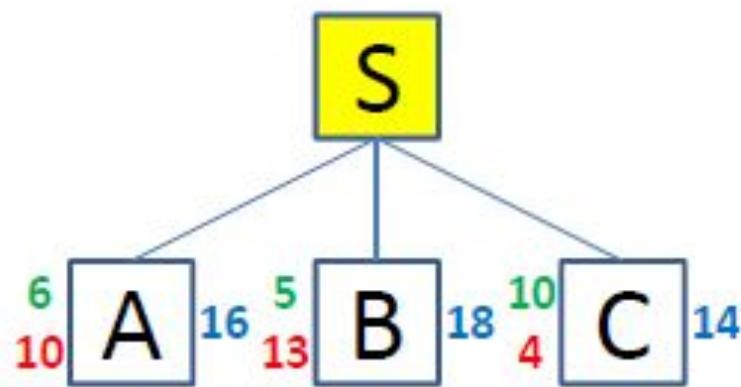
A* Search

0
17 S 17



QUEUE:
S

A* Search



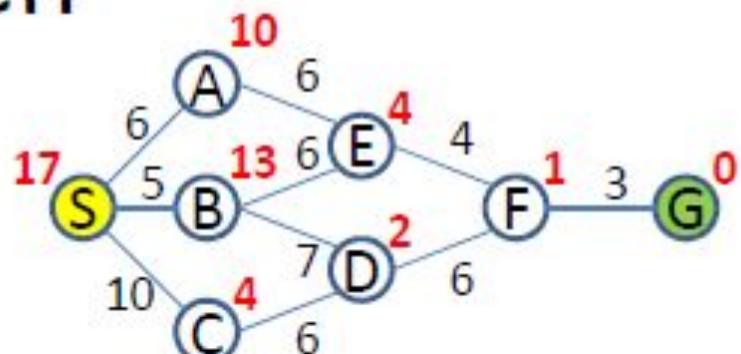
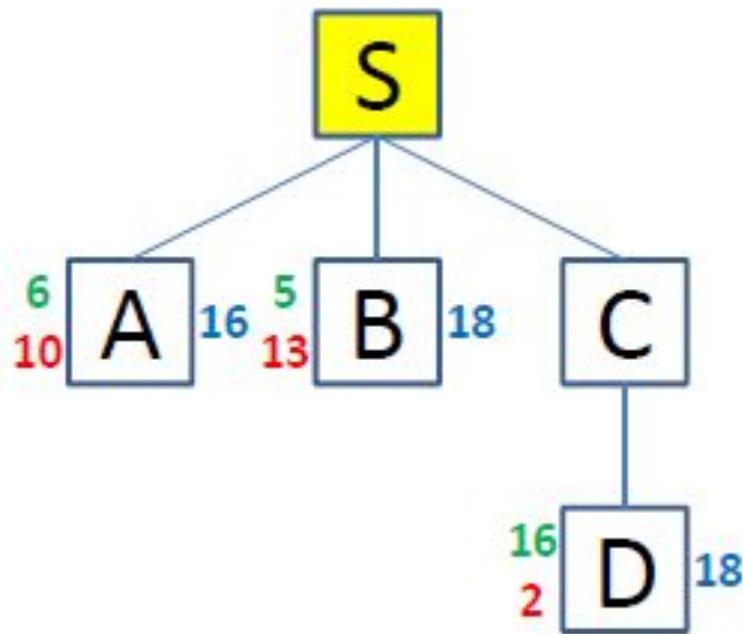
QUEUE:

SC

SA

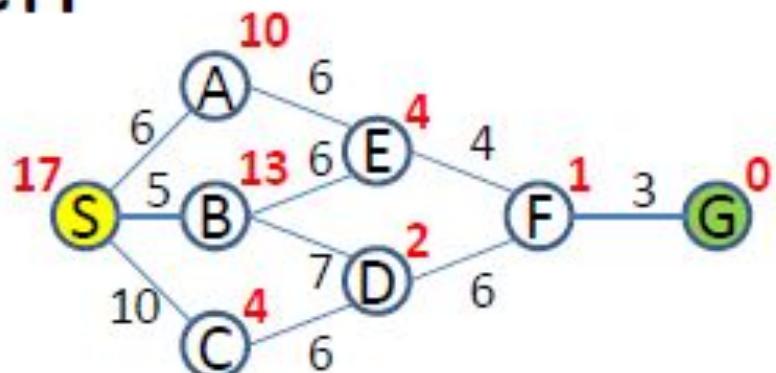
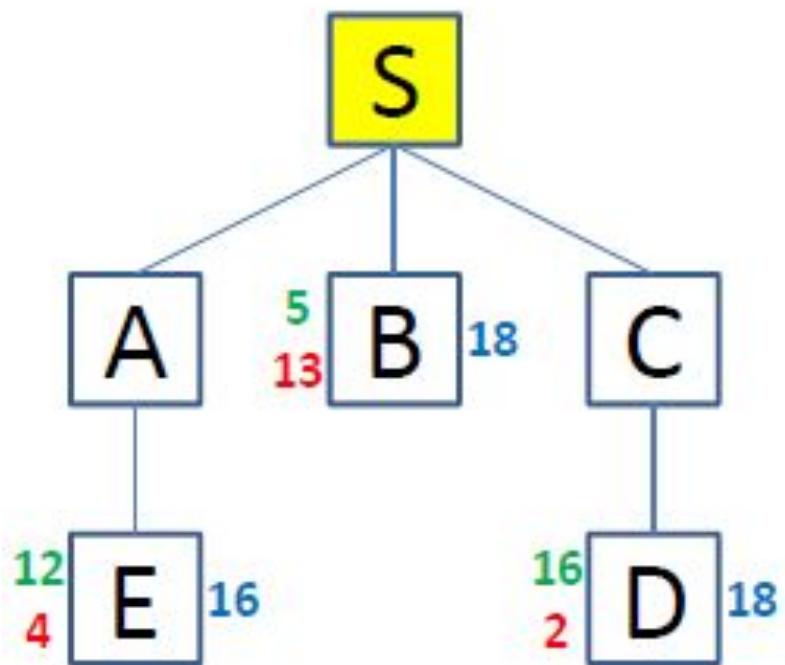
SB

A* Search



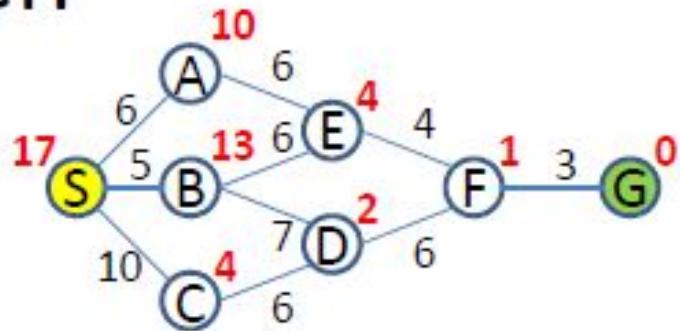
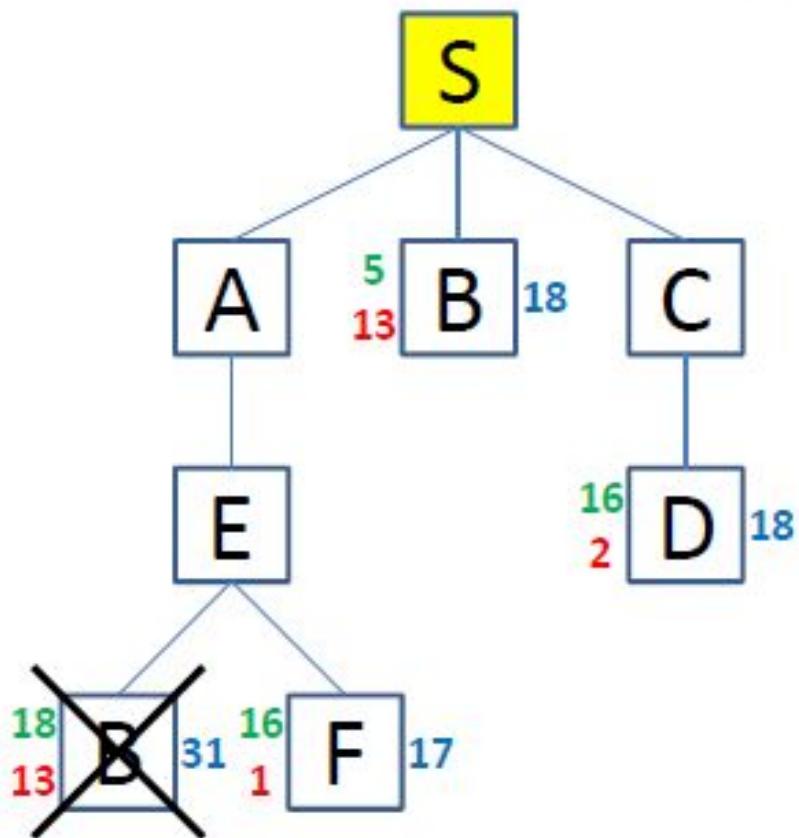
QUEUE:
SA
SCD
SB

A* Search



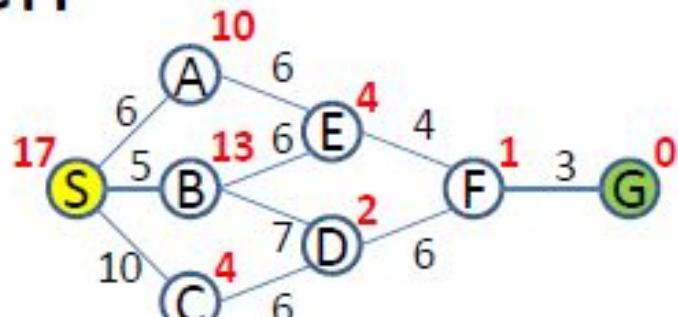
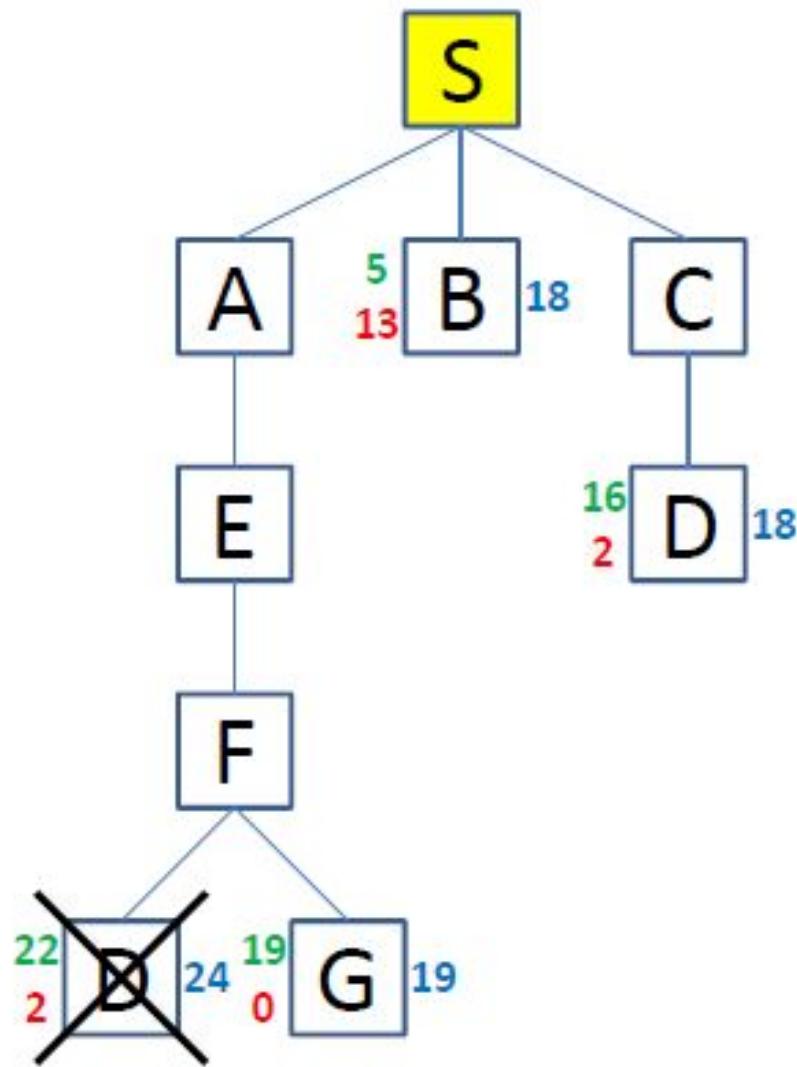
QUEUE:
SAE
SCD
SB

A* Search



QUEUE:
SAEF
SCD
SB
SAEB

A* Search



QUEUE:

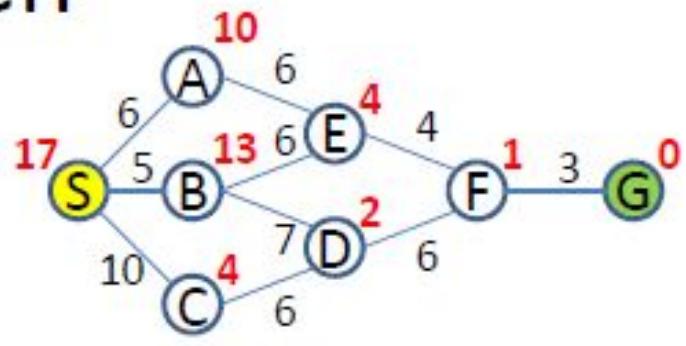
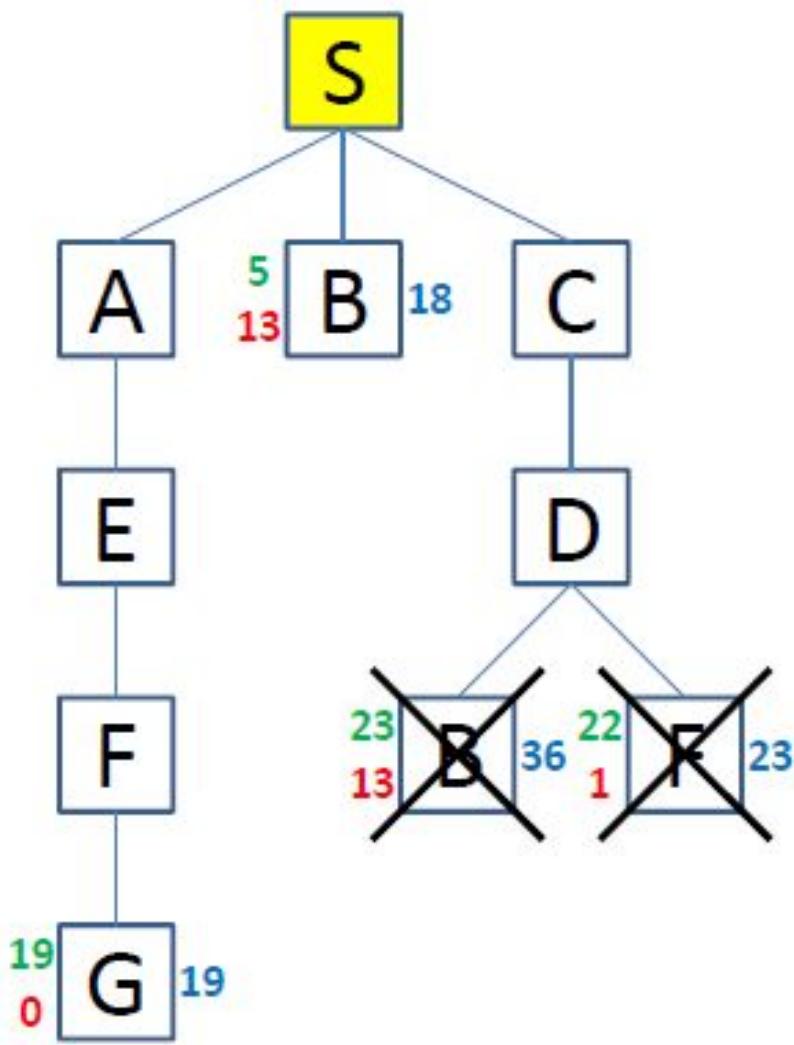
SCD

SB

SAEFG

SAEFD

A* Search



QUEUE:

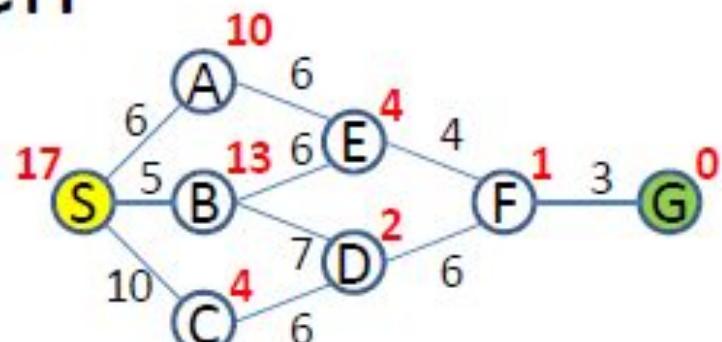
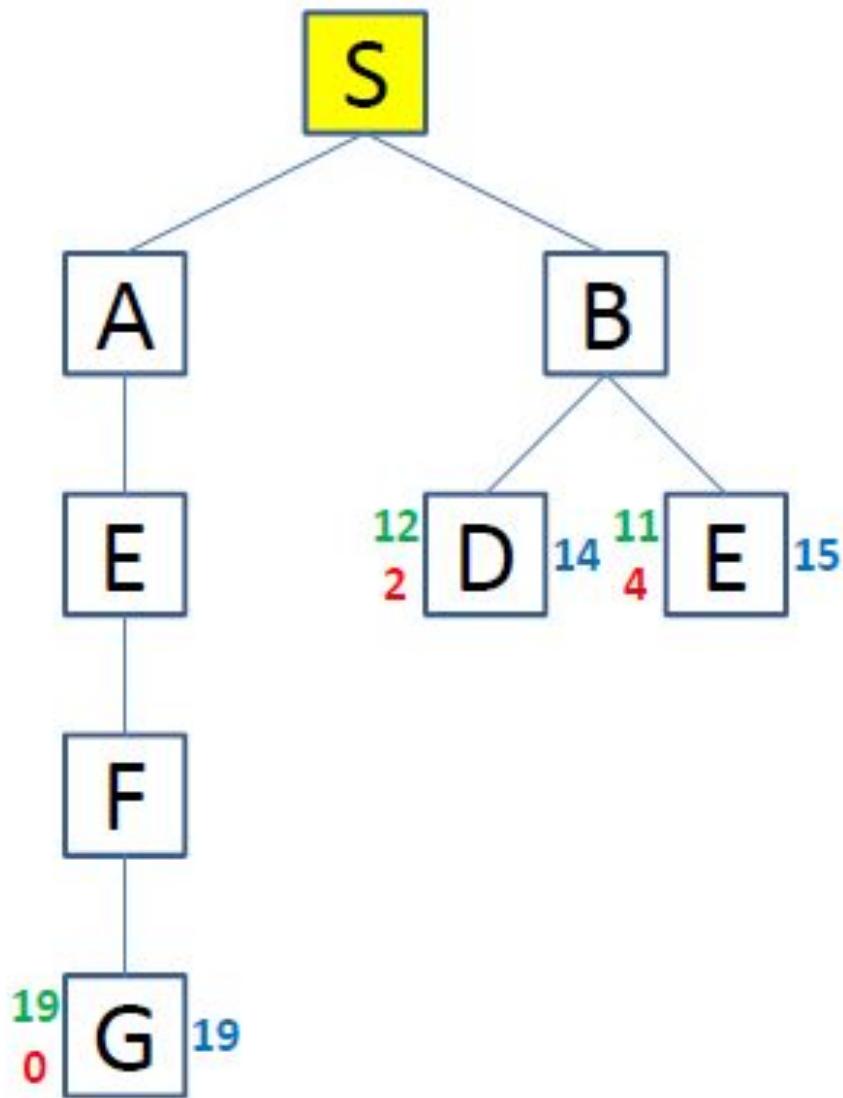
SB

SAEFG

SCDF

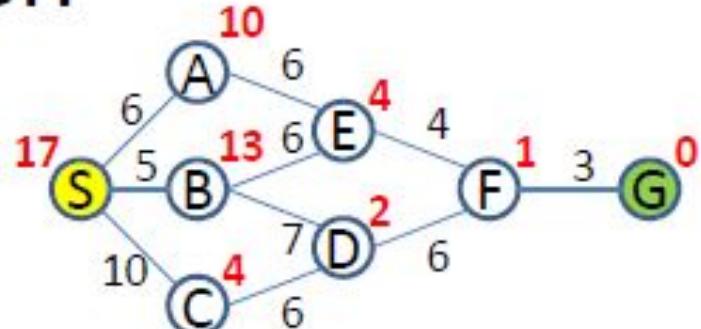
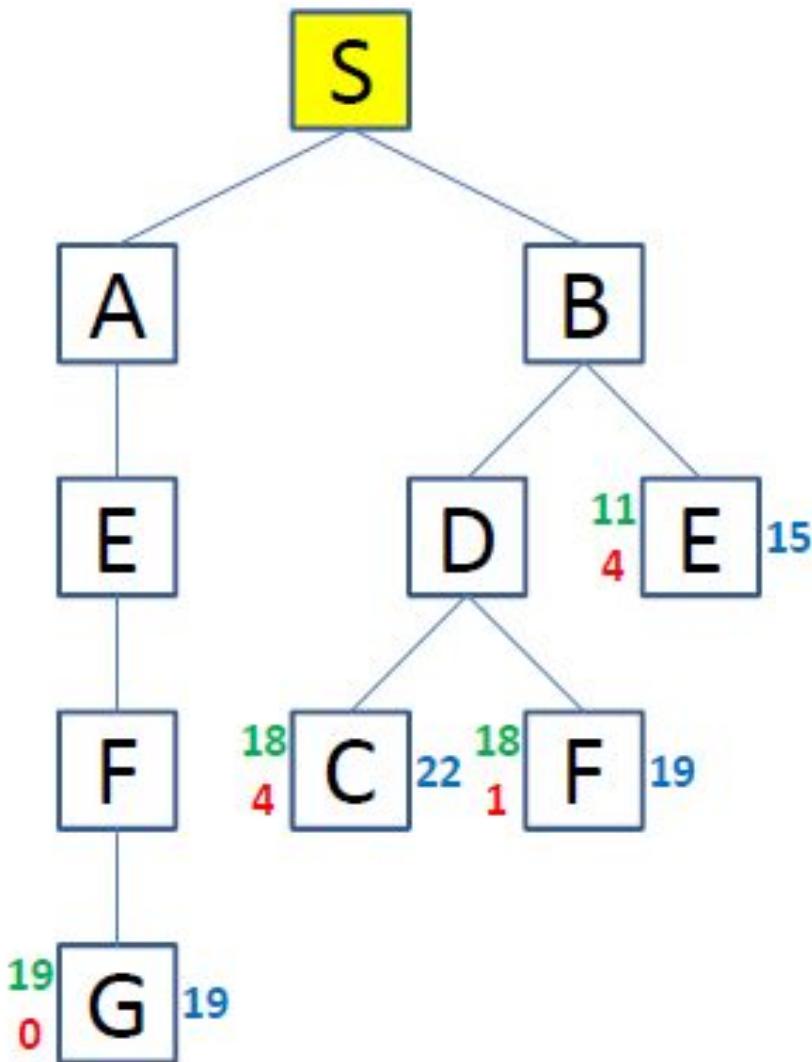
SCDB

A* Search



QUEUE:
SBD
SBE
SAEFG

A* Search



QUEUE:

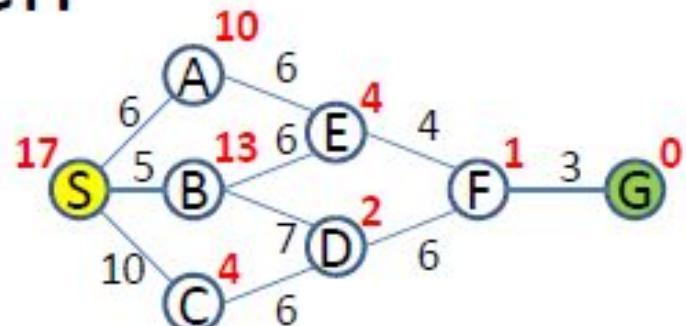
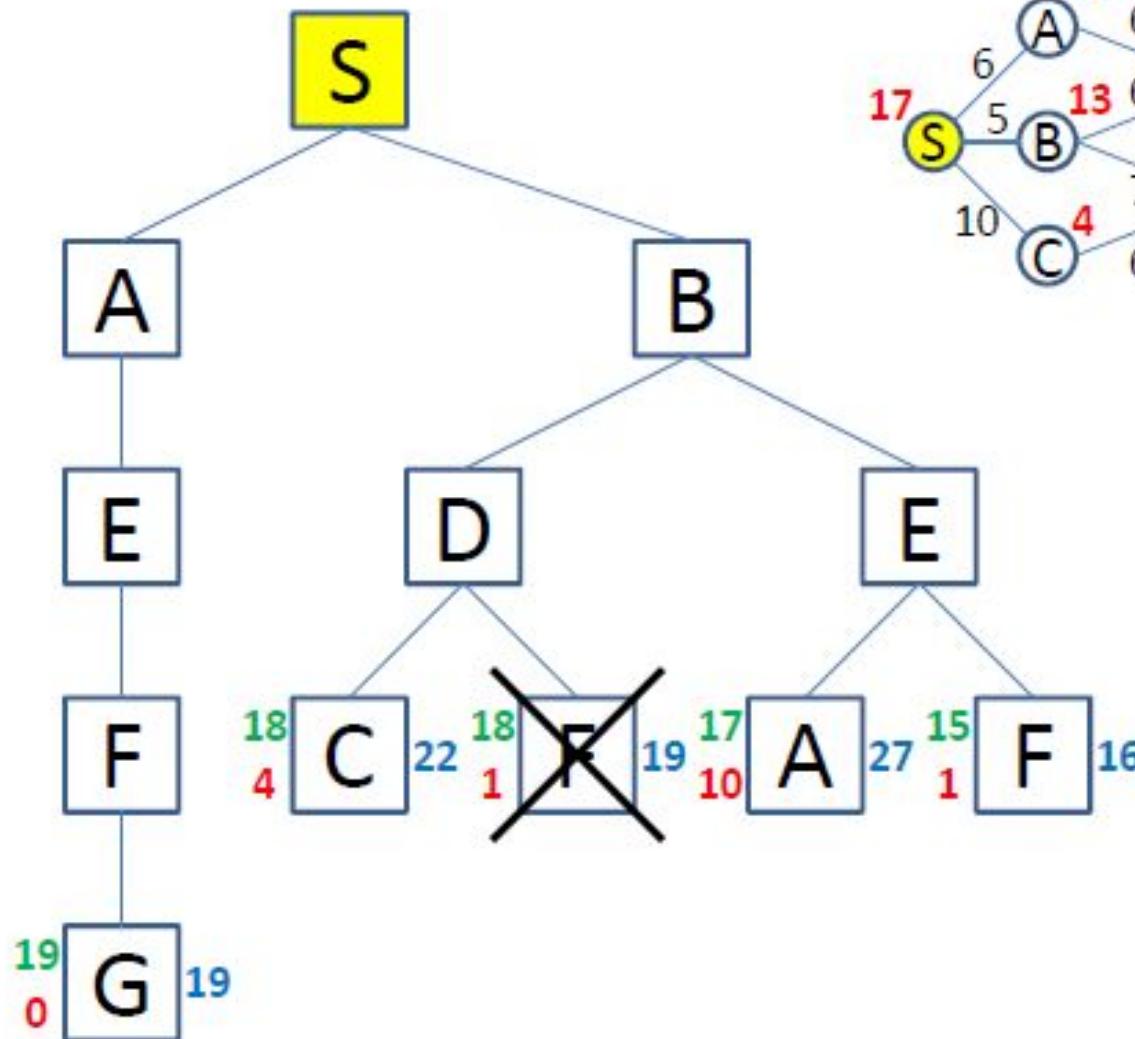
SBF

SBDF

SAEFG

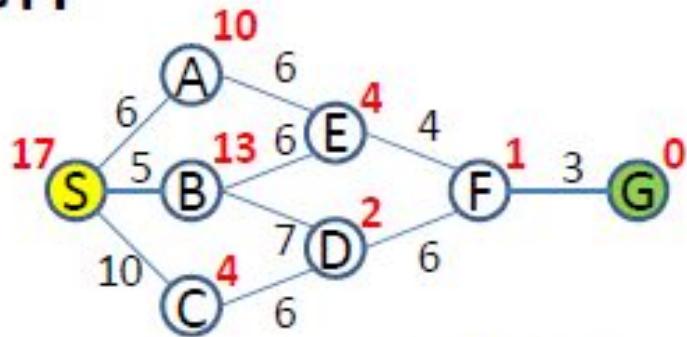
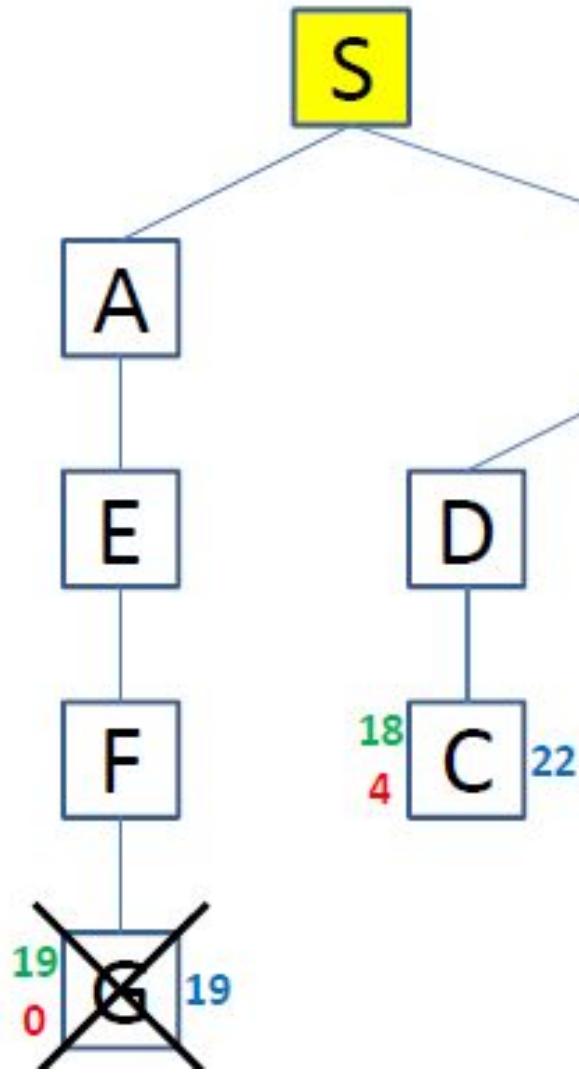
SBDC

A* Search



QUEUE:
SBEF
SAEFG
SBDF
SBDC
SBEA

A* Search



QUEUE:
SBEFG
SAEFG
SBDC
SBEFD
SBEA

Problem Reduction

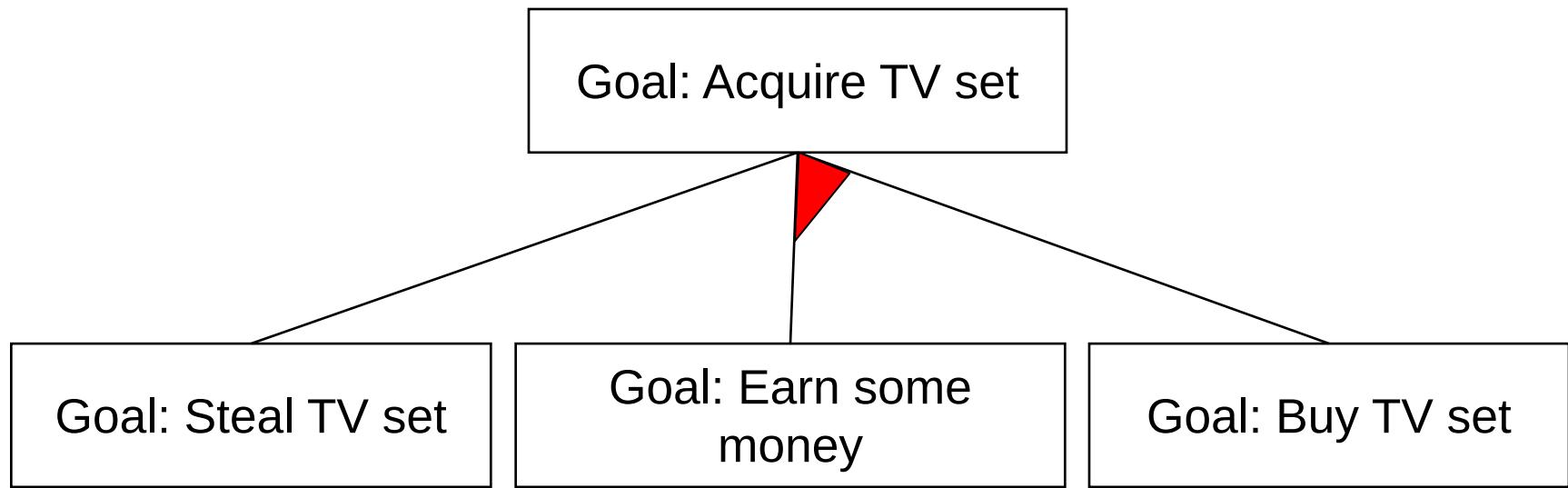
(Heuristic Search Techniques)

Problem reduction

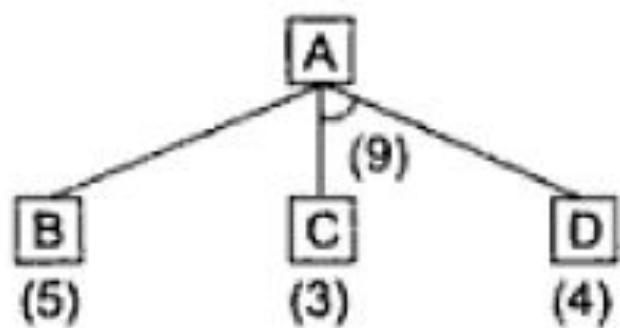
- Sometimes problems only seem hard to solve.
- A hard problem may be one that can be reduced to a number of simple problems...and, when each of the simple problems is solved, then the hard problem has been solved.
- This is the basic perception behind the method of problem reduction.

- To represent problem reduction techniques we need to use an AND-OR graph/tree.
- Problem reduction technique using AND-OR graph is useful for representing a solution of problem that can be solved by decomposing it into a set of smaller (sub)problems all of which must be solved.

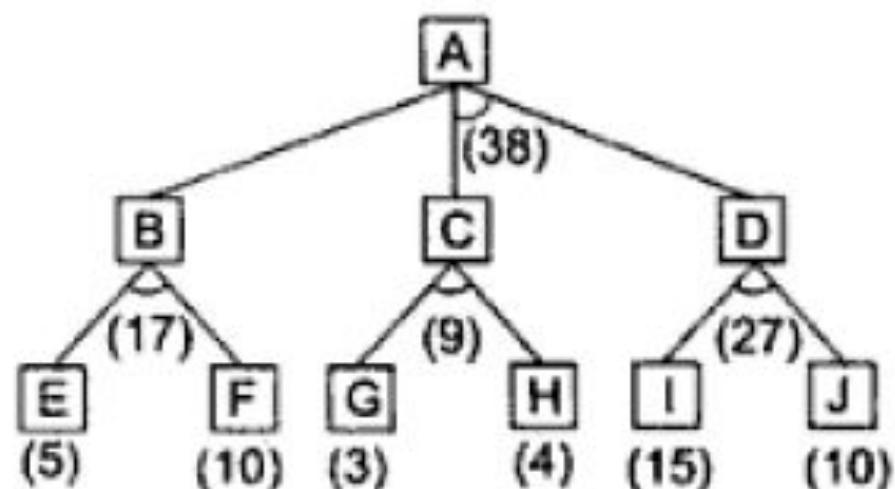
- This decomposition or reduction, generates arcs that we call AND arcs.
- One AND arc may point to a number of successor nodes, all of which must be solved in order for the arc to point to a solution.
- As in OR graph, several arcs may emerge from a single node, indicating the variety of ways in which the original problem might be solved. That is why it is called AND-OR graph



AND-OR Graphs



(a)



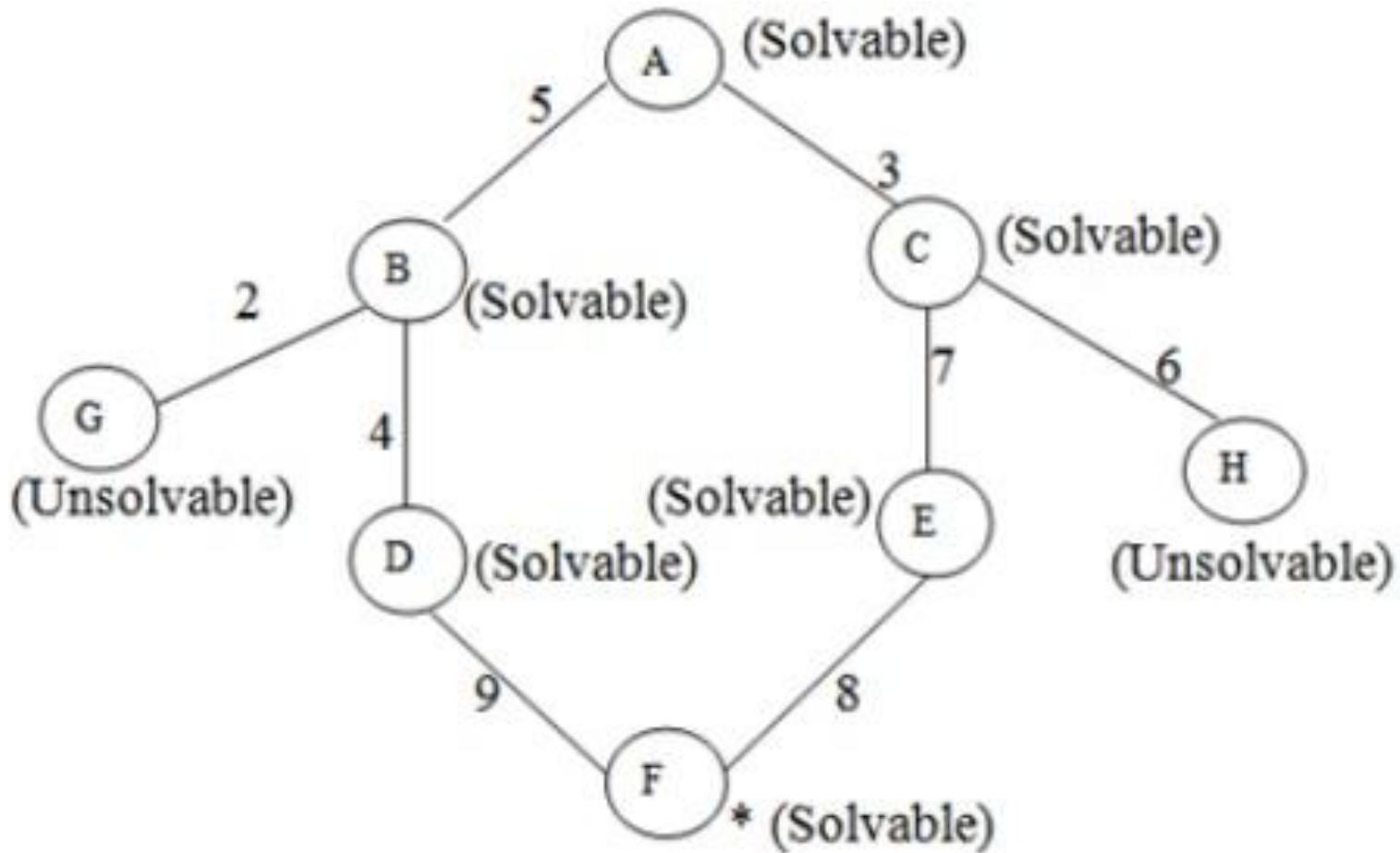
(b)

Fig. 3.7 AND-OR Graphs

Problem reduction: Algorithm

1. Initialize the graph to the starting node.
2. Loop until the starting node is labelled **SOLVED** or until its cost goes above **FUTILITY**:
 - i. Traverse the graph, starting at the initial node and following the current best path and accumulate the set of nodes that are on that path and have not yet been expanded.
 - ii. Pick one of these unexpanded nodes and expand it. If there are no successors, assign **FUTILITY** as the value of this node. Otherwise, add its successors to the graph and for each of them compute $f(n)$. If $f(n)$ of any node is 0, mark that node as **SOLVED**.
 - iii. Change the $f(n)$ estimate of the newly expanded node to reflect the new information provided by its successors. Propagate this change backwards through the graph. If any node contains a successor arc whose descendants are all solved, label the node itself as **SOLVED**.

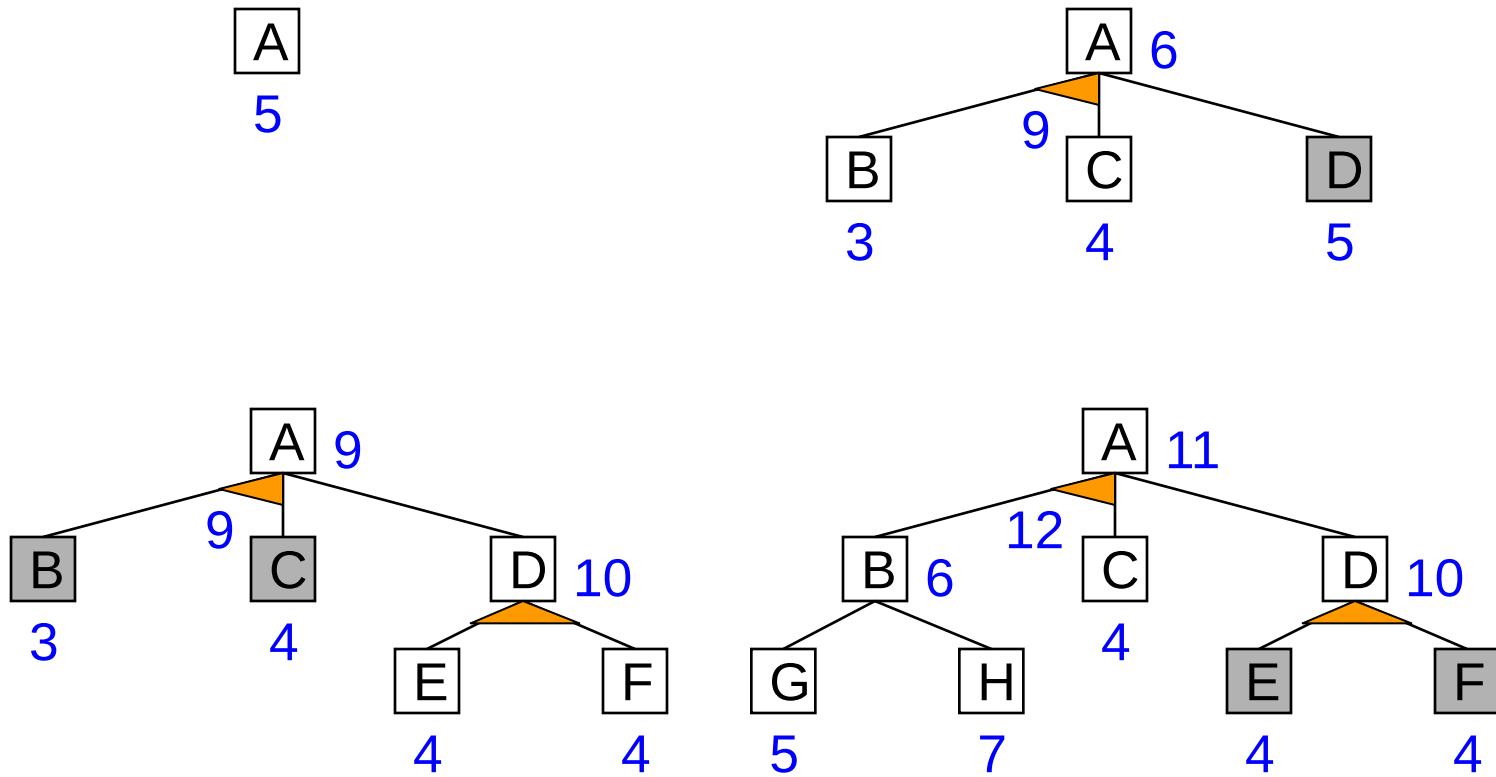
Example: Problem Reduction



ALGORITHM: AO*

1. Let G be a graph with only starting node **INIT**.
2. Repeat the followings until **INIT** is labeled **SOLVED** or $h(\text{INIT}) > \text{FUTILITY}$
 - a) *Select an unexpanded node from the most promising path from INIT (call it NODE)*
 - b) Generate successors of **NODE**. If there are none, set $h(\text{NODE}) = \text{FUTILITY}$ (i.e., **NODE** is unsolvable); otherwise for each **SUCCESSOR** that is not an ancestor of **NODE** do the following:
 - i. Add **SUCCESSOR** to G .
 - ii. If **SUCCESSOR** is a terminal node, label it **SOLVED** and set $h(\text{SUCCESSOR}) = 0$.
 - iii. If **SUCCESSOR** is not a terminal node, compute its h
 - c) Propagate the newly discovered information up the graph by doing the following: let S be set of **SOLVED** nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to **Node**. Until S is empty repeat the followings:
 - i. Remove a node from S and call it **CURRENT**.
 - ii. Compute the cost of each of the arcs emerging from **CURRENT**. Assign minimum cost of its successors as its h .
 - iii. Mark the best path out of **CURRENT** by marking the arc that had the minimum cost in step ii
 - iv. Mark **CURRENT** as **SOLVED** if all of the nodes connected to it through new labeled arc have been labeled **SOLVED**
 - v. If **CURRENT** has been labeled **SOLVED** or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of **CURRENT** to S .

AO* Example



Hill Climbing

(local search algo, greedy approach, no backtrack)

(Heuristic Search Techniques)

Hill Climbing

- Searching for a **goal state** = Climbing to the **top of a hill**
- Generate-and-test + **direction to move.**
- **Heuristic function** to estimate how close a given state is to a goal state.
- Types:
 - ***Simple Hill Climbing***
 - ***Steepest-Ascent Hill Climbing***
 - ***Simulated Annealing***

(1) Simple Hill Climbing :

Algorithm :

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

(2) Steepest-Ascent Hill Climbing (Gradient Search)

- Considers **all the moves** from the current state.
- Selects **the best one** as the next state.

Simple vs. steepest-ascent hill climbing

- simple hill climbing is an algorithm that helps to climb a mountain in 2D Space.
- In Steepest ascent hill climbing, you can conceptualize that the mountain is in 3D space.

Algorithm

1. Evaluate the initial state. If it is a goal state, then return it and quit it. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or a complete iteration produces no change to current state:
 - SUCC = a state such that any possible successor of the current state will be better than SUCC (the worst state).
 - For each operator that applies to the current state, evaluate the new state:
 - goal → quit
 - better than SUCC → set SUCC to this state
 - SUCC is better than the current state → set the current state to SUCC.

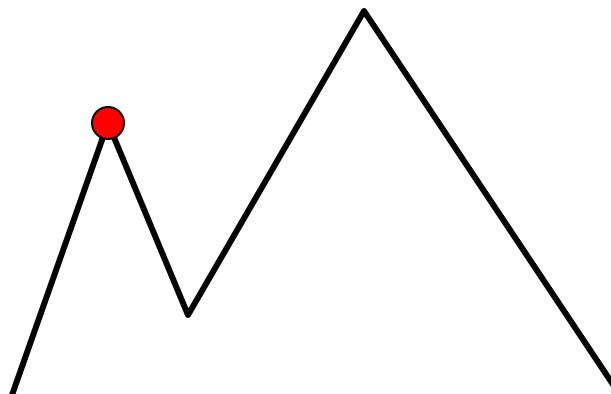
- **Both basic and steepest-ascent hill climbing**
 - May fail to find a solution.
 - Either algorithm may terminate not by finding a goal state but by getting to a state from which no better states can be generated.
 - This will happen if the program has reached either a local maximum, a plateau, or a ridge.
- **Both Hill climbing and Steepest ascent hill climbing suffers**
 - the limitation of getting stuck in local maximum which can be addressed by simulated annealing .

- A ***local maximum*** is a state that is better than all its neighbors but is not better than some state farther away. At a local maximum, all moves appear to make things worse. Local maxima are particularly frustrating because they often occur almost within sight of a solution. In this case, they are called ***foothills***.
- A ***plateau*** is a flat area of the search space in which a whole set of neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.
- A ***ridge*** is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope.

Hill Climbing: Disadvantages

Local maximum

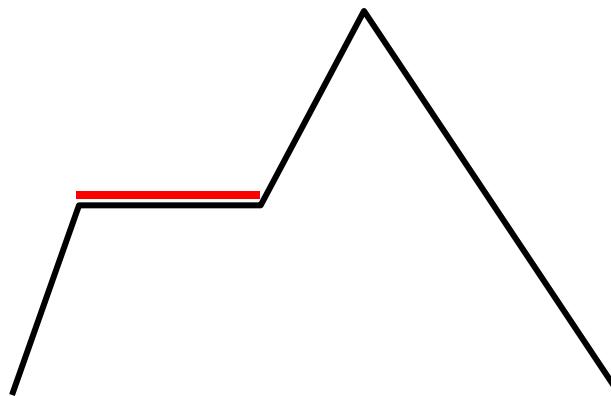
A state that is better than all of its neighbours, but not better than some other states far away.



Hill Climbing: Disadvantages

Plateau

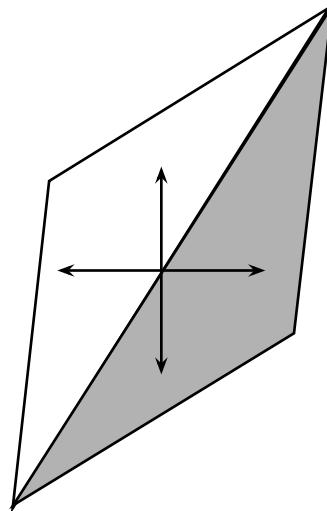
A flat area of the search space in which all neighbouring states have the same value.



Hill Climbing: Disadvantages

Ridge

The orientation of the high region, compared to the set of available moves, makes it impossible to climb up. However, two moves executed serially may increase the height.

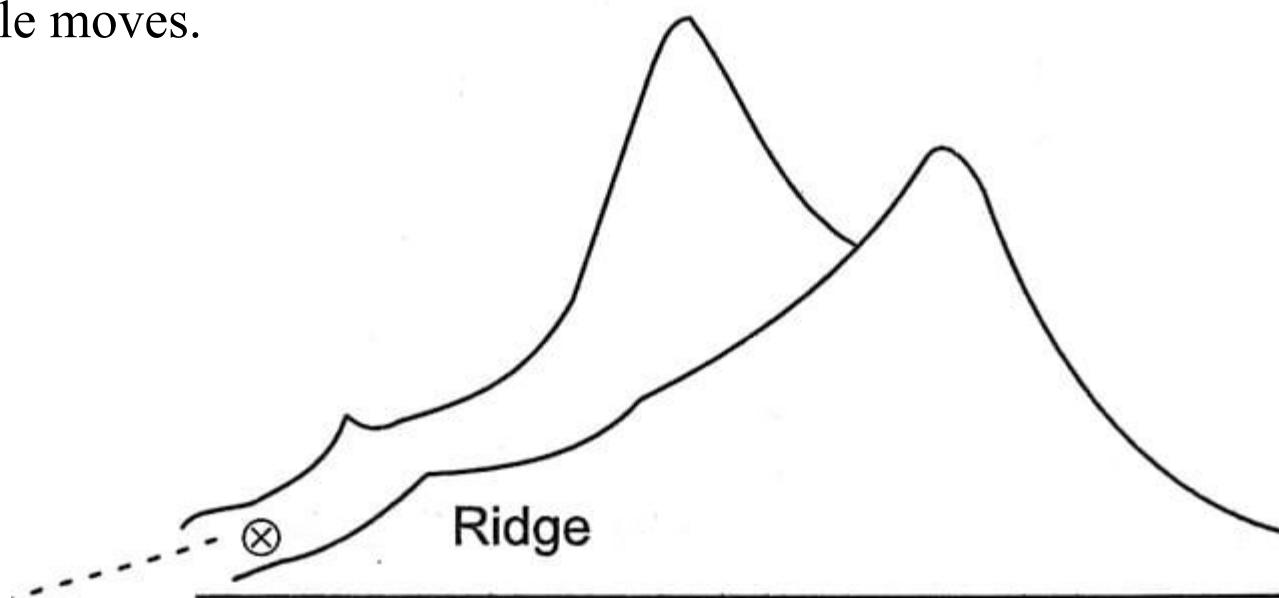




(a) Local Maximum
(b) Plateau
(c) Ridge



- **Local maximum:** It is a state that is better than all its neighbors but is not better than some other states farther away .
- **Plateau:** It is a flat area of the search space in which a whole set of neighboring states have the same value. on a plateau it is not possible to determine the best direction in which to move by making local comparisons.
- **Ridge:** It is special case of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope. But the orientation of the high region, compared to the set of available moves and directions in which they move, makes it impossible to traverse a ridge by single moves.



Solution to problems

Local Maxima : Backtrack to some earlier node and try going in a different direction.

Plateaus: Make a **big jump** to try to get in a new section of search space.

Ridges: Moving in **several directions at once**(apply two or more rules before doing test).

(3) Simulated Annealing :

- Simulated Annealing is a variation of hill climbing in which, at the beginning of the process, some downhill moves may be made.

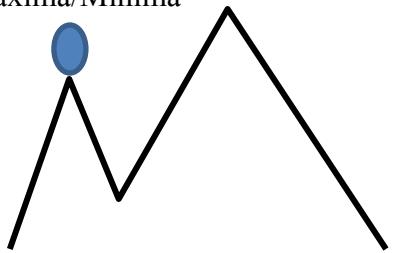
Physical Annealing

- Physical substances are melted and then **gradually cooled** until some solid state is reached.
- The goal is to produce a **minimal-energy** state.
- **Annealing schedule**: if the temperature is lowered sufficiently slowly, then the goal will be attained.
- Nevertheless, there is some probability for a transition to a higher energy state: $e^{-\Delta E/T}$.

Algorithm: Simulated Annealing

1. Energy E , Change in Energy ΔE
Temperature T

Local Maxima/Minima



```
C = Cinit
for T=Tmax to Tmin
    EC=E (C)
    N=Next (C)
    EN=E (N)
    ΔE=EN - EC
```

- 2.
3. Current Configuration to New Configuration

C $\overset{\text{Move}}{N}$

```
if (ΔE > 0) (positive – good)
    C = N
elseif (e(ΔE/T) > rand (0,1))
    C = N
```

Probability Function: $e^{(\Delta E/T)}$

For high temperature we accept **more bad** moves,
as we progress, with low temperature, the probability to accept bad move is also **low**

Algorithm: Simulated Annealing

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

(b) Evaluate the new state. Compute

$$\Delta E = \text{Val}(\text{current state}) - \text{Val}(\text{new state})$$

- If the new state is a goal state then return it and quit.
- If it is not a goal state but it is better than the current state then make it the current state. Also set BEST-SO-FAR to this new state.
- If it is not better than the current state, then make it the current state with probability p' . This step is usually implemented by invoking a random number generator to produce a number in the range $[0,1]$. If the number is less than p' , then the move is accepted. Otherwise do nothing.

(c) Revise T as necessary according to the annealing schedule.

(5) Return BEST-SO-FAR, as the answer.

Ways Out

- Backtrack to some earlier node and try going in a different direction.
- Make a big jump to try to get in a new section.
- Moving in several directions at once.

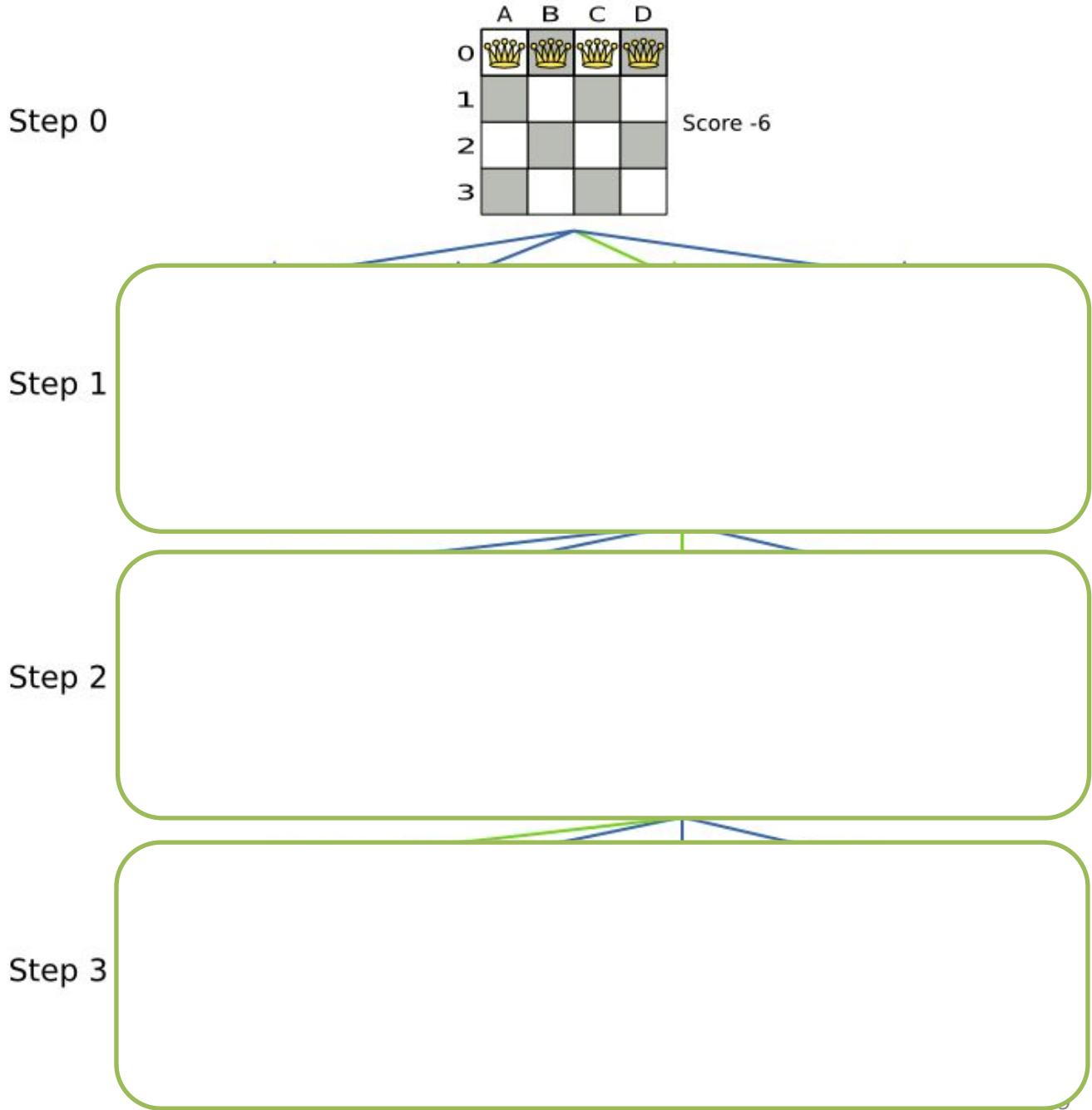
Hill Climbing: Disadvantages

- Hill climbing is a **local method**:
Decides what to do next by looking only at the
“immediate” consequences of its choices.
- **Global information** might be encoded in
heuristic functions.

Hill Climbing: Conclusion

- Can be **very inefficient** in a large, rough problem space.
- Global heuristic may have to pay for **computational complexity**.
- **Often useful** when combined with other methods, getting it started right in the right general neighbourhood.

4-Queen Problem using Hill Climbing



Local Beam Search

- Beam search is a heuristic search algorithm that explores a graph by expanding the most optimistic node in a limited set. Beam search is an *optimization* of best-first search that reduces its memory requirements.
- Best-first search is a graph search that orders all partial solutions according to some heuristic. But in beam search, only a predetermined number of best partial solutions are kept as candidates. Therefore, it is a *greedy* algorithm.
- Beam search uses *breadth-first search* to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. However, it only stores a predetermined number (β), of best states at each level called the *beamwidth*. Only those states are expanded next.

- The greater the beam width, the fewer states are pruned. No states are pruned with infinite beam width, and beam search is identical to breadth-first search. The beamwidth bounds the memory required to perform the search.
- Since a *goal state* could potentially be pruned, beam search sacrifices completeness (the guarantee that an algorithm will terminate with a solution if one exists). Beam search is not optimal, which means there is no guarantee that it will find the best solution.
- In general, beam search returns the *first* solution found. Once reaching the configured maximum search depth (i.e., translation length), the algorithm will evaluate the solutions found during a search at various depths and return the best one that has the highest probability.
- The beam width can either be *fixed* or *variable*. One approach that uses a variable beam width starts with the width at a minimum. If no solution is found, the beam is widened, and the procedure is repeated.

Example: Beam Search

- For example, let's take the value of $\beta = 2$ for the tree shown below. So, follow the following steps to find the goal node.

Step 1: OPEN = {A}

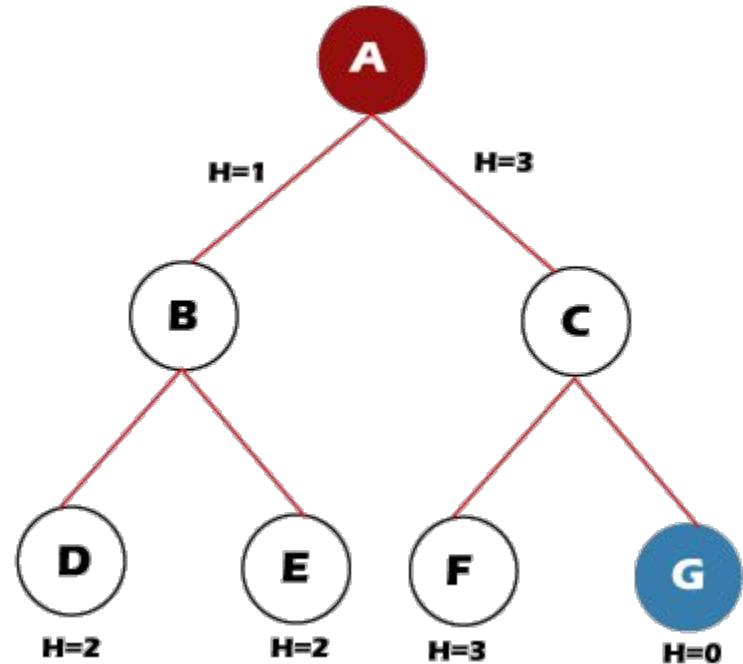
Step 2: OPEN = {B, C}

Step 3: OPEN = {D, E}

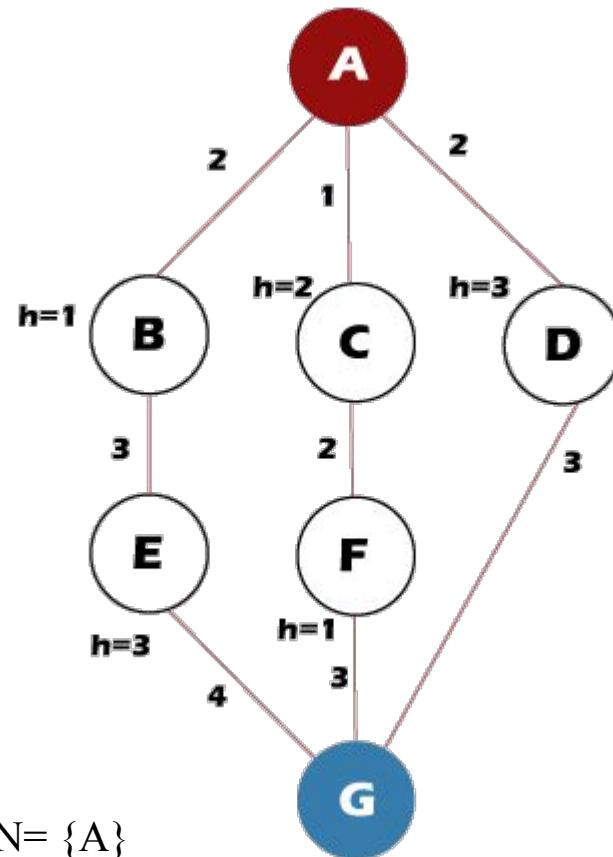
Step 4: OPEN = {E}

Step 5: OPEN = {}

The open set becomes empty **without** finding the goal node.



- The beam width and an inaccurate heuristic function may cause the algorithm to *miss* expanding the shortest path.
- A more *precise* heuristic function and a *larger* beam width can make Beam Search more likely to find the optimal path to the goal.
- For example, we have a tree with heuristic values as shown here:



Step 1: OPEN= {A}
 Step 2: OPEN= {B, C}
 Step 3: OPEN= {C, E}
 Step 4: OPEN= {F, E}
 Step 5: OPEN= {G, E}
 Step 6: Found the goal node {G}, now stop.
Path: A-> C-> F-> G

But the *Optimal Path* is: A-> D-> G

Unit-4

Constraint Satisfaction Problems

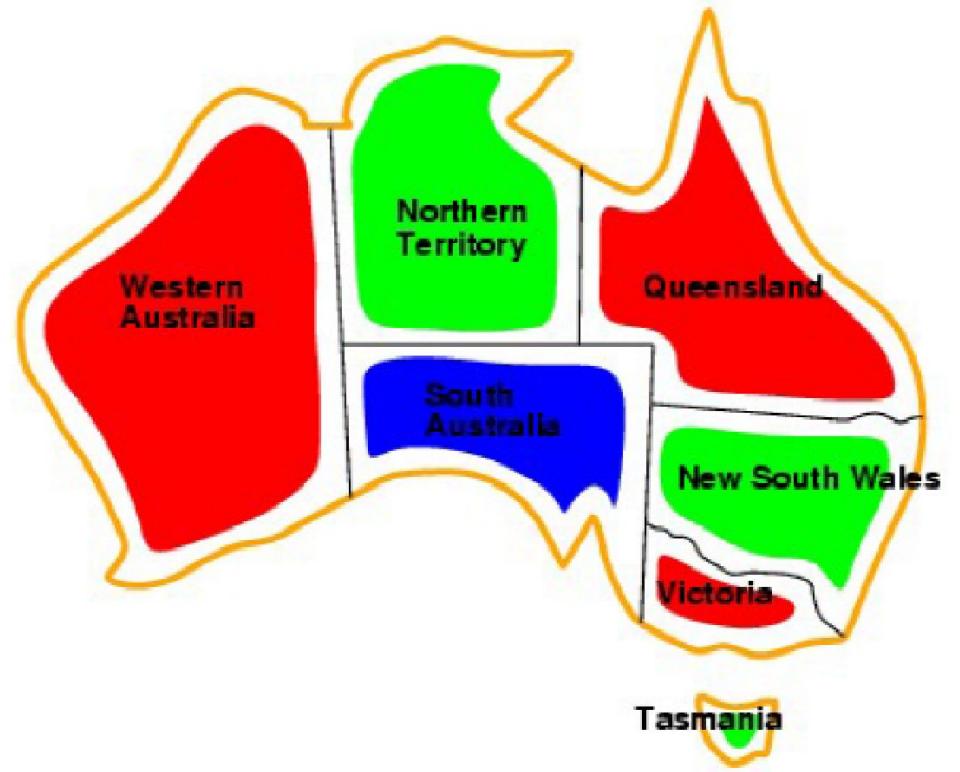
What is Constraint Satisfaction Problems (CSP)?

CSP is defined by 3 components (X, D, C):

- **state**: a set of variables X, each X_i , with values from domain D_i
- **goal test**: a set of constraints C, each C_i involves some subset of the variables and specifies the allowable combinations of values for that subset
- Each constraint C_i consists of a pair $\langle \text{scope}, \text{rel} \rangle$, where scope is a tuple of variables and rel is the relation, either represented explicitly or abstractly
- For example,
 - A and B are variables and respective domains are $\{1,2\}$ and $\{2,4\}$
 - Constraint $(A \neq B)$
 - The possible pairs are $(1,2), (1,4), (2,4)$ but can not have $(2,2)$

Example: Map Coloring

- Variables: $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$
- Domains: $D_i = \{\text{red, green, blue}\}$
- Constraints: adjacent regions must have different colors
- Solution?



{ WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red }

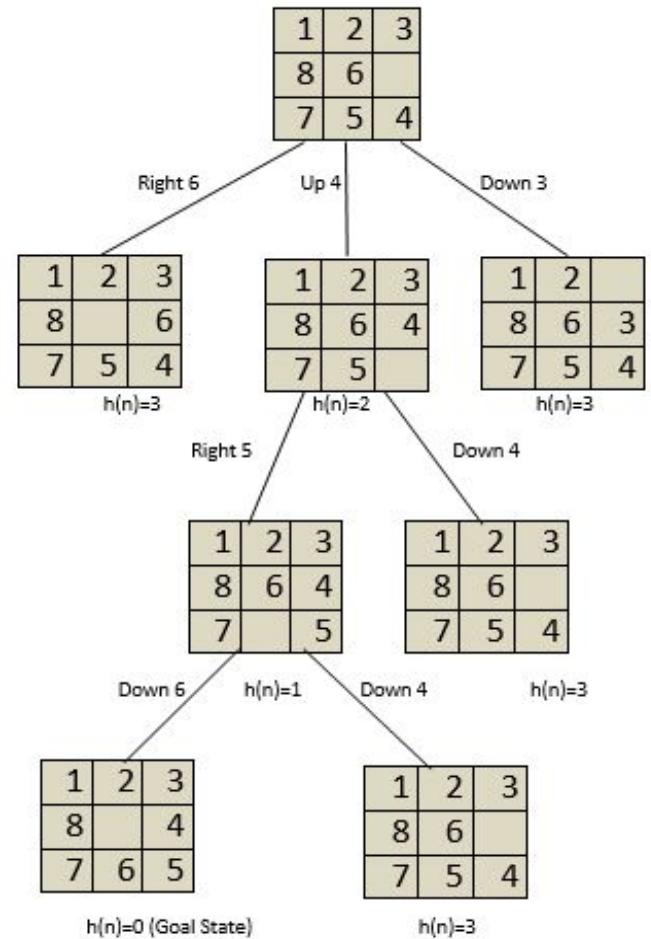
Local search (constraint satisfaction)

- In constraint satisfaction, local search is an **incomplete** method for finding a solution to a problem. It is based on iteratively **improving** an assignment of the variables until all constraints are satisfied.
- In particular, local search algorithms typically **modify** the value of a variable in an assignment at each step. The new assignment is close to the previous one in the space of assignment, hence the name **local search**.
- All local search algorithms use a **function** that evaluates the quality of assignment, for example the number of constraints violated by the assignment.
- This amount is called the cost of the assignment. The **aim of local search** is that of finding an assignment of minimal cost, which is a solution if any exists.

Local search (What is already covered?)

Two classes of **local search** algorithms exist.

- The **first** one is that of greedy or non-randomized algorithms. These algorithms proceed by changing the current assignment by always trying to decrease (or at least, non-increase) its cost.
- The main problem of these algorithms is the possible presence of plateaus (**see the figure**), which are regions of the space of assignments where no local move decreases cost.
- The **second** class of local search algorithm have been invented to solve this problem. They escape these plateaus by doing random moves, and are called randomized local search algorithms. (**Simulated annealing**)



Crypt-arithmetic: Constraint Satisfaction Problem

- Crypt-arithmetic is a type of constraint satisfaction problem.
- Constraints:
 - No two letters have same value
 - Some of digit must be as shown in problem
 - There should be only one carry forward

$$\begin{array}{r} \boxed{T} \quad \boxed{O} \\ + \quad \boxed{G} \quad \boxed{O} \\ \hline \boxed{O} \quad \boxed{U} \quad \boxed{T} \end{array}$$

$$\begin{array}{l} T = ? \\ G = ? \\ O = ? \\ U = ? \end{array}$$

$$\begin{array}{l} T = ? \\ G = ? \\ O = 1 \\ U = ? \end{array}$$

$$\begin{array}{l} T = 2 \\ G = ? \\ O = 1 \\ U = ? \end{array}$$

$$\begin{array}{r} \boxed{2} \quad \boxed{1} \\ + \quad \boxed{G} \quad \boxed{1} \\ \hline \boxed{1} \quad \boxed{U} \quad \boxed{2} \end{array}$$

$$\begin{array}{r} \boxed{2} \quad \boxed{1} \\ + \quad \boxed{8} \quad \boxed{1} \\ \hline \boxed{1} \quad \boxed{0} \quad \boxed{2} \end{array}$$

Now G can be
8 or 9

If G is 9, then U must be 1,
which is not possible as O is 1

So G must be
8 hence U is 0

$$\begin{array}{l} T = 2 \\ G = 8 \\ O = 1 \\ U = 0 \end{array}$$

$$\begin{array}{r}
 \boxed{S} \quad \boxed{E} \quad \boxed{N} \quad \boxed{D} \\
 + \quad \boxed{M} \quad \boxed{O} \quad \boxed{R} \quad \boxed{E} \\
 \hline
 \boxed{M} \quad \boxed{O} \quad \boxed{N} \quad \boxed{E} \quad \boxed{Y}
 \end{array}$$

M=1

$$\begin{array}{r}
 \boxed{C1=1} \quad \boxed{S} \quad \boxed{E} \quad \boxed{N} \quad \boxed{D} \\
 + \quad \boxed{1} \quad \boxed{O} \quad \boxed{R} \quad \boxed{E} \\
 \hline
 \boxed{1} \quad \boxed{O} \quad \boxed{N} \quad \boxed{E} \quad \boxed{Y}
 \end{array}$$

If S=9
O=0

$$\begin{array}{r}
 \boxed{9} \quad \boxed{E} \quad \boxed{N} \quad \boxed{D} \\
 + \quad \boxed{1} \quad \boxed{0} \quad \boxed{R} \quad \boxed{E} \\
 \hline
 \boxed{1} \quad \boxed{0} \quad \boxed{N} \quad \boxed{E} \quad \boxed{Y}
 \end{array}$$

E + 0 = N, Which is not possible,
hence $(+1) + E + 0 = N$ (using
carry)
Hence, $1 + E + 0 = N$

$$\begin{array}{r}
 \boxed{C2=1} \\
 \boxed{9} \quad \boxed{E} \quad \boxed{N} \quad \boxed{D} \\
 + \quad \boxed{1} \quad \boxed{0} \quad \boxed{R} \quad \boxed{E} \\
 \hline
 \boxed{1} \quad \boxed{0} \quad \boxed{N} \quad \boxed{E} \quad \boxed{Y}
 \end{array}$$

Now, $N + R (+C3) = E$,
that is $N + R (+1) = E$
Now $N + R$ must be greater than 10 to
generate a carry,
So $N + R = E + 10$.
Now substitute the value of N from
previous step

So $E + 1 + R (+1) = E + 10$
 $R = 10 - 1 = 9$ if no carry
 $R = 10 - 2 = 8$ if carry

Now 9 is already assigned so, R must be
equal to 8

$$\begin{array}{r}
 \boxed{C3=1} \\
 \boxed{9} \quad \boxed{E} \quad \boxed{N} \quad \boxed{D} \\
 + \quad \boxed{1} \quad \boxed{0} \quad \boxed{8} \quad \boxed{E} \\
 \hline
 \boxed{1} \quad \boxed{0} \quad \boxed{N} \quad \boxed{E} \quad \boxed{Y}
 \end{array}$$

Now, $D + E = Y$
here carry is required so the
possible values of D & E are:
5, 6, 7 (3, 4 can not be included
because the sum gives 0 or 1 which
is already assigned)
Let's assume D = 7 and E = 5
Then $7 + 5 = 12$, which gives Y = 2

$$\begin{array}{r}
 \boxed{9} \quad \boxed{5} \quad \boxed{6} \quad \boxed{7} \\
 + \quad \boxed{1} \quad \boxed{0} \quad \boxed{8} \quad \boxed{5} \\
 \hline
 \boxed{1} \quad \boxed{0} \quad \boxed{6} \quad \boxed{5} \quad \boxed{2}
 \end{array}$$

Practice problems: Crypt-arithmetic

- A S + A = M O M
- S O + S O = T O O
- L E O + L E E = A L L
- B A S E + B A L L = G A M E S
- D O N A L D + G E R A L D = R O B E R T

Adversarial Search

- Adversarial search is search when there is an "enemy" or "opponent" changing the state of the problem every step **in a direction you do not want**.
- **Examples:** Chess, business, trading, war.
- You change state, but then you don't **control** the next state.
- Opponent will change the next state in a way:
 - ✓ **unpredictable**
 - ✓ **hostile** to you

Different Game Scenarios using Adversarial search

1. Games with Perfect information

These are open and transparent games where each player has all complete information on the status of the game, and the move of the opponent is quite visible to the players. Chess, Checkers, Ethello and Go are some of such games.

2. Games with Imperfect information

In these games, players will not have any information on the status of the game, and it will have blind moves by the players assuming certain conditions and predicting the outcomes. Tic-Tac-Toe, Battleship, Blind and Bridge are some examples of this type of game.

3. Deterministic Games

These games follow pre-defined rules and pattern, and it is played in a transparent manner. Each player can see the moves of the opponents clearly. Chess, Checkers, Tic-tac-toe, Go are some of the games played in this manner.

4. Non-Deterministic Games

Luck, Chance, Probability plays a major role in these types of game. The outcome of these games is highly unpredictable, and players will have to take random shots relying on luck. Poker, Monopoly, Backgammon are few of the games played in this method.

5. Zero-Sum games

These games are purely competitive in nature, and one player's gain is compensated by other player's loss. The net value of gain and loss is zero, and each player will have different conflicting strategies in these games.

Depending on the game environment and game situation, each player will either try to maximize the gain or minimize the loss.

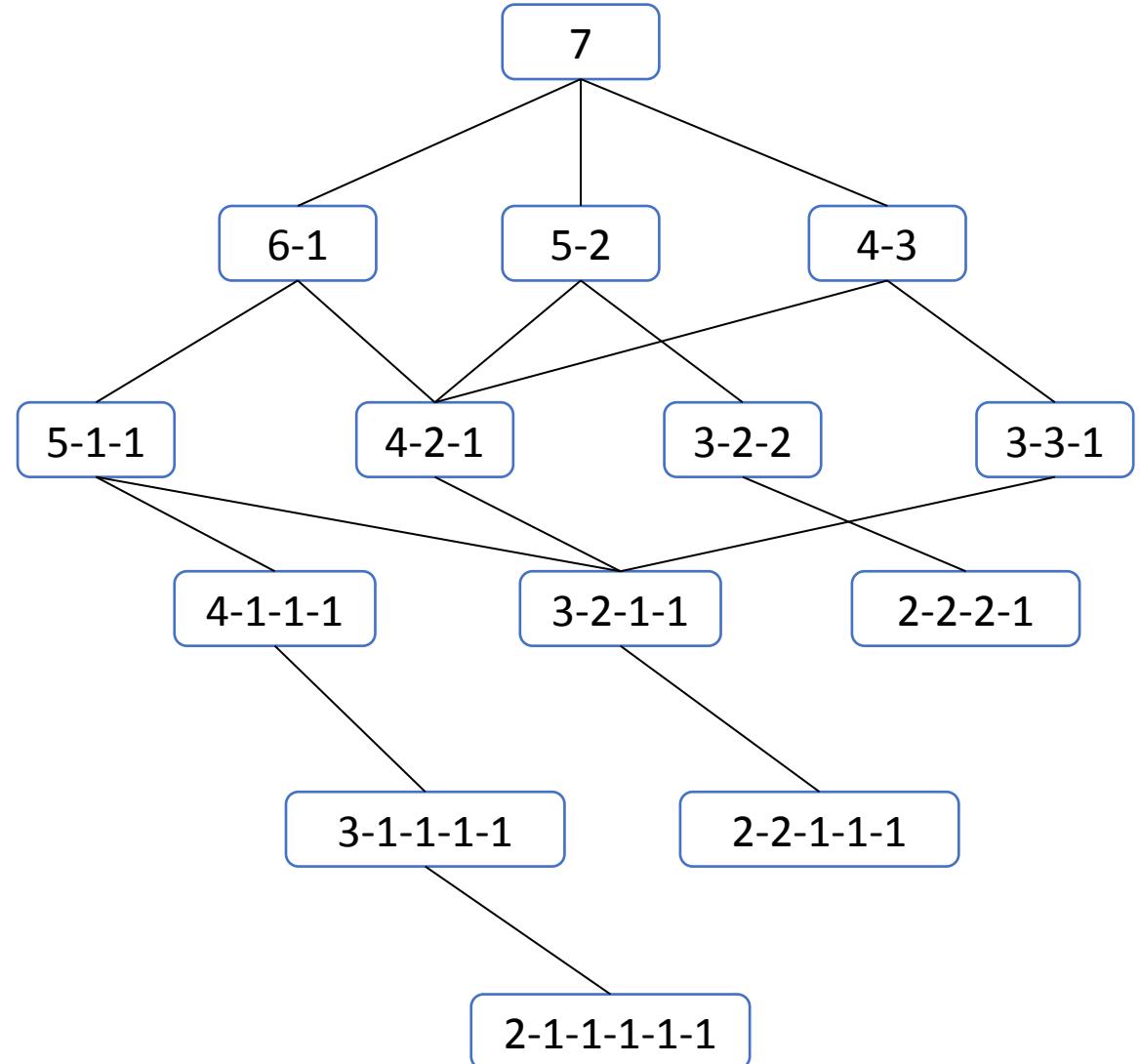
Games in AI

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, $\frac{1}{2}$. And for tic-tac-toe, utility values are +1, -1, and 0.

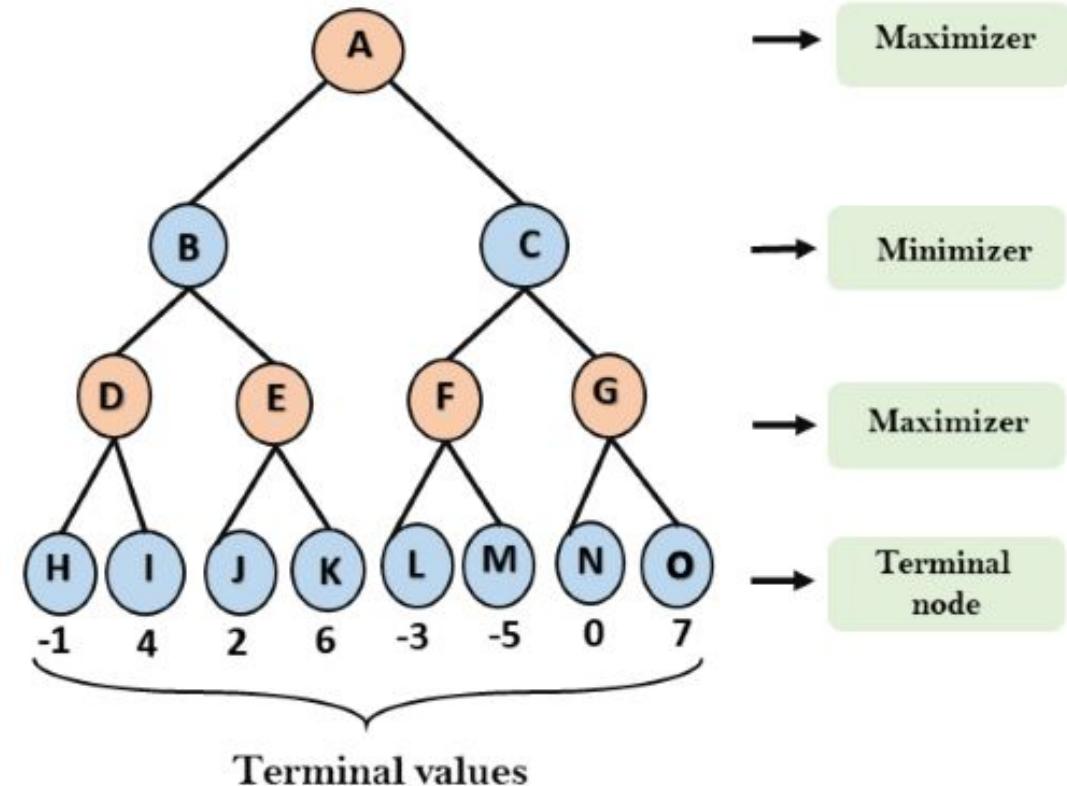
Minimax: Example of Adversarial Search

- The Minimax algorithm tries to predict the opponent's behaviour.
- It predicts the opponent will take the worst action from our viewpoint.
- We are MAX - trying to maximise our score/move to best state.
- Opponent is MIN - tries to minimise our score/move to worst state for us.
- **The Game of Nim:** At each move the player must divide a pile of tokens into 2 piles of different sizes. The first player who is unable to make a move loses the game.
- Example: 7 Piles



Minimax: another example

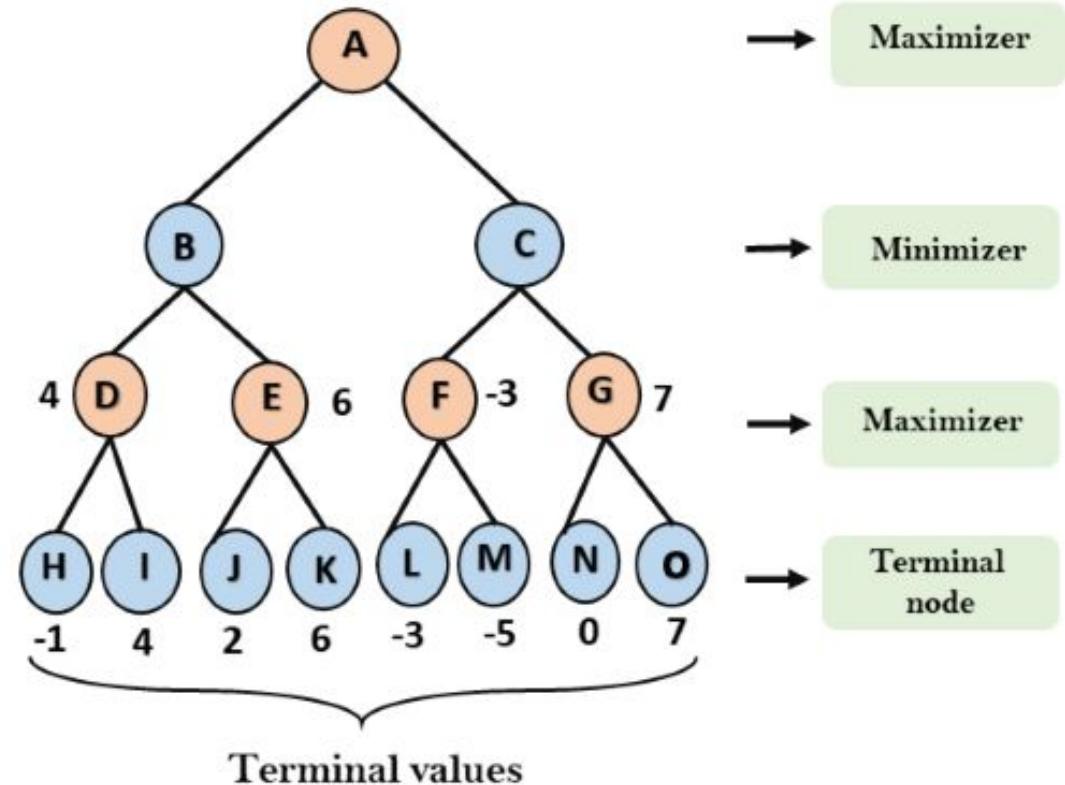
- **Step-1:** In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.
- In the tree diagram, let's take A is the initial state of the tree.
- Suppose maximizer takes first turn which has worst-case initial value = $-\infty$, and minimizer will take next turn which has worst-case initial value = $+\infty$.



Cont'd

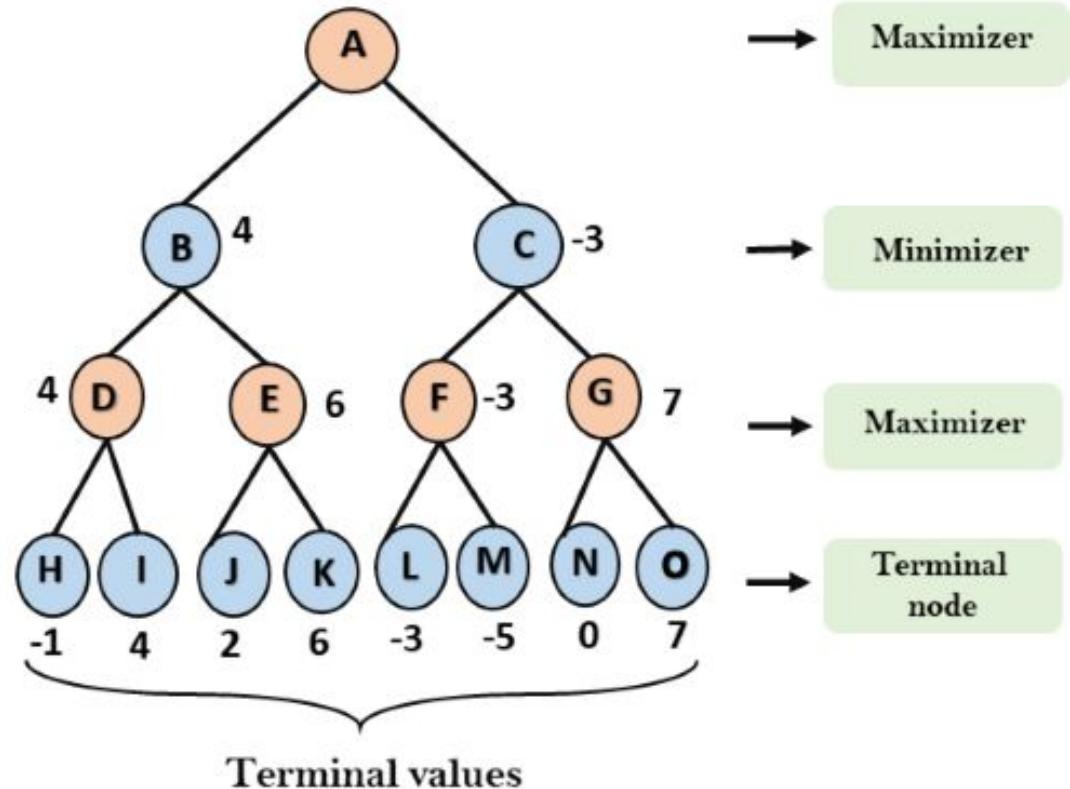
- **Step 2:** Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

For node D	$\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
For Node E	$\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
For Node F	$\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
For node G	$\max(0, -\infty) \Rightarrow \max(0, 7) = 7$



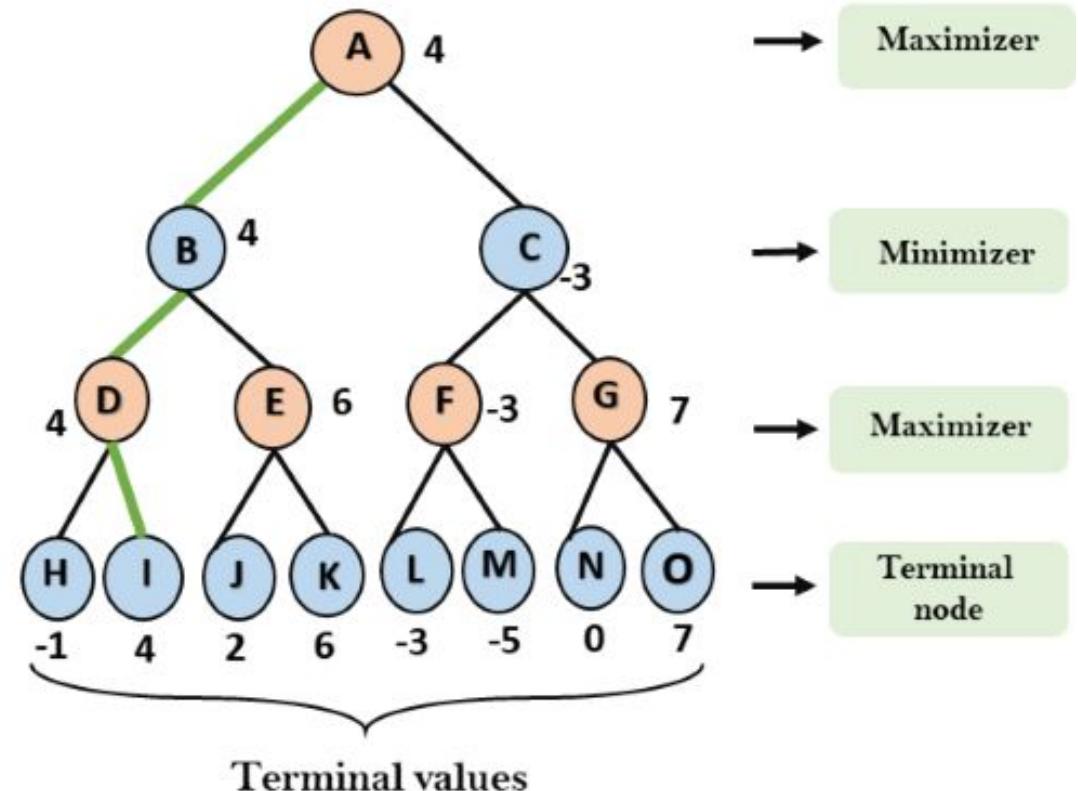
Cont'd

- **Step 3:** In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.
- For node B= $\min(4,6) = 4$
- For node C= $\min (-3, 7) = -3$



Cont'd

- **Step 4:** Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.
- For node A $\max(4, -3) = 4$



That was the complete workflow of the minimax two player game.

Steps for the minimax algorithm

In AI, Minimax algorithm can be stated as follows:

1. Create the entire game tree.
2. Evaluate the scores for the leaf nodes based on the evaluation function.
3. Backtrack from the leaf to the root nodes:
4. At the root node, choose the node with the maximum value and select the respective move.

For Maximizer, choose the node with the maximum score.

For Minimizer, choose the node with the minimum score.

Properties of minimax algorithm in AI

1. The algorithm is complete, meaning in a finite search tree, a solution will be certainly found.
2. It is optimal if both the players are playing optimally.
3. Due to Depth-First Search (DFS) for the game tree, the time complexity of the algorithm is $O(b^d)$, where b is the branching factor and d is the maximum depth of the tree.
4. Like DFS, the space complexity of this algorithm is $O(b^d)$.

Minimax: Advantages & Limitations

Advantages

A thorough assessment of the search space is performed.

Decision making in AI is easily possible.

New and smart machines are developed with this algorithm.

Limitations

Because of the huge branching factor, the process of reaching the goal is slower.

Evaluation and search of all possible nodes and branches degrades the performance and efficiency of the engine.

Both the players have too many choices to decide from.

If there is a restriction of time and space, it is not possible to explore the entire tree.

***** But with Alpha-Beta Pruning, the algorithm can be improved.***

```
function minimax(node, depth, maximizingPlayer)
if depth ==0 or node is a terminal node then
    return static evaluation of node
if MaximizingPlayer then // for Maximizer Player
    maxEva= -infinity
    for each child of node do
        eva= minimax(child, depth-1, false)
        maxEva= max(maxEva,eva) //Maximum of the values
    return maxEva
else // for Minimizer player
    minEva= +infinity
    for each child of node do
        eva= minimax(child, depth-1, true)
        minEva= min(minEva, eva) // minimum of the values
    return minEva
```

Minimax using alpha-beta pruning

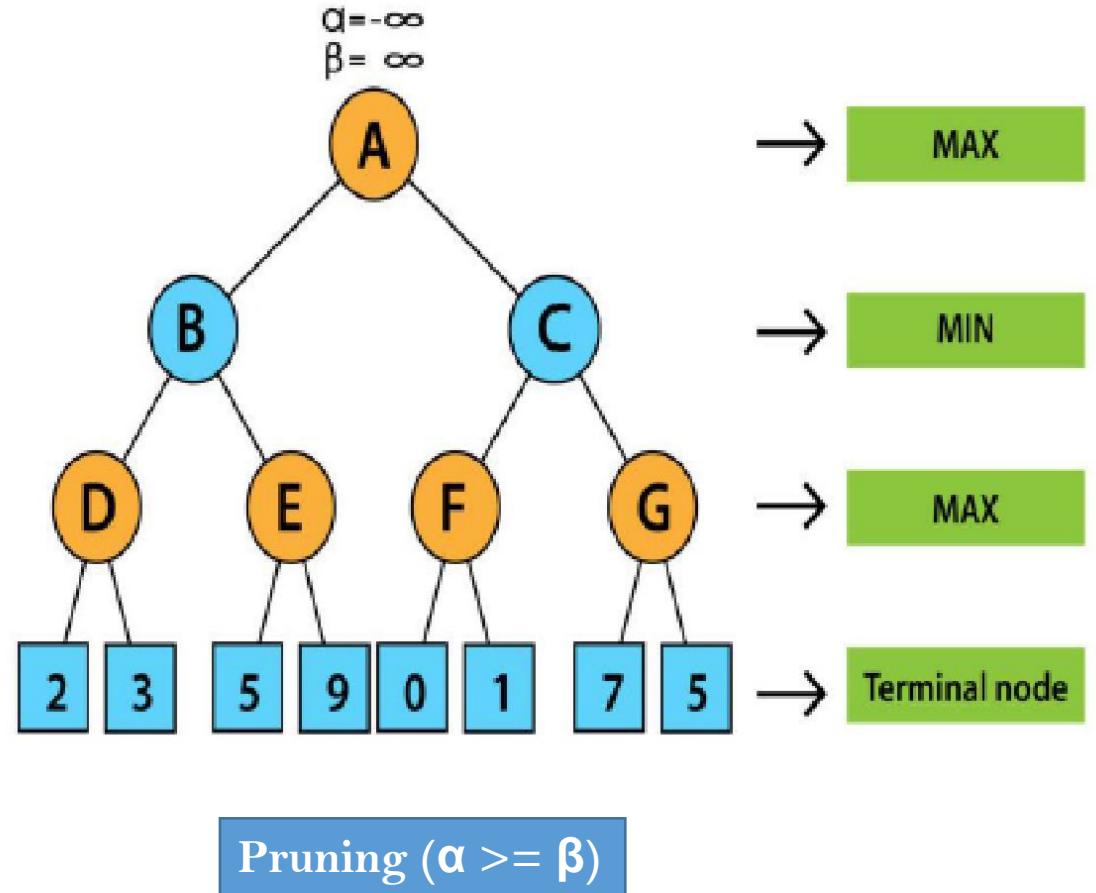
- Minimax procedure is a **depth-first** process
- One path is explored as far as time allows, the **static evolution function** is applied to the game positions at the last step of the path.
- The efficiency of the depth-first search can improve by **branch and bound** technique in which partial solutions that clearly worse than known solutions can abandon early.
- It is necessary to modify the branch and bound strategy to include **two bounds**, one for each of the players.
- This modified strategy called **alpha-beta pruning**.

Key points in Alpha-Beta Pruning

- **Alpha:** Alpha is the best choice or the **highest** value that we have found at any instance along the path of Maximizer. The initial value for alpha is $-\infty$.
- **Beta:** Beta is the best choice or the **lowest** value that we have found at any instance along the path of Minimizer. The initial value for beta is $+\infty$.
- The condition for Alpha-beta **Pruning** is that $\alpha \geq \beta$.
- Each node has to keep track of its alpha and beta values. **Alpha** can be updated only when it's **MAX**'s turn and, similarly, **beta** can be updated only when it's **MIN**'s chance.
- MAX will update only **alpha** values and MIN player will update only **beta** values.
- The node values will be passed to upper nodes instead of values of alpha and beta during go into reverse of tree.
- Alpha and Beta values only be passed to **child** nodes.

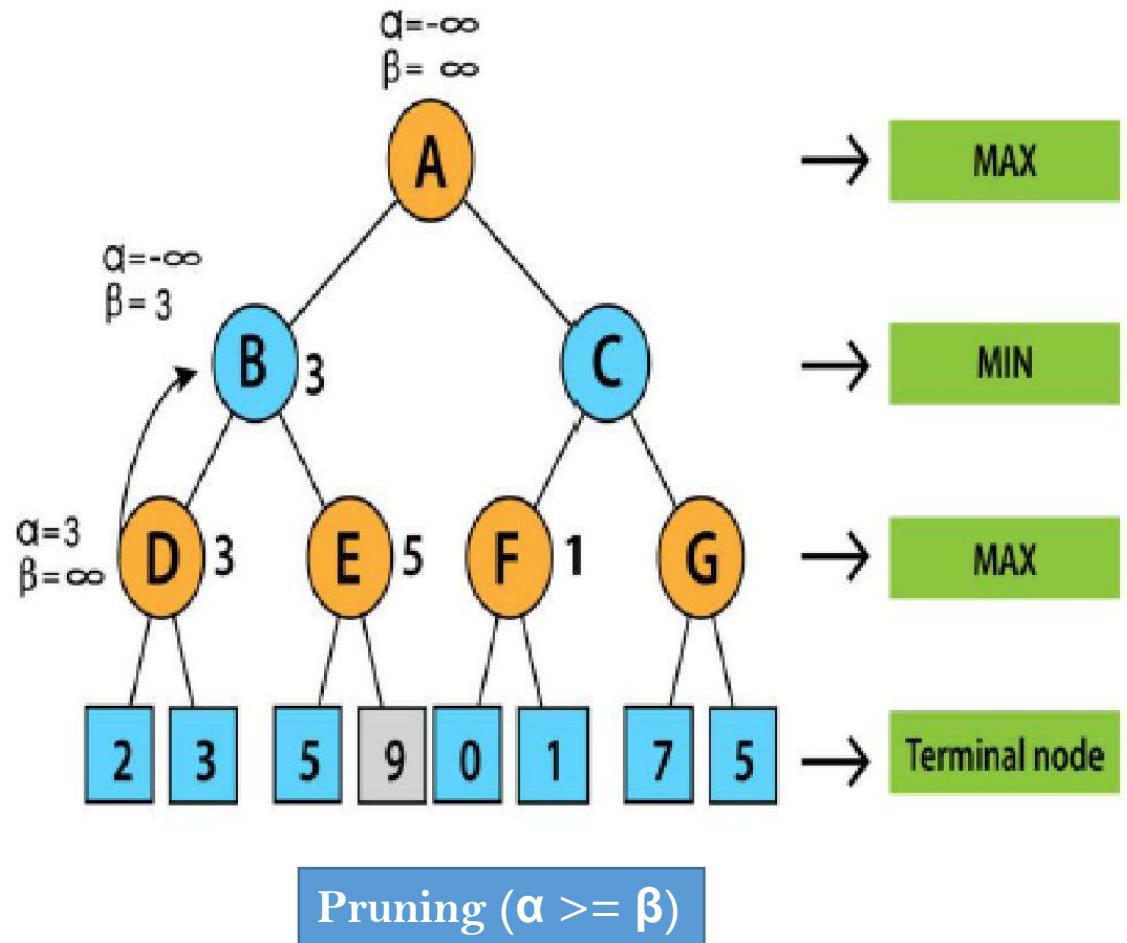
The working of Alpha-Beta Pruning - I

- We will first start with the initial move.
- We will initially define the α and β values as the **worst** case i.e. $\alpha = -\infty$ and $\beta = +\infty$.
- We will **prune** the node only when α becomes greater than or equal to β . ($\alpha \geq \beta$)



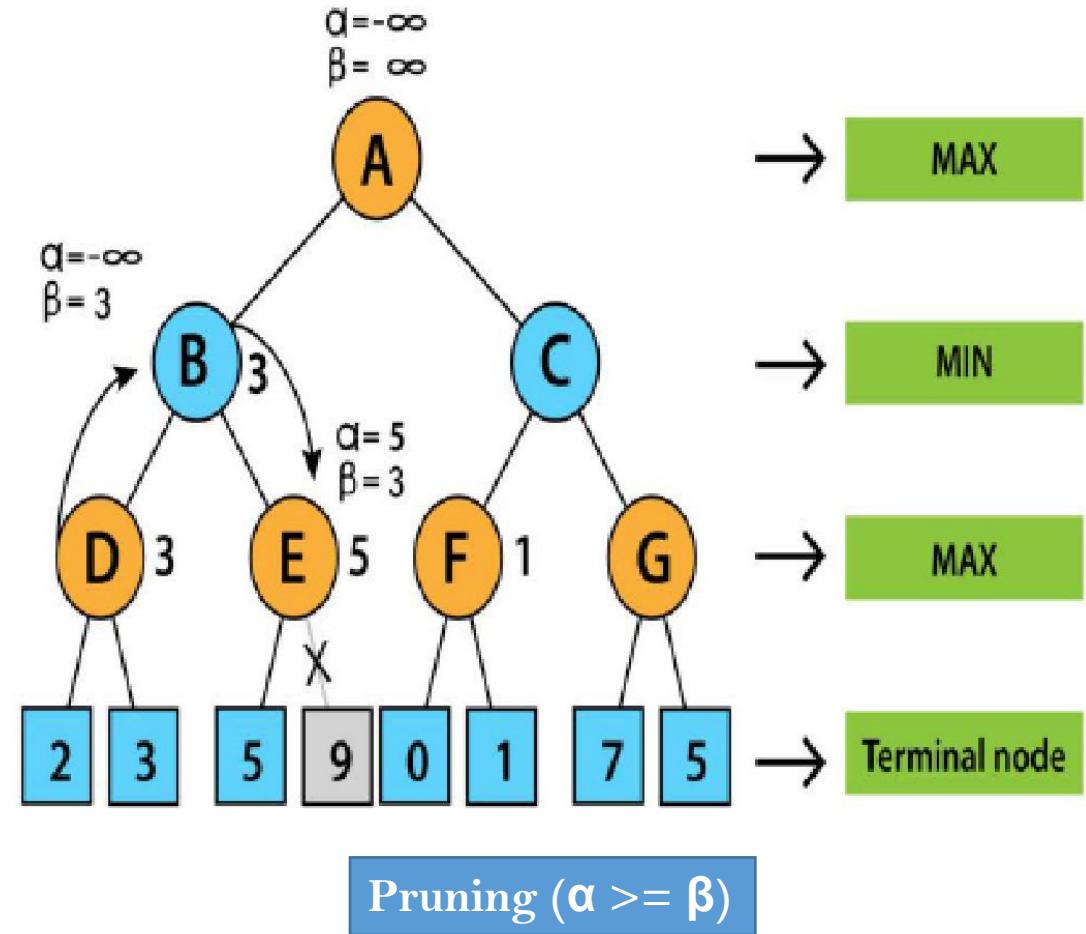
The working of Alpha-Beta Pruning - II

- Since the initial value of α is less than β so we **didn't prune** it. Now it's turn for MAX. So, at node D, value of α will be calculated. The value of α at node D will be max (2, 3). So, value of **α at node D will be 3**.
- Now the next move will be on node B and its turn for MIN now. So, at node B, the value of α, β will be min (3, ∞). So, at node B values will be **alpha= $-\infty$** and $\beta = 3$.
- In the next step, algorithms traverse the next successor of Node B which is node E, and the values of $\alpha= -\infty$, and $\beta = 3$ will also be passed.



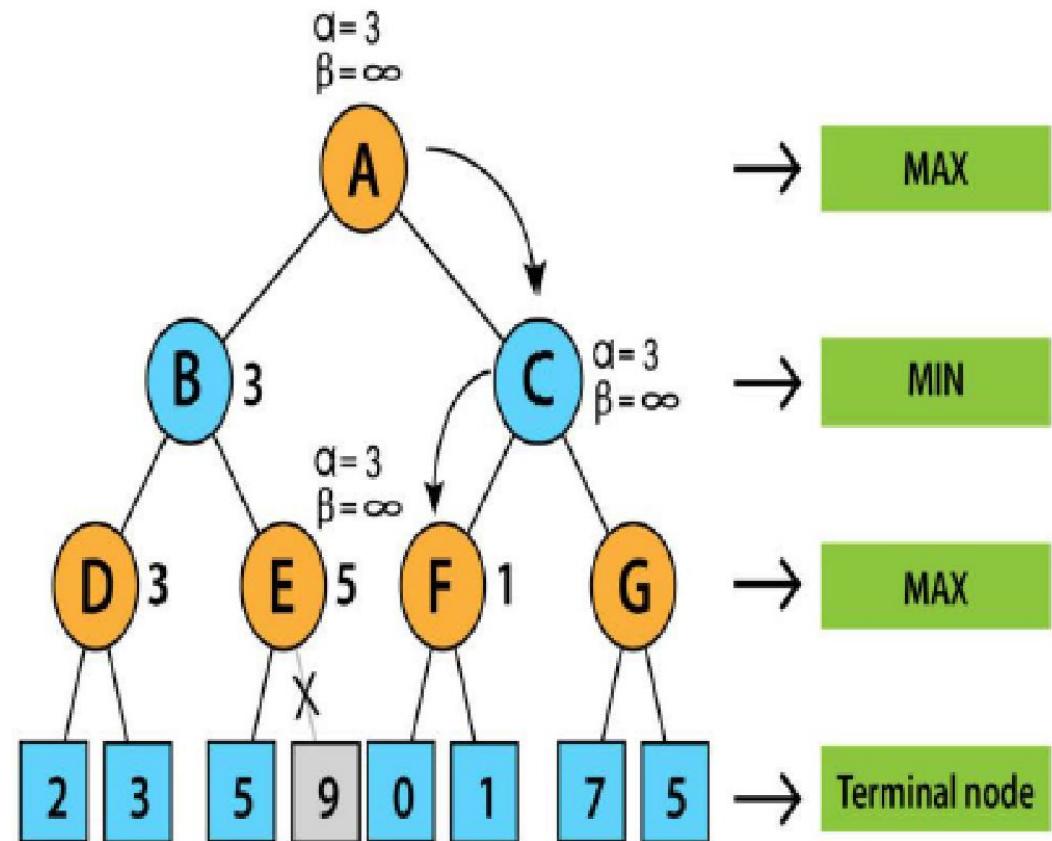
The working of Alpha-Beta Pruning - III

- Now it's turn for MAX. So, at node E we will look for MAX. The current value of α at E is $-\infty$ and it will be compared with 5. So, MAX ($-\infty, 5$) will be 5.
- So, at node E, $\alpha = 5, \beta = 3$.
- Now as we can see that α is greater than β which is satisfying the pruning condition.
- So we can prune the right successor of node E and algorithm will not be traversed and the value at node E will be 5.



The working of Alpha-Beta Pruning - IV

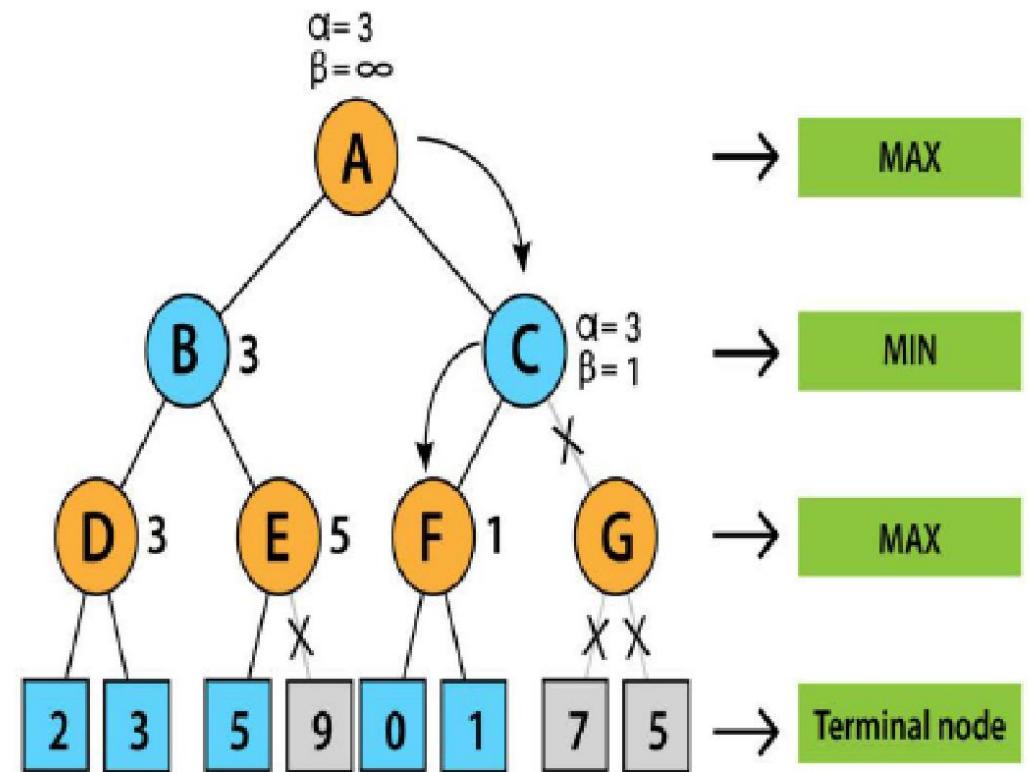
- In the next step the algorithm again comes to node A from node B. At node A, α will be changed to maximum value as MAX ($-\infty, 3$).
- So now the value of α and β at node A will be $(3, +\infty)$ respectively and will be transferred to node C. These same values will be transferred to node F.
- At node F the value of α will be compared to the left branch which is 0.
- So, MAX $(0, 3)$ will be 3 and then compared with the right child which is 1, and $\text{MAX}(3, 1) = 3$ still α remains 3, but the node value of F will become 1.



Pruning ($\alpha \geq \beta$)

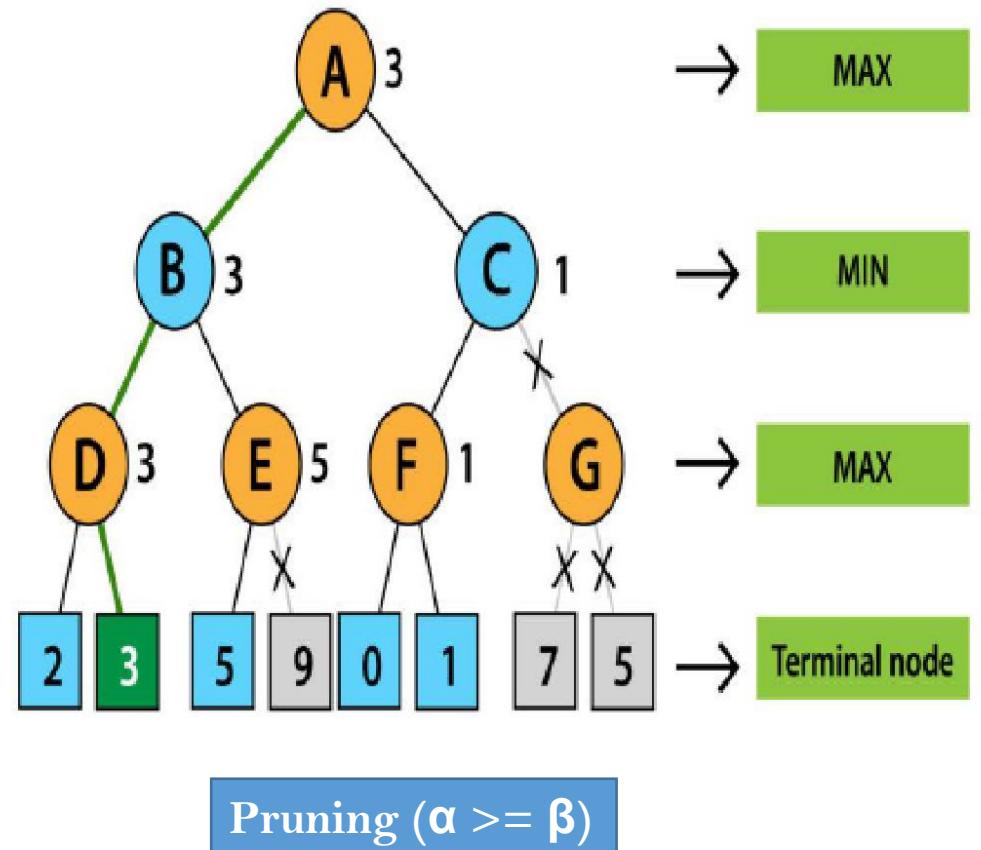
The working of Alpha-Beta Pruning - V

- Now node F will return the node value 1 to C and will compare to β value at C. Now its turn for MIN.
- So, MIN ($+\infty, 1$) will be 1. Now at node C, $\alpha = 3$, and $\beta = 1$ and α is greater than β which again satisfies the pruning condition.
- So, the next successor of node C i.e. G will be pruned and the algorithm didn't compute the entire subtree G.



The working of Alpha-Beta Pruning - VI

- Now, C will return the node value to A and the best value of A will be MAX (1, 3) will be 3.
- The represented tree is the final tree which is showing the nodes which are computed and the nodes which are not computed. So, for this example the optimal value of the maximizer will be 3.



```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
```

```
    if node is a leaf node :
```

```
        return value of the node
```

```
    if isMaximizingPlayer :
```

```
        bestVal = -INFINITY
```

```
        for each child node :
```

```
            value = minimax(node, depth+1, false, alpha, beta)
```

```
            bestVal = max( bestVal, value)
```

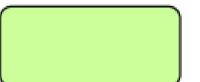
```
            alpha = max( alpha, bestVal)
```

```
            if beta <= alpha:
```

```
                break
```

```
    return bestVal
```

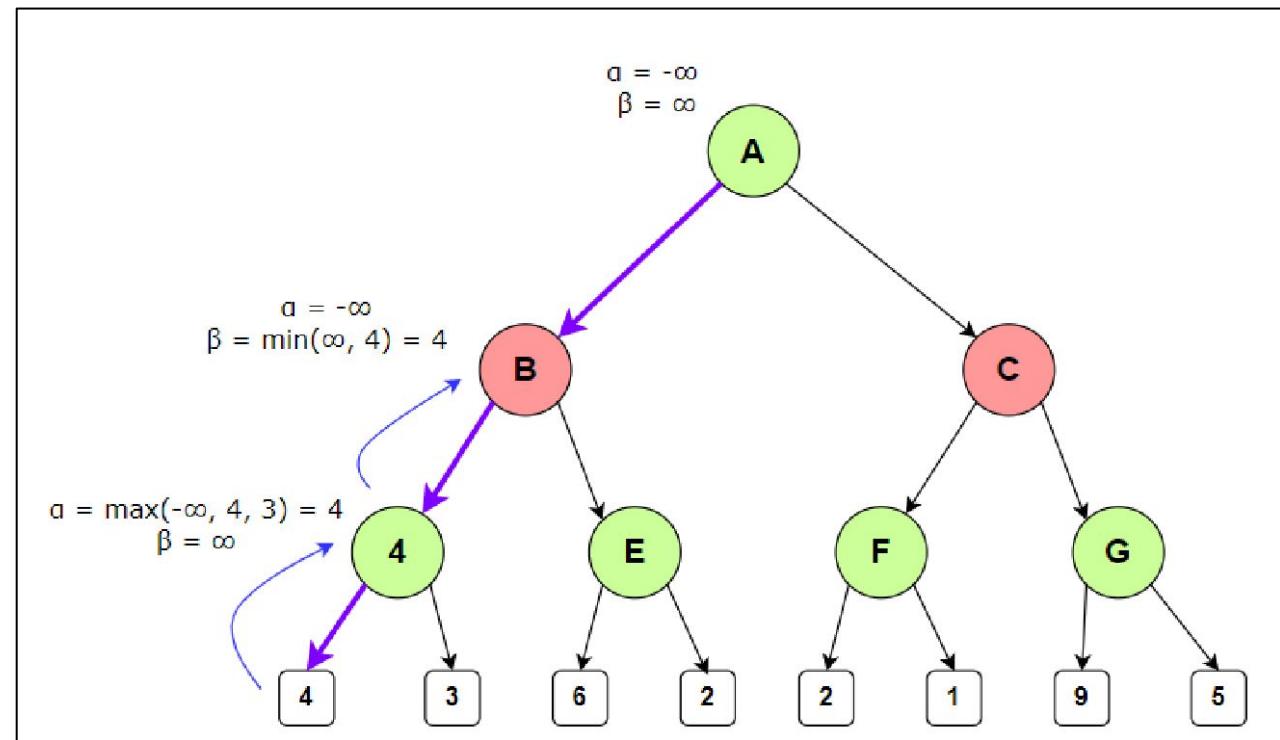
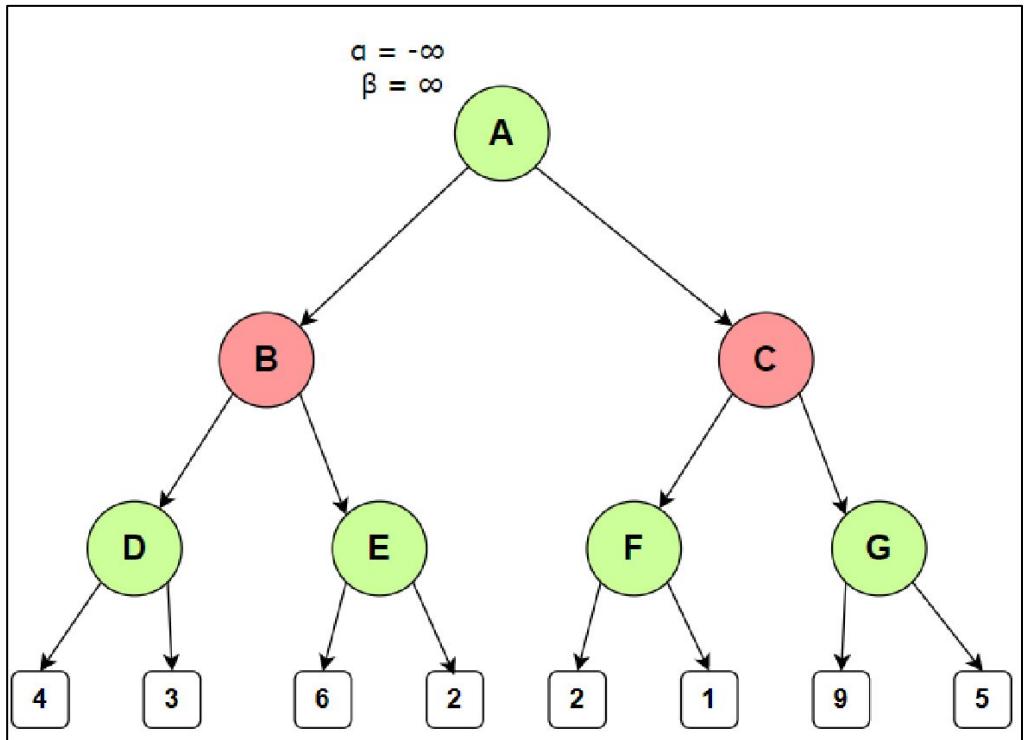
Example: Alpha-Beta Pruning

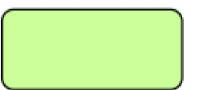


MAX



MIN

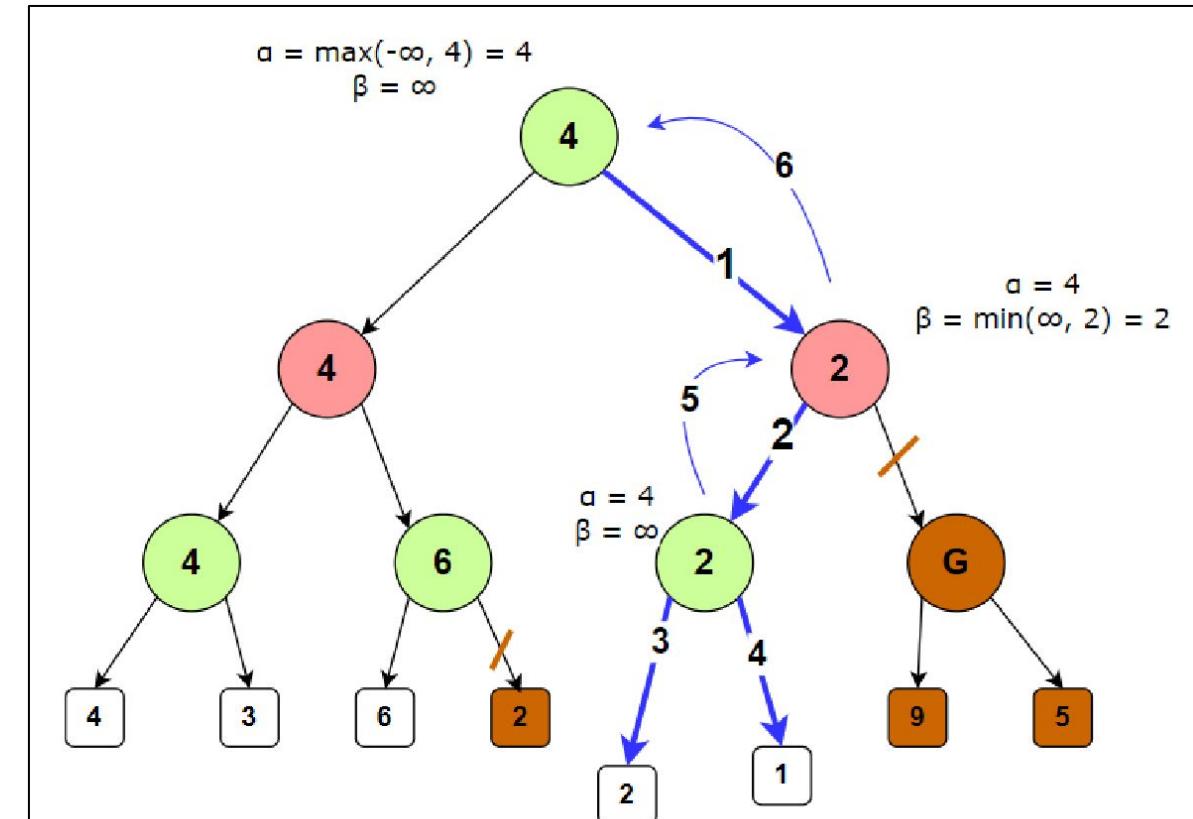
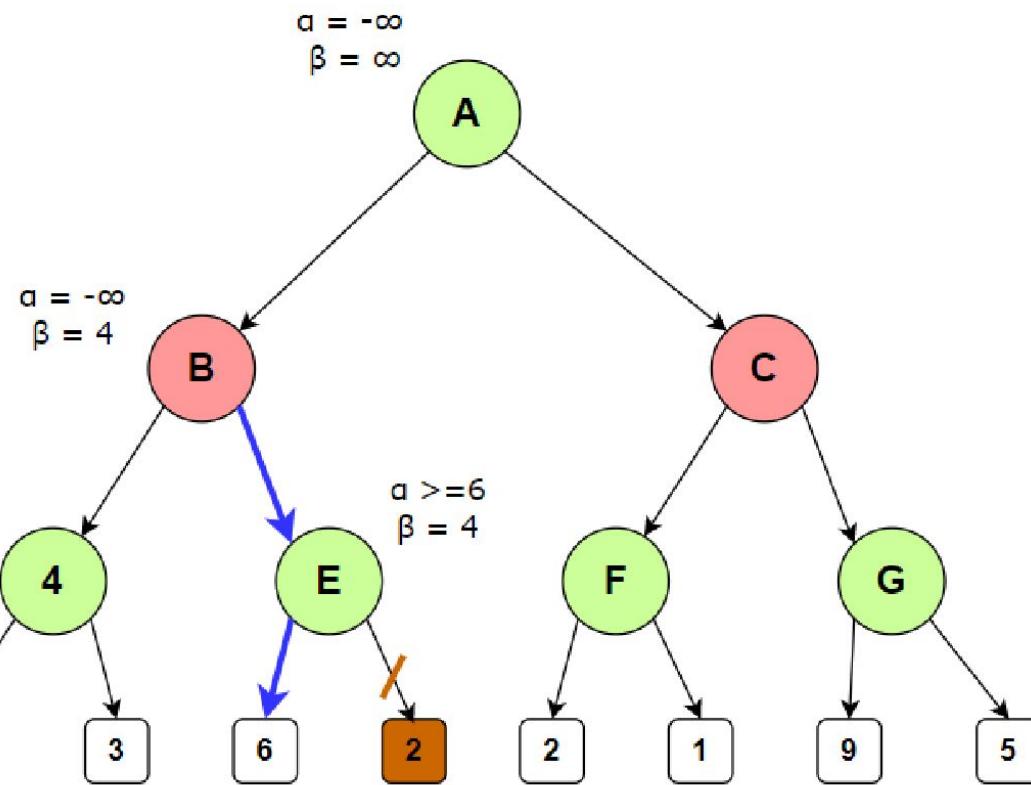




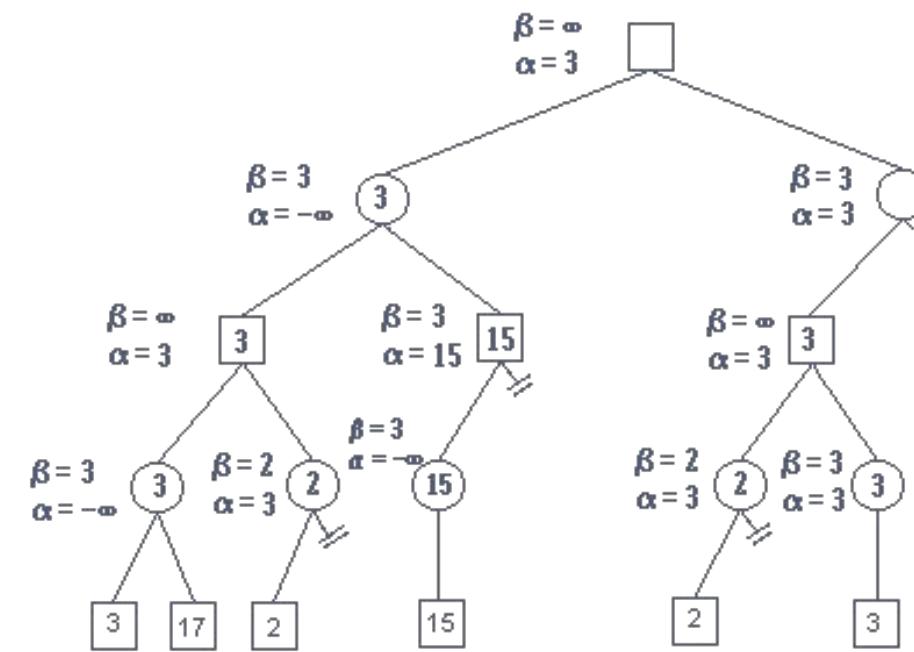
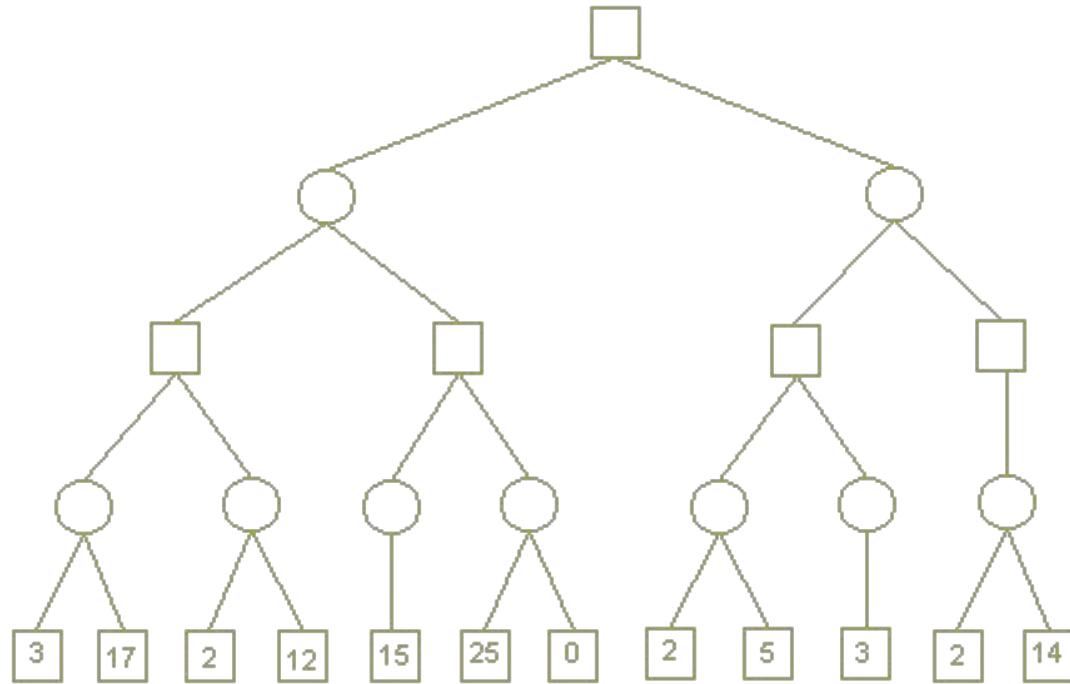
MAX



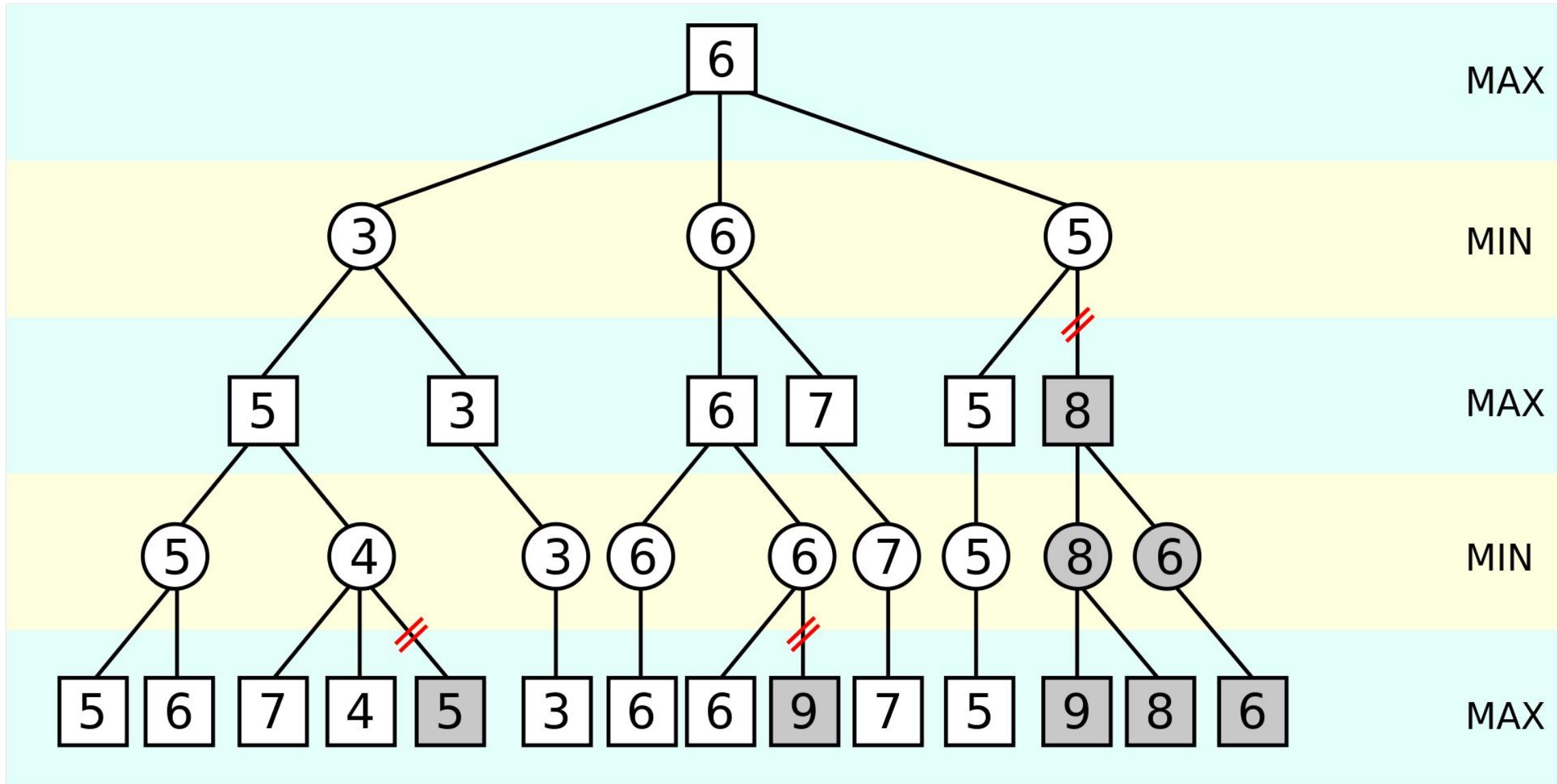
MIN



Another Example: Alpha-Beta Pruning



Another Example: Alpha-Beta Pruning



Additional Refinements (for MiniMax)

- We have seen how game playing domain is different than other domains and how one needs to change the method of search. We have also seen how MiniMax search algorithm is applied and also seen how alpha beta pruning helps improve the efficiency of the MiniMax search algorithm.
- In fact it is possible to improve the functioning of algorithm by providing few **additional refinements** to the process of searching.
- There are a few simple **refinements** one can deploy while playing game playing algorithm.
 - **First** is waiting for stable states where the change of heuristic values across layers is not drastic.
 - **Second** is to use secondary search to avoid horizon effect.
 - **Third** is to use book moves for typical non-search part of the game playing.
 - **Fourth** is to use other than MiniMax algorithm.

Assignment

- Explain and implement Tic-Tac-Toe game using MiniMax Algorithm

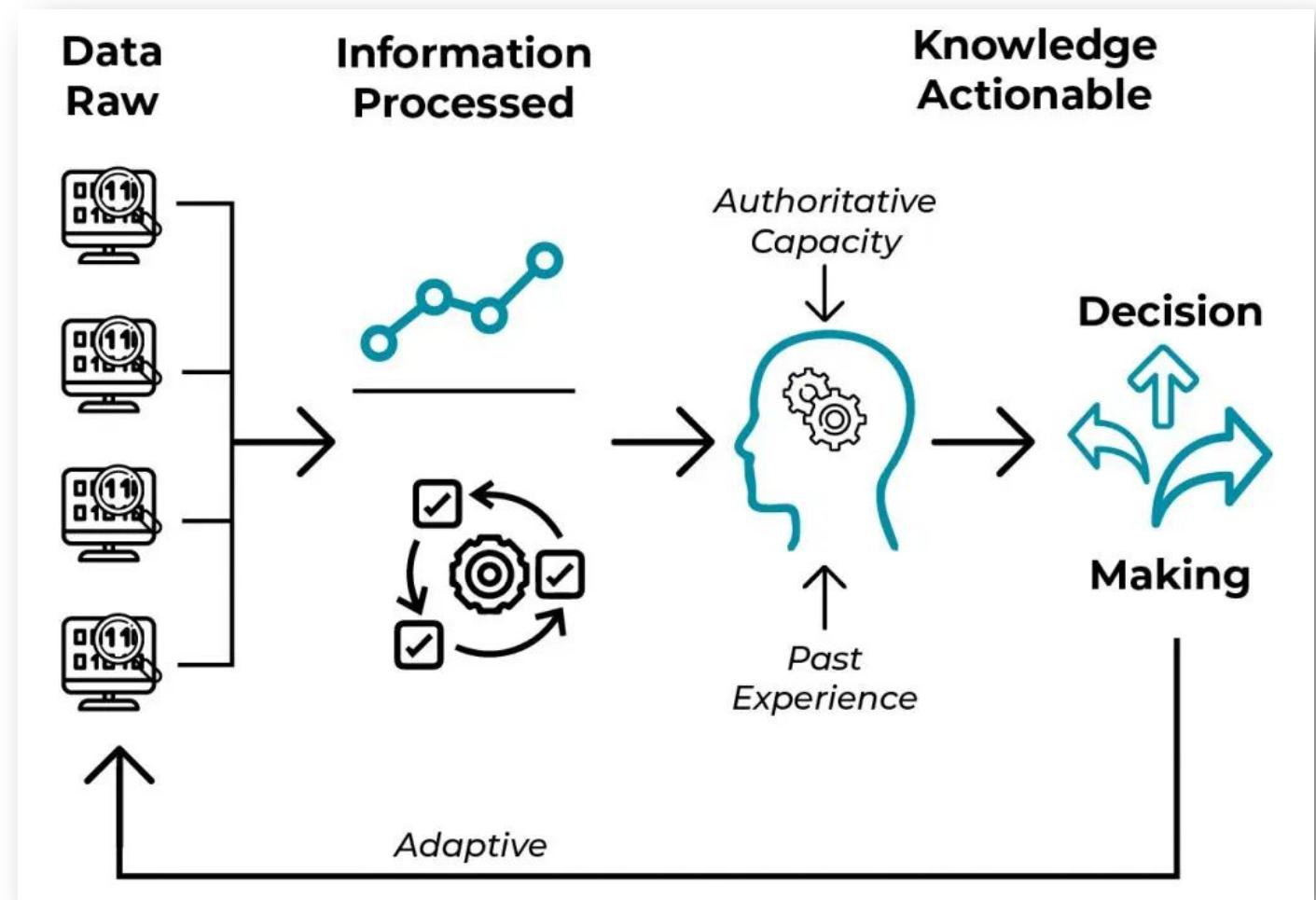
UNIT 5: Knowledge & Reasoning

Part - A

Knowledge Representation

What makes humans different from other animals or machines is our conscience

- **Conscience:** a person's moral sense of right and wrong, viewed as acting as a guide to one's behavior.
- Sum of our **memories**, i.e., all the knowledge we have gathered so far.
- This knowledge makes different personalities and makes humans **behave** differently and take different **actions**.
- To make AI more sophisticated, complex information about world requires, which leads to the concept of **Knowledge Representation** in Artificial Intelligence.



Data, Information and Knowledge

- **DATA** : Primitive verifiable facts

E.g. name of novels available in a library

- **INFORMATION** : Analyzed Data

E.g. the novel that is frequently asked by the member of library is “Harry Potter and the chamber of secrets”.

- **KNOWLEDGE**: Analyzed information that is often used for further information deduction.

E.g. since the librarian knows the name of the novel that is frequently asked by members, s/he will ask for more copies of the novel the next time s/he place an order.

What is Knowledge Representation?

- Knowledge Representation in Artificial Intelligence refers to that concept where **ways are identified** to provide machines with the knowledge so that AI systems can act intelligently.
- We have concepts that are completely unknown to the machine such as intuition, intentions, prejudices, beliefs, judgments, common sense, etc.
- Some knowledge is straight forward such as $3+4=7$ while others are more complex like shot a basketball into the hoop.
- Knowledge representation is **not just storing data** into some database, but it also enables an intelligent machine to **learn** from that **knowledge and experiences** so that it can behave **intelligently** like a human.

What is to be presented?

- **Objects**: all facts about the object, e.g. cars have wheels, piano has keys
- **Events**: the actions which occur, e.g. war, achievements, advancement
- **Performance**: how to perform actions in different situations
- **Facts**: factual description of the world (truths about the real world)
- **Meta-Knowledge**: knowledge of what we know
- **Knowledge-base**: group of information regarding any discipline, field, e.g., knowledge-base for road construction

Types of Knowledge



Declarative Knowledge – It includes concepts, facts, and objects and expressed in a declarative sentence.

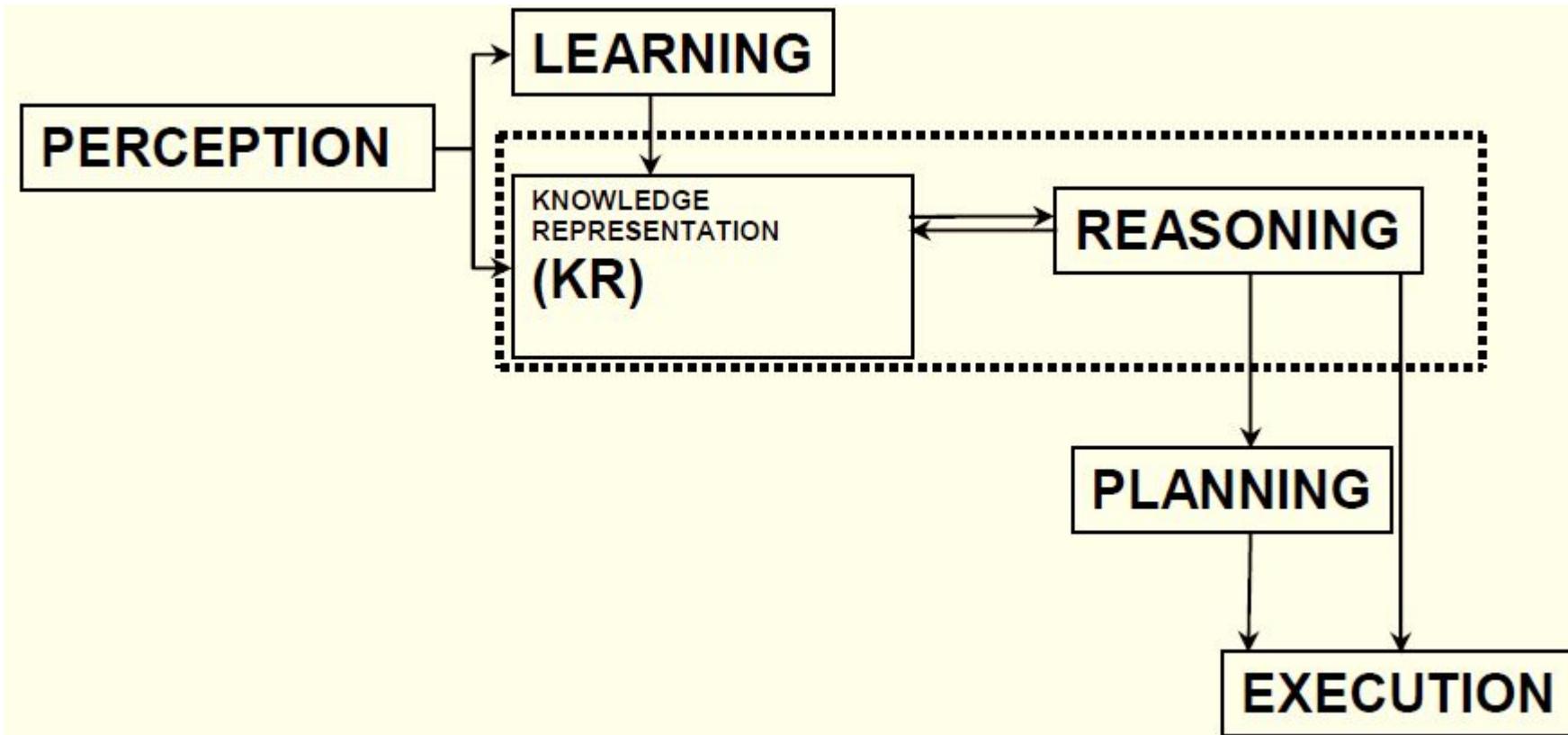
Structural Knowledge – It is a basic problem-solving knowledge that describes the relationship between concepts and objects.

Procedural Knowledge – This is responsible for knowing how to do something and includes rules, strategies, procedures, etc.

Meta Knowledge – Meta Knowledge defines knowledge about other types of Knowledge.

Heuristic Knowledge – This represents some expert knowledge in the field or subject.

Knowledge Representation in AI Cycle



Techniques of Knowledge Representation in AI



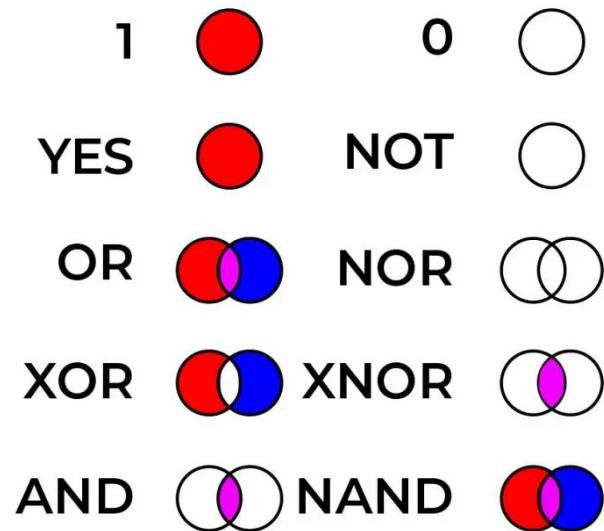
1. Logical Representation

- well-defined syntax with proper rules
- no ambiguity

Logical Representation can be of two types-

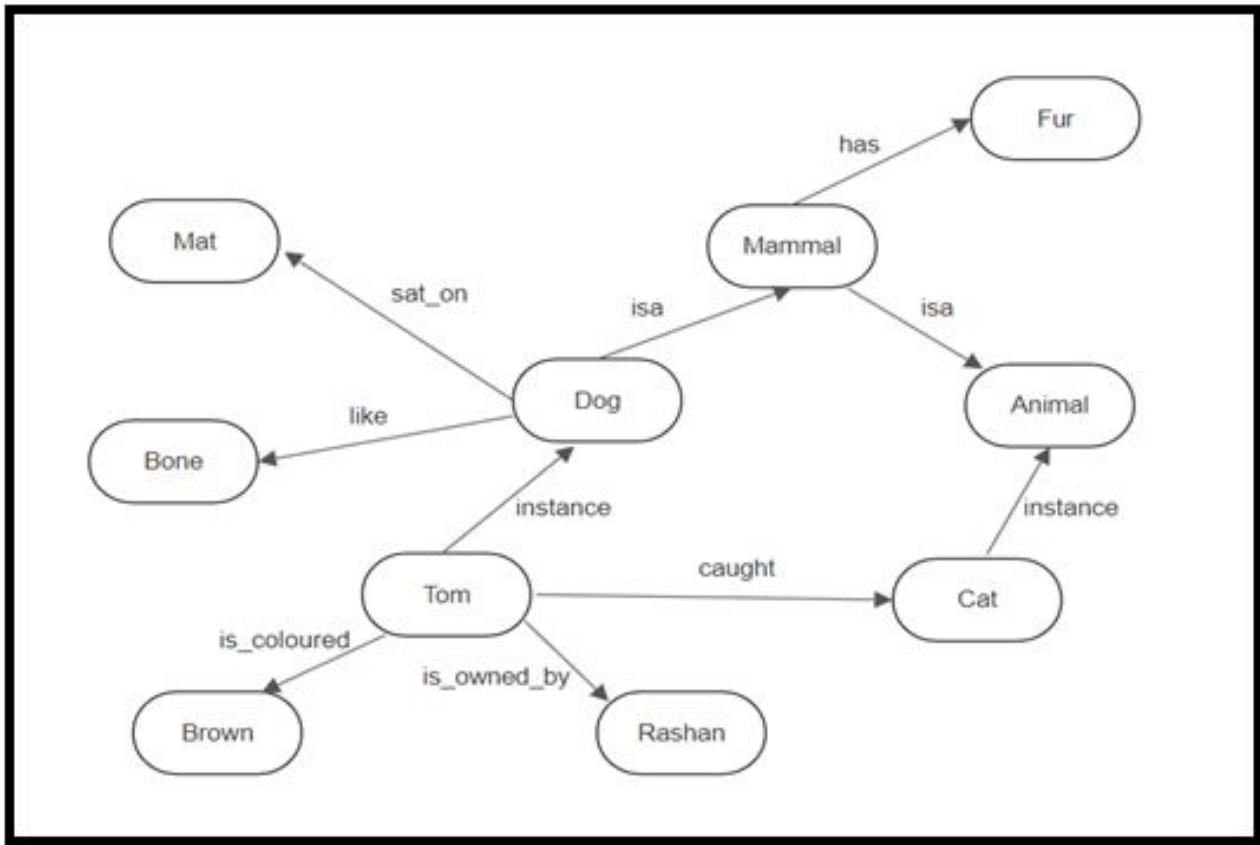
- **Propositional Logic**
 - ✓ Works in a Boolean, i.e., True or False method
- **Predicate Logic** (First-order Logic)
 - ✓ represents the objects in quantifiers and predicates
 - ✓ advanced version of propositional logic

LOGICAL REPRESENTATION



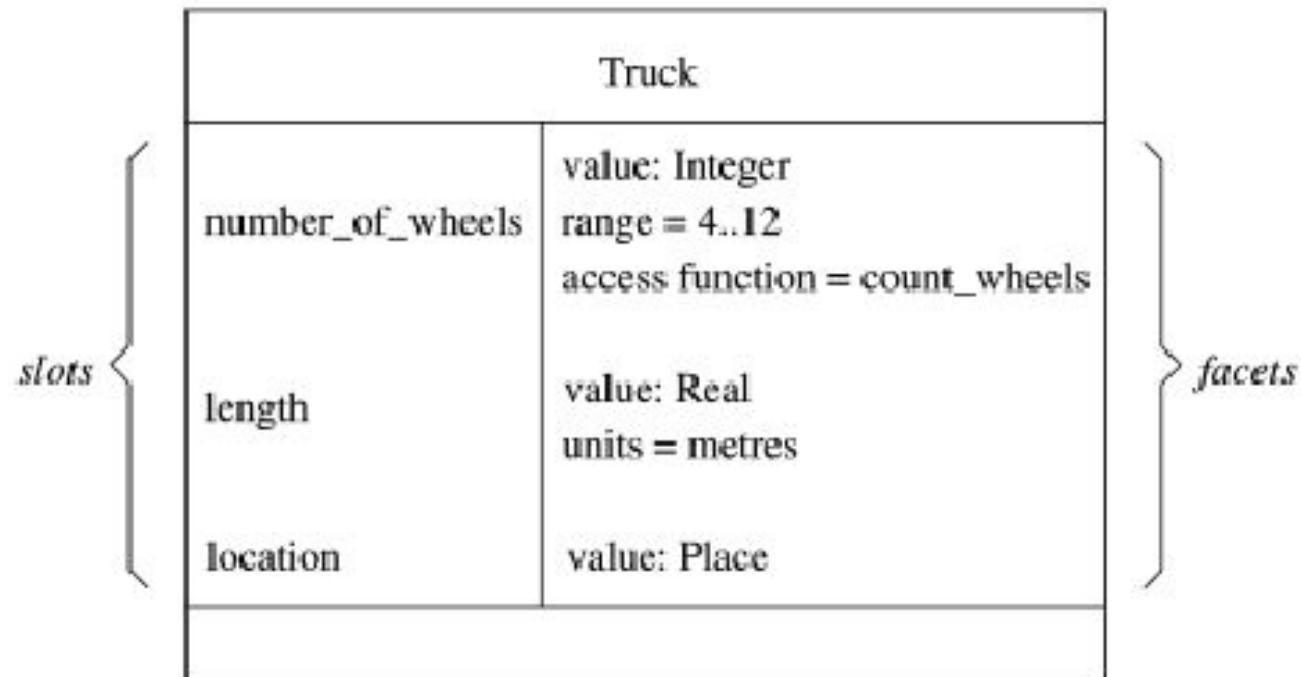
2. Semantic Networks

- Mathematically a semantic net can be defined as a **labeled directed graph**.
- How the objects are **connected**
- Semantic net allows us to perform inheritance reasoning as all members of a class will **inherit** all the properties of superclass
- **Is-A** relationship
- Ex. Google knowledge graph



3. Frame Representation

- A **record like structure** - consists of a collection of attributes and its values to describe an entity in the world.
- AI data structure - which divides knowledge into substructures by representing stereotypes situations.
- Consists of a collection of **slots** and slot values. These slots may be of any type and sizes.
- **Slots have names and values which are called facets.**

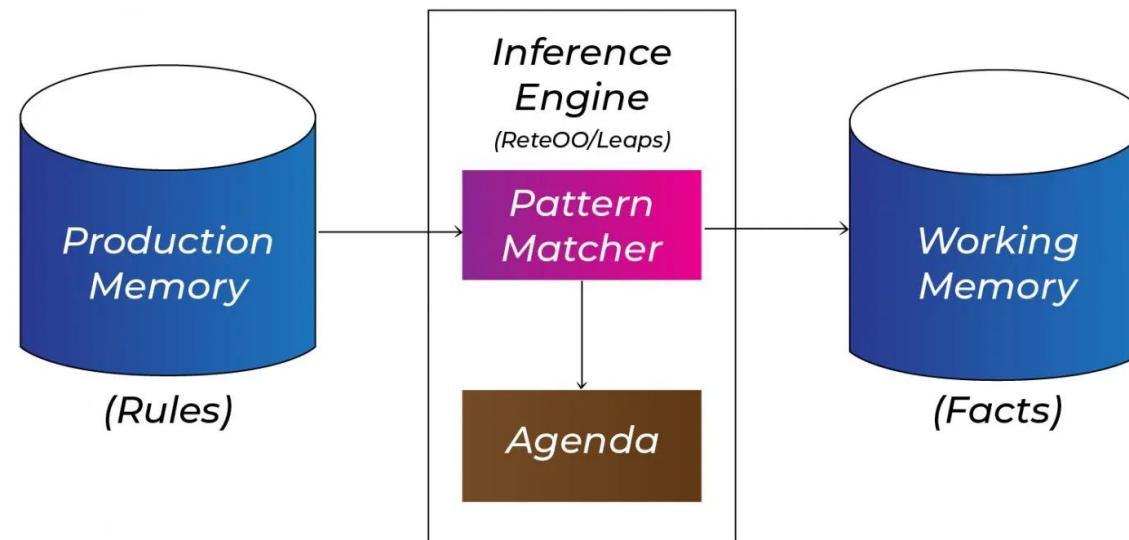


4. Production Rules

- Production rules system consist of (condition, action) pairs which mean, "If condition then action". It has mainly **three** parts:

1. The set of production rules
2. Working Memory
3. The recognize-act-cycle

E.g. water-jug problem



Summary: Knowledge Representation schemes

Logical schemes

- Predicate calculus
- Propositional calculus

Networked schemes

- Semantic nets
- Conceptual graphs

Procedural schemes

- IF..THEN..

Logical schemes

Logic, Propositional Logic, Predicate Logic

Logic

Logic is a formal language for expressing knowledge and ways of reasoning.

Logic is defined by:

- **A set of sentences**

- A sentence is constructed from a set of primitives according to **syntax** rules.

- **A set of interpretations**

- An interpretation gives a **semantic** to primitives. It associates primitives with values.

- **The valuation (meaning) function V**

- Assigns a value (typically the truth value) to a given sentence under some interpretation

Language of numerical constraints:

- **A sentence:**

$$x + 3 \leq z \text{ (where } x, z \text{ are variable symbol)}$$

- **An interpretation:**

$$I : \quad x = 5, z = 2$$

variables mapped to specific real numbers

- **Valuation (meaning) function V:**

$v(x+3 \leq z, I)$ is false for $I: x=5, z=2$

is True for $I: x=5, z=10$

$$V : \text{sentence} \times \text{interpretation} \rightarrow \{\text{True}, \text{False}\}$$

Propositional Logic

True or False but not BOTH

Basics of Propositional Logic

- There are **four** different types of sentences.
- These sentences help us to define propositional logic.

Declarative sentences assert or declare something, like “Richmond is the capital of Virginia.”

Exclamatory sentences are emotional expressions, such as “watch out!”

Interrogative sentences ask questions, like “what time is it?”

Imperative sentences give commands, such as “turn right at the traffic light.”

- And the sentences we are most interested in are **declarative!**
- Because propositions, also called statements, are declarative sentences that are either true or false, but not both.
- This means that every proposition is either true (T) or false (F).

Propositions Examples

- Let's look at a few examples of how we determine the type of sentence illustrated, and if it is a proposition, we will identify its truth value.
- Notice for our last two examples, that while the sentences are declarative, they are not a proposition because we don't know the value of "she" or "x" or "y" — hence, we are unable to determine the truth value for the sentence.
- These types of scenarios are called paradoxes and open sentences, respectively.

	Determine the type of Sentence	If a proposition determine its truth value
5 is a prime number.	Declarative and Proposition	T
8 is an odd number.	Declarative and proposition	F
Did you lock the door?	Interrogative	
Happy Birthday!	Exclamatory	
Jane Austen is the author of Pride and Prejudice.	Declarative and Proposition	T
Please pass the salt.	Imperative	
She walks to school.	Declarative	
$ x + y \leq x + y $	Declarative	

Compound Statement

- The combination of simple statements using logical connectives is called a **compound** statement
- The symbols we use to represent propositional variables and operations are called **symbolic logic**.

Connective	Symbol	Statement
and	\wedge	Conjunction
or	\vee	Disjunction (inclusive or)
or	\oplus, \vee	Exclusive or
not	\sim, \neg, p	Negation
if...then	\rightarrow, \Rightarrow	Implication (conditional statement)
if and only if	$\leftrightarrow, \Leftarrow\Rightarrow$	Biconditional

I make you dinner or dessert
 I only make you dinner
 I only make you dessert
 I make neither dinner nor dessert

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Conjunction

I make you dinner or dessert
 I only make you dinner
 I only make you dessert
 I make neither dinner nor dessert

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Logical Disjunction

For Negation:

P	$\neg P$
True	False
False	True

For Conjunction:

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

For disjunction:

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

For Implication:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

For Biconditional:

P	Q	$P \Leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

Precedence of connectives

- Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators.
- This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Logical equivalence

- Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.
- Let's take two propositions A and B, so for logical equivalence, we can write it as $A \Leftrightarrow B$. In below truth table we can see that column for $\neg A \vee B$ and $A \rightarrow B$, are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators

Commutativity:

$$P \wedge Q = Q \wedge P, \text{ or}$$

$$P \vee Q = Q \vee P.$$

Associativity:

$$(P \wedge Q) \wedge R = P \wedge (Q \wedge R),$$

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

Identity element:

$$P \wedge \text{True} = P,$$

$$P \vee \text{True} = \text{True}.$$

Distributive:

$$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R).$$

Limitations of Propositional logic

1. We cannot represent relations like ALL, some, or none with propositional logic.
Example:
All the students are intelligent.
Some apples are sweet.
2. Propositional logic has limited expressive power.
3. In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Basic facts about propositional logic

- Propositional logic is also called **Boolean** logic as it works on 0 and 1.
- In propositional logic, we use **symbolic** variables to represent the logic, and we can use any symbol for representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either **true** or **false**, but it cannot be both.
- Propositional logic consists of an **object**, relations or **function**, and logical **connectives**.
- These **connectives** are also called **logical** operators.
- The propositions and connectives are the **basic elements** of the propositional logic.
- Connectives can be said as a logical operator which connects **two sentences**.
- A proposition formula which is **always true** is called **tautology**, and it is also called a **valid** sentence.
- A proposition formula which is **always false** is called **Contradiction**.

Inference

- In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, so generating the conclusions from evidence and facts is termed as Inference.
- Terminologies related to inference rules

Implication: It is one of the logical connectives which can be represented as $P \rightarrow Q$. 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 $Q \rightarrow P$.

Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.

Inverse: The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.

Inference – Cont'd

- Hence from the above truth table, we can prove that $P \rightarrow Q$ is equivalent to $\neg Q \rightarrow \neg P$, and $Q \rightarrow P$ is equivalent to $\neg P \rightarrow \neg Q$.

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$\neg Q \rightarrow \neg P$	$\neg P \rightarrow \neg Q$
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:

1. **Modus Ponens** (or Implication Elimination or Affirming the Antecedent), in Propositional Logic, is a rule of inference. It can be summarized as “P implies Q and P is asserted to be true, therefore Q must be true.”

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

Example:

- Statement-1: "If I am sleepy then I go to bed" $\Rightarrow P \rightarrow Q$
Statement-2: "I am sleepy" $\Rightarrow P$
Conclusion: "I go to bed." $\Rightarrow Q$.
Hence, we can say that, if $P \rightarrow Q$ is true and P is true then Q will be true.

An example of an argument that fits the form modus ponens:

- If today is Tuesday, then John will go to work.
- Today is Tuesday.
- Therefore, John will go to work.

2. Modus Tollens:

1. You can't log into the network
2. If you have a current password, then you can log into the network

Therefore, You don't have a current password.

This is an argument of the form:

- $\neg q$
- $p \rightarrow q$
- $\therefore \neg p$

Compare this with modes ponens

1. If you have a current password, then you can log on to the network
2. You have a current password

Therefore: You can log on to the network

This has the form:

- $p \rightarrow q$
- p
- $\therefore q$

Modes Ponens vs. Modes Tollens

To help you understand good and bad way of logical constructions, here are some examples. The basic ideas are:

- There are two **consistent** logical argument constructions: modus ponens ("the way that affirms by affirming") and modus tollens ("the way that denies by denying").
 - **Modus Ponens:** "If A is true, then B is true. A is true. Therefore, B is true."
 - **Modus Tollens:** "If A is true, then B is true. B is not true. Therefore, A is not true."
- There are two related **incorrect** and **inconsist** constructions: affirming the consequent and denying the antecedent.
 - Affirming the Consequent: "If A is true, then B is true. B is true. Therefore, A is true."
 - Denying the Antecedent: "If A is true, then B is true. A is not true. Therefore, B is not true."
- "If it is a car, then it has wheels. It is a car. Therefore, it has wheels." (Modus Ponens - CORRECT)
- "If it is a car, then it has wheels. It does not have wheels. Therefore, it is not a car." (Modus Tollens - CORRECT)
- "If it is a car, then it has wheels. It has wheels. Therefore, it is a car." (Affirming the Consequent - INCORRECT.)
- "If it is a car, then it has wheels. It is not a car. Therefore, it does not have wheels." (Denying the Antecedent - INCORRECT)

Examples:

- "If Xyrplex is 9, Guffaw is 1. Guffaw is 2. Therefore, Xyrplex is not 9."
(ANSWER: Modus Tollens)
- "If Nagini is a Snake, Snape is a goner. Nagini is a snake. Therefore, Snape is a goner."
(ANSWER: Modus Ponens)
- "If Blurts are Flurts, Green is Grue. Green is Grue. Therefore, Blurts are Flurts."
(ANSWER: INCORRECT – Affirming the Consequent)
- "If Sagan has hair, Tyson is awesome. Sagan has hair. Therefore, Tyson is awesome."
(ANSWER: Modus Ponens)
- "If Fordham brings a ram, Peruna will kick. Fordham did not bring a ram. Therefore, Peruna did not kick."
(ANSWER: INCORRECT – Denying the Antecedent)

3. Hypothetical Syllogism:

- **Syllogism** : an instance of a form of reasoning in which a conclusion is drawn from two given or assumed propositions (premises); a common or middle term is present in the two premises but not in the conclusion, which may be invalid (e.g. *all dogs are animals; all animals have four legs; therefore all dogs have four legs*).
- The **Hypothetical Syllogism** rule state that if $P \rightarrow R$ is true whenever $P \rightarrow Q$ is true, and $Q \rightarrow R$ is true.

Example:

- Statement-1: If you have my home key then you can unlock my home. $P \rightarrow Q$
- Statement-2: If you can unlock my home then you can take my money. $Q \rightarrow R$
- Conclusion: If you have my home key then you can take my money. $P \rightarrow R$

4. Disjunctive Syllogism:

- The Disjunctive syllogism rule state that if $P \vee Q$ is true, and $\neg P$ is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism:
$$\frac{P \vee Q, \quad \neg P}{Q}$$

Example:

- Statement-1: Today is Sunday or Monday. $\Rightarrow P \vee Q$
- Statement-2: Today is not Sunday. $\Rightarrow \neg P$
- Conclusion: Today is Monday. $\Rightarrow Q$

5. Addition:

- The Addition rule is one the common inference rule, and it states that If P is true, then $P \vee Q$ will be true.

Notation of Addition:
$$\frac{P}{P \vee Q}$$

Example:

- Statement-1: I have a vanilla ice-cream. $\Rightarrow P$
- Statement-2: I have Chocolate ice-cream. $\Rightarrow Q$
- Conclusion: I have vanilla or chocolate ice-cream. $\Rightarrow (P \vee Q)$

6. Simplification:

- The simplification rule state that if $P \wedge Q$ is true, then Q or P will also be true. It can be represented as:

Notation of Simplification rule:
$$\frac{P \wedge Q}{Q}$$
 Or
$$\frac{P \wedge Q}{P}$$

7. Resolution:

- The Resolution rule state that if $P \vee Q$ and $\neg P \wedge R$ is true, then $Q \vee R$ will also be true. It can be represented as

Notation of Resolution
$$\frac{P \vee Q, \quad \neg P \wedge R}{Q \vee R}$$

Translation

Assume the following sentences:

- It is not sunny this afternoon and it is colder than yesterday.
- We will go swimming only if it is sunny.
- If we do not go swimming then we will take a canoe trip.
- If we take a canoe trip, then we will be home by sunset.

Denote:

- p = It is sunny this afternoon
- q = it is colder than yesterday
- r = We will go swimming
- s= we will take a canoe trip
- t= We will be home by sunset

Translation

Assume the following sentences:

- It is not sunny this afternoon and it is colder than yesterday.

$$\neg p \wedge q$$

- We will go swimming only if it is sunny.

$$r \rightarrow p$$

- If we do not go swimming then we will take a canoe trip.

$$\neg r \rightarrow s$$

- If we take a canoe trip, then we will be home by sunset.

$$s \rightarrow t$$

- p = It is sunny this afternoon
- q = it is colder than yesterday
- r = We will go swimming
- s= we will take a canoe trip
- t= We will be home by sunset

Knowledge Representation and Reasoning

(putting all together)

Knowledge Representation & Reasoning

Knowledge representation is the study of how knowledge about the world can be represented and what kinds of reasoning can be done with that knowledge.

We will discuss two different systems that are commonly used to represent knowledge in machines and perform algorithmic reasoning:

- **Propositional calculus**
- **Predicate calculus**

Propositional Calculus

In propositional calculus,

- features of the world are represented by **propositions**,
- relationships between features (constraints) are represented by **connectives**.

Example:

$$\text{LECTURE_BORING} \wedge \text{TIME_LATE} \Rightarrow \text{SLEEP}$$

This expression in propositional calculus represents the fact that for some agent in our world, if the features LECTURE_BORING and TIME_LATE are both true, the feature SLEEP is also true.

Propositional Calculus

You see that the language of propositional calculus can be used to represent aspects of the world.

When there are

- a **language**, as defined by a syntax,
- **inference rules** for manipulating sentences in that language, and
- **semantics** for associating elements of the language with elements of the world,

then we have a system called **logic**.

The Language

Atoms (and atomic sentences):

The atoms T and F and all strings that begin with a capital letter, for instance, P, Q, LECTURE_BORING, and so on.

Literals: P and $\neg P$

Connectives:

- \vee “or”
- \wedge “and”
- \Rightarrow “implies” or “if-then”
- \neg “not”

Sentences: can be formed using the connectives

The Language

Syntax of well-formed formulas (wffs):

- Any atom is a wff.
- If ω_1 and ω_2 are wffs, so are
 - $\omega_1 \wedge \omega_2$ (conjunction)
 - $\omega_1 \vee \omega_2$ (disjunction)
 - $\omega_1 \Rightarrow \omega_2$ (implication)
 - $\neg \omega_1$ (negation)
- There are no other wffs.

The Language

- Atoms and negated atoms are called **literals**.
- In $\omega_1 \Rightarrow \omega_2$, ω_1 is called the **antecedent**, and ω_2 is called the **consequent** of the implication.
- **Examples of wffs (sentences):**

$$(P \wedge Q) \Rightarrow \neg P$$

$$P \Rightarrow \neg P$$

$$P \vee P \Rightarrow P$$

$$(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$$

$$\neg \neg P$$

- The **precedence order** of the above operators is
 $\neg \wedge \vee \Rightarrow$
For example, $\neg P \vee Q \Rightarrow R$ means $((\neg P) \vee Q) \Rightarrow R$.

The Language

- **General Form:**

$$((q \wedge \neg r) \Rightarrow s) \wedge \neg(s \wedge t)$$

- **Conjunction Normal Form (CNF):**

$$(\neg q \vee r \vee s) \wedge (\neg s \vee \neg t)$$

Set Notation: $\{(\neg q, r, s), (\neg s, \neg t)\}$ □ set of clauses

Empty clauses (): false

- **Binary Clauses: 1 or 2 literals per clause**

$$(\neg q \vee r) \quad (\neg s \vee \neg t)$$

- **Horn Clauses: 0 or 1 positive literals per clause**

$$(\neg q \vee \neg r \vee s) \quad (\neg s \vee \neg t)$$

Rules of Inference

We use **rules of inference** to generate new wffs from existing ones.

One important rule is called **modus ponens** or the **law of detachment**. It is based on the tautology $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$. We write it in the following way:

$$\begin{array}{c} P \\ P \Rightarrow Q \\ \hline \therefore Q \end{array}$$

The two **hypotheses** P and $P \Rightarrow Q$ are written in a column, and the **conclusion** below a bar, where \therefore means “therefore”.

Rules of Inference

$$\frac{P}{\therefore P \vee Q}$$

Addition

$$\frac{P \wedge Q}{\therefore P}$$

Simplification

$$\frac{P \\ Q}{\therefore P \wedge Q}$$

Conjunction

$$\frac{\neg Q \\ P \Rightarrow Q}{\therefore \neg P}$$

Modus tollens

$$\frac{P \Rightarrow Q \\ Q \Rightarrow R}{\therefore P \Rightarrow R}$$

Hypothetical syllogism

$$\frac{P \vee Q \\ \neg P}{\therefore Q}$$

Disjunctive syllogism

Proofs

The sequence of wffs $\{\omega_1, \omega_2, \dots, \omega_n\}$ is called a **proof** (or a **deduction**) of ω_n from a set of wffs Δ iff (if and only if) each ω_i in the sequence is either in Δ or can be inferred from one or more wffs earlier in the sequence by using one of the **rules of inference**.

If there is a proof of ω_n from Δ , we say that ω_n is a **theorem** of the set Δ . We use the following notation:

$$\Delta \vdash \omega_n$$

In this notation, we can also indicate the set of inference rules \mathcal{R} that we use:

$$\Delta \vdash_{\mathcal{R}} \omega_n$$

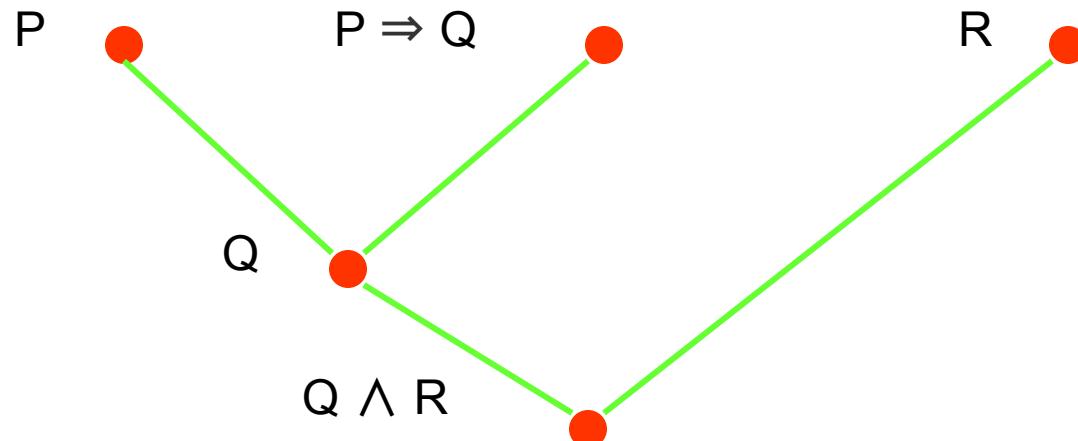
Proofs

Example:

Given a set of wffs $\Delta = \{P, R, P \Rightarrow Q\}$, the following sequence is a proof of $Q \wedge R$ given the inference rules that we discussed earlier:

$$\{P, P \Rightarrow Q, Q, R, Q \wedge R\}$$

Tree representation:



Semantics

- In propositional logic, we **associate** atoms with propositions about the world.
- We thereby specify the **semantics** of our logic, giving it a **“meaning”**.
- Such an association of atoms with propositions is called an **interpretation**.
- In a given interpretation, the proposition associated with an atom is called the **denotation** of that atom.
- Under a given interpretation, atoms have values – **True** or **False**. We are willing to accept this idealization (otherwise: **fuzzy logic**).

Semantics

Example:

“Sonali is either intelligent or a good actor.
If Sonali is intelligent, then she can count
from 1 to 10.
Sonali can only count from 1 to 2.
Therefore, Sonali is a good actor.”

Step 1: $\neg C$ Hypothesis
Step 2: $I \Rightarrow C$ Hypothesis
Step 3: $\neg I$ Modus Tollens Steps 1 & 2
Step 4: $A \vee I$ Hypothesis
Step 5: A Disjunctive Syllogism
Steps 3 & 4

Propositions:

I: “Sonali is intelligent.”
A: “Sonali is a good actor.”
C: “Sonali can count from 1 to 10.”

Conclusion: **A** (“Sonali is a good actor.”)

Semantics

Let us consider an **agent** which cause action when it gets some input from the sensors

- The sensors inform the agent about a **set of features** of the outside world.
- We can **associate propositions** with these features such as “There is a wall in the cell on the right hand side.”
- The sensors tell the agent whether each of these propositions is currently true or false.
- This propositional information that is available to the agent is called its **knowledge base**.

Semantics

- We say that an interpretation **satisfies** a wff if the wff is assigned the value **True** under that interpretation.
- An interpretation that satisfies a wff is called a **model** of that wff.
- An interpretation that satisfies each wff in a **set of wffs** is called a model of that set.
- The **more wffs** we have that describe the world, the **fewer models** there are.
- This means that the more we know about the world, the less uncertainty there is.

Semantics

- If no interpretation satisfies a wff (or a set of wffs), the wff is said to be **inconsistent** or **unsatisfiable**, for example, $P \wedge \neg P$.
- A wff is said to be **valid** if it has value **True** under all interpretations of its constituent atoms, for example, $P \vee \neg P$.
- Neither valid wffs nor inconsistent wffs tell us anything about the world.

Semantics

- Two wffs are said to be **equivalent** if and only if their truth values are identical under all interpretations. For two equivalent wffs ω_1 and ω_2 we write $\omega_1 \equiv \omega_2$.
- If a wff ω has value True under all of those interpretations for which each of the wffs in a set Δ has value True, then we say that
 - Δ **logically entails** ω
 - ω **logically follows** from Δ
 - ω is a **logical consequence** of Δ
 - $\Delta \models \omega$

Soundness and Completeness

- **Soundness:** if something is provable, it is valid.

- If, for any set of wffs Δ and wff ω , $\Delta \vdash_{\mathcal{R}} \omega$ implies $\Delta \models \omega$, we say that the set of inference rules \mathcal{R} is **sound**.

- **Completeness:** if something is valid, it is provable.

- If, for any set of wffs Δ and wff ω , it is the case that whenever $\Delta \models \omega$, there exists a proof of ω from Δ using the set of inference rules \mathcal{R} , we say that \mathcal{R} is **complete**.
- When \mathcal{R} is **sound and complete**, we can determine whether one wff follows from a set of wffs by searching for a **proof** instead of using a **truth table** (increased efficiency).

Resolution

- Multiple rules of inference can be combined into one rule, called **resolution**.
- A **clause** is a set of literals, which is a short notation for the disjunction of all the literals in the set. For example, the wff $\{P, Q, \neg R\}$ is the same as the wff $P \vee Q \vee \neg R$.
- The **empty clause** $\{ \}$ (or NIL) is equivalent to F.

Resolution

Resolution rule for the propositional calculus:

From $\{\lambda\} \cup \Sigma_1$ and $\{\neg \lambda\} \cup \Sigma_2$ (where Σ_1 and Σ_2 are sets of literals and λ is an atom), we can infer $\Sigma_1 \cup \Sigma_2$, which is called the **resolvent** of the two clauses.

The atom λ is the **atom resolved upon**, and the process is called **resolution**.

Examples:

- Resolving $R \vee P$ and $\neg P \vee Q$ yields $R \vee Q$.
In other words: $\{R, P\}, \{\neg P, Q\}$ yields $\{R, Q\}$.
- Resolving R and $\neg R \vee P$ yields P .
In other words: $\{R\}, \{\neg R, P\}$ yields $\{P\}$.

Resolution

And yet Another Example:

Resolving $P \vee Q \vee \neg R$ with $P \vee W \vee \neg Q \vee R$

- on Q yields $P \vee \neg R \vee R \vee W$, which is **True**.
- on R yields $P \vee Q \vee \neg Q \vee W$, which is also **True**.

You cannot resolve on Q and R simultaneously!

Note:

- Any **set of wffs** containing ω and $\neg \omega$ is unsatisfiable,
i.e. if one wff is the negation of another wff in that set,
it is impossible that all wffs in that set are true.
- Any **clause** that contains λ and $\neg \lambda$ has value **True**
regardless of the value of λ .

Converting wffs to Conjunctions of Clauses

Resolution is a powerful tool for algorithmic inference, but we can only apply it to **conjunctions of clauses** (conjunctive normal form, CNF).

So is there a way to **convert** any wff into such a conjunction of clauses?

Fortunately, there is such a way, allowing us to apply resolution to **any** wff.

Converting wffs to Conjunctions of Clauses

Example: $\neg(P \Rightarrow Q) \vee (R \Rightarrow P)$.

Step 1: Eliminate implication operators:

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

Step 2: Reduce the scopes of \neg operators by using DeMorgan's laws and eliminating double \neg operators:

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

Step 3: Convert to CNF by using the associative and distributive laws:

$$(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P), \text{ and then}$$

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

Convert into CNF (1st step in Resolution)

It is a canonical formula, that is, any logic can be converted into canonical formula.

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Eliminate \Leftrightarrow , replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$:

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

2. Eliminate \Rightarrow , replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2} \vee P_{2,1}) \vee B_{1,1}) \text{ i.e. part 1 is clause but part 2 is not}$$

3. Move \neg inwards using de Morgan's rules and double-negation:

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

4. Apply distributive law (\vee over \wedge):

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

Examples: Convert into CNF

1. $(A \rightarrow B) \rightarrow C$

2. $A \rightarrow (B \rightarrow C)$

3. $(A \rightarrow B) \vee (B \rightarrow A)$

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

1. $(A \rightarrow B) \rightarrow C$

Answer:

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

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

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

2. $A \rightarrow (B \rightarrow C)$

Answer:

$$\neg A \vee \neg B \vee C$$

3. $(A \rightarrow B) \vee (B \rightarrow A)$

Answer:

$$\underline{(\neg A \vee B) \vee (\neg B \vee A)}$$

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

Answer:

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

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

More Examples: Convert into CNF

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

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

Answer:

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

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

Answer:

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

Resolution

- If the unicorn is mythical, then it is immortal, but if the unicorn is not mythical, it is a mammal. If the unicorn is either immortal or mammal, then it is horned.
- Prove: the unicorn is horned.
- **Correctness:**
- If $S_1 \vdash_R S_2$ then $S_1 \models S_2$
- **Refutation Completeness:**
- If S is unsatisfiable then $S \vdash_R ()$, that is S is *false* (empty clause)

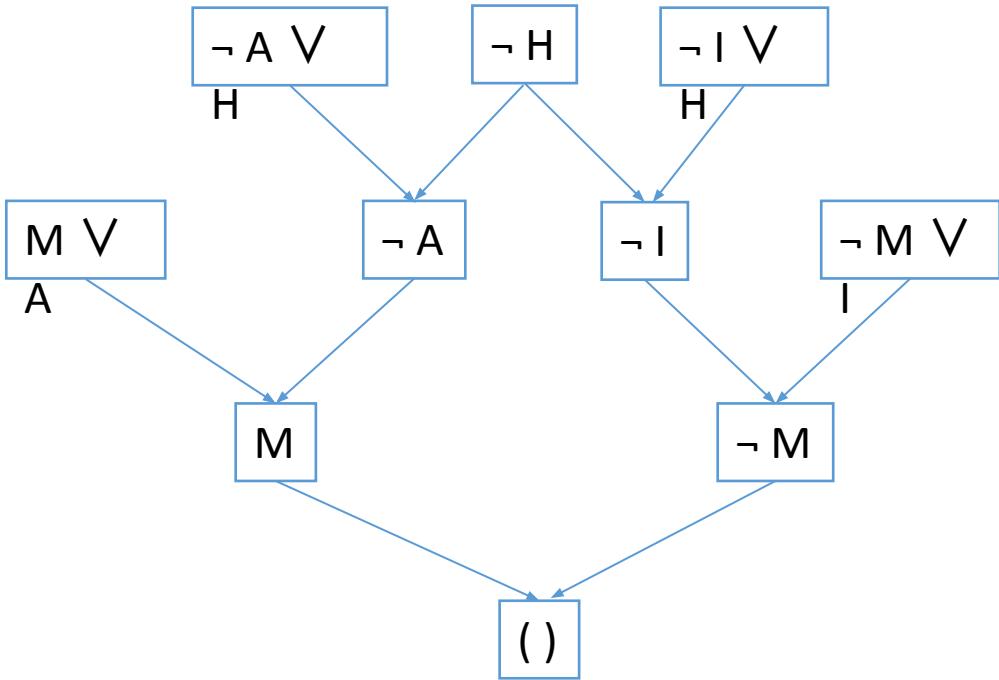
Gödel's theorem: you can not have a proof system which is sufficiently expressive and also complete, i.e. you can not build something which will prove everything

Resolution Example

- If the unicorn is mythical, then it is immortal, but if the unicorn is not mythical, it is a mammal. If the unicorn is either immortal or mammal, then it is horned.
- **Prove:** the unicorn is horned.

- ✓ M: mythical
- ✓ I: immortal
- ✓ A: mammal
- ✓ H: horned

first statement, $M \Rightarrow I$ clause is $\neg M \vee I$
second statement, $M \vee A$
the last statement is,
 $A \Rightarrow H$, i.e., $\neg A \vee H$ and $I \Rightarrow H$, i.e., $\neg I \vee H$



Resolution Algorithm

Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

function PL-RESOLUTION(KB, α) **returns** true or false

inputs: KB , the knowledge base, a sentence in propositional logic
 α , the query, a sentence in propositional logic

$clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$

$new \leftarrow \{ \}$

loop do

for each C_i, C_j **in** $clauses$ **do**

$resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)

if $resolvents$ contains the empty clause **then return** true

$new \leftarrow new \cup resolvents$

if $new \subseteq clauses$ **then return** false

$clauses \leftarrow clauses \cup new$

UNIT 5: Knowledge & Reasoning

PART - B

Predicate Logic

Also known as

First-order logic, quantificational logic, first-order predicate calculus

- In this topic, we begin exploring one particular way of representing facts — the language of logic.
- The logical formalism is appealing because it immediately suggests a powerful way of deriving new knowledge from old — mathematical deduction.
- In this formalism, we can conclude that a new statement is true by proving that it follows from the statements that are already known.
- Thus the idea of a proof, as developed in mathematics as a rigorous way of demonstrating the truth of an already believed proposition, can be extended to include deduction as a way of deriving answers to questions and solutions to problems.

What is Predicate Logic?

- It is a **collection** of formal systems used in mathematics, philosophy, linguistics, and computer science.
- First-order logic uses **quantified** variables over non-logical objects, and allows the use of sentences that contain variables, so that rather than propositions such as "**Socrates is a man**", one can have expressions in the form "**there exists x such that x is Socrates and x is a man**", where "**there exists**" is a quantifier, while **x** is a variable.
- This **distinguishes** it from propositional logic, which does not use quantifiers or relations; in this sense, propositional logic is the foundation of first-order logic.

First-order logic

- First-order logic is another way of **knowledge representation** in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently **expressive** to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic** or **First-order predicate logic**.
- First-order logic is a powerful language that develops information about the **objects** in a more easy way and can also express the **relationship** between those objects.
- First-order logic (like natural language) does not only assume that the world contains **facts** like propositional logic but also assumes the following things in the world:
 - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, ...
 - **Relations:** It can be **unary** relation such as: red, round, is adjacent, or **n-ary** relation such as: the sister of, brother of, has color, comes between
 - **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic has **two** main parts:
 - Syntax
 - Semantics

Proposition Vs. Predicate Logic

- The predicate refers to the property that the subject of the statement can take on.

$$\underbrace{i^2 + 3k}_{\text{subject}} \geq \underbrace{10 + j}_{\text{predicate}}$$

- We can assign values to each variable — thus, creating a true or false proposition, as seen in the example below.

Question:

$$P(x) : x + y \geq 6$$

Possible Solutions:

Let $P(7,1)$ $P(\textcolor{red}{7},\textcolor{green}{1}) : (\textcolor{red}{7}) + (\textcolor{green}{1}) \geq 6$ True propositional statement
 $8 \geq 6$

Let $P(3,2)$ $P(\textcolor{blue}{3},\textcolor{pink}{2}) : (\textcolor{blue}{3}) + (\textcolor{pink}{2}) \geq 6$ False propositional statement
 $5 \not\geq 6$

Proposition Vs. Predicate Logic – Cont'd

- But this isn't **always effective** or **helpful**, as we ultimately want the predicate to be factual **over a range** of elements, not just the ones that we've hand-selected.
- How do we do this?
- We use **quantifiers** to create propositional functions. This process is called **quantification**.
- Quantifiers express the extent to which a predicate is **true over a range** of elements.
- For example, imagine we have the statement: "Every person who is 18 years of age or older is able to vote. Sarah is 18 years old."
- While it would seem logical to conclude that Sarah would then be able to vote legally, propositional logic alone is ill-equipped with handling quantified variables, namely, what does "every person" really mean?
- As the range of possibilities is too broad, we need to apply one of two types of **quantifiers** to help us achieve our goal:
 - **Universal Quantifier:** \forall
 - **Existential Quantifier:** \exists

Basic Elements of First-order logic

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf,
Connectives	\wedge , \vee , \neg , \Rightarrow , \Leftrightarrow
Equality	$=$
Quantifier	\forall , \exists

Points to remember:

The main connective for universal quantifier \forall is implication \rightarrow .

The main connective for existential quantifier \exists is and \wedge .

Universal Quantifier

- Universal Quantification is the proposition that a property is true for all the values of a variable in a particular domain, sometimes called the domain of discourse or the universe of discourse.
- Usually, universal quantification takes on any of the following forms:
 - $P(x)$ is true for all values of x
 - For all x , $P(x)$
 - For each x , $P(x)$
 - For every x , $P(x)$
 - Given any x , $P(x)$
- And is symbolically denoted

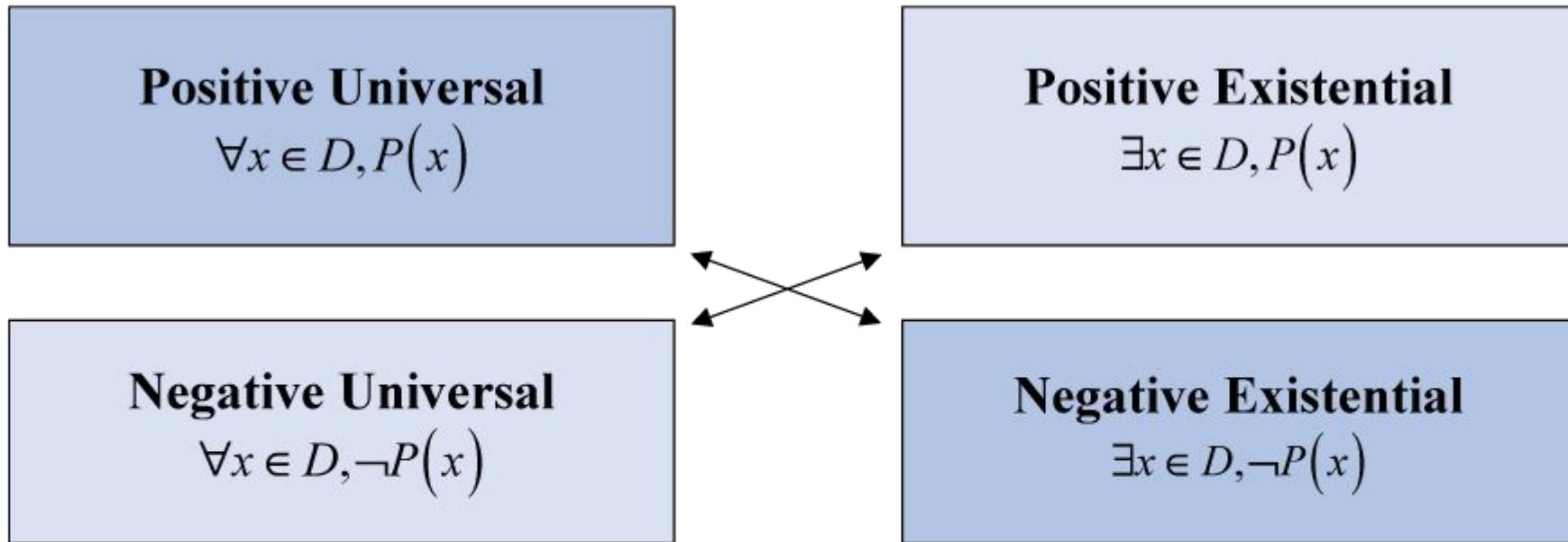
$\forall x \in D, P(x)$ “for all x , belonging to domain D , $p(x)$ is true”

Existential Quantifier

- Existential quantification is the proposition that a property is true for some value in a particular domain.
- Customarily, existential quantification takes on one of the following forms:
 - There exists an x such that $P(x)$
 - There exists an element x in the domain such that $P(x)$
 - For some x , $P(x)$
 - There is some x such that $P(x)$
- And is symbolically denoted

$\exists x \in D, P(x)$ “there exists x , belonging to domain D , such that $p(x)$ is true”

Universal vs. Existential Predicate



$$\neg(\forall x \in D, P(x)) \equiv \exists x \in D, \neg P(x)$$

$$\neg(\exists x \in D, P(x)) \equiv \forall x \in D, \neg P(x)$$

Some Examples of FOL using quantifier

- All birds fly.

$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$

- Every man respects his parent.

$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$

- Some boys play cricket.

$\exists x \text{ boys}(x) \wedge \text{play}(x, \text{cricket}).$

- Not all students like both Mathematics and Science.

$\neg \forall (x) [\text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science})].$

WFF - Well Formed Formula

- Every wizard who is not Voldemort is mortal

$$\forall x ((\text{WIZARD}(x) \wedge \neg x = v) \rightarrow \text{MORTAL}(x))$$

For every entity **x**, if **x** is a wizard and **x** is not equal to Voldemort, then **x** is mortal

CAT(x) The **1-place** predicate CAT and the variable **x** combine to form a wff.

x is a cat

B. LIKE(x,s) The **2-place** predicate LIKE combines with the variable **x** and individual **s** to form a wff.

x liked Sue

C. LIKE(x,y) The 2-place predicate LIKE combines with the variables **x** and **y** to form a wff.

x liked y

Examples: Predicate logic / First order logic

1. Some green dragon is sleeping
2. No green dragon is sleeping
3. Every green dragon is sleeping
4. Not every green dragon is sleeping

1. $\exists x ((G(x) \wedge D(x)) \wedge S(x))$
2. $\neg \exists x ((G(x) \wedge D(x)) \wedge S(x))$
3. $\forall x ((G(x) \wedge D(x)) \rightarrow S(x))$
4. $\neg \forall x ((G(x) \wedge D(x)) \rightarrow S(x))$

1. Some dragon is sleeping or twitching
2. No dragon is sleeping or twitching
3. Every dragon is sleeping or twitching
4. Not every dragon is sleeping or twitching

1. $\exists x [D(x) \wedge (S(x) \vee T(x))]$
2. $\neg \exists x [D(x) \wedge (S(x) \vee T(x))]$
3. $\forall x [D(x) \rightarrow (S(x) \vee T(x))]$
4. $\neg \forall x [D(x) \rightarrow (S(x) \vee T(x))]$

Using Predicate Logic

- Marcus was a man.

man(Marcus)

- Marcus was a Pompeian.

Pompeian(Marcus)

- All Pompeians were Romans.

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

- Caesar was a ruler.

ruler(Caesar)

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

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

Representing INSTANCE and ISA Relationships

- The **first part** of the figure contains the representations we have already discussed. In these representations, class membership is represented with **unary predicates** (such as Roman), each of which corresponds to a class.
- The **second part** of the figure contains representations that use the **instance** predicate explicitly.
- The **third part** contains representations that use both the instance and isa predicates explicitly. The use of the **isa** predicate simplifies the representation of sentence 3, but it requires that one additional axiom (shown here as number 6) be provided.

- | |
|--|
| <ol style="list-style-type: none">1. Man(Marcus).2. Pompeian(Marcus).3. $\forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x).$4. ruler(Caesar).5. $\forall x: \text{Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar}).$ |
| <ol style="list-style-type: none">1. instance(Marcus, man).2. instance(Marcus, Pompeian).3. $\forall x: \text{instance}(x, \text{Pompeian}) \rightarrow \text{instance}(x, \text{Roman}).$4. instance(Caesar, ruler).5. $\forall x: \text{instance}(x, \text{Roman}). \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar}).$ |
| <ol style="list-style-type: none">1. instance(Marcus, man).2. instance(Marcus, Pompeian).3. isa(Pompeian, Roman)4. instance(Caesar, ruler).5. $\forall x: \text{instance}(x, \text{Roman}). \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar}).$6. $\forall x: \forall y: \forall z: \text{instance}(x, y) \wedge \text{isa}(y, z) \rightarrow \text{instance}(x, z).$ |

Computable Functions and Predicates

- To express simple facts, such as the following greater-than and less-than relationships:
 $gt(1,0)$ $It(0,1)$ $gt(2,1)$ $It(1,2)$ $gt(3,2)$ $It(2,3)$
- It is often also useful to have computable functions as well as computable predicates.
- Thus we might want to be able to evaluate the truth of $gt(2 + 3,1)$
- To do so requires that we first compute the value of the plus function given the arguments 2 and 3, and then send the arguments 5 and 1 to gt .

Consider the following set of facts, again involving Marcus:

1) Marcus was a man.

man(Marcus)

2) Marcus was a Pompeian.

Pompeian(Marcus)

3) Marcus was born in 40 A.D.

born(Marcus, 40)

4) All men are mortal.

$\forall x: \text{man}(x) \rightarrow \text{mortal}(x)$

5) All Pompeians died when the volcano erupted in 79 A.D.

$\text{erupted(volcano, 79)} \wedge \forall x : [\text{Pompeian}(x) \rightarrow \text{died}(x, 79)]$

6) No mortal lives longer than 150 years.

$\forall x: \forall t1: \forall t2: \text{mortal}(x) \wedge \text{born}(x, t1) \wedge \text{gt}(t2 - t1, 150) \rightarrow \text{died}(x, t2)$

7) It is now 2022.

$\text{now} = 1991$

So, Now suppose we want to answer the question “**Is Marcus alive?**”

- The statements suggested here, there may be two ways of deducing an answer.
- Either we can show that Marcus is dead because he was killed by the volcano or we can show that he must be dead because he would otherwise be more than 150 years old, which we know is not possible.
- Also, As soon as we attempt to follow either of those paths rigorously, however, we discover, just as we did in the last example, that we need some additional knowledge. For example, our statements talk about dying, but they say nothing that relates to being alive, which is what the question is asking.

So we add the following facts:

8) Alive means not dead.

$\forall x: \forall t: [\text{alive}(x, t) \rightarrow \neg \text{dead}(x, t)] \wedge [\neg \text{dead}(x, t) \rightarrow \text{alive}(x, t)]$

9) If someone dies, then he is dead at all later times.

$\forall x: \forall t1: \forall t2: \text{died}(x, t1) \wedge \text{gt}(t2, t1) \rightarrow \text{dead}(x, t2)$

So, based on these facts, we can answer the question “Is Marcus alive?” by proving: **$\neg \text{alive}(\text{Marcus}, \text{now})$**

One way of proving that Marcus is not alive...

1. $\text{man}(\text{Marcus})$ $\neg \text{alive}(\text{Marcus}, \text{now})$
2. $\text{Pompeian}(\text{Marcus})$ ↑ (9, substitution)
3. $\text{born}(\text{Marcus}, 40)$ $\text{dead}(\text{Marcus}, \text{now})$
4. $x: \text{man}(x) \rightarrow \text{mortal}(x)$
5. $\forall x: \text{Pompeian}(x) \rightarrow \text{died}(x, 79)$ $\text{died}(\text{Marcus}, t_1) \wedge \text{gt}(\text{now}, t_1)$
6. $\text{erupted}(\text{volcano}, 79)$ ↑ (5, substitution)
7. $\forall x: \forall t1: \forall t2: \text{mortal}(x) \wedge \text{born}(x, t1) \wedge \text{gt}(t2 - t1, 150) \rightarrow \text{died}(x, t2)$ $\text{Pompeian}(\text{Marcus}) \wedge \text{gt}(\text{now}, 79)$
8. $\text{now} = 1991$ ↑ (2)
9. $\forall x: \forall t: [\text{alive}(x, t) \rightarrow \neg \text{dead}(x, t)] \wedge [\neg \text{dead}(x, t) \rightarrow \text{alive}(x, t)]$ $\text{gt}(1991, 79)$
10. $\forall x: \forall t1: \forall t2: \text{died}(x, t1) \wedge \text{gt}(t2, t1) \rightarrow \text{dead}(x, t2)$ ↑ (compute gt)
nil

Natural Deduction

- Testing whether a proposition is a tautology by testing every possible truth assignment is expensive—there are exponentially many. We need a deductive system, which will allow us to construct proofs of tautologies in a step-by-step fashion.
- The system we will use is known as natural deduction. The system consists of a set of rules of inference for deriving consequences from premises. One builds a proof tree whose root is the proposition to be proved and whose leaves are the initial assumptions or axioms (for proof trees, we usually draw the root at the bottom and the leaves at the top).
- For example, one rule of our system is known as modus ponens. Intuitively, this says that if we know P is true, and we know that P implies Q , then we can conclude Q .

Propositional logic

1/2

Syntax
↓

Alphabet \rightarrow Language (set of sentences)
 $\wedge, \vee, \neg, P, Q, R, \dots, (,)$

PROOF / PROVABILITY

$\{$
 $KB \vdash \alpha$
 $\}$

Language + Rules of Inference

Semantics
↓

Truth values

Tautologies Contradictions Contingency

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

ENTAILMENT

$KB \models \alpha$



Satisfiability
SAT

Proof systems

MP: $P, P \supset Q \vdash Q$

MP is valid because

$$((P \wedge (P \supset Q)) \supset Q)$$

MODUS PONENS

$$\frac{P}{\begin{array}{c} P \\ P \supset Q \\ \hline Q \end{array}}$$

PROPOSITIONAL VARIABLES

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

$$\underline{((A \wedge B) \vee C) \supset RVS}$$

RVS

Pick some rule

with matching antecedents

Add the consequent to KB

until some termination criteria

Sentence 1

$$\underline{\text{Sentence 1} \supset \text{Sentence 2}}$$

Sentence 2

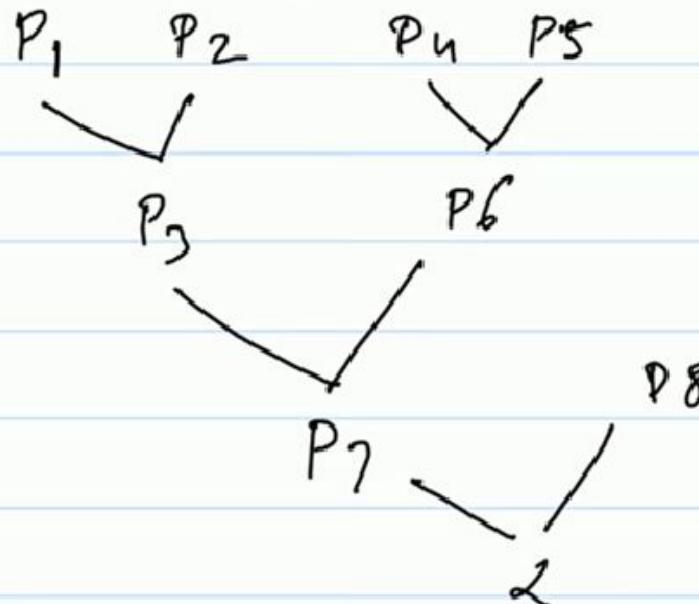
Termination

Direct Proof / Natural Deduction

↓
"until α is added to KB"

GENTZEN

Proof Tree



Indirect Proof

↓
"proof by contradiction"

if you add negation of goal
 $\{\neg\alpha\} \cup \text{KB} \vdash \perp$

Natural Deduction: Example

Given / Premises

1. $(P \wedge Q) \Rightarrow R$
 2. $(\neg Q \vee S) \Rightarrow T$
 3. $\neg T \wedge P$
 4. $\neg T$ 3, Simplification
 5. P - do -
 6. $\neg(\neg Q \vee S)$ 4, 2, Modus Tollens
 7. $Q \wedge \neg S$ 6 de Morgan's
 8. Q 7 Simplification
 9. $(P \wedge Q)$ 5, 8, Addition
 10. R 9, 1, MP
- $\vdash R$

Direct proof : needs a set of
starting sentences

AXIOMS

logical
(Tautologies)

Domain Specific

$(P \wedge Q) \Rightarrow R, (\neg Q \vee S) \Rightarrow T, \neg T \wedge P \vdash R$

Inference in Predicate Logic

Also known as

First-order logic, quantificational logic, first-order predicate calculus

- 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

- 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:** $\neg (x = y)$ which is equivalent to $x \neq y$.

FOL inference rules for quantifier

There are four quantifier rules of inference that allow you to remove or introduce a quantifier

Rule of Inference	Name
$\begin{array}{c} \forall x P(x) \\ \therefore P(c) \end{array}$	Universal instantiation
$\begin{array}{c} P(c) \text{ for an arbitrary } c \\ \therefore \forall x P(x) \end{array}$	Universal generalization
$\begin{array}{c} \exists x P(x) \\ \therefore P(c) \text{ for some element } c \end{array}$	Existential instantiation
$\begin{array}{c} P(c) \text{ for some element } c \\ \therefore \exists x P(x) \end{array}$	Existential generalization

1. Universal Generalization

- Universal generalization is a valid inference rule which states that if premise $P(c)$ is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as $\forall x P(x)$.
- It can be represented as: .
$$\frac{P(c)}{\forall x P(x)}$$
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.
- **Example:** Let's represent, $P(c)$: "**A byte contains 8 bits**", so for $\forall x P(x)$ "**All bytes contain 8 bits.**", it will also be true.

2. 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 $P(c)$ by substituting a ground term c (a constant within domain x) from **$\forall x P(x)$ for any object in the universe of discourse.**

It can be represented as::

- **Example:1.**

IF "Every person like ice-cream" $\Rightarrow \forall x P(x)$ so we can infer that
"John likes ice-cream" $\Rightarrow P(c)$

- **Example: 2.**

Let's take a famous example,

"All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

- $\forall x \text{ king}(x) \wedge \text{greedy}(x) \rightarrow \text{Evil}(x),$

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

- **King(John) \wedge Greedy (John) \rightarrow Evil (John),**
- **King(Richard) \wedge Greedy (Richard) \rightarrow Evil (Richard),**
- **King(Father(John)) \wedge Greedy (Father(John)) \rightarrow Evil (Father(John)),**

3. 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.
- This rule states that one can infer $P(c)$ from the formula given in the form of $\exists x P(x)$ for a new constant symbol c .
- The restriction with this rule is that c used in the rule must be a new term for which $P(c)$ is true.
- It can be represented as:
$$\frac{\exists x P(x)}{P(c)}$$

- **Example:**
- From the given sentence: $\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, \text{John})$,
- So we can infer: **Crown(K) \wedge 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**.

4. Existential generalization/introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P , then we can infer that there exists something in the universe which has the property P .
- It can be represented as:
$$\frac{P(c)}{\exists x P(x)}$$
- **Example:** Let's say that,
"Priyanka got good marks in English."
"Therefore, someone got good marks in English."

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 p_i, p'_i, q . Where there is a substitution θ such that $\text{SUBST}(\theta, p'_i) = \text{SUBST}(\theta, p_i)$, it can be represented as:

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

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.
- Here let say, $p1'$ is $\text{king}(\text{John})$ $p1 \text{ is } \text{king}(x)$
- $p2'$ is $\text{Greedy}(y)$ $p2 \text{ is } \text{Greedy}(x)$
- θ is $\{x/\text{John}, y/\text{John}\}$ $q \text{ is } \text{evil}(x)$
- $\text{SUBST}(\theta, q)$.
- This is known as UNIFICATION.

What is 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 $\text{Unify}\{\text{King}(x), \text{King}(\text{John})\}$
 - Let $\Psi_1 = \text{King}(x)$, $\Psi_2 = \text{King}(\text{John})$,
- Substitution $\theta = \{\text{John}/x\}$** 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**.

Unification Example

Ψ_1	Ψ_2	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	{fail}

- E.g. Let's say there are two different expressions, $P(x, y)$, and $P(a, f(z))$.
- In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.
 - $P(x, y) \dots\dots\dots (i)$
 - $P(a, f(z)) \dots\dots\dots (ii)$
- Substitute x with a , and y with $f(z)$ in the first expression, and it will be represented as a/x and $f(z)/y$.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: $[a/x, f(z)/y]$.

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:

```
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 ≠ NIL then do,  
        a. Apply S to the remainder of both L1 and L2.  
        b. SUBST= APPEND(S, SUBST).  
Step.6: Return SUBST.
```

Most General Unifier

- **1. Find the MGU of $\{p(f(a), g(Y))$ and $p(X, X)\}$**
- Sol: $S_0 \Rightarrow$ Here, $\Psi_1 = p(f(a), g(Y)),$ and $\Psi_2 = p(X, X)$
 $\text{SUBST } \theta = \{f(a) / X\}$
 $S1 \Rightarrow \Psi_1 = p(f(a), g(Y)),$ and $\Psi_2 = p(f(a), f(a))$
 $\text{SUBST } \theta = \{f(a) / g(y)\},$
Unification failed.
- Unification is not possible for these expressions.

Most General Unifier

- **Find the MGU of $\{p(b, X, f(g(Z)))$ and $p(Z, f(Y), f(Y))\}$**
- Here, $\Psi_1 = p(b, X, f(g(Z)))$, and $\Psi_2 = p(Z, f(Y), f(Y))$
 $S_0 \Rightarrow \{ p(b, X, f(g(Z))); p(Z, f(Y), f(Y)) \}$
SUBST $\theta = \{b/Z\}$
- $S_1 \Rightarrow \{ p(b, X, f(g(b))); p(b, f(Y), f(Y)) \}$
SUBST $\theta = \{f(Y) /X\}$
- $S_2 \Rightarrow \{ p(b, f(Y), f(g(b))); p(b, f(Y), f(Y)) \}$
SUBST $\theta = \{g(b) /Y\}$
- $S_2 \Rightarrow \{ p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))) \}$ **Unified Successfully.**
And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

Most General Unifier

- **3. Find the MGU of {p (X, X), and p (Z, f(Z))}**
- Here, $\Psi_1 = \{p (X, X), \text{ and } \Psi_2 = p (Z, f(Z))\}$
 $S_0 \Rightarrow \{p (X, X), p (Z, f(Z))\}$
SUBST $\theta = \{X/Z\}$
 $S1 \Rightarrow \{p (Z, Z), p (Z, f(Z))\}$
SUBST $\theta = \{f(Z) / Z\}$, **Unification Failed.**

Most General Unifier

- **4. Find the MGU of UNIFY(prime(11), prime(y))**
- Here, $\Psi_1 = \{\text{prime}(11)\}$, and $\Psi_2 = \{\text{prime}(y)\}$
 $S_0 \Rightarrow \{\text{prime}(11), \text{prime}(y)\}$
SUBST $\theta = \{11/y\}$
- $S_1 \Rightarrow \{\text{prime}(11), \text{prime}(11)\}$, **Successfully unified.**
Unifier: $\{11/y\}$.

Most General Unifier

- **5. Find the MGU of $Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)\}$**
- Here, $\Psi_1 = Q(a, g(x, a), f(y))$, and $\Psi_2 = Q(a, g(f(b), a), x)$
 $S_0 \Rightarrow \{Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)\}$
SUBST $\theta = \{f(b)/x\}$
 $S_1 \Rightarrow \{Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))\}$
- SUBST $\theta = \{b/y\}$
 $S_1 \Rightarrow \{Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))\}$, **Successfully Unified.**
- **Unifier: [a/a, f(b)/x, b/y].**

Most General Unifier

- **6. UNIFY(knows(Richard, x), knows(Richard, John))**
- Here, $\Psi_1 = \text{knows}(\text{Richard}, x)$, and $\Psi_2 = \text{knows}(\text{Richard}, \text{John})$
 $S_0 \Rightarrow \{ \text{knows}(\text{Richard}, x); \text{knows}(\text{Richard}, \text{John}) \}$
SUBST $\theta = \{ \text{John}/x \}$
 $S_1 \Rightarrow \{ \text{knows}(\text{Richard}, \text{John}); \text{knows}(\text{Richard}, \text{John}) \},$
- **Successfully Unified.**
Unifier: {John/x}.

Skolemization

- Conversion of sentences **FOL** to **CNF** requires skolemization.
- **Skolemization:** remove existential quantifiers by introducing new function symbols.
- **Special case:** introducing constants – **Skolem Contant**

Skolemization: Example

- Every philosopher writes at least one book.

$$\forall x[\text{Philo}(x) \rightarrow \exists y[\text{Book}(y) \wedge \text{Write}(x, y)]]$$

- Eliminate Implication:

$$\forall x[\neg\text{Philo}(x) \vee \exists y[\text{Book}(y) \wedge \text{Write}(x, y)]]$$

- Skolemize: substitute y by g(x) in order to remove \exists

$$\forall x[\neg\text{Philo}(x) \vee [\text{Book}(g(x)) \wedge \text{Write}(x, g(x))]]$$

here **g(x)** is **skolem** constant.

Prenex Normal Form

- It is often more convenient to deal with formulas in which all quantifiers have been moved to the front of the expression.
- These types of formulas are said to be in prenex normal form.
- **Definition:** A formula is in prenex normal form if it is of the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n B$$

where $Q_i (i = 1, \dots, n)$ is \forall or \exists and the formula B is quantifier free.

Connection between \forall and \exists

- $\forall x \neg \text{Likes}(x, \text{Chips}) \equiv \neg \exists x \text{ Likes}(x, \text{Chips})$
- $\forall x \text{ Likes}(x, \text{IceCream}) \equiv \neg \exists x \neg \text{Likes}(x, \text{IceCream})$

$$1. \quad \forall x \neg P \equiv \neg \exists x P$$

$$2. \quad \neg \forall x P \equiv \exists x \neg P$$

$$3. \quad \forall x P \equiv \neg \exists x \neg P$$

$$4. \quad \exists x P \equiv \neg \forall x \neg P$$

$$1. \quad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

$$2. \quad \neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

$$3. \quad P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$

$$4. \quad P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

Resolution in Predicate Logic: Example

- All hounds howl at night.

$$\forall x (HOUND(x) \rightarrow HOWL(x))$$

- Anyone who has any cats will not have any mice.

$$\forall x \forall y (HAVE(x,y) \wedge CAT(y) \rightarrow \neg \exists z (HAVE(x,z) \wedge MOUSE(z)))$$

- Light sleepers do not have anything which howls at night.

$$\forall x (LS(x) \rightarrow \neg \exists y (HAVE(x,y) \wedge HOWL(y)))$$

- John has either a cat or a hound.

$$\exists x (HAVE(John,x) \wedge (CAT(x) \vee HOUND(x)))$$

- (Conclusion) If John is a light sleeper, then John does not have any mice.

$$LS(John) \rightarrow \neg \exists z (HAVE(John,z) \wedge MOUSE(z))$$

The next step is to transform each wff into Prenex Normal Form, skolemize, and rewrite as clauses in conjunctive normal form; these transformations are shown below.

$$1. \quad \forall x (HOUND(x) \rightarrow HOWL(x))$$

$$\neg HOUND(x) \vee HOWL(x)$$

$$2. \quad \forall x \forall y (HAVE(x,y) \wedge CAT(y) \rightarrow \neg \exists z (HAVE(x,z) \wedge MOUSE(z)))$$

$$\forall x \forall y (HAVE(x,y) \wedge CAT(y) \rightarrow \forall z \neg (HAVE(x,z) \wedge MOUSE(z)))$$

$$\forall x \forall y \forall z (\neg (HAVE(x,y) \wedge CAT(y)) \vee \neg (HAVE(x,z) \wedge MOUSE(z)))$$

$$\neg HAVE(x,y) \vee \neg CAT(y) \vee \neg HAVE(x,z) \vee \neg MOUSE(z)$$

$$3. \quad \forall x (LS(x) \rightarrow \neg \exists y (HAVE(x,y) \wedge HOWL(y)))$$

$$\forall x (LS(x) \rightarrow \forall y \neg (HAVE(x,y) \wedge HOWL(y)))$$

$$\forall x \forall y (LS(x) \rightarrow \neg HAVE(x,y) \vee \neg HOWL(y))$$

$$\forall x \forall y (\neg LS(x) \vee \neg HAVE(x,y) \vee \neg HOWL(y))$$

$$\neg LS(x) \vee \neg HAVE(x,y) \vee \neg HOWL(y)$$

4. $\exists x (\text{HAVE}(\text{John},x) \wedge (\text{CAT}(x) \vee \text{HOUND}(x)))$

$\text{HAVE}(\text{John},a) \wedge (\text{CAT}(a) \vee \text{HOUND}(a))$

5. $\neg [\text{LS}(\text{John}) \rightarrow \neg \exists z (\text{HAVE}(\text{John},z) \wedge \text{MOUSE}(z))] \text{ (negated conclusion)}$

$\neg [\neg \text{LS}(\text{John}) \vee \neg \exists z (\text{HAVE}(\text{John},z) \wedge \text{MOUSE}(z))]$

$\text{LS}(\text{John}) \wedge \exists z (\text{HAVE}(\text{John},z) \wedge \text{MOUSE}(z))$

$\text{LS}(\text{John}) \wedge \text{HAVE}(\text{John},b) \wedge \text{MOUSE}(b)$

1. $\neg \text{HOUND}(x) \vee \text{HOWL}(x)$

2. $\neg \text{HAVE}(x,y) \vee \neg \text{CAT}(y) \vee \neg \text{HAVE}(x,z) \vee \neg \text{MOUSE}(z)$

3. $\neg \text{LS}(x) \vee \neg \text{HAVE}(x,y) \vee \neg \text{HOWL}(y)$

4.

a) $\text{HAVE}(\text{John},a)$

b) $\text{CAT}(a) \vee \text{HOUND}(a)$

5.

a) $\text{LS}(\text{John})$

b) $\text{HAVE}(\text{John},b)$

c) $\text{MOUSE}(b)$

Now we proceed to prove the conclusion by resolution using the above clauses. Each result clause is numbered; the numbers of its parent clauses are shown to its left.

[1.,4.(b):] 6. $\text{CAT}(a) \vee \text{HOWL}(a)$

[2.,5.(c):] 7. $\neg \text{HAVE}(x,y) \vee \neg \text{CAT}(y) \vee \neg \text{HAVE}(x,b)$

[7.,5.(b):] 8. $\neg \text{HAVE}(\text{John},y) \vee \neg \text{CAT}(y)$

[6.,8:] 9. $\neg \text{HAVE}(\text{John},a) \vee \text{HOWL}(a)$

[4.(a),9:] 10. $\text{HOWL}(a)$

[3.,10:] 11. $\neg \text{LS}(x) \vee \neg \text{HAVE}(x,a)$

[4.(a),11:] 12. $\neg \text{LS}(\text{John})$

[5.(a),12:] 13. $()$

Logic Programming

- Logic programming is a programming paradigm in which logical assertions are viewed as programs.
- These are several logic programming systems, PROLOG is one of them.
- *A PROLOG program consists of several logical assertions where each is a horn clause i.e. a clause with at most one positive literal.*
- Ex : $P, P \vee Q, P \rightarrow Q$
 - The facts are represented on Horn Clause for two reasons.
 - Because of a uniform representation, a simple and efficient interpreter can be written.
 - The logic of Horn Clause is decidable.

Consider the following example

Logical representation

- $\forall x : \text{pet}(x) \wedge \text{small}(x) \rightarrow \text{apartmentpet}(x)$
- $\forall x : \text{cat}(x) \vee \text{dog}(x) \rightarrow \text{pet}(x)$
- $\forall x : \text{poodle}(x) \rightarrow \text{dog}(x) \wedge \text{small}(x)$
- poodle (fluffy)

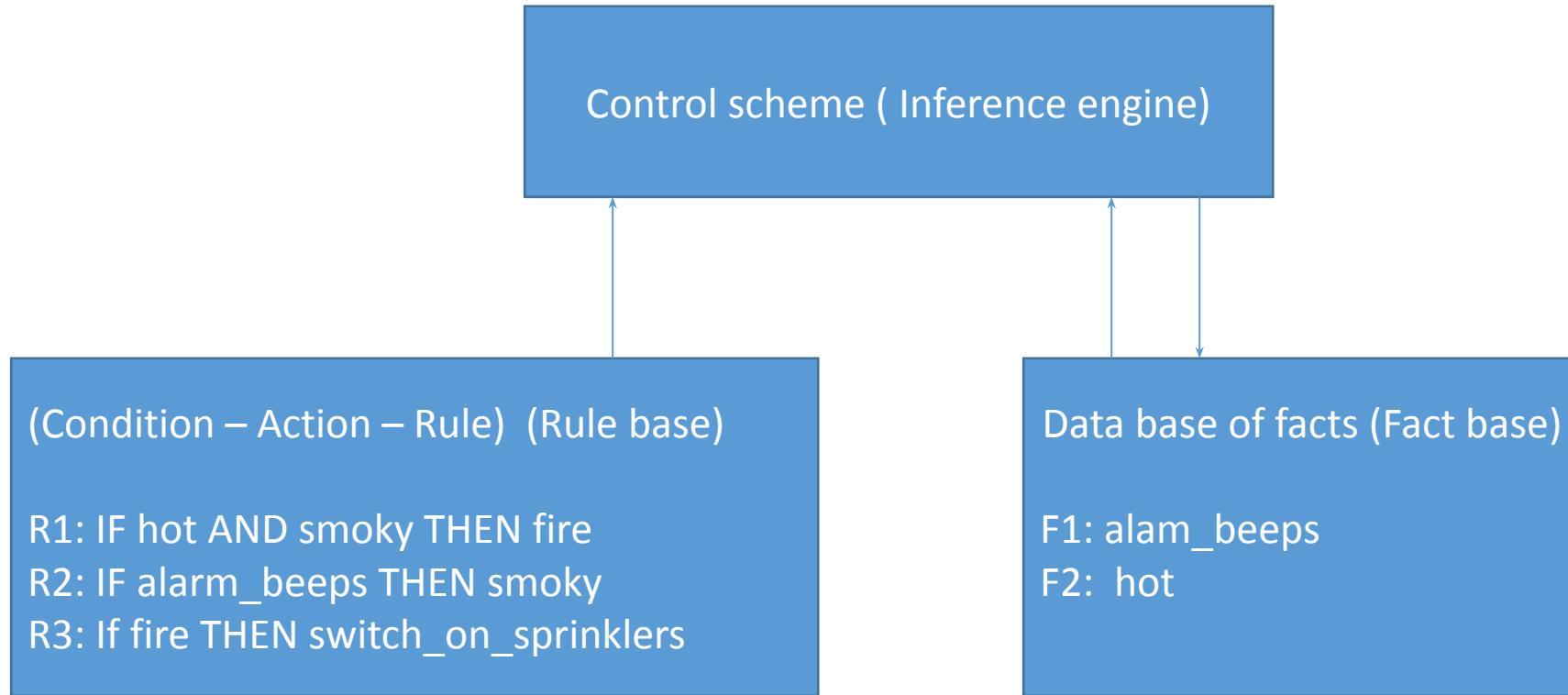
Prolog representation

- apartmentpet(x) :- pet(x), small(x).
- pet(x) :- cat(x).
- pet(x) :- dog(x).
- dog(x) :- poodle(x).
- small(x) :- poodle(x).
- poodle (fluffy).

Rule-Based System Architecture

- **A collection of facts**
 - Assertions that represent domain specific knowledge.
 - E.g. Anil is hardworking.
- **A collection of rules**
 - assertions given in implicational form
 1. **Implication:** the ‘ \rightarrow ’ in the rule
 2. **Antecedent:** the part of the rule before ‘ \rightarrow ’
 3. **Consequent:** the part of the rule after ‘ \rightarrow ’
 - Read a rule as “antecedent implies consequent” or
 - IF antecedent THEN consequent
- **An inference engine**
 - “A generic control mechanism that applies the axiomatic knowledge present in the knowledge base to the task-specific data to arrive at some conclusion”.

Rule-Based System Architecture



Two control strategies:

1. forward chaining

- working from the facts to a conclusion.

2. backward chaining

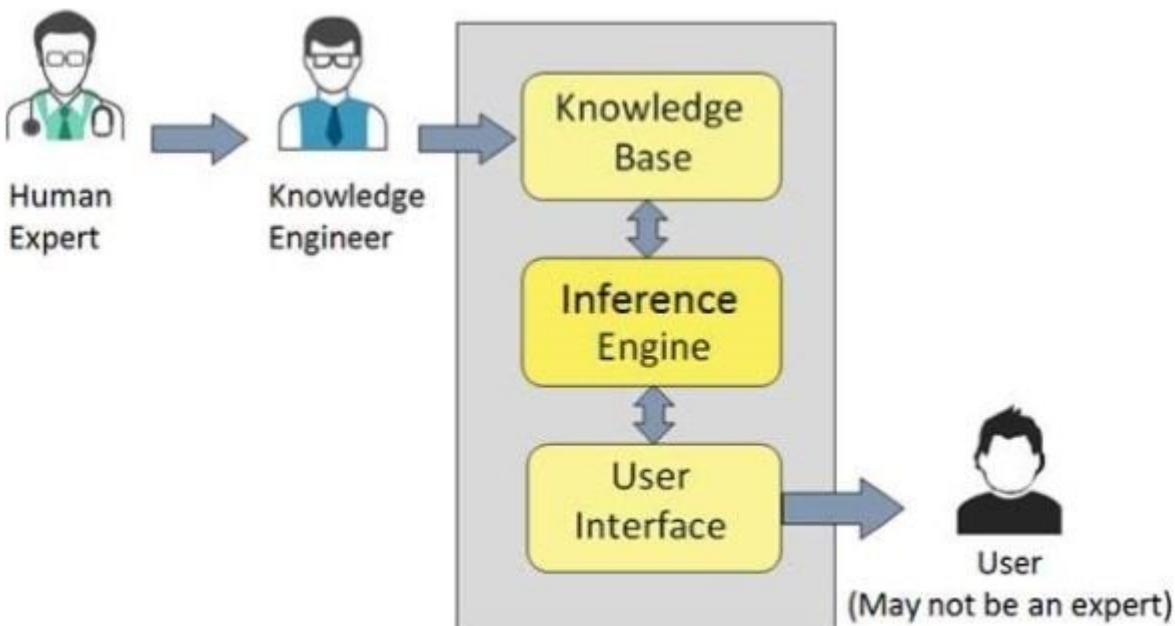
- reasoning from goals back to facts

Introduction to the Expert System

- A brief overview of an expert system can help us gain more insights on the origin of **backward** and **forward** chaining in artificial intelligence.
- An expert system is a computer application that uses rules, approaches, and facts to provide solutions to complex problems.
- Examples of expert systems include **MYCIN** and **DENDRAL**.
- MYCIN uses the backward chaining technique to diagnose bacterial infections.
- DENDRAL employs forward chaining to establish the structure of chemicals.

Expert system and Chaining: Relationship

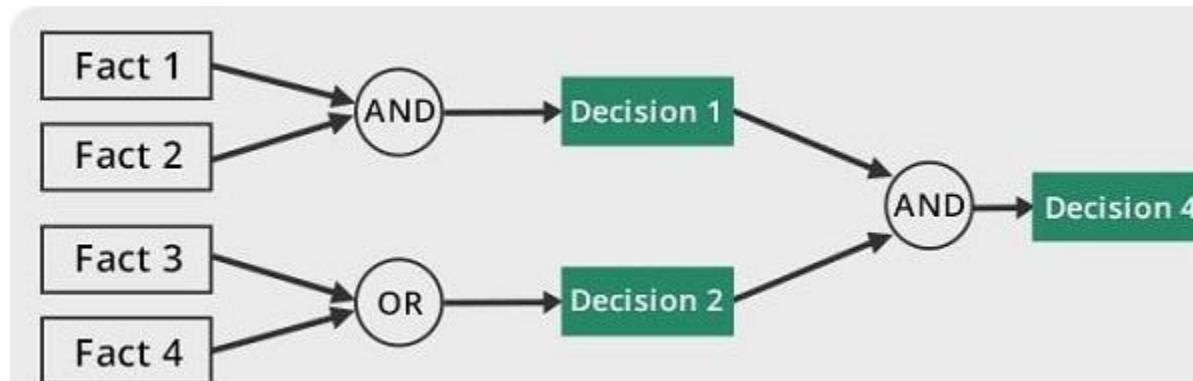
- There are **three** components in an expert system: **user interface**, **inference engine**, and **knowledge base**.
- The user interface enables users of the system to interact with the expert system. High-quality and domain-specific knowledge is stored in the **knowledge base**.
- Backward and forward chaining stem from the **inference engine** component.
- This is a component in which **logical rules** are applied to the knowledge base to get new information or make a decision.
- The backward and forward chaining techniques are used by the inference engine as strategies for **proposing solutions** or **deducing information** in the expert system.



Forward chaining / Forward reasoning

- Forward chaining is a method of reasoning in artificial intelligence in which inference rules are applied to existing data to extract additional data until an endpoint (goal) is achieved.
- In this type of chaining, the inference engine starts by evaluating existing facts, derivations, and conditions before deducing new information.
- An endpoint (goal) is achieved through the manipulation of knowledge that exists in the knowledge base.

“What can happen next?”



Properties of forward chaining

- The process uses a down-up approach (bottom to top).
- It starts from an initial state and uses facts to make a conclusion.
- This approach is data-driven.
- It's employed in expert systems and production rule system.

Examples of forward chaining

A simple example of forward chaining can be explained in the following sequence.

A

$A \rightarrow B$

B

A is the starting point. $A \rightarrow B$ represents a fact. This fact is used to achieve a decision B. (**Modes Ponens**)

A practical example will go as follows;

Tom is running (A)

If a person is running, he will sweat ($A \rightarrow B$)

Therefore, Tom is sweating. (B)

Advantages & Disadvantages: Forward Chaining

Advantages

- It can be used to draw multiple conclusions.
- It provides a good basis for arriving at conclusions.
- It's more flexible than backward chaining because it does not have a limitation on the data derived from it.

Disadvantages

- The process of forward chaining may be time-consuming.
- It may take a lot of time to eliminate and synchronize available data.
- Unlike backward chaining, the explanation of facts or observations for this type of chaining is not very clear.

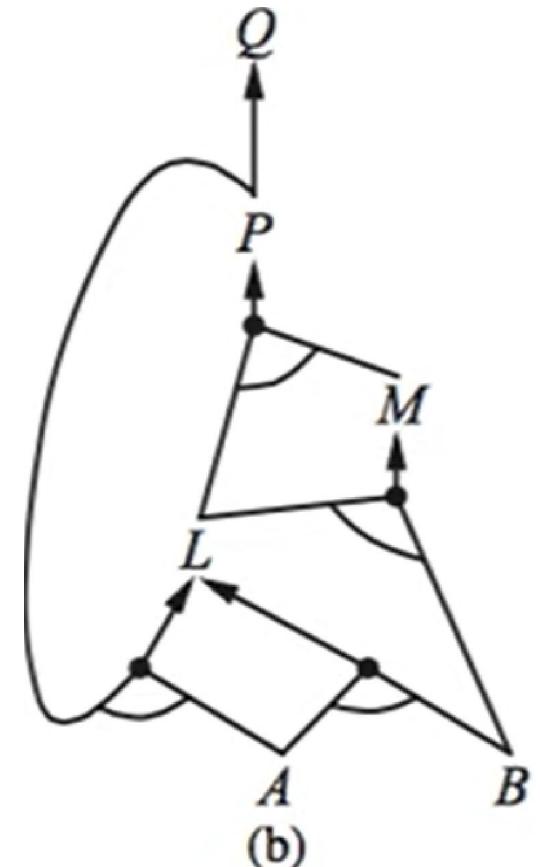
Forward Chaining Example: I

suppose these are the things we have in our knowledge base:

- P implies Q
- L and M implies P
- B and L implies M
- A and P implies L
- A and B implies L
- A is true
- B is true.
- And now I asked the question is Q true?

$$\begin{aligned}P &\Rightarrow Q \\L \wedge M &\Rightarrow P \\B \wedge L &\Rightarrow M \\A \wedge P &\Rightarrow L \\A \wedge B &\Rightarrow L \\A \\B\end{aligned}$$

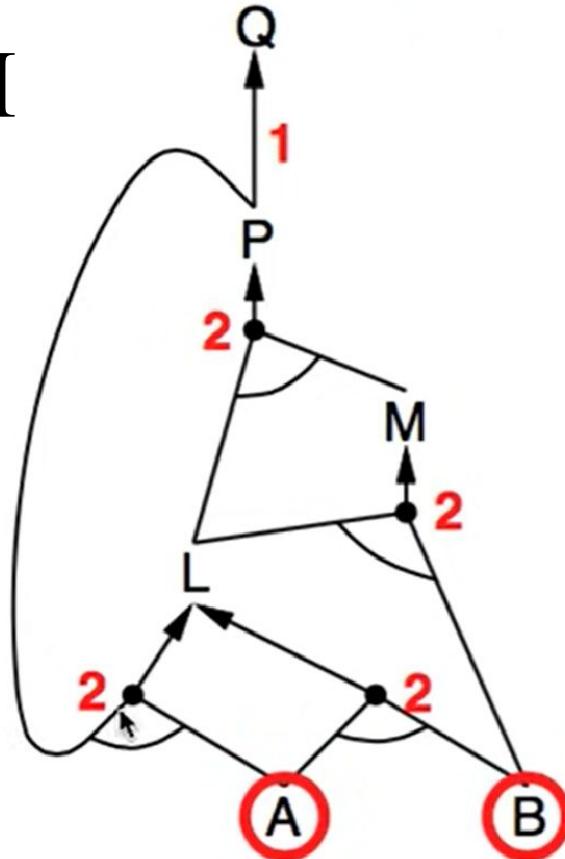
(a)



(b)

Forward Chaining Example: II

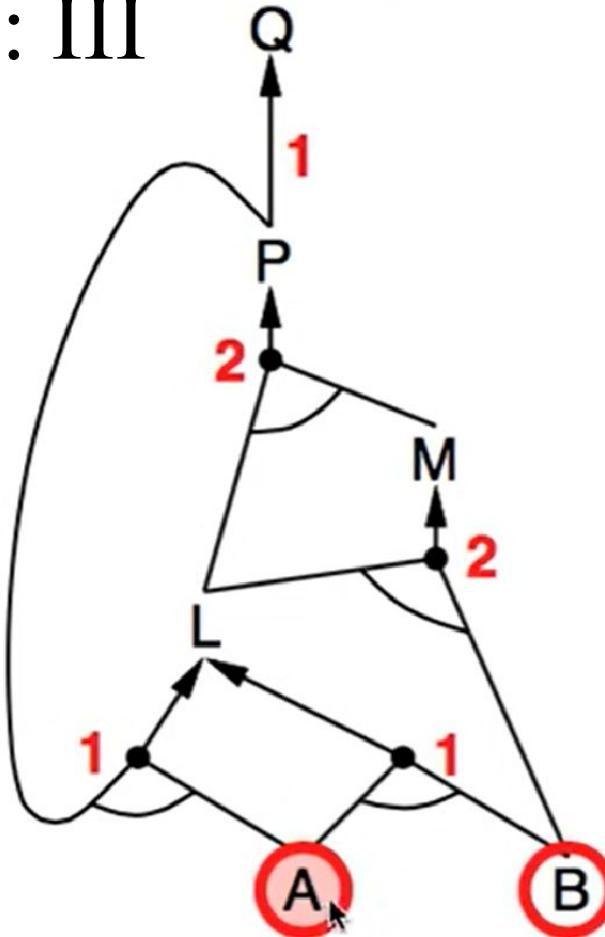
- Q has only one literal
- L has 2 literals A and P
- M has 2 literals B and L
- And so on...



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

- And I put A and B in my queue, because those are the 2 things I know to be true.
- Now, I can run a simple forward chaining algorithm. Because A is true I pop it out from the queue.

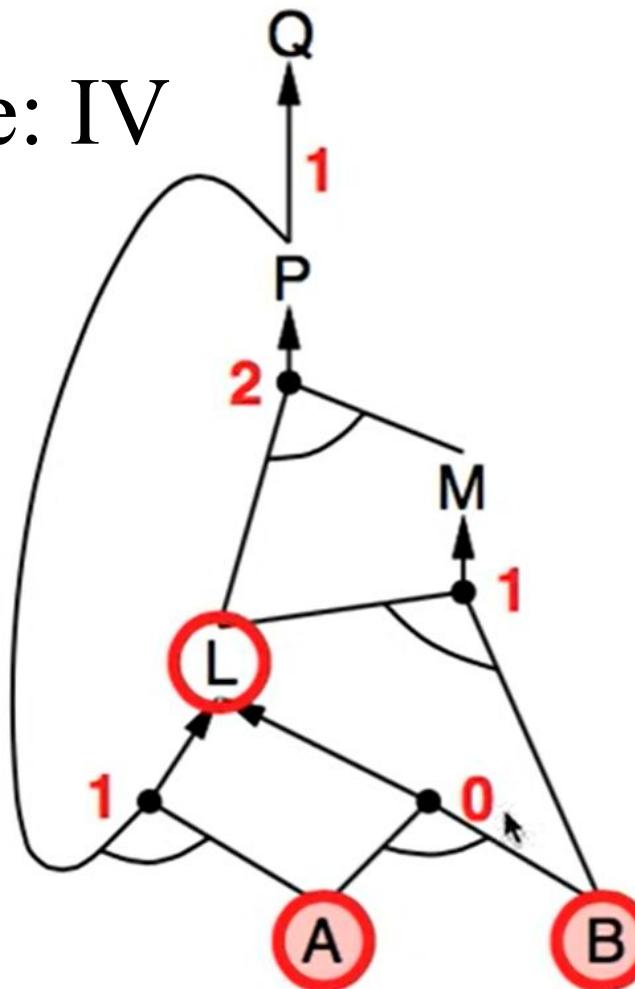
Forward Chaining Example: III



$$\begin{aligned}P &\Rightarrow Q \\L \wedge M &\Rightarrow P \\B \wedge L &\Rightarrow M \\A \wedge P &\Rightarrow L \\A \wedge B &\Rightarrow L \\A \\B\end{aligned}$$

- As soon as I know A is true, I know that the dot for A and B (the node for A and B), no longer needs 2 things to be true it only needs one more thing to be true.

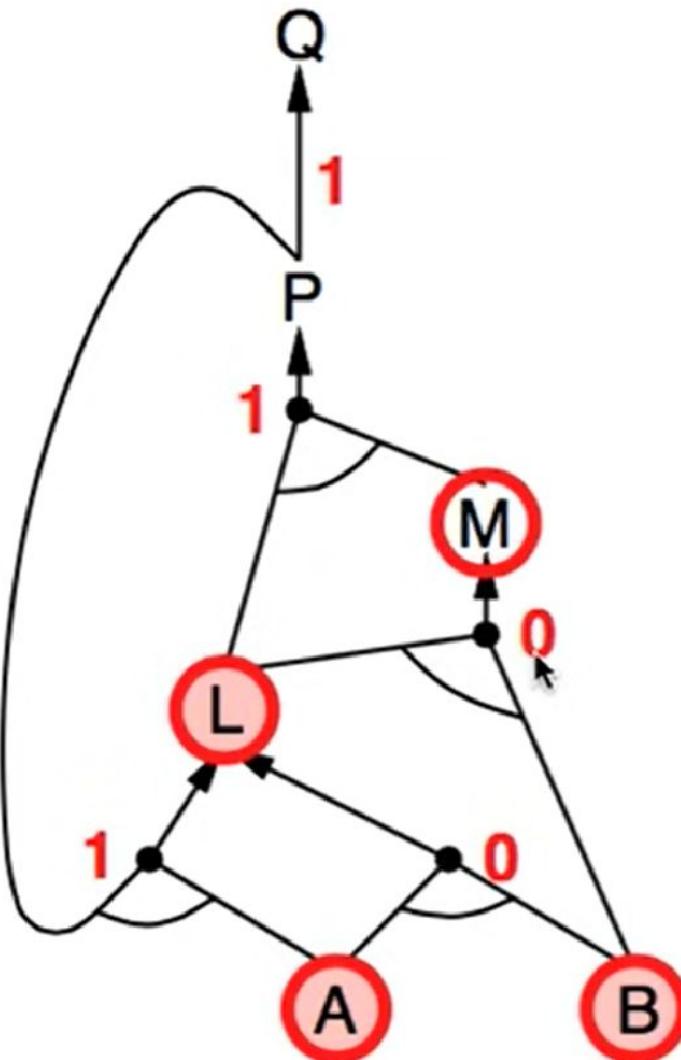
Forward Chaining Example: IV



$P \Rightarrow Q$
 $L \wedge M \Rightarrow P$
 $\neg B \wedge L \Rightarrow M$
 $A \wedge P \Rightarrow L$
 $\underline{A \wedge B \Rightarrow L}$
A
B

- As soon as I pop B, now I know that I do not need to satisfy both the things of this antecedent are satisfied.
- Because both the things of this antecedent are satisfied now I can prove L because I can prove L, I put it in the queue and I repeat this process.

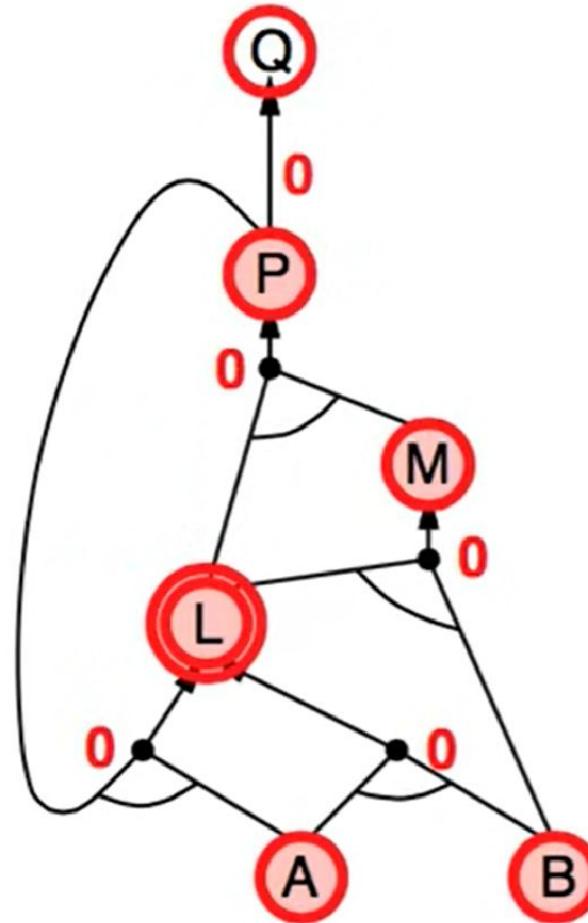
Forward Chaining Example: V



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

Forward Chaining Example: VI

- And suppose, I keep running this procedure, and at some point, I get to Q to be true and when I can get Q to be true, that means I have solved the question I had at hand.
- So now you see that this is an inference procedure, which is using a mechanical process for computing new sentences.
- And in this case, if all the sentences are horn clauses (a clause with at most one positive literal) of the form, A and B and C and so, implies something and my query is a single variable, or a conjunction of multiple variables, but not a disjunction.
- So, if my query is a single variable, then I can say that this procedure is guaranteed to prove it and it will be also complete.



$$\begin{aligned}P &\Rightarrow Q \\L \wedge M &\Rightarrow P \\B \wedge L &\Rightarrow M \\A \wedge P &\Rightarrow L \\A \wedge B &\Rightarrow L \\A \\B\end{aligned}$$

Example: Forward Chaining

Suppose that the goal is to conclude the **color of a pet named Fritz**, given that **he croaks** and **eats flies**, and that the rule base contains the following four rules:

1. If X croaks and X eats flies - Then X is a frog
2. If X chirps and X sings - Then X is a canary
3. If X is a frog - Then X is green
4. If X is a canary - Then X is yellow

Let us illustrate forward chaining by following the pattern of a computer as it evaluates the rules.

Assume the following facts:

Fritz croaks

Fritz eats flies

With forward reasoning, the inference engine can derive that **Fritz is green** in a series of steps:

1. Since the base facts indicate that "Fritz croaks" and "Fritz eats flies", the antecedent of rule #1 is satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is a frog

2. The antecedent of rule #3 is then satisfied by substituting Fritz for X, and the inference engine concludes:

Fritz is green

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American. Prove that West is a criminal.

- We first represent these facts in first-order definite clauses.

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

$$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \rightarrow \text{Criminal}(x)$$

RES: $\neg \text{American}(x) \vee \neg \text{Weapon}(y) \vee \neg \text{Sells}(x, y, z) \vee \neg \text{Hostile}(z) \vee \text{Criminal}(x)$

2. "Nono ... has some missiles": $\exists x \text{ Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$ is transformed into two definite clauses by Existential Instantiation, introducing a new constant M₁:

$\text{Owns}(\text{Nono}, M_1)$

$\text{Missile}(M_1)$

RES: $\text{Owns}(\text{Nono}, M_1)$

$\text{Missile}(M_1)$

- All of the missiles were sold to country Nono by Colonel West.

$$\forall x \text{Missiles}(x) \wedge \text{Owns}(\text{Nono}, x) \rightarrow \text{Sells}(\text{West}, x, \text{Nono})$$

RES: $\neg \text{Missiles}(x) \vee \neg \text{Owns}(\text{Nono}, x) \vee \text{Sells}(\text{West}, x, \text{Nono})$

- We will also need to know that “missiles are weapons”.

$$\text{Missile}(x) \rightarrow \text{Weapons}(x)$$

RES: $\neg \text{Missile}(x) \vee \text{Weapons}(x)$

- We know that enemy of America counts as “hostile”.

$$\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)$$

RES: $\neg \text{Enemy}(x, \text{America}) \vee \text{Hostile}(x)$

- West, who is American ...

$\text{American}(\text{West})$ **RES:** $\text{American}(\text{West})$

- The country Nono, an enemy of America ...

$\text{Enemy}(\text{Nono}, \text{America})$ **RES:** $\text{Enemy}(\text{Nono}, \text{America})$

Step1:

American(West)

Missile(M1)

Owns(Nono,M1)

Enemy(Nono,America)

Step2:

Weapon(M1)

Sells(West,M1,Nono)

Hostile(Nono)

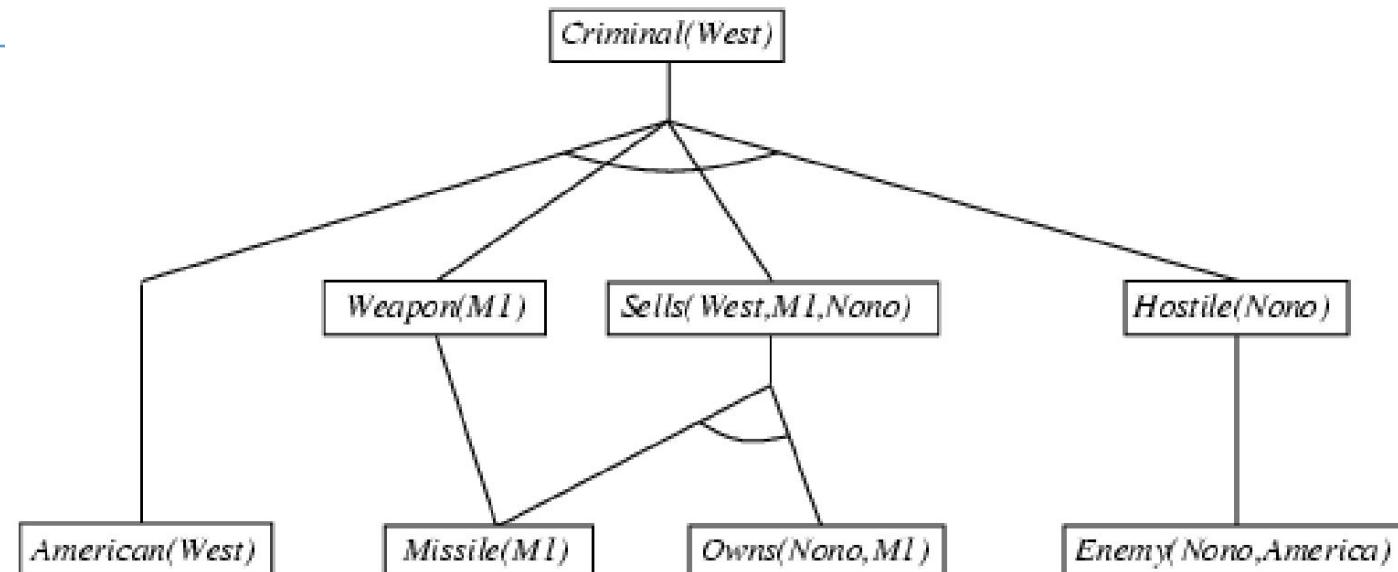
American(West)

Missile(M1)

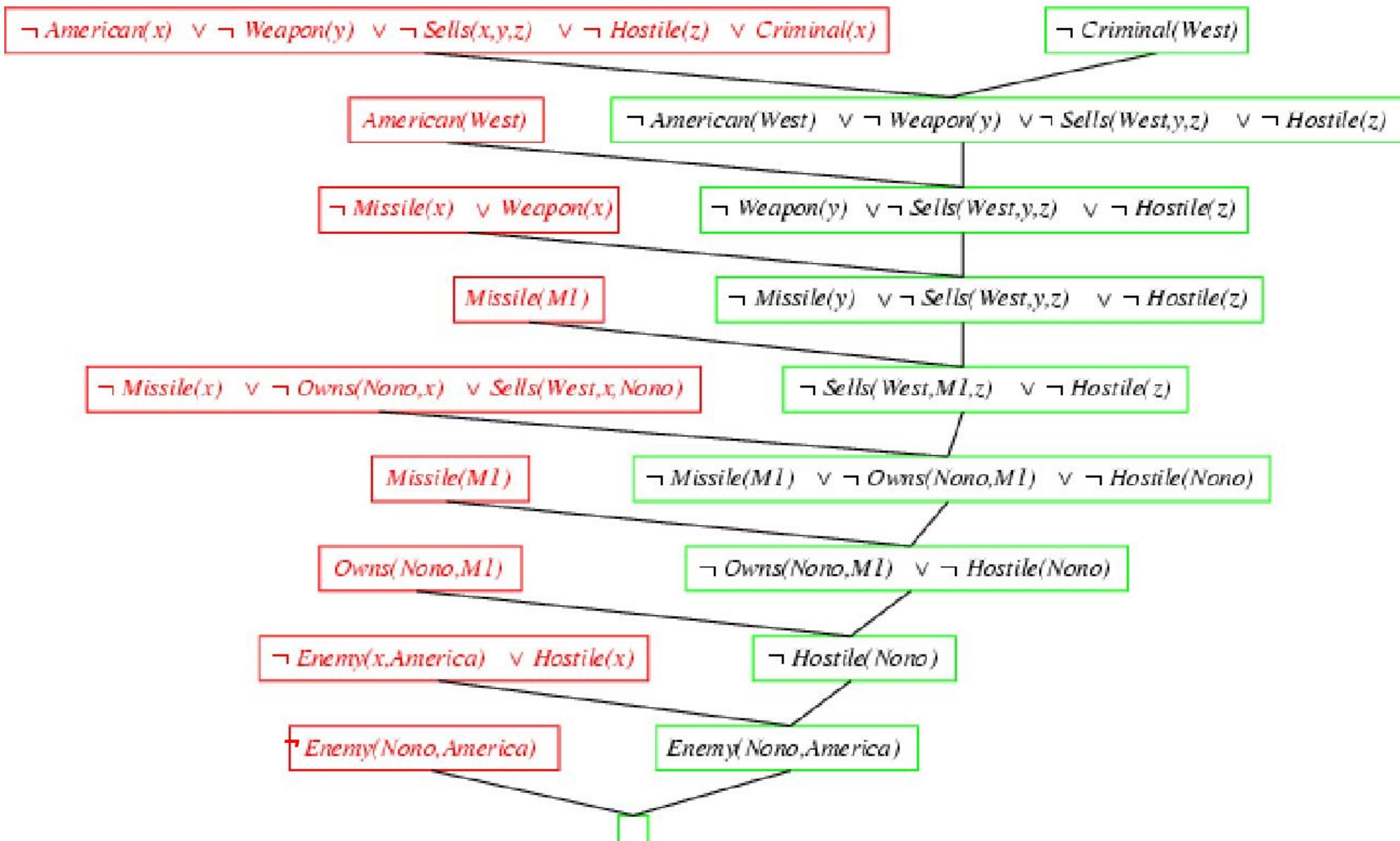
Owns(Nono,M1)

Enemy(Nono,America)

Step3: We can check the given statement that needs to be checked and check whether it is satisfied with the substitution which infers all the previously stated facts. Thus we reach our goal.



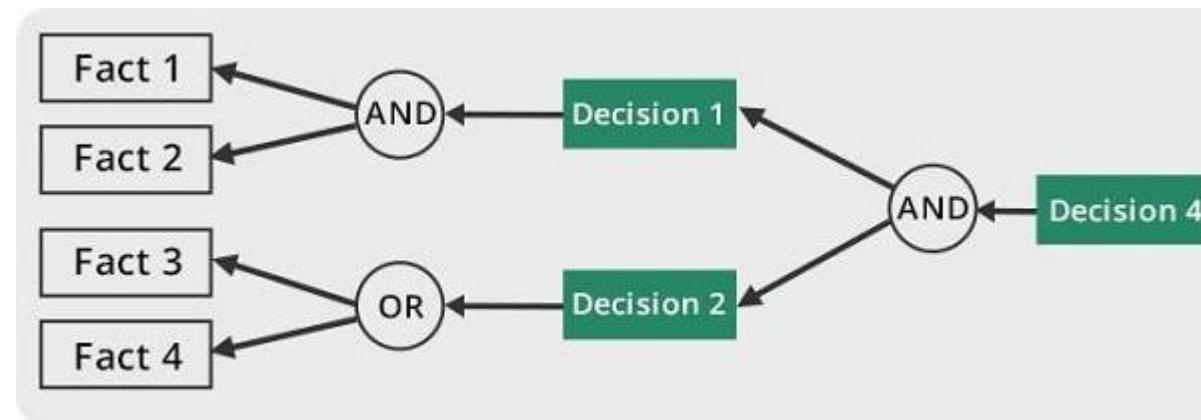
Resolution proof: West is a criminal (previous example)



Backward Chaining

- Backward chaining is a concept in artificial intelligence that involves backtracking from the endpoint or goal to steps that led to the endpoint.
- This type of chaining starts from the goal and moves backward to comprehend the steps that were taken to attain this goal.
- The backtracking process can also enable a person establish logical steps that can be used to find other important solutions.

“Why this happened?”



Unit-6

Probabilistic Reasoning

Uncertainty in AI: Motivation

- Now, moving towards Traditional AI to Modern AI
 - Think about what is it that we have resolved so far in the field?
 - I give you a new puzzle, I give you a new setting, I put you in a different setting you will not train for it - you should be able to solve it. You should be able to reason about it.
- ✓ let us come up with a very general algorithm – search problem
- ✓ uninformed systematic search to heuristic function in form systematic search to local search and to backtracking search and so on.
- ✓ But sometimes the problem has some characteristics which we can do the reason which to make some inferences.
- ✓ So we said okay search and inference together is even better than search right.

Cont'd

- ✓ Earlier in search our language was just a program blackbox but now we said let us go inside the black box let us define the language.
- ✓ we define the language of constraint satisfaction problems
- ✓ We said let us define a more interesting language we define the language of logic.
- ✓ we learned about some algorithms for inference and logic.
- ✓ then finally we said that all of this for single agent problems but we often have multi agent problems.
- ✓ and we talked about the minimax algorithm
- ✓ and we talked about the alpha-beta pruning algorithm

Cont'd

- All these traditional AI considered only deterministic problem.
- There is no notion of randomness, there is no notion of uncertainty
- Think for a robot picking up a cup, what's possible outcome?
 - Heavy, stuck to the bottom, not cup at all and so on...
- In logic, we have to define a variable for each such possibility.

- Suppose, cuff and cold if these are the only symptoms I give you, which disease you will say?
- you say common cold. Why did not you say throat cancer?
- It is less probable? you have some intuition?

Cont'd

- Can logic deal with all of that?
- Can logic say that you could have throat cancer but the probability of throat cancer is ridiculously small and you probably have common cold because that is the most frequent.
- And unfortunately the traditional AI could not do this.
- And therefore logic went out of fashion and what came into fashion at the time is **probabilistic modeling** which is what we are going to start with.

Probabilistic reasoning in Artificial intelligence

- Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates.
- With this knowledge representation, we might write $A \rightarrow B$, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called **uncertainty**.
- So to represent uncertain knowledge, where we are not sure about the predicates, we need **uncertain reasoning or probabilistic reasoning**.
- Causes of uncertainty in real world:
 - Information occurred from unreliable sources.
 - Experimental Errors
 - Equipment fault
 - Temperature variation
 - Climate change

Probabilistic reasoning

- Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge.
- In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.
- We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
- In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as
 - "It will rain today,"
 - "behavior of someone for some situations,"
 - "A match between two teams or two players."
- These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- Bayes' rule
- Bayesian Statistics

[Note] As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms.

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

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

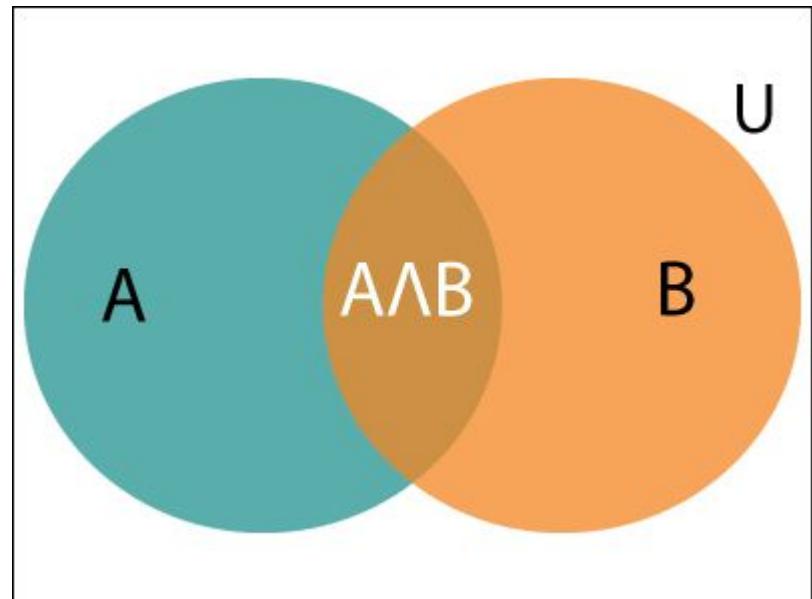
- **Event:** Each possible outcome of a variable is called an event
- **Sample space:** The collection of all possible events is called sample space.
- **Random variables:** Random variables are used to represent the events and objects in the real world.
- **Prior probability:** The prior probability 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.
- 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.



Example

- In a class, there are 70% of the students who like English and 40% of the students who likes English and Mathematics.
- What is the percent of students those who like English also like Mathematics?
- Let, A is an event that a student likes Mathematics
- B is an event that a student likes English.

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

- Hence, 57% are the students who like English also like Mathematics.

BAYE'S THEOREM: Describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

↪ In Probability theory it relates the Conditional probability & Marginal probabilities of two random events.

$$\rightarrow P(H|E) = \frac{\text{no. of times } H \text{ and } E}{\text{no. of times } E}$$

↪ Calculate $P(B|A)$ with knowledge of $P(A|B)$.

$$P(A \cap B) = P(A|B) \cdot P(B) - (i) \quad \left. \begin{array}{l} \text{from (i) and (ii)} \\ \text{L.H.S are equal.} \end{array} \right\}$$

$$P(H|E) = \frac{P(H \cap E)}{P(E)} \quad \left. \begin{array}{l} \text{Prob. of } H \\ \text{when } E \text{ is true.} \end{array} \right\}$$

$$\rightarrow P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$

So,

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

Baye's theorem formula.

Posterior (Prob. of A when B is marginal true),
 Prior Prob (Prob. of hypothesis)

Likelihood (Prob. of evidence)

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:

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

Bayes' Theorem Example #1

- Find a patient's probability of having liver disease if they are an alcoholic. "Being an alcoholic" is the test (kind of like a litmus test) for liver disease.
- A could mean the event "Patient has liver disease." Past data tells you that 10% of patients entering your clinic have liver disease. $P(A) = 0.10$.
- B could mean the litmus test that "Patient is an alcoholic." Five percent of the clinic's patients are alcoholics. $P(B) = 0.05$.
- You might also know that among those patients diagnosed with liver disease, 7% are alcoholics. This is your $B|A$: the probability that a patient is alcoholic, given that they have liver disease, is 7%.

Bayes' theorem tells you:

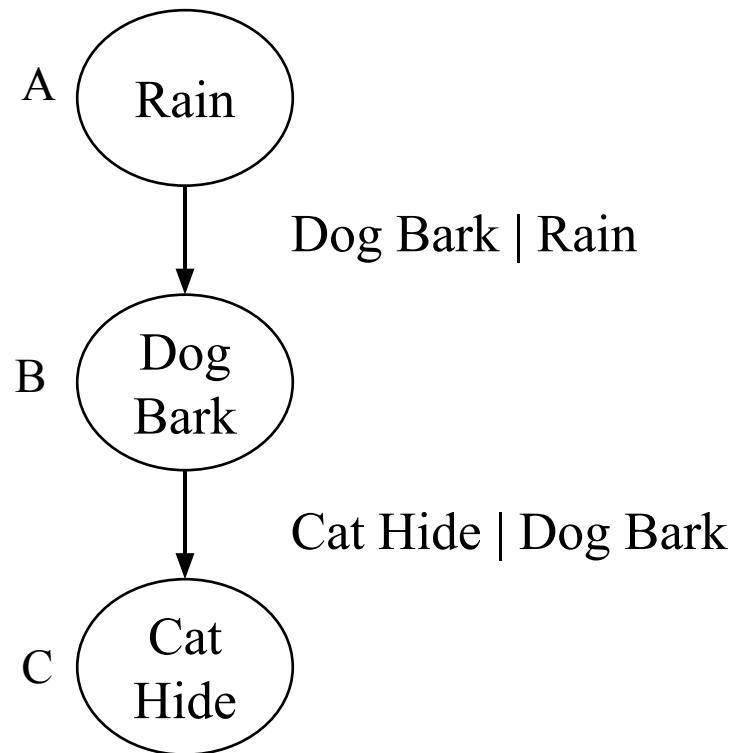
- $P(A|B) = (0.07 * 0.1)/0.05 = 0.14$
- In other words, if the patient is an alcoholic, their chances of having liver disease is 0.14 (14%). This is a large increase from the 10% suggested by past data. But it's still unlikely that any particular patient has liver disease.

Bayesian Belief Network in artificial intelligence

- A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph.
- It is also called a **Bayes network**, **belief network**, **decision network**, or **Bayesian model**.
- Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.
- Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:
 - **Directed Acyclic Graph**
 - **Table of conditional probabilities.**

Bayesian Belief Network

Directed Acyclic Graph (DAG)



If C is conditionally independent of A given B, then we can
Write $P(A,B,C) = P(A)P(B|A)P(C|B)$

Conditional Probability Table

	R	$\sim R$
B	9/48	18/48
$\sim B$	3/48	18/48

$$B = T \ \& \ R = T \quad \square \ 0.19$$

$$B = T \ \& \ R = F \quad \square \ 0.375$$

$$B = F \ \& \ R = T \quad \square \ 0.375$$

$$B = F \ \& \ R = F \quad \square \ 0.375$$

Convenient for representing probabilistic relation between multiple events

Example

- Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called Harry.

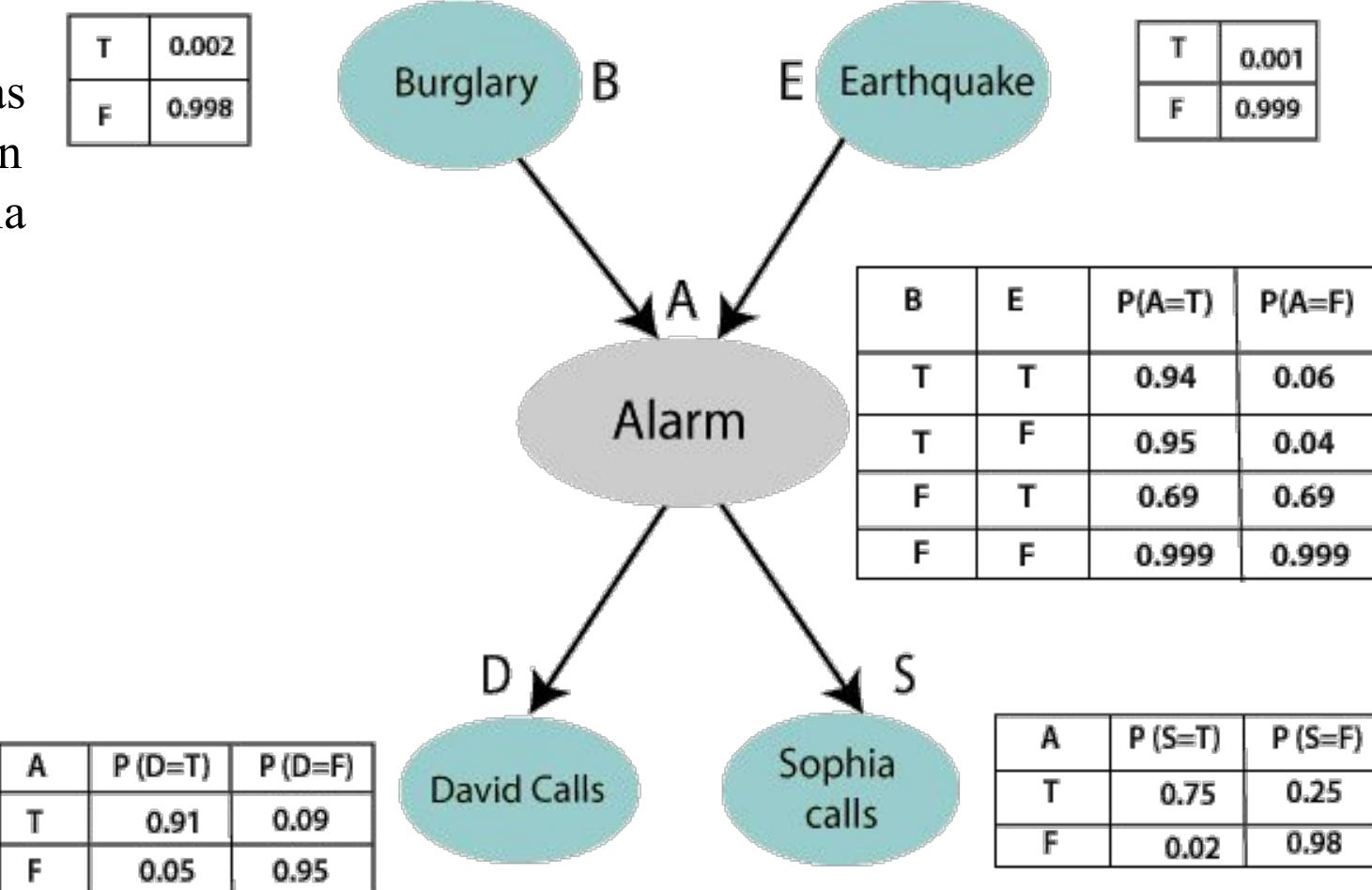
$$A, \sim B, \sim E, D, S$$

$$P(S, D, A, \neg B, \neg E)$$

$$= P(S|A) * P(D|A) * P(A|\neg B, \neg E) * P(\neg B) * P(\neg E)$$

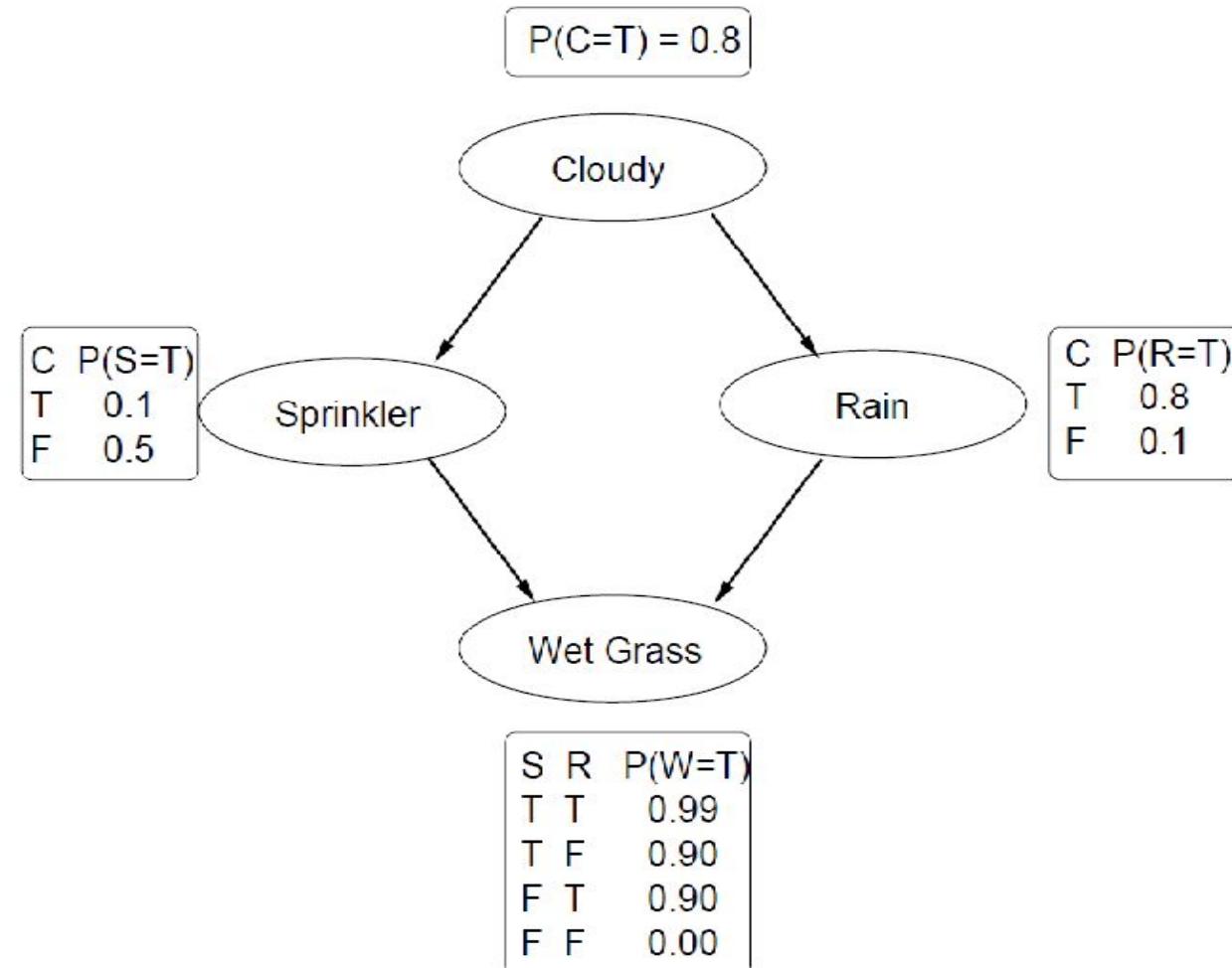
$$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$$

$$= \mathbf{0.00068045}.$$



Another Example

- Whether the grass is wet, **W**, depends on whether the sprinkler has been used, **S**, or whether it has rained, **R**.
- Whether the sprinkler is used depends on whether it is cloudy, similarly for whether it has rained.
- The probability of the grass being wet (**W**) is conditionally independent of it being cloudy, given information about the sprinklers and whether it has rained.
- This joint probability may be expressed as
$$P(C, S, R, W) = P(C)P(S|C)P(R|C)P(W|S,R)$$

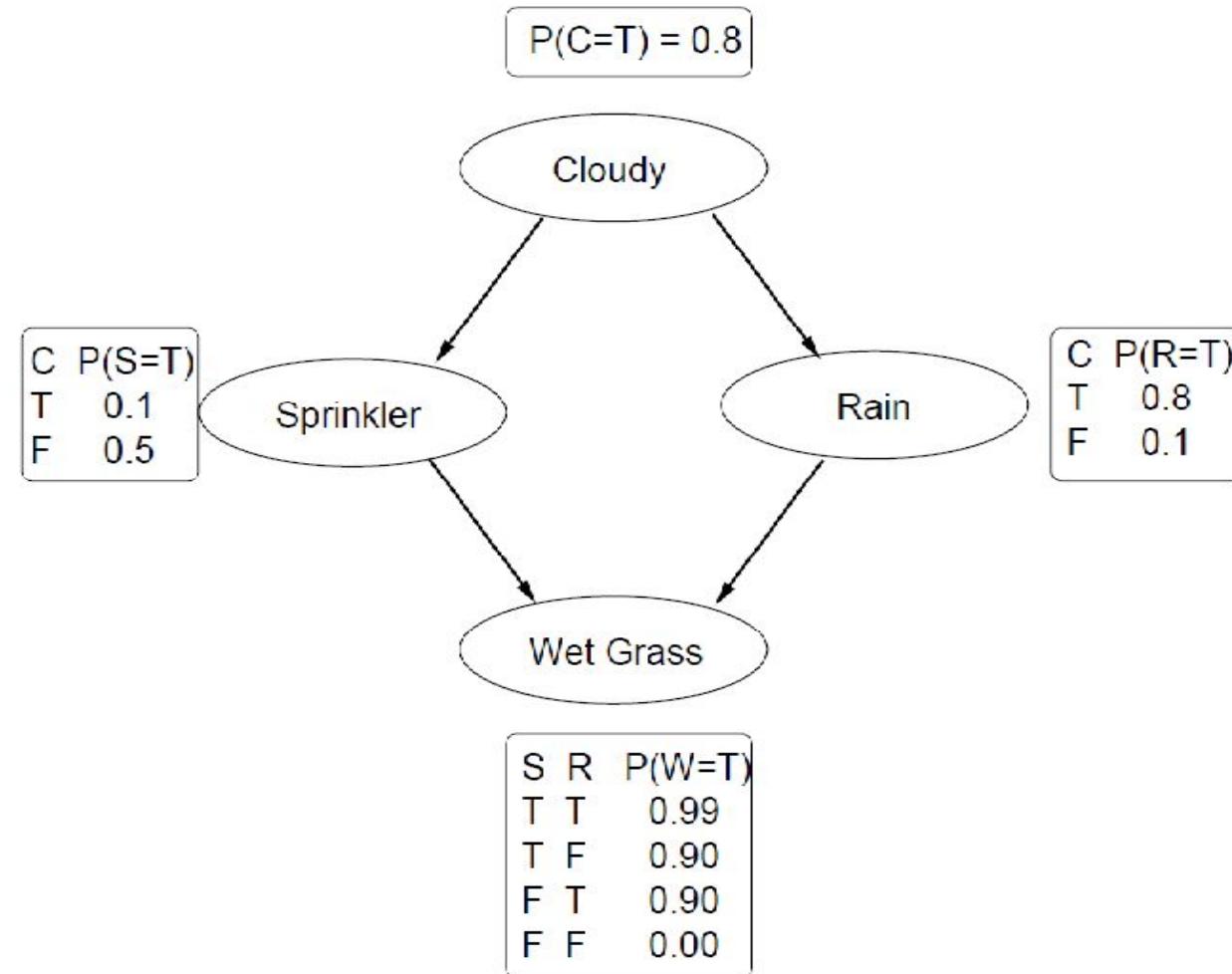


It is cloudy, what's the probability that the grass is wet?

The basic task is given some observation infer the probability of an event. So a question may be

- It is cloudy, what's the probability that the grass is wet?
- so we want to compute $P(W = T|C = T)$.
(Note: for simplicity of notation $P(W_T|C_T)$ will be used for $P(W = T|C = T)$.)
- Re-expressing this request in terms of the joint probability

$$P(W_T|C_T) = \frac{P(W_T, C_T)}{P(C_T)}$$

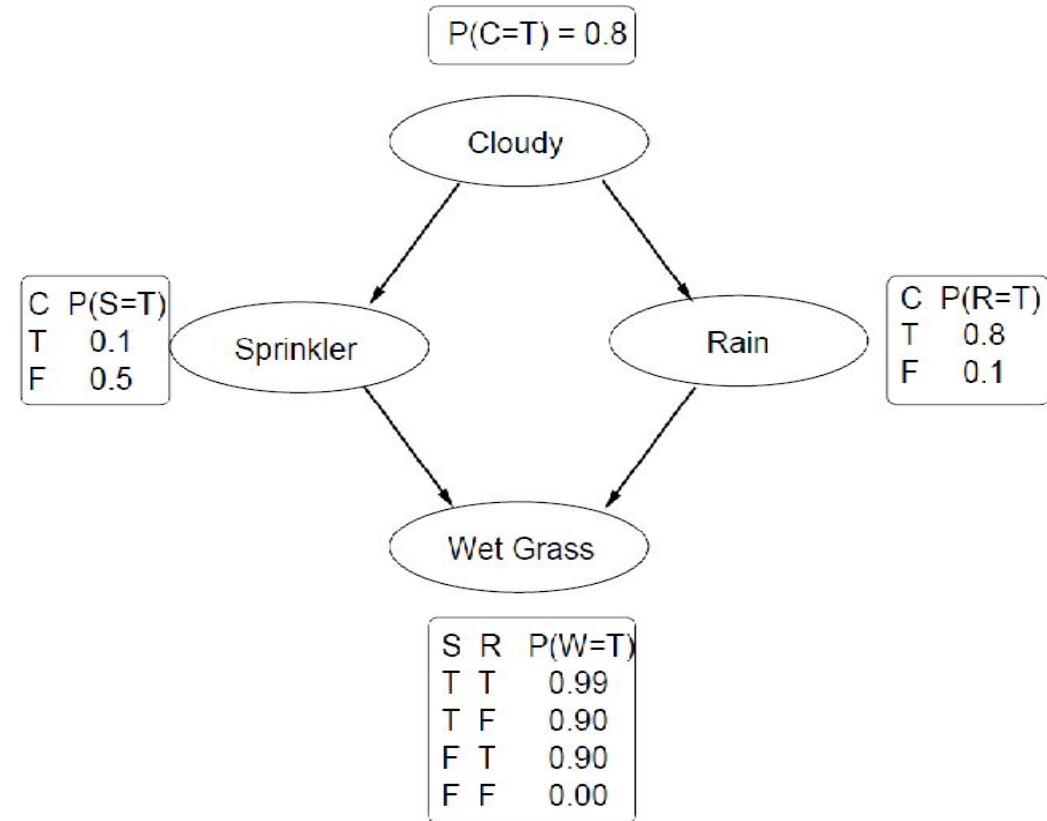


It is cloudy, what's the probability that the grass is wet?

The denominator is known (0.8). The numerator may expressed as a marginal distribution

$$\begin{aligned} P(W_T, C_T) &= \sum_S \sum_R P(W_T, S, R, C_T) \\ &= \sum_S \sum_R P(W_T | S, R) P(S | C_T) P(R | C_T) P(C_T) \end{aligned}$$

- $P(A, B) = P(A | B) P(B)$
(rule of conditional probability)
- $P(W_T, S, R, C_T)$
 $= P(W_T | S, R, C_T) P(S, R, C_T)$
 $= P(W_T | S, R) P(S | R, C_T) P(R, C_T)$
 $= P(W_T | S, R) P(S | C_T) P(R | C_T) P(C_T)$



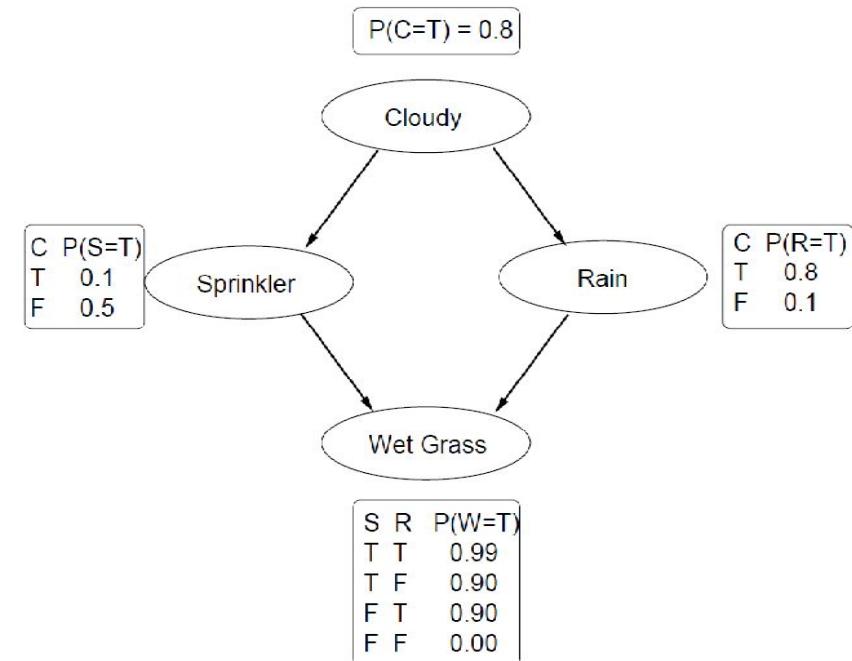
$$P(W_T | S, R) P(S | C_T) P(R | C_T)$$

where the summation are over the variable being T, or F.

From the simple example this is (note $P(C_T)$ has simply been cancelled from the numerator and denominator)

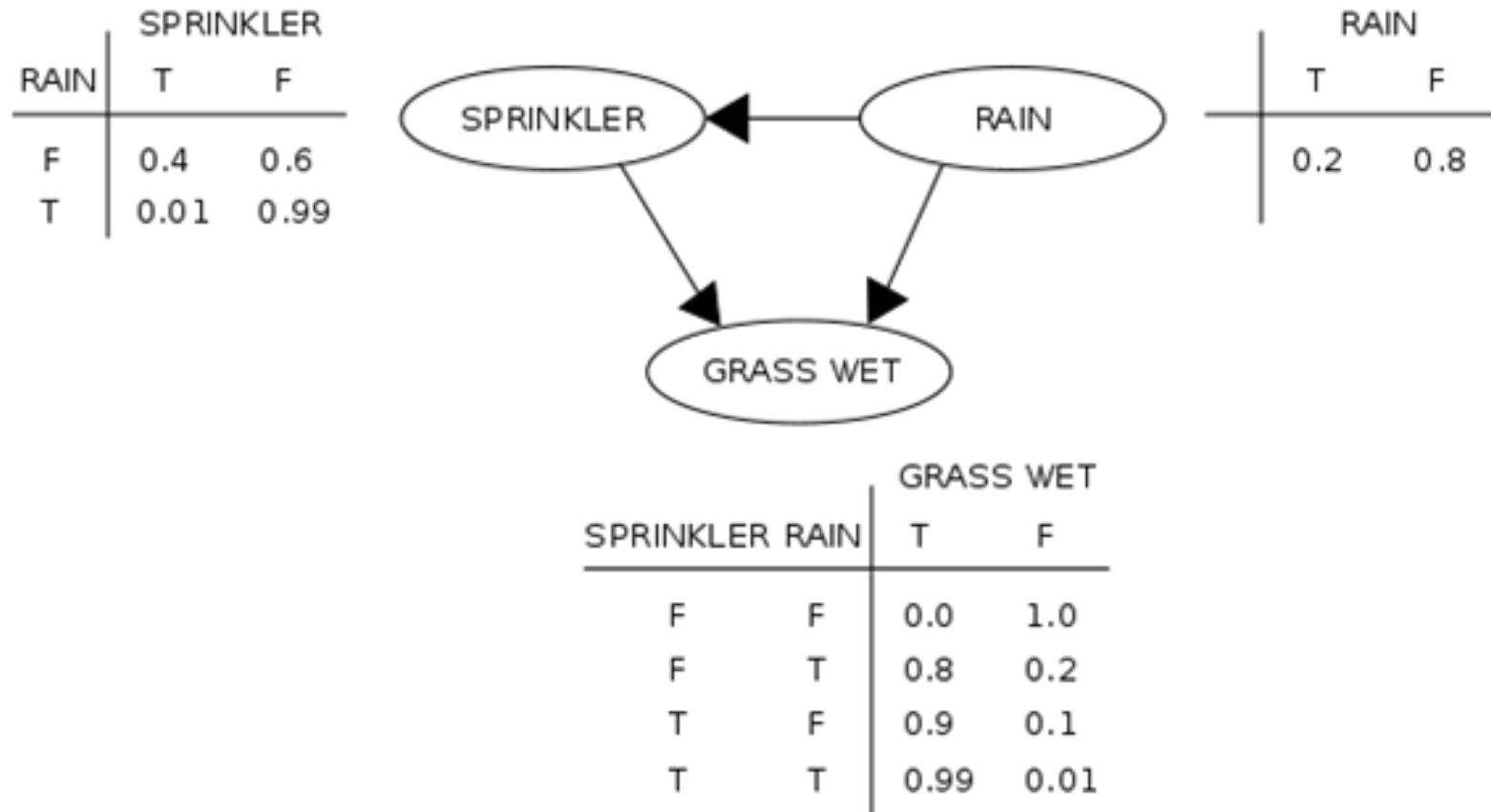
$$\begin{aligned}
 &= P(W_T | S_T, R_T) P(S_T | C_T) P(R_T | C_T) \quad (S = T \text{ & } R = T) \\
 &+ P(W_T | S_F, R_T) P(S_F | C_T) P(R_T | C_T) \quad (S = F \text{ & } R = T) \\
 &+ P(W_T | S_T, R_F) P(S_T | C_T) P(R_F | C_T) \quad (S = T \text{ & } R = F) \\
 &+ P(W_T | S_F, R_F) P(S_F | C_T) P(R_F | C_T) \quad (S = F \text{ & } R = F)
 \end{aligned}$$

$P(W_T C_T) = 0.99 \times 0.1 \times 0.8 + 0.90 \times 0.1 \times 0.2$
$+ 0.90 \times 0.9 \times 0.8 + 0.00 \times 0.9 \times 0.2$
$= 0.7452$



Example

- The grass is wet, what is the probability of rain?



Dempster-Shafer theory

- The **theory of belief functions**, also referred to as **evidence theory** or **Dempster–Shafer theory (DST)**, is a general framework for reasoning with uncertainty, with understood connections to other frameworks such as probability, possibility and imprecise probability theories.
- First introduced by **Arthur P. Dempster** in the context of statistical inference, the theory was later developed by **Glenn Shafer** into a general framework for modeling epistemic uncertainty—a mathematical theory of evidence.
- The theory allows one to **combine evidence** from different sources and arrive at a **degree of belief** (represented by a mathematical object called **belief function**) that takes into account all the available evidence.
- Dempster–Shafer theory is a generalization of the **Bayesian theory of subjective probability**.



[Arthur P. Dempster](#)

Basic

- Beside the probability, there is also a thing called belief.
- That is belief of a person when doing inference.
- **Example:** Suppose a person come to you with a coin and says that if you toss the coin then 90% of time it will be head.
- Now you do not **believe** that character.
- **X** is the event of coin to be head then my belief **Bel(X) = 0** and **Bel(not X) = 0** because I don't have any evidence to support the fact.
- Now, a very reliable person in which my confidence is 90%, says that the coin is fair.
- So my **Bel(X) = 0.5 * 0.9 = 0.45** and **Bel(not X) = 0.5 * 0.9 = 0.45**
- Total probability is **0.90** (**0.1** less than total probability of **1**)
- That show my **lack of confidence** on the person.

- **Probability theory limitation**

- Assign a single number to measure any situation, no matter how it is complex
- Cannot deal with missing evidence, heuristics, and limited knowledge

- **Dempster-Shafer theory**

- Extend probability theory
- Consider a set of propositions as a whole
- Assign a set of propositions an interval [believe, plausibility] to constraint the degree of belief for each individual propositions in the set
- The belief measure bel is in $[0,1]$
 - 0 – no support evidence for a set of propositions
 - 1 – full support evidence for a set of propositions
- The plausibility of p ,

- **$\text{pl}(p) = 1 - \text{bel}(\neg(p))$**

- Reflect how evidence of $\neg(p)$ relates to the possibility for belief in p
- $\text{Bel}(\text{not}(p))=1$: full support for $\text{not}(p)$, no possibility for p
- $\text{Bel}(\text{not}(p))=0$: no support for $\text{not}(p)$, full possibility for p
- Range is also in $[0,1]$

Need for Dempster-Shafer theory

- DST is a mathematical theory of evidence-based on belief functions and plausible reasoning. It is used to combine separate pieces of information (evidence) to calculate the probability of an event.
- DST offers an alternative to traditional probabilistic theory for the mathematical representation of uncertainty.
- DST can be regarded as, a more general approach to represent uncertainty than the Bayesian approach.
- DST is an evidence theory, it combines all possible outcomes of the problem. Hence it is used to solve problems where there may be a chance that different evidence will lead to some different result.
- For example, Let A represent the proposition "Moore is attractive". Then the axioms of probability insist that $P(A) + P(\neg A) = 1$.
- Now suppose that Andrew does not even know who "Moore" is, then
- We cannot say that Andrew believes the proposition if he has no idea what it means. Also, it is not fair to say that he disbelieves the proposition. It would therefore be meaningful to denote Andrew's belief B of B(A) and B($\neg A$) as both being 0.

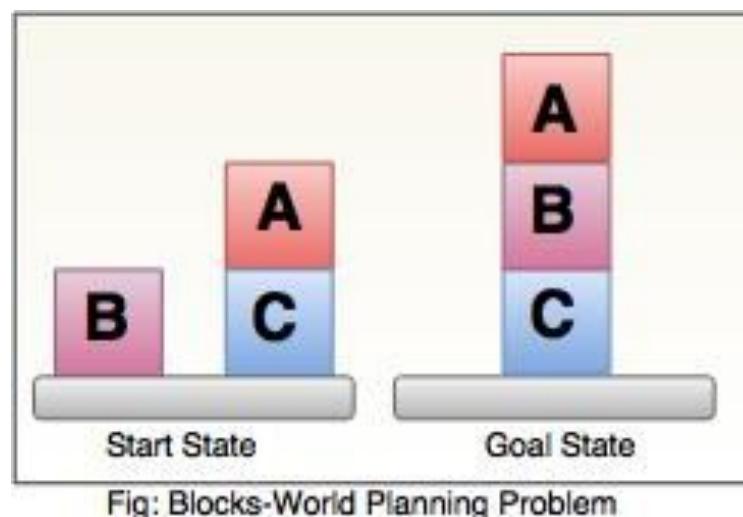
Planning

Planning in AI

- The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.
- The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

Blocks-World planning problem

- The blocks-world problem is known as Sussman Anomaly.
- Noninterleaved planners of the early 1970s were unable to solve this problem, hence it is considered as anomalous.
- When two subgoals G1 and G2 are given, a noninterleaved planner produces either a plan for G1 concatenated with a plan for G2, or vice-versa.
- In blocks-world problem, three blocks labeled as 'A', 'B', 'C' are allowed to rest on the flat surface. The given condition is that only one block can be moved at a time to achieve the goal



Components of Planning System

The planning consists of following important steps:

- Choose the best rule for applying the next rule based on the best available heuristics.
- Apply the chosen rule for computing the new problem state.
- Detect when a solution has been found.
- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

AI Planners

- Well-known AI Planners:
 - STRIPS (Fikes and Nilsson, 1971): theorem proving system
 - ABSTRIPS (Sacerdoti, 1974): added hierarchy of abstractions
 - HACKER (Sussman, 1975): use library of procedures to plan
 - NOAH (Sacerdoti, 1975): problem decomposition and plan reordering

Goal stack planning

- This is one of the most important planning algorithms, which is specifically used by STRIPS.
- The stack is used in an algorithm to hold the action and satisfy the goal. A knowledge base is used to hold the current state, actions.
- Goal stack is similar to a node in a search tree, where the branches are created if there is a choice of an action.

- The important steps of the algorithm are as stated below:
 - i. Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied subgoals on the stack.
 - ii. If stack top is a single unsatisfied goal then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
 - iii. If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.
 - iv. If stack top is a satisfied goal, pop it from the stack.

Partial-Order Planning

Idea:

- works on several subgoals independently
- solves them with subplans
- combines the subplans
- flexibility in ordering the subplans
- least commitment strategy:
 - delaying a choice during search
 - Example, leave actions unordered, unless they must be sequential

Hierarchical Planning

- Hierarchical planning is a planning method based on Hierarchical Task Network (HTN) or HTN planning.
- It combines ideas from Partial Order Planning & HTN Planning.
- HTN planning is often formulated with a single “top level” action called Act, where the aim is to find an implementation of Act that achieves the goal.
- In HTN planning, the initial plan is viewed as a very high level description of what is to be done.
- This plan is refined by applying decomposition actions.
- Each action decomposition reduces a higher level action to a partially ordered set of lower-level actions.
- This decomposition continues until only the primitive actions remain in the plan.

Advantages of Hierarchical Planning:

- The key benefit of hierarchical structure is that, at each level of the hierarchy, plan is reduced to a small number of activities at the next lower level, so the computational cost of finding the correct way to arrange those activities for the current problem is small.
- HTN methods can create the very large plans required by many real-world applications.
- Hierarchical structure makes it easy to fix problems in case things go wrong.
- For complex problems hierarchical planning is much more efficient than single level planning.

Non-linear planning

- This planning is used to set a goal stack and is included in the search space of all possible subgoal orderings. It handles the goal interactions by interleaving method.
- **Advantage of non-Linear planning**

Non-linear planning may be an optimal solution with respect to plan length (depending on search strategy used).

- **Disadvantages of Nonlinear planning**

1. It takes larger search space, since all possible goal orderings are taken into consideration.
2. Complex algorithm to understand.

Algorithm

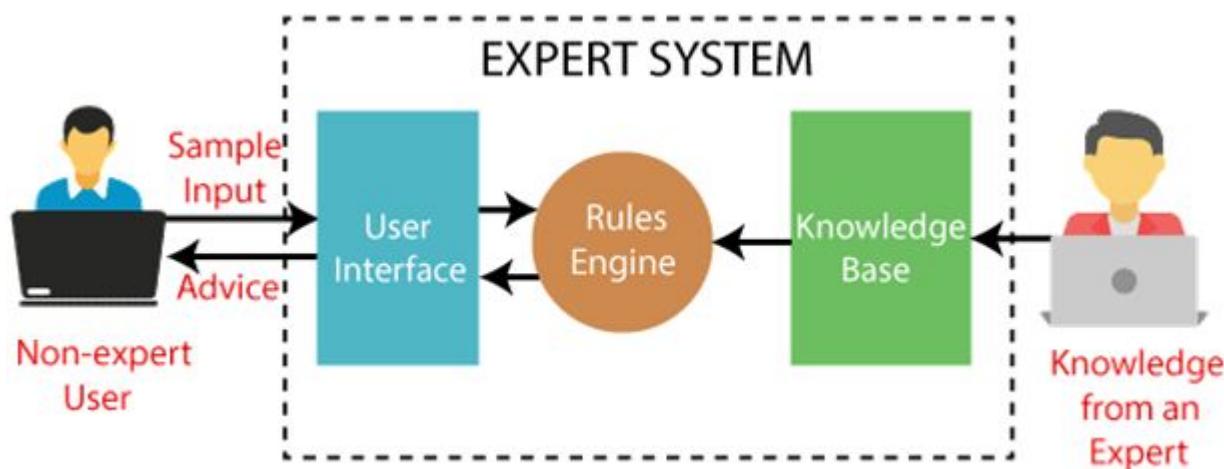
1. Choose a goal 'g' from the goalset
2. If 'g' does not match the state, then
 - Choose an operator 'o' whose add-list matches goal g
 - Push 'o' on the opstack
 - Add the preconditions of 'o' to the goalset
3. While all preconditions of operator on top of opstack are met in state
 - Pop operator o from top of opstack
 - state = apply(o, state)
 - plan = [plan; o]

Unit-7

Expert Systems

What is an Expert System?

- Computer program that is designed to solve complex problems and to provide decision-making ability like a human expert.
- Extract knowledge from its knowledge base using the reasoning and inference rules according to the user queries.
- 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.



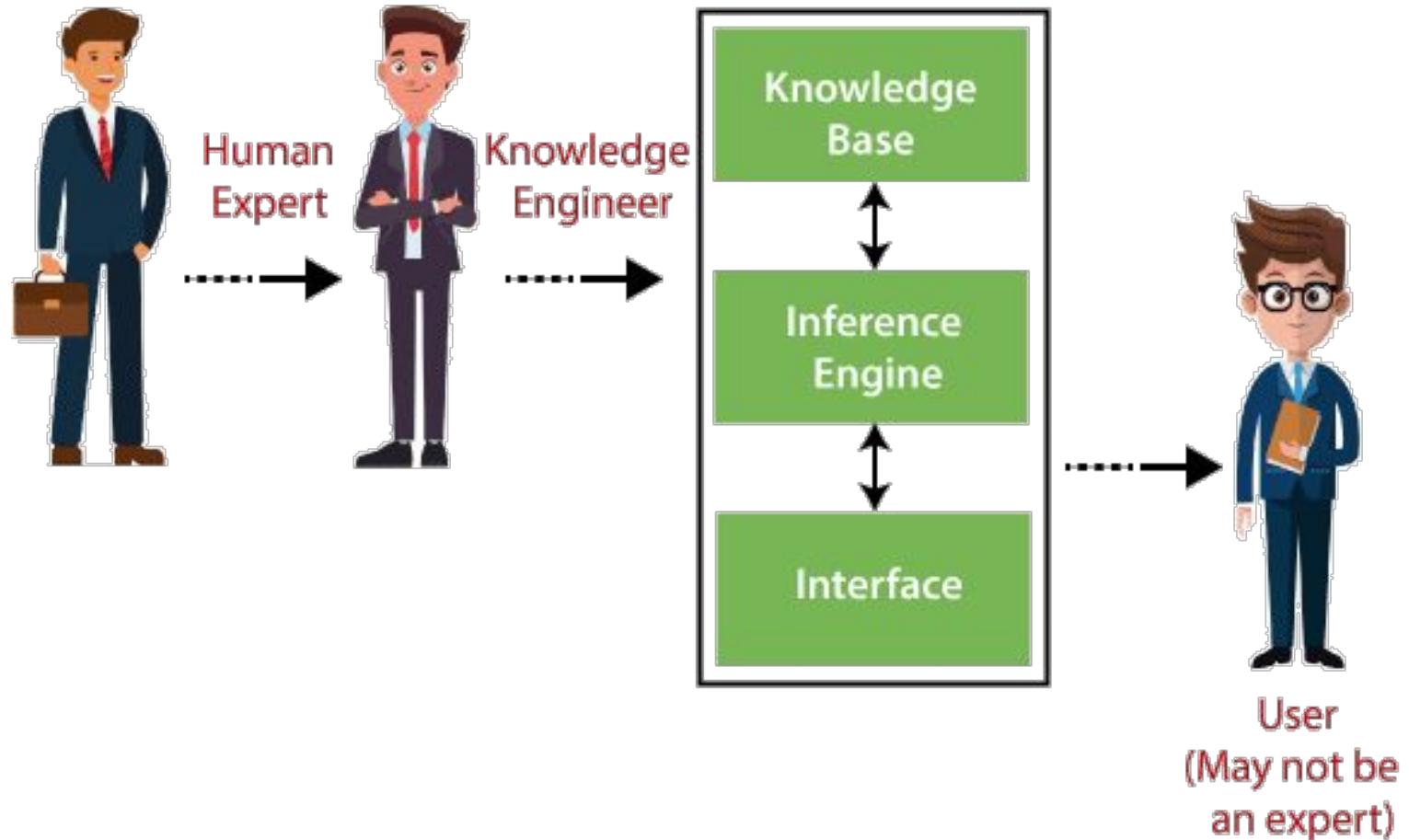
Some Expert Systems

- **DENDRAL:** chemical analysis
- **MYCIN:** Medical
- **PXDES:** Lung cancer
- **CaDeT:** Detect cancer at early stages

Characteristics of Expert System

- **High Performance**
- **Understandable**
- **Reliable**
- **Highly responsive**

Components of Expert System



1. User Interface

- With the help of a user interface, the expert system interacts with the user, takes **queries** as an input in a readable format, and passes it to the inference engine.
- After getting the response from the inference engine, it displays the output to the user.
- In other words, it is an interface that helps a non-expert user to communicate with the expert system to find a solution.

2. Inference Engine

- brain of the expert system as it is the main processing unit of the system
- **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on facts and rules.
- **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.
- Inference Engine uses **forward chaining / backward chaining** to derive the solution.

3. Knowledge Base

- stores knowledge acquired from the different experts (similar to database)
- collections of objects and their attributes
- Two components: **Factual Knowledge & Heuristic Knowledge**

Knowledge Acquisition in Expert Systems

- Knowledge acquisition is the process of extracting, structuring and organizing knowledge from one source, usually human experts, so it can be used in software such as an ES.
- This is often the major obstacle in building an ES.
- There are **three** main topic areas central to knowledge acquisition that require consideration in all ES projects.
 - **First**, the domain must be evaluated to determine if the type of knowledge in the domain is suitable for an ES.
 - **Second**, the source of expertise must be identified and evaluated to ensure that the specific level of knowledge required by the project is provided.
 - **Third**, if the major source of expertise is a person, the specific knowledge acquisition techniques and participants need to be identified.

Advantages of Expert System

- highly reproducible
- They can be used for risky places where the human presence is not safe.
- Error possibilities are less if the KB contains correct knowledge.
- The performance of these systems remains steady as it is not affected by emotions, tension, or fatigue.
- They provide a very high speed to respond to a particular query.

Limitations of Expert System

- The response of the expert system may get wrong if the knowledge base contains the wrong information.
- Like a human being, it cannot produce a creative output for different scenarios.
- Its maintenance and development costs are very high.
- Knowledge acquisition for designing is much difficult.
- For each domain, we require a specific ES, which is one of the big limitations.
- It cannot learn from itself and hence requires manual updates.