IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Problem Solving Methods & Introduction to Search Strategies

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 01
Presentation Date – 21/09/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

1. **Make use of** Problem-solving methods in AI to find solutions for complex issues.
2. **Identify** the search strategies in AI which provide systematic methods for exploring possible solutions to problems.

Problem Solving Methods

Problem Solving Methods

- The method of solving problem through AI ***involves the process of defining the search space, deciding start and goal states then finding the path from start to goal state through search space.***
- The movement from start state to goal state is guided by set of rules specifically designed for that particular problem.

Problem

- It is the question which is to be solved. For solving a problem it needs to be ***precisely defined.***
- The definition means, ***defining the start state, goal state, other valid states and transitions.***
- ***Finding the Solution***
- After representation of the problem and related knowledge in the suitable format, the ***appropriate methodology is chosen*** which uses the knowledge and transforms the start state to goal state.
- The Techniques of finding the solution are called ***search techniques.***
- Various search techniques are developed for this purpose.

Problem Solving Agents

- In Artificial Intelligence, ***Search techniques*** are universal problem-solving methods.
- ***Rational agents or Problem-solving agents*** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.
- Problem-solving agents are the ***goal-based agents*** and use atomic representation. So here, we will discuss various problem-solving search algorithms.

Search Algorithm Terminologies:

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

- ***Search Space***: Search space represents *a set of possible solutions*, which a system may have.
- ***Start State***: It is a *state from which begins the search.*
- ***Goal test***: It is a function *where agent observe the current state and returns whether the goal state is achieved or not*.

Search Algorithm Terminologies: (Contd..)

- ***Search tree***: A tree representation of search problem is called Search tree. The *root of the search tree is the root node* which is corresponding to the initial state.
- ***Actions***: It gives the description of *all the available actions* to the agent.
- ***Transition model***: A description of *what each action do*, can be represented as a transition model.

Search Algorithm Terminologies: (Contd..)

- ***Path Cost***: It is a function which *assigns a numeric cost to each path.*
- ***Solution***: It is an *action sequence* which leads from the start node to the goal node.
- ***Optimal Solution***: If a solution *has the lowest cost* among all solutions.

Properties of Search Algorithms:

- **Completeness**: A search algorithm is said to be completed *if it guarantees to return a solution if at least any solution exists for any random input.*
- **Optimality**: If a solution found for an algorithm is guaranteed to be the best solution (*lowest path cost*) among all other solutions, then such a solution for is said to be an optimal solution.
- **Time Complexity**: Time complexity is a *measure of time for an algorithm to complete its task.*
- **Space Complexity**: It is the *maximum storage space required at any point during the search*, as the complexity of the problem.

Search Strategies / Control Strategies

Search Strategies

- It is one of the most important component of problem solving that ***describes the order of application of the rules to the current state.*** That means:
  - Helps us to decide ***which rule to apply next.***
  - What to do ***when there are more than 1 matching rules?***

[NOTE: This question usually arises when we have more than one rule (and sometimes fewer than one rule) that will match with the left side of the current state.]

Requirements of Search Strategies

There are mainly **<u>two requirements to choose a control strategy.</u>**

- The first requirement of a good control strategy is that ***it should cause motion towards solution.***
- The second requirement of a good control strategy is that ***it should explore the solution space in a systematic manner.***

Requirements of Search Strategies (Contd..)

Example:

Consider the water jug problem. On each cycle, choose at random from among the applicable rules. This strategy is better than the first. It causes motion. It will lead to a solution eventually. But we are ***likely to arrive at the same state several times during the process and to use many more steps than are necessary***. Because the ***control strategy is not systematic***, we may explore a particular useless sequence of operators several times before we finally find a solution.

Search Directions

To solve real world problems, effective control strategy must be needed.

There are ***2 directions in which search could proceed.***

1. **Data driven search (forward chaining)** [start state → goal state]
2. **Goal driven search (backward chaining)** [goal state → start state]

Search Directions

Forward Chaining

- Forward chaining ***starts from known facts and applies inference rules to extract more data until it reaches to the goal.***
- It is a ***bottom-up approach.***
- It is known as ***data driven inference technique*** as we reach to the goal using available data.
- It applies breadth first search strategy.

Backward Chaining

- It ***starts from the goal and works backward through inference rules to find the required facts that support the goal.***
- It is a ***top down approach***.
- It is known as ***goal driven technique*** as we start from the goal and divide into subgoals to extract the facts.
- It applies depth first search strategy.

Search Directions

Forward Chaining

- Forward chaining tests for all available rules.
- It is suitable for planning, monitoring, control and interpretation application.

Backward Chaining

- Backward chaining only tests for few required rules.
- It is suitable for diagnostic, prescription and debugging applications.

Types of Search Strategies

Types of Search Strategies

Based on the search problems, there are mainly ***two types of search strategies:***

1. **Uninformed search (**Blind search / Exhaustive search / Brute force search)
2. **Informed search (**Heuristic search / Intelligent search)

1. Uninformed / Blind / Exhaustive search

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

1.Uninformed / Blind / Exhaustive search (Contd..)

Following are the various types of uninformed search algorithms:

1. ***Breadth-first search***
2. ***Depth-first search***
3. ***Depth limited search***
4. ***Iterative deepening depth-first search***
5. ***Uniform cost search***
6. ***Bidirectional Search***

2. Informed / Intelligent Search

- Informed search algorithms ***use domain knowledge***. In an informed search, ***problem information is available which can guide the search.***
- Informed search strategies ***can find a solution more efficiently than an uninformed search*** strategy. Informed search is also called a ***Heuristic search.***
- A <u>**heuristic**</u> is a way which might not always be guaranteed for best solutions but ***guaranteed to find a good solution*** in reasonable time.
- Informed search can ***solve much complex problem*** which could not be solved in another way.

2. Informed / Intelligent Search (Contd..)

- It uses ***Heuristic function***, which maps each state to a numerical value to depict the goodness of a node.

- Represented as:

H(n)=value

(Where ,H(n) is a heuristic function and ‘n’ is the current state.)

2. Informed / Intelligent Search (Contd..)

Following are the various types of informed search algorithms:

1. ***Hill Climbing***
2. ***Best-First Search***
3. ***A* Algorithm***
4. ***AO* Algorithm***
5. ***Beam Search***
6. ***Constraint Satisfaction***

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Uninformed Search (BFS, DFS)

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 02
Presentation Date – 21/09/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Understand the concept of uninformed search where little or no domain-specific information (heuristics) is available to guide the search process.

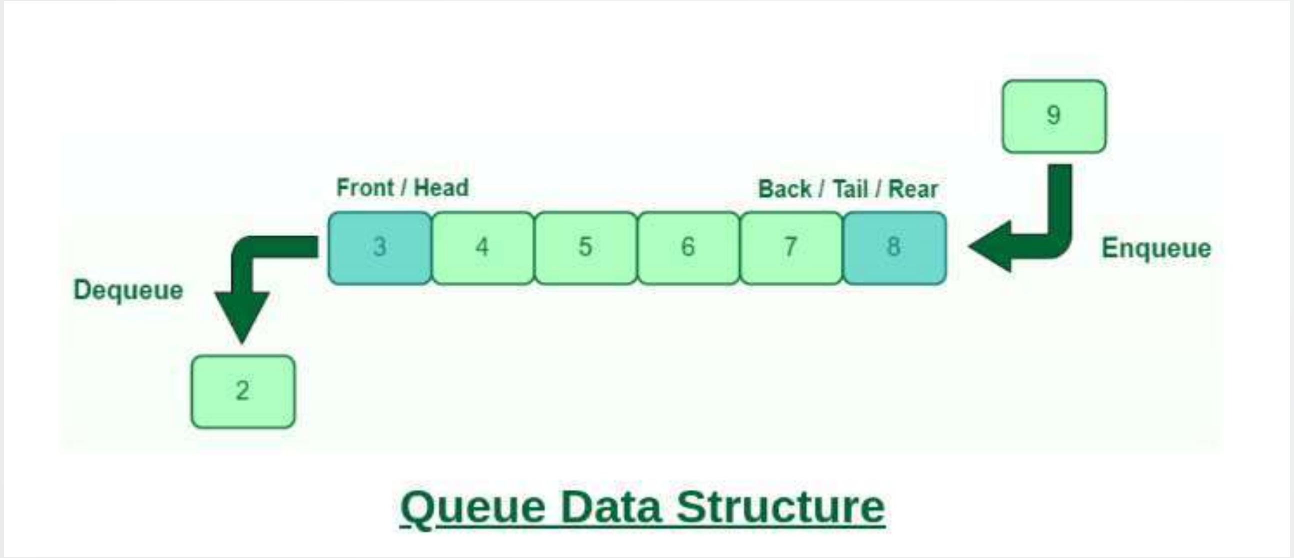
Uninformed / Exhaustive / Blind Search (Breadth First Search)

Breadth First Search (BFS)

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm ***searches breadth wise*** in a tree or graph, so it is called breadth-first search.
- BFS algorithm ***explores all the nodes at a given depth*** before proceeding to the next level.
- The breadth-first search algorithm is an example of a ***general-graph search algorithm.***
- BFS is ***used where the given problem is very small*** and ***space complexity is not considered.***

Breadth First Search (BFS) [Contd..]

- Breadth-first search ***implemented using FIFO queue data structure.***



Breadth First Search (BFS) [Contd..]

Algorithm:

- **S1:** Enter the starting node on Queue.
- **S2**: If Queue is empty, then return fail and stop.
- **S3**: If first element on Queue is the Goal node, then return success and stop.
- **S4**: Remove and expand first element from the Queue and place children at end of the Queue.
- **S5**: Goto S2.

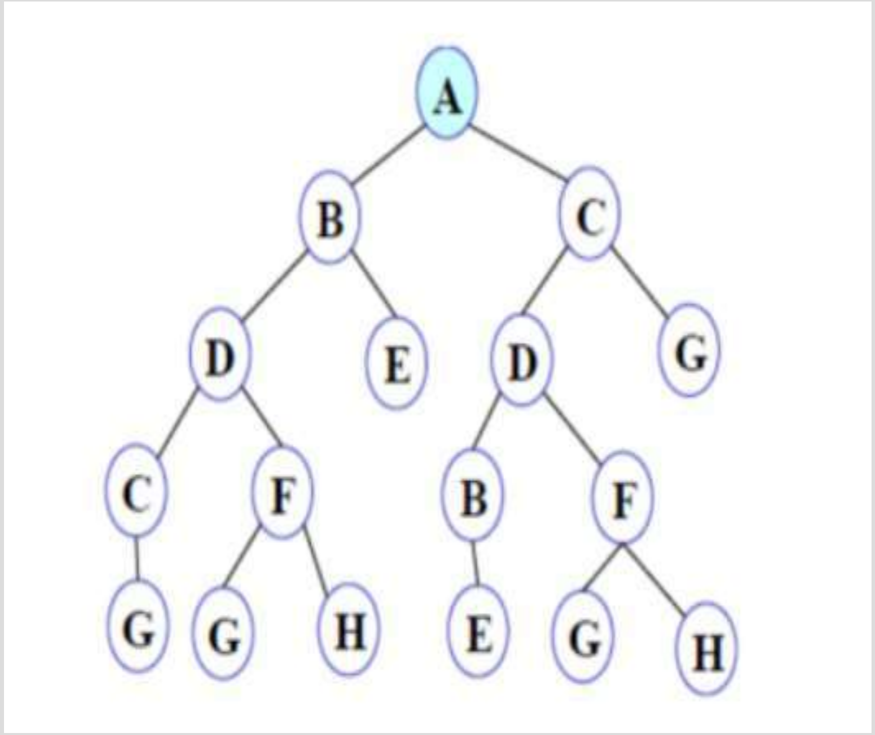
Example-1(BFS)

Consider the following State Space to be searched:

Here node **'A'** is the source or start or initial node and node **'G'** is the goal node.

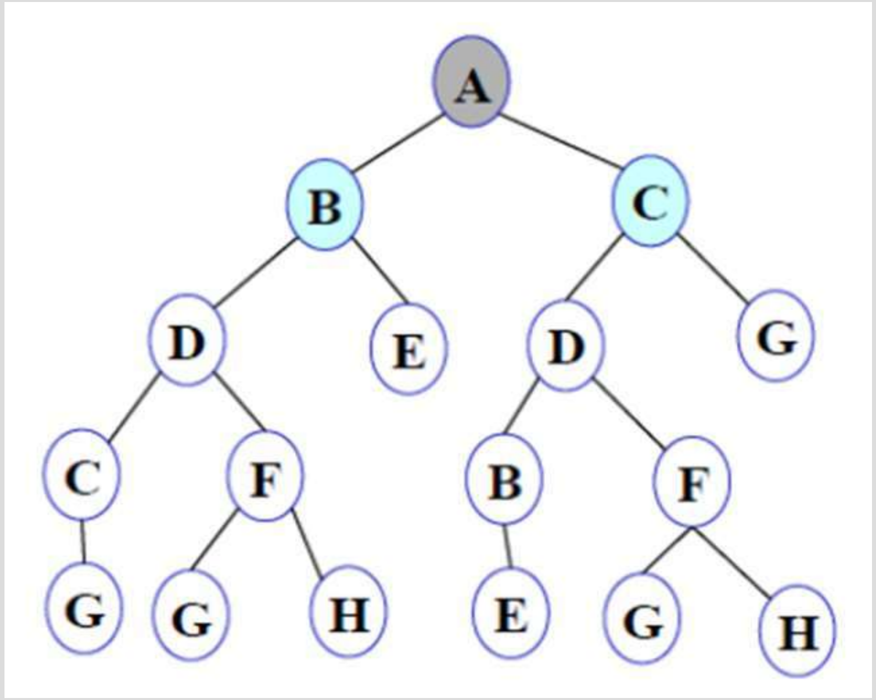
- **Step 1:** Initially Queue contains only one node corresponding to the source state A.

i.e., Queue : A



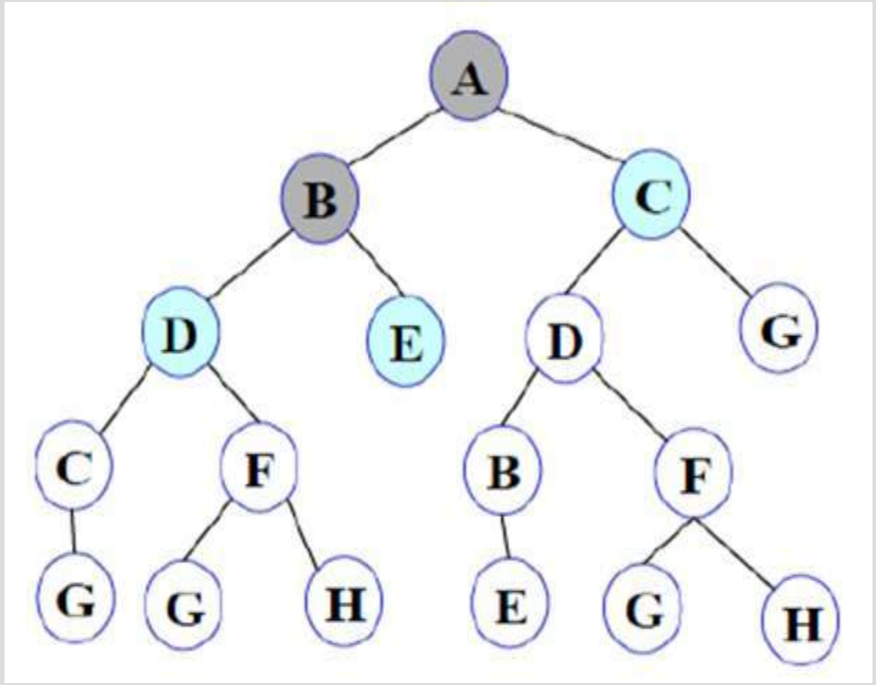
- **Step 2**: Node A is removed from Queue. The node is expanded, and its children B and C are generated. They are placed at the back of Queue.

  i.e., Queue : B C



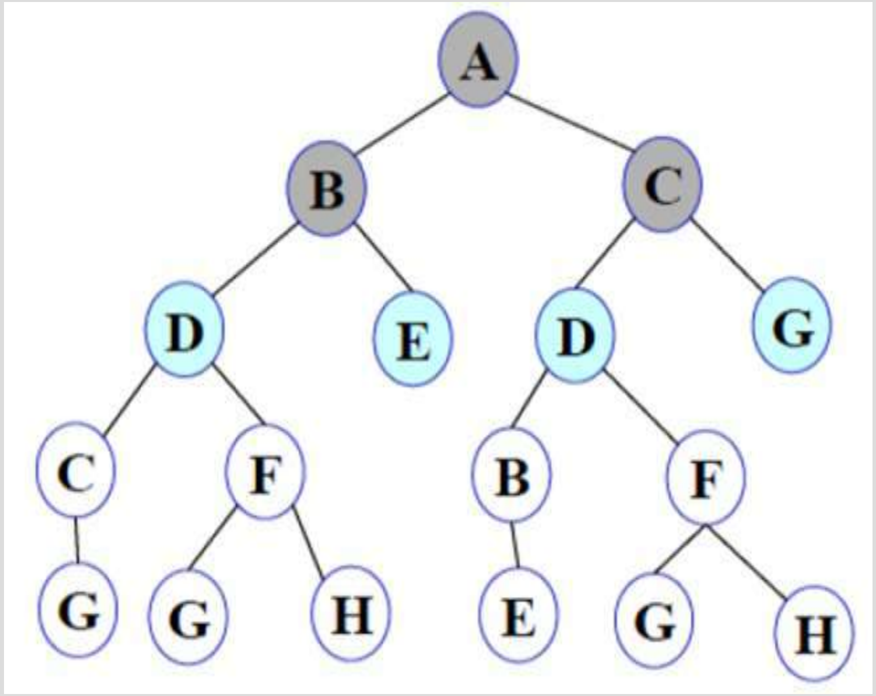
- **Step 3**: Node B is removed from Queue and is expanded. Its children D, E are generated and put at the back of Queue.

  i.e., Queue : C D E



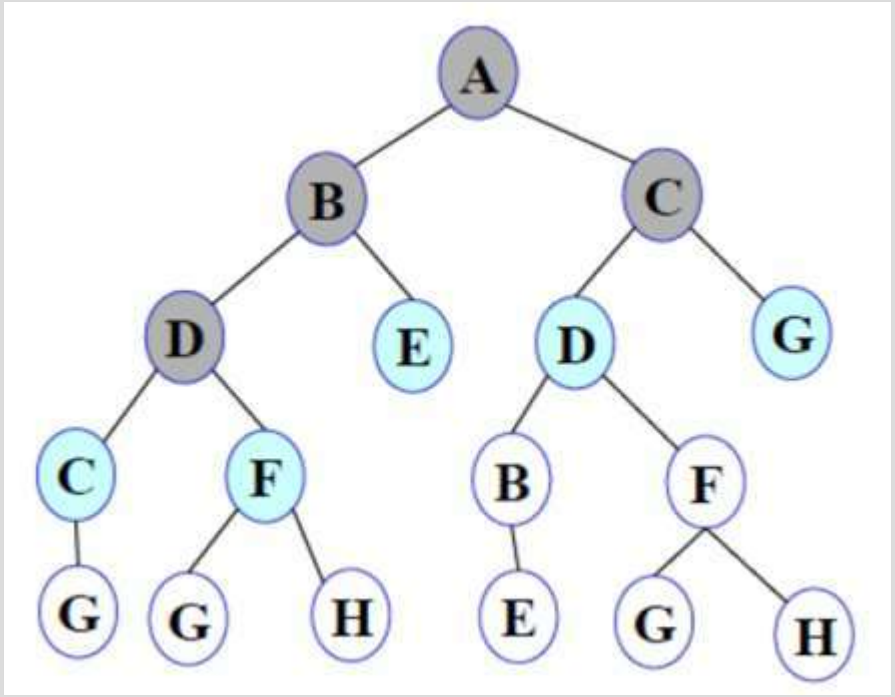
- **Step 4**: Node C is removed from Queue and is expanded. Its children D and G are added to the back of Queue.

  i.e., Queue: D E D G

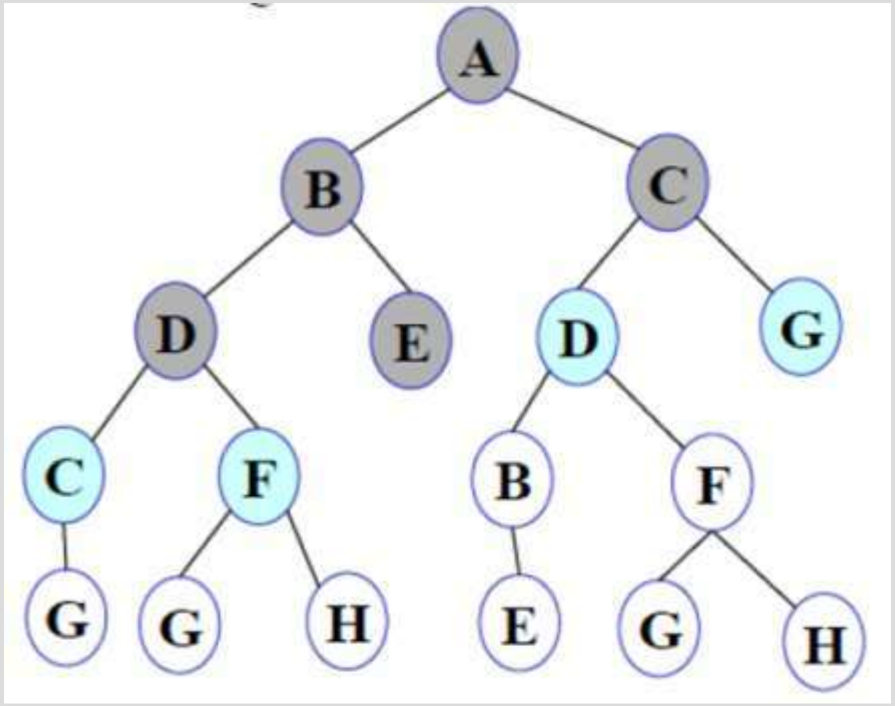


- **<u>Step 5</u>:** Node D is removed from Queue. Its children C and F are generated and added to the back of Queue.

  i.e., Queue: E D G C F

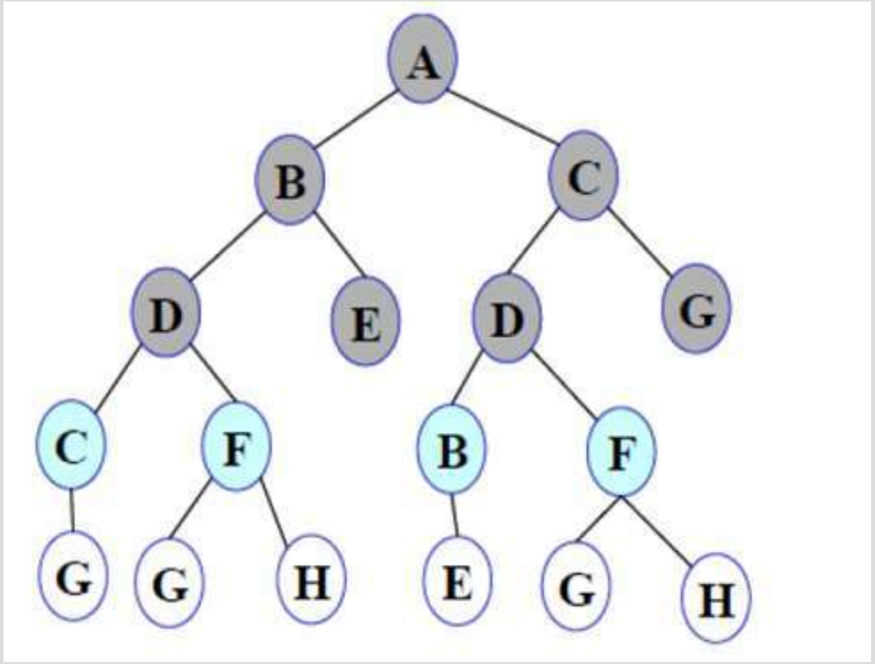


- **<u>Step 6</u>:** Node E is removed from Queue. It has no children.
- i.e., Queue: D G C F

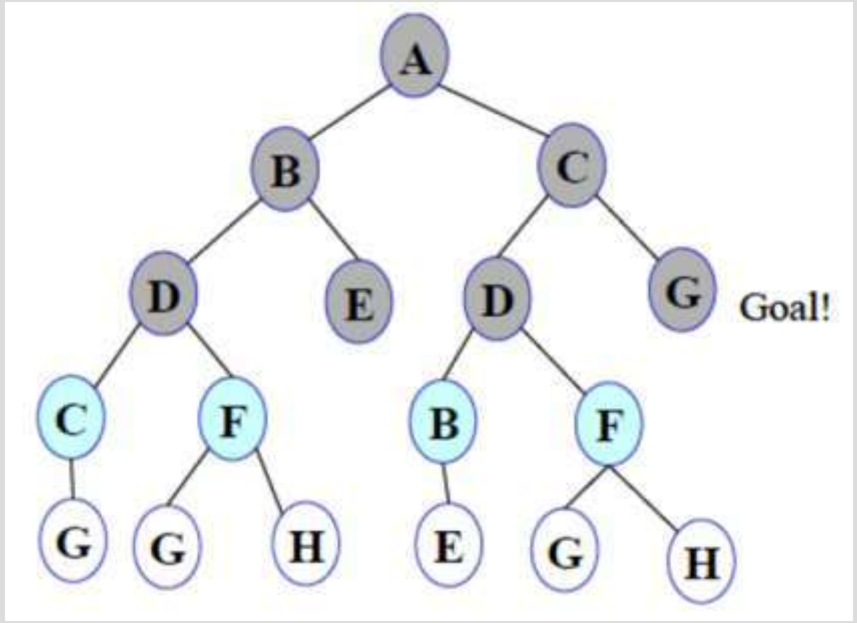


- **Step 7:** D is expanded; B and F are put in OPEN.

  i.e., Queue: G C F B F



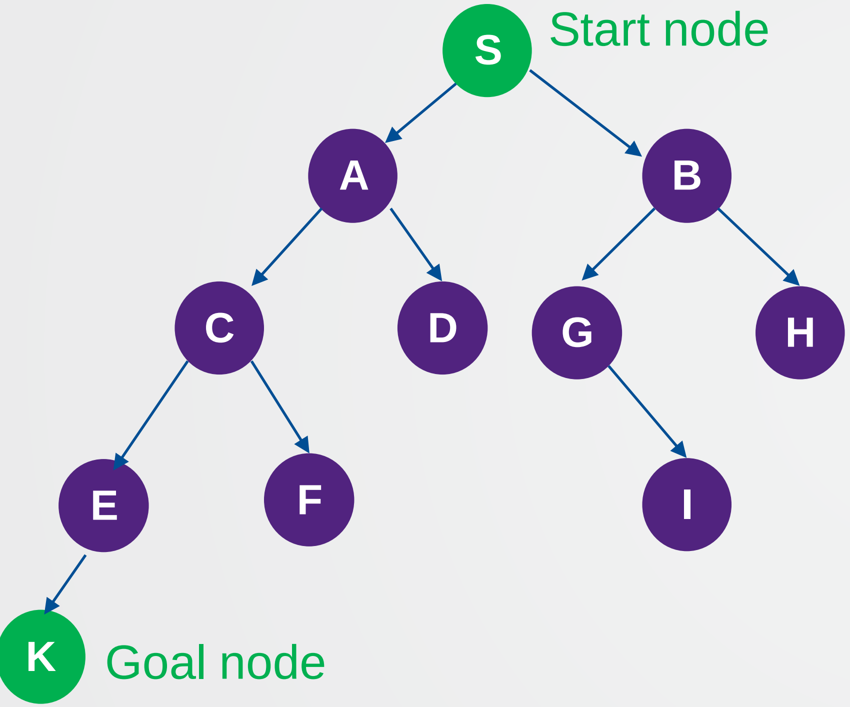
- **Step 8: G is selected for expansion. It is found to be a goal node**. So the algorithm returns the path **A B C D E D G** by following the parent pointers of the node corresponding to G. The algorithm terminates.



GOAL NODE FOUND!!

TRAVERSAL ORDER: A B C D E D G

Example-2 (BFS)



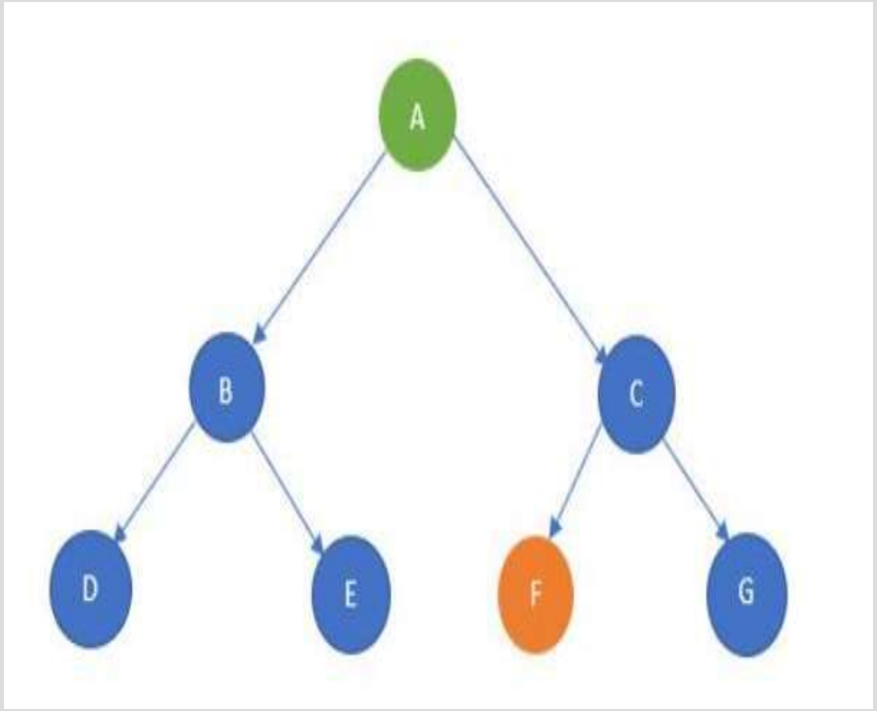
Queue: S (S is not goal node, remove and expand)
Queue: A B (A is not goal node, remove and expand)
Queue: B C D (B is not goal node, remove and expand)
Queue: C D G H (C is not goal node, remove and expand)
Queue: D G H E F (D is not goal node, no expansion, G is not the goal node, remove and expand)
Queue: H E F I (H is not goal node, no expansion, E is not the goal node, remove and expand)
Queue: F I K (Remove F, I as no expansion) K goal node found.

TRAVERSAL ORDER: S→A→B→C→D→G→H→E→F→I→K

Example-3 (BFS)

Consider the following State Space to be searched:

Let A be the start state and F be the final or goal state to be searched.



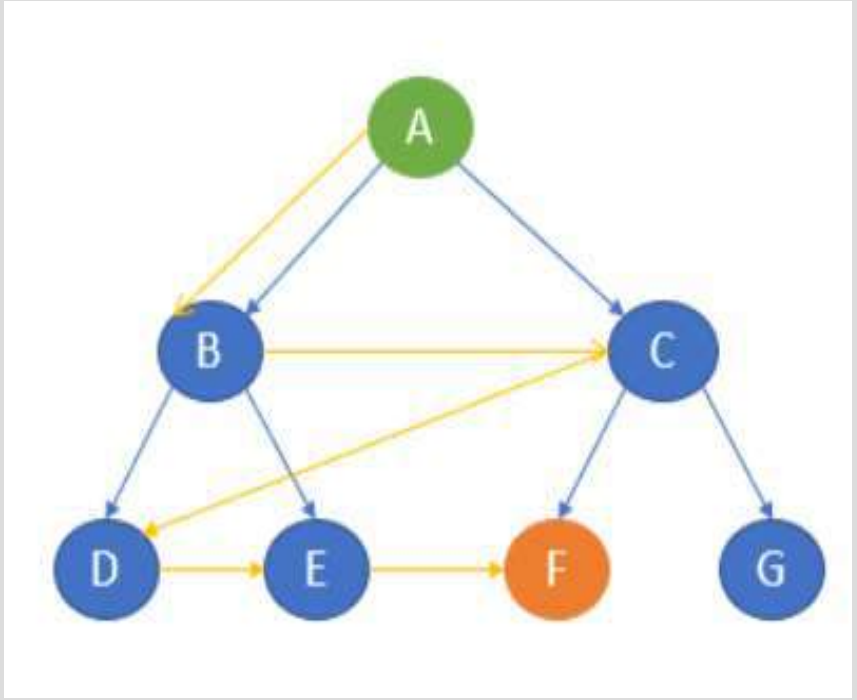
Shortcut method for BFS Traversal:

It starts from A and then travel to the next level and visits B and C and then

travel to the next level and visits D, E, F and G. Here, the goal state is defined as F. So, the traversal will stop at F.

The path of traversal is:

A —-> B —-> C —-> D —-> E —-> F



Breadth First Search (BFS) - Advantages

- One of the ***simplest search strategies.***
- BFS will ***provide a solution if any solution exists***.
- If there are more than one solutions for a given problem, then BFS will ***provide the minimal solution*** which requires the least number of steps. (refer example-2)

Breadth First Search (BFS) - Disadvantages

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

Breadth First Search (BFS) - Analysis

- **Time Complexity**: $O(b^d)$
- **Space Complexity**: $O(b^d)$

  [where b→ branching factor and d→ depth]
- **Solution** : Optimal.

Breadth First Search (BFS) – What is the need?

Some of the most vital aspects that make this algorithm your first choice are:

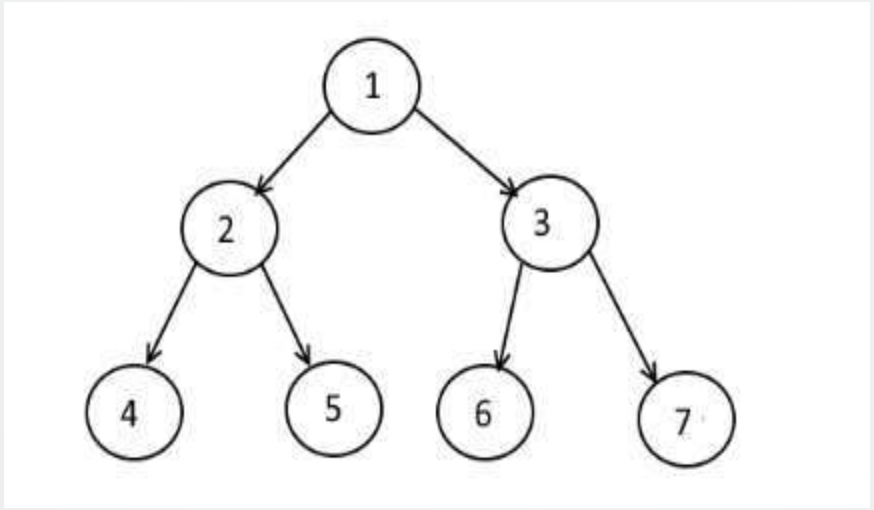
- BFS can ***traverse through a graph in the smallest number of iterations***.
- The architecture of the BFS algorithm is ***simple and robust.***
- The result of the BFS algorithm ***holds a high level of accuracy*** in comparison to other algorithms.
- ***No possibility of this algorithm getting caught up in an infinite loop problem.***

Breadth First Search (BFS) – Applications

- **Un-weighted Graphs:** BFS algorithm can easily create the shortest path and a minimum spanning tree to visit all the vertices of the graph in the shortest time possible with high accuracy.
- **P2P Networks:** BFS can be implemented to locate all the nearest or neighboring nodes in a peer to peer network. This will find the required data faster.
- **Navigation Systems:** BFS can help to find all the neighboring locations from the main or source location.
- **Network Broadcasting:** A broadcasted packet is guided by the BFS algorithm to find and reach all the nodes it has the address for.

Try

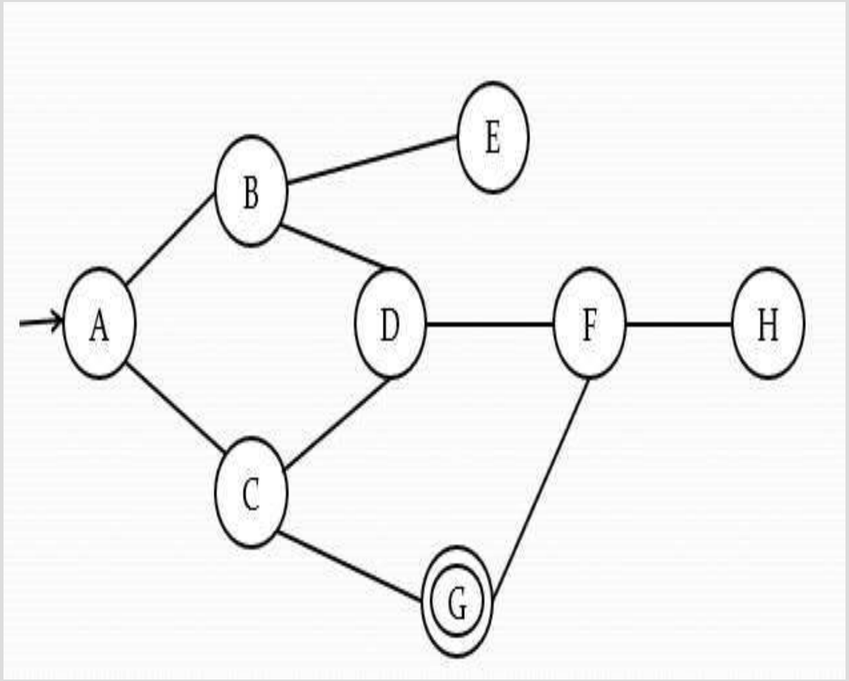
The graph is given with nodes from 1 to 7. Find the shortest path from 1 to 7 using BFS traversal.



Try

The graph is given with nodes from A to H. Find the shortest path from A to G using BFS traversal.

.



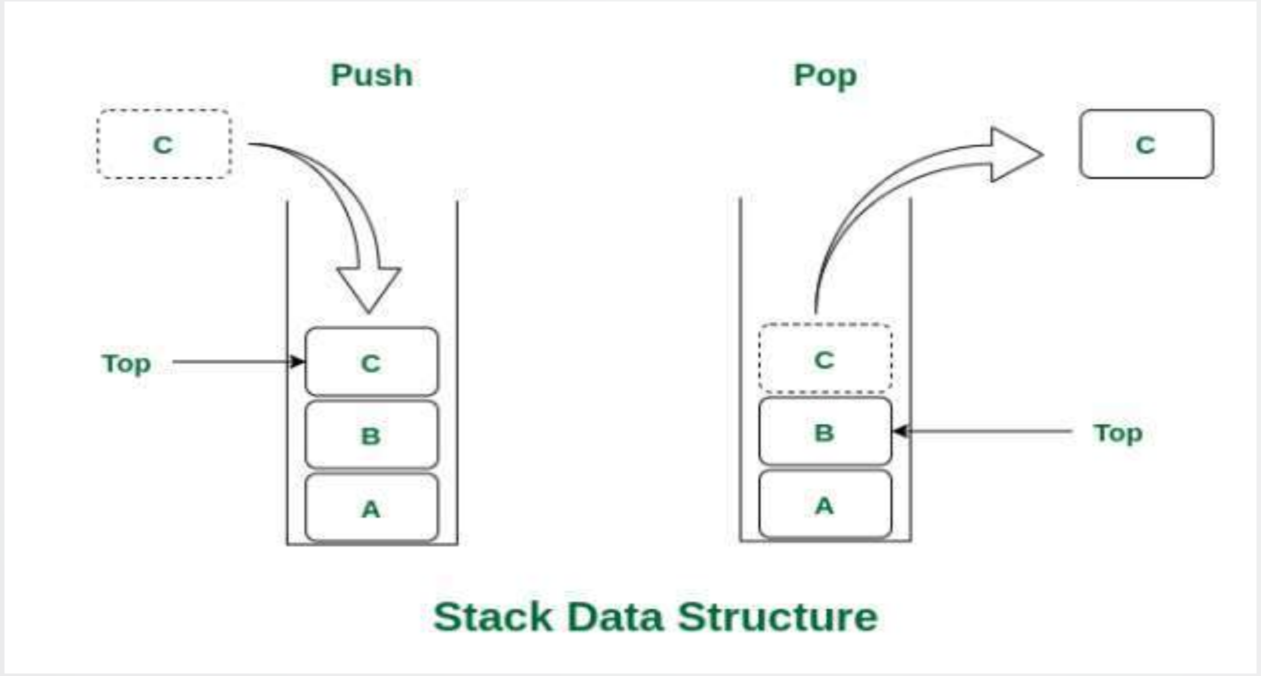
Uninformed / Exhaustive / Blind Search (Depth First Search)

Depth First Search (DFS)

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

Depth First Search (DFS) [Contd..]

- DFS ***uses a stack data structure (LIFO)*** for its implementation.



Depth First Search (DFS) [Contd..]

Algorithm:

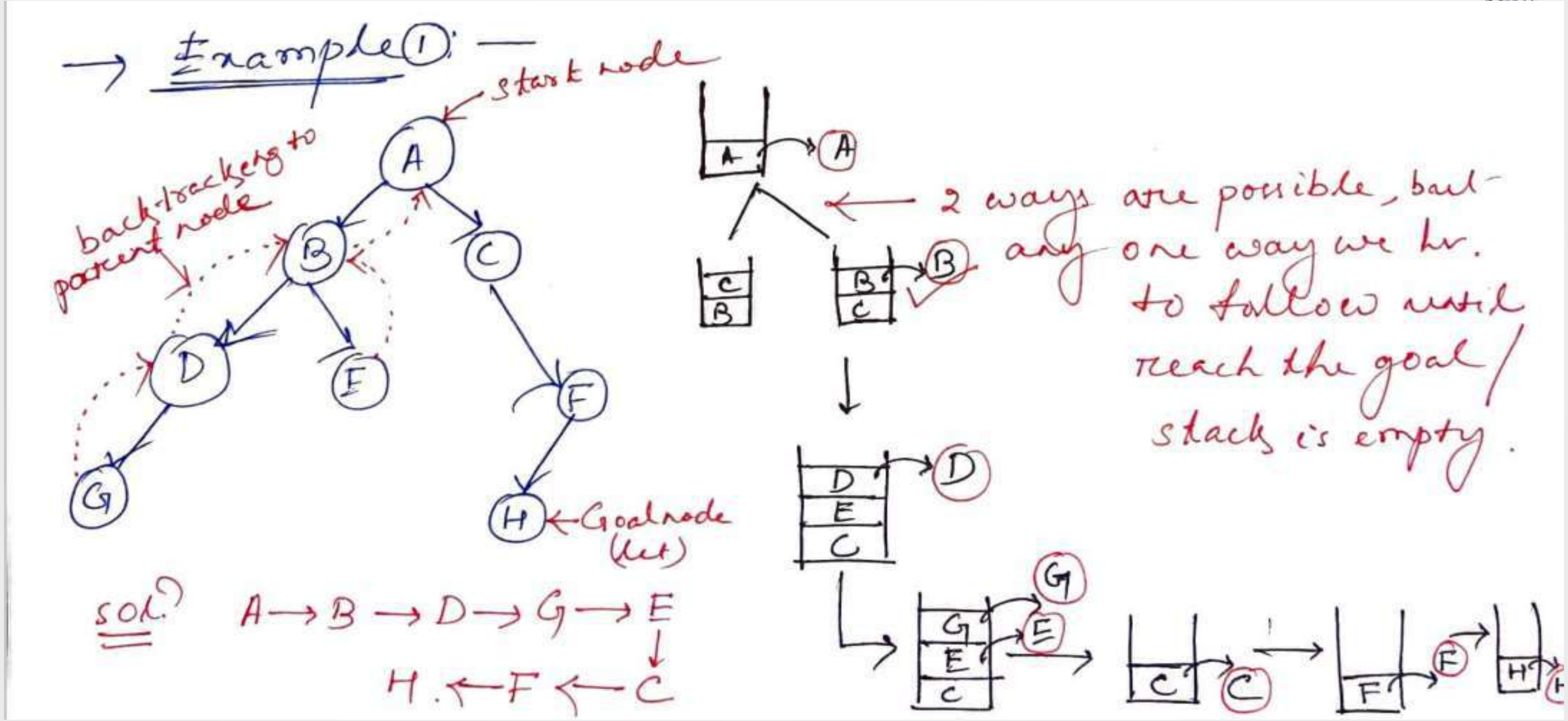
- **S1**: If initial state is a goal state, quit and return success.
- **S2**: Otherwise do the following until success or failure is reported:

  a) Generate a successor ‘E’ of the initial state. If there are no more successors, signal failure.

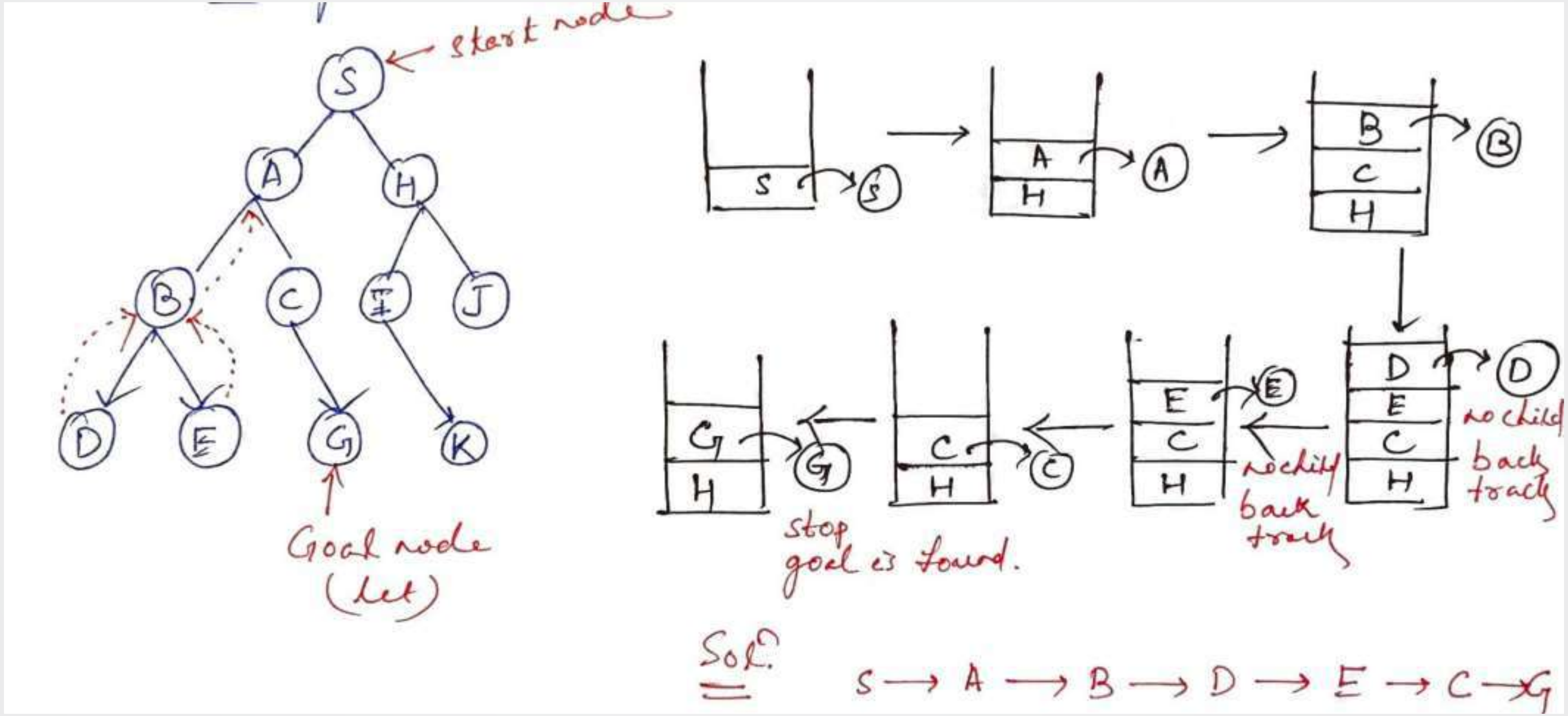
  b) Call Depth-First-Search with ‘E’ as the initial state.

  c) If success is obtained, return success, otherwise continue in this loop.

Example-1 (DFS)


→ Example ①: —
start node
back-tracking to parent node
A
B
C
D
E
F
G
H
Goal node (let)
sol^n A → B → D → G → E ↓ C → F → H.
2 ways are possible, but any one way we hv. to follow until reach the goal / stack is empty.
A
C B
B C
D E C
G E C
C
F
H

Example-2 (DFS)

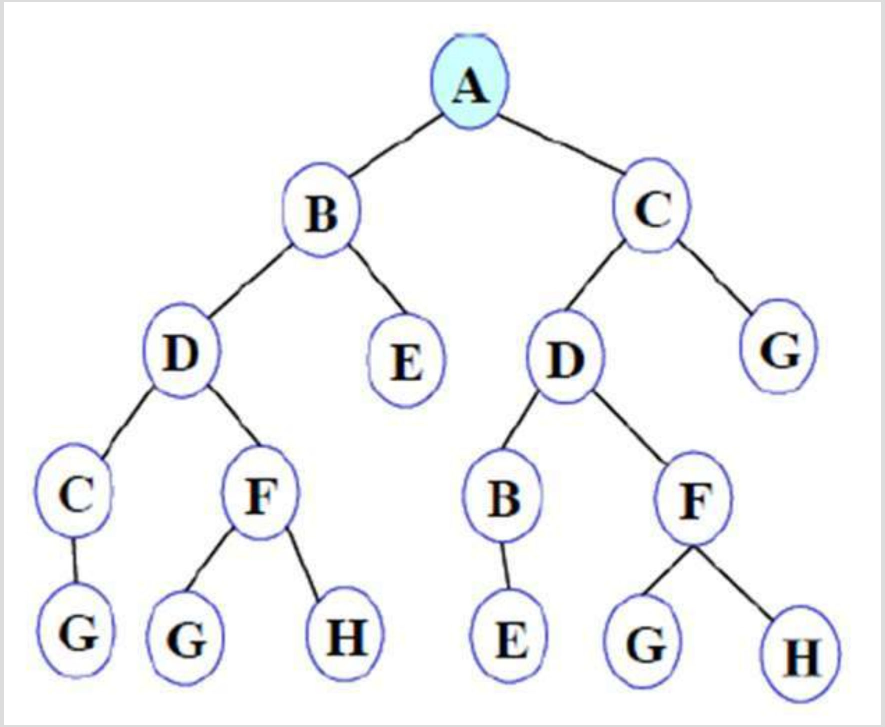


Example-3 (DFS)

Let us consider the following tree. Here node 'A' is the source or start or initial node and node 'G' is the goal node.

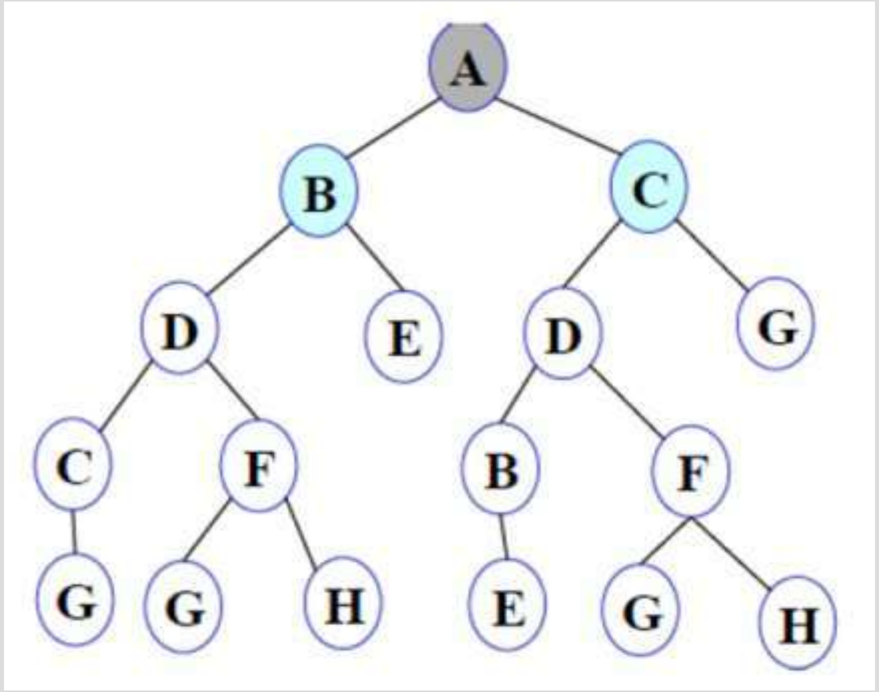
Step 1: Initially Stack contains only one node corresponding to the source state A.

i.e., Stack : A



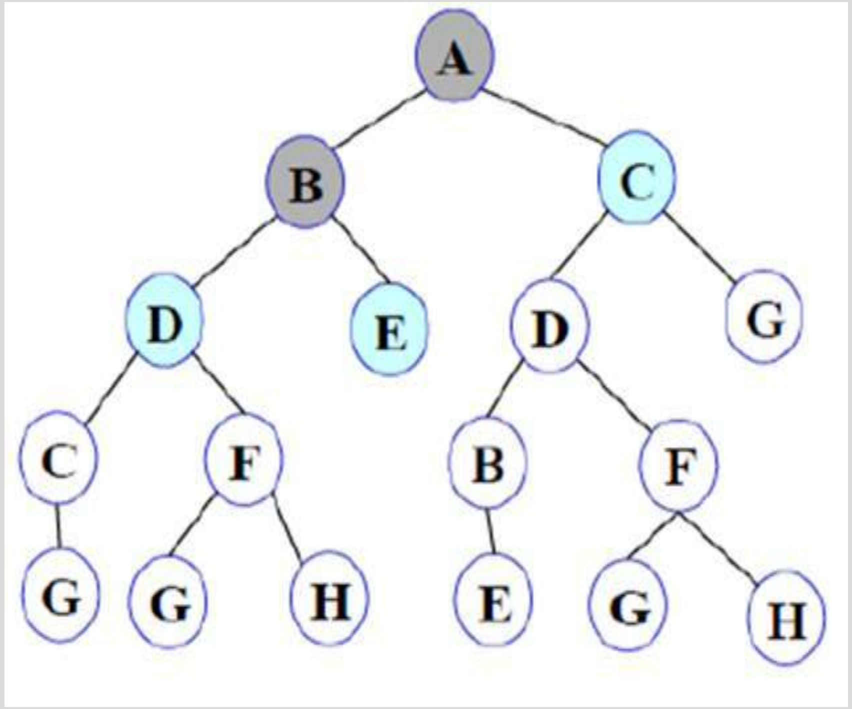
Step 2: Node A is removed from Stack. A is expanded and its children B and C are put on the top of the Stack.

i.e., Stack : B C



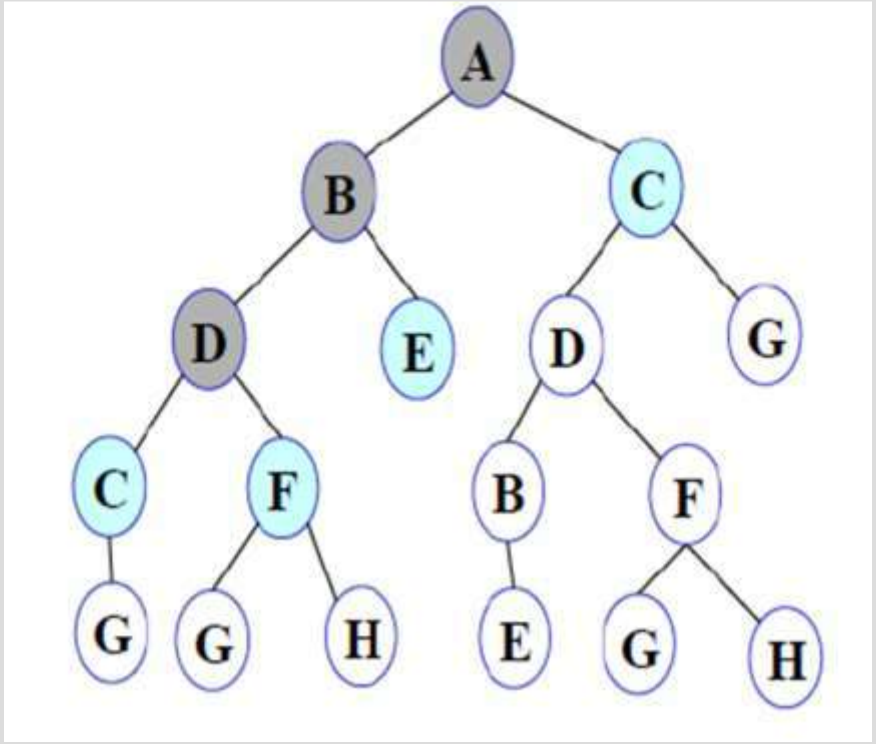
Step 3: B is not the goal node. So it is removed from the stack and expanded.

i.e., Stack : D E C



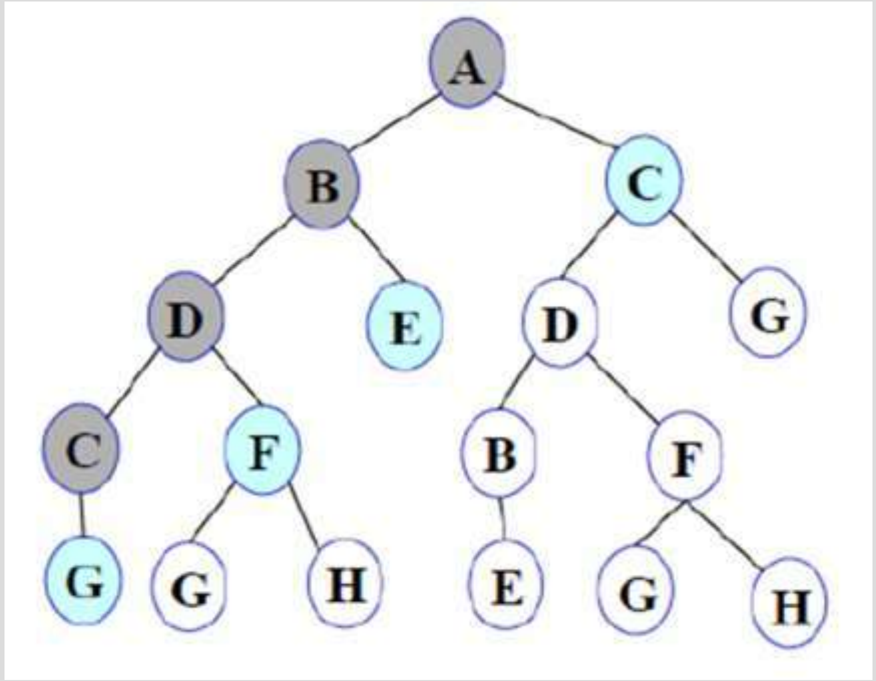
Step 4: D is not the goal node. So it is removed from the stack and expanded.

i.e., Stack : C F E C



Step 5: C is not the goal node. So it is removed from the stack and expanded.

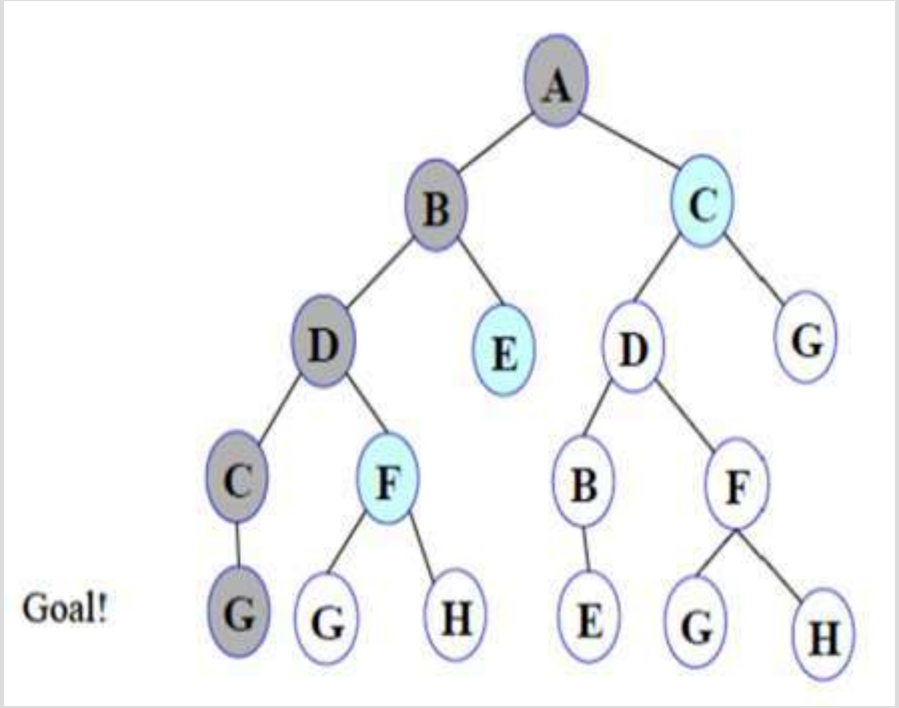
i.e., Stack : G F E C



Step 6: Node G is expanded and found to be a goal node.

i.e., Stack : G F E C

TRAVERSAL ORDER: A-B-D-C-G

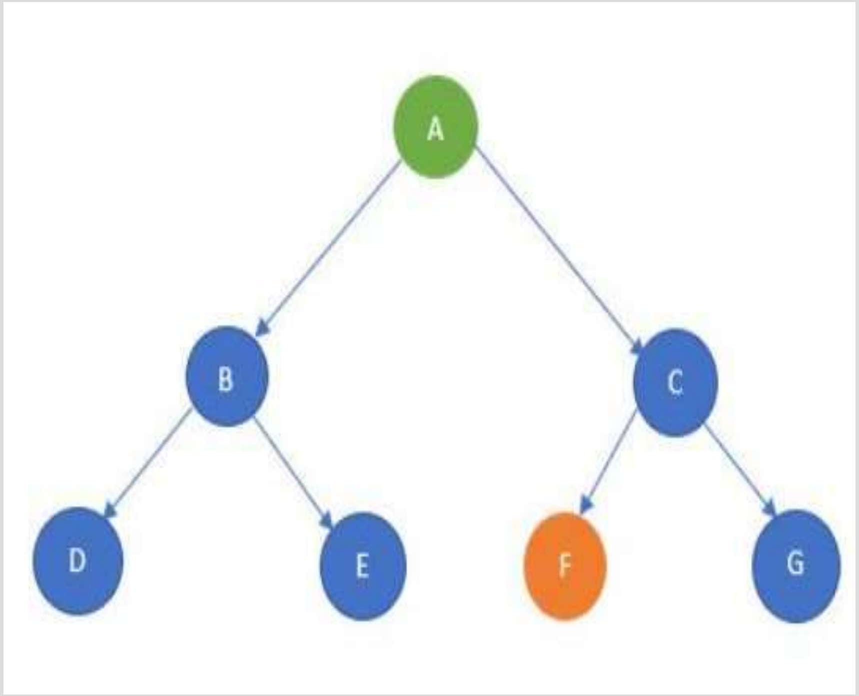


GOAL NODE FOUND!!

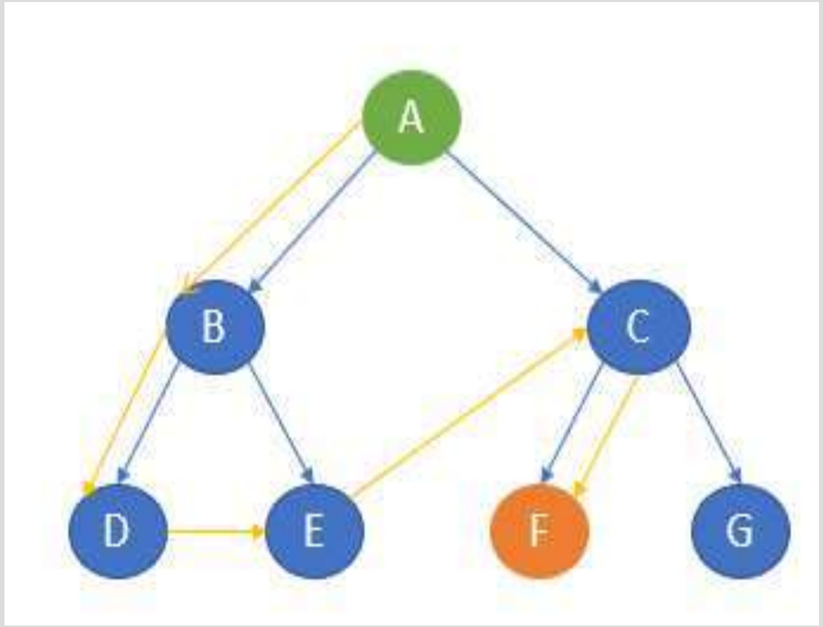
Example-4 (DFS)

Consider the following State Space to be searched:

Let A be the start state and F be the final or goal state to be searched.



Here, it starts from the start state A and then travels to B and then it goes to D. After reaching D, it backtracks to B. B is already visited, hence it goes to the next depth E and then backtracks to B. as it is already visited, it goes back to A. A is already visited. So, it goes to C and then to F. F is our goal state and it stops there.



The path of traversal is:

A —-> B —-> D —-> E —-> C —-> F

Depth First Search (DFS) - Advantages

- DFS ***requires very less memory (as compared to BFS)*** as it only needs to store a stack of the nodes on the path from root node to the current node.
- It ***takes less time to reach to the goal node*** than BFS algorithm (if it traverses in the right path).
- DFS does not require much more search.

Depth First Search (DFS) - Disadvantages

- There is the possibility that many states keep re-occurring, and there is ***no guarantee of finding the solution***.
- DFS algorithm goes for deep down searching and sometime it ***may go to the infinite loop.***

Depth First Search (DFS) - Analysis

- **Time Complexity**: $O(b^d)$
- **Space Complexity**: $O(bd)$

  [where b→ branching factor and d→ depth]
- **Solution** : No particular solution. (not optimal).

Depth First Search (DFS) - Applications

- For finding the path.
- To test if the graph is bipartite.
- For finding the strongly connected components of a graph.
- For detecting cycles in a graph.

Difference Between BFS & DFS

BFS

- BFS stands for ***Breadth First Search.***
- BFS(Breadth First Search) uses ***Queue data structure*** for finding the shortest path.
- BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we ***reach a vertex with minimum number of edges*** from a source vertex.

DFS

- DFS stands for ***Depth First Search.***
- DFS(Depth First Search) uses ***Stack data structure.***
- In DFS, we might ***traverse through more edges*** to reach a destination vertex from a source.

Difference Between BFS & DFS

BFS

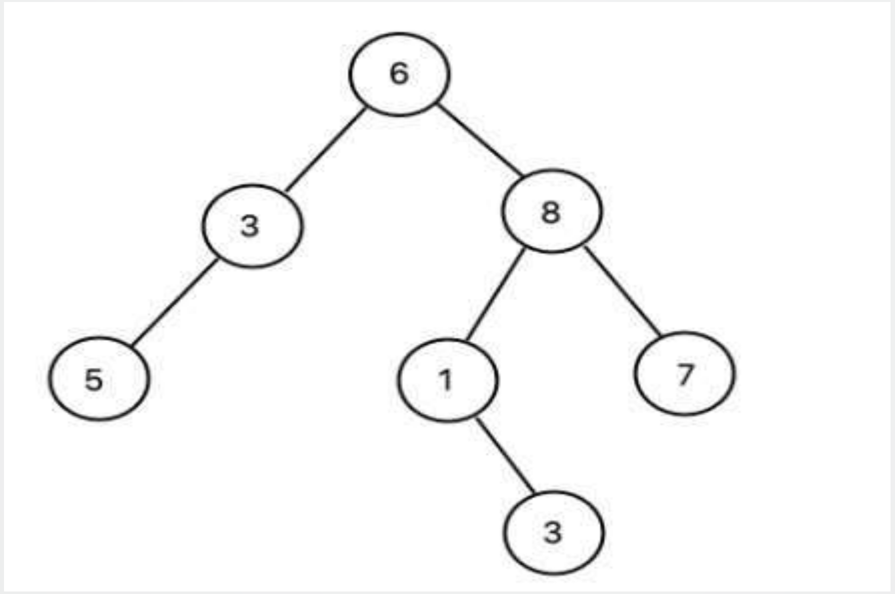
- BFS is more ***suitable for searching vertices which are closer to the given source.***
- BFS considers all neighbours first and therefore ***not suitable for decision making trees used in games or puzzles.***

DFS

- DFS is more ***suitable when there are solutions away from source.***
- DFS is more ***suitable for game or puzzle problems.*** We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.

Try

Perform the DFS traversal to find the shortest path from 6 to 7.



References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Uninformed Search (DLS, IDDFS, UCS, BS)

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 03
Presentation Date – 25/09/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Understand the concept of uninformed search where little or no domain-specific information (heuristics) is available to guide the search process.

Uninformed / Exhaustive / Blind Search (Depth Limited Search(DLS / DLDFS))

Shortcomings of DFS

- There is the possibility that many states keep re-occurring, and there is ***no guarantee of finding the solution***.
- DFS algorithm goes for deep down searching and sometime it ***may go to the infinite loop.***

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

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

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

Depth Limited Search-Example

Here, Start node = S, Goal node = J

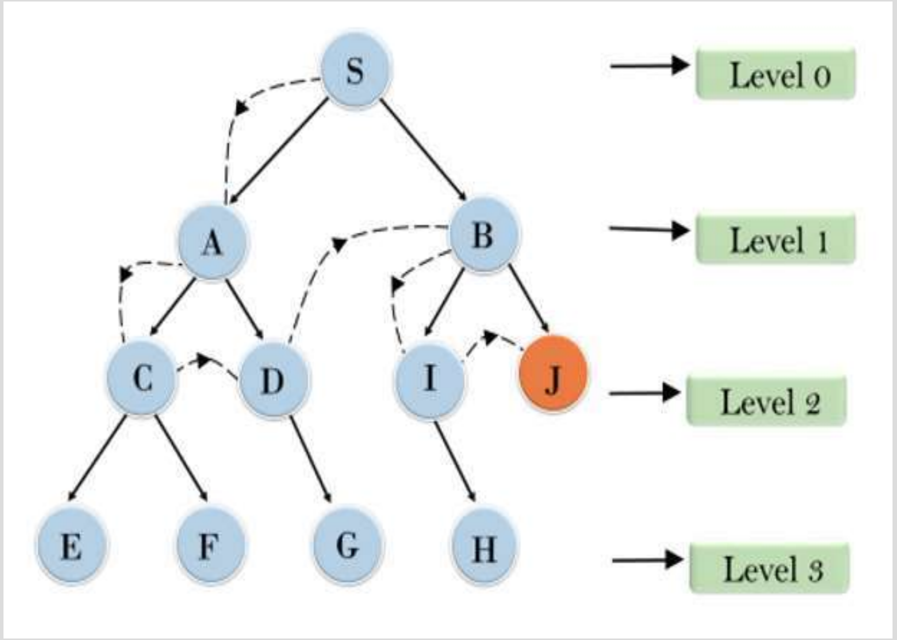
Let Depth (d) = 2

Then path : S→A→C→D→B→I→J

[**NOTE:**

If we define **Depth (d) = 2 and Goal node = H**

In this case the result will have **no solution.(**as per 2nd failure criteria**)]**



Depth Limited Search - Advantages

Depth-limited search is ***Memory efficient.***

Depth Limited Search - Disadvantages

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

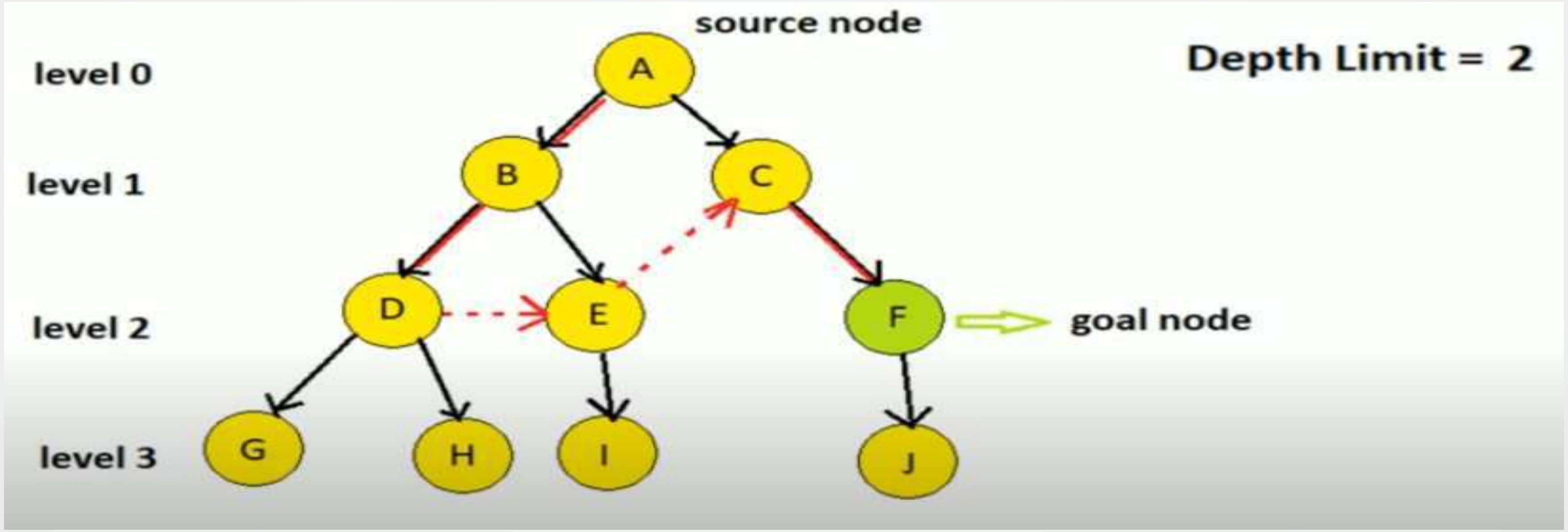
Depth Limited Search - Analysis

- **Completeness**: DLS search algorithm is *complete if the solution is above the depth-limit.*
- **Time Complexity**: Time complexity of DLS algorithm is O(b^d).
- **Space Complexity**: Space complexity of DLS algorithm is O(bd).
- **Optimal**: Depth-limited search can be viewed as a special case of DFS, and it is also *not optimal* even if b>d.

Try

Here, Start node = A, Goal node = F, Depth Limit = 2

Perform the DLS traversal to find the shortest path between **A and F.**



Uninformed / Exhaustive / Blind Search
(Iterative Deepening Depth First Search (IDDFS))

Iterative Deepening DFS (IDDFS)

- This is the ***modified version of Depth Limited Search algorithm.***
- This is type of Uninformed search technique also called as ***blind search.***
- This algorithm ***Works on only present value.***
- This algorithm is a ***combination of DFS & BFS***. (This Search algorithm combines the benefits of Breadth-first search's ***fast search*** and depth-first search's ***memory efficiency***.)
- It ***uses STACK (LIFO)*** to perform search operations.

Iterative Deepening DFS (IDDFS)

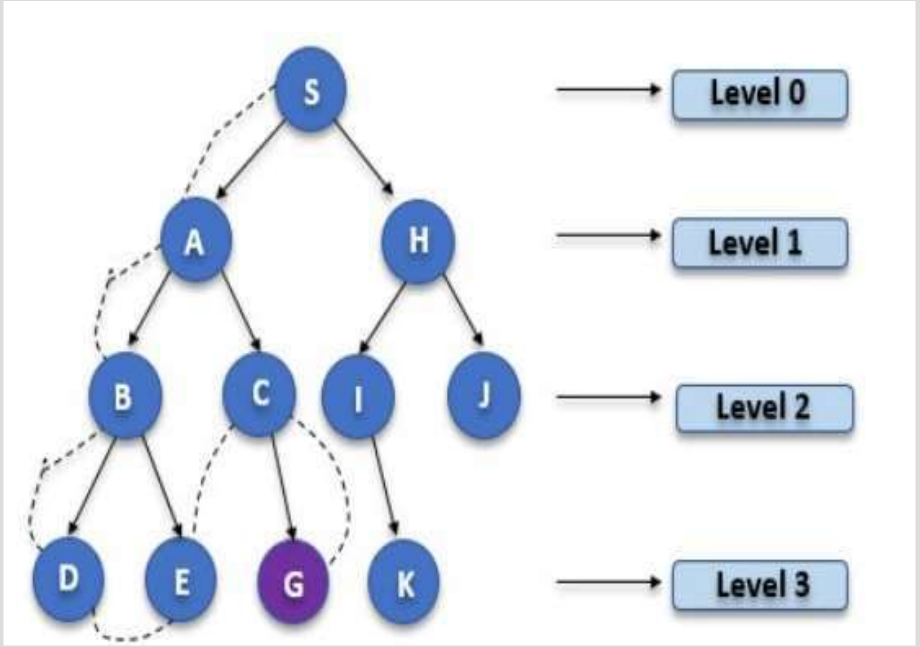
- Here, the best depth limit is found out by gradually increasing depth limit on each iterations.
- ***Initially depth limit is =0***
- ***On each iteration depth increased by exactly one.***
- The iterative search algorithm is ***useful when search space is large, and depth of goal node is unknown.***

Iterative Deepening DFS (IDDFS) – Example - 1

Here, Start node = S, Goal node = G

1'st Iteration → S (d=0)
2'nd Iteration → S, A, H (d=0+1)
3'rd Iteration → S, A, B, C, H, I, J (d=1+1)
4'th Iteration → S, A, B, D, E, C, G (d=2+1)

In the fourth iteration, the algorithm will find the goal node.



Then path : S→A→B→D→E→C→G

Iterative Deepening DFS (IDDFS) - Advantages

- Combine the ***advantage of both DFS and BFS***.
- It is ***Complete & Optimal***. (definitely find optimal solution)
- Will not go in infinite loop.
- ***Fast searching***.
- Less memory requirement. (i.e. ***memory efficient***)

Iterative Deepening DFS (IDDFS) - Disadvantages

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

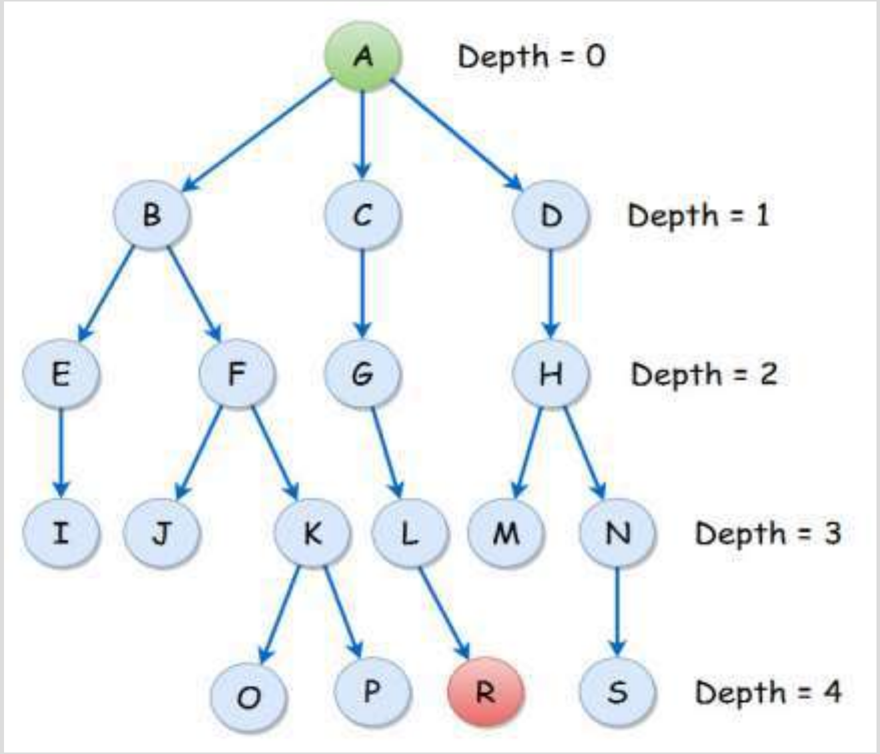
Iterative Deepening DFS (IDDFS) - Analysis

- **Completeness:** This algorithm is complete if the branching factor is finite.
- **Time Complexity:** Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$.**
- **Space Complexity**: The space complexity of IDDFS will be **O(d).**
- **Optimal:** IDDFS algorithm is optimal.

Try

Here, Start node = A, Goal node = R

Perform the IDDFS traversal to find the shortest path between A and R.



Uninformed / Exhaustive / Blind Search (Uniform Cost Search (UCS))

Uniform Cost Search (UCS)

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

Uniform Cost Search (UCS)

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

Uniform Cost Search (UCS) - Algorithm

- **S1**: Insert the root node into the priority queue.
- **S2**: Remove the element with the highest priority.
- **S3**: If the removed node is the goal node,

      Print total cost and stop the algorithm.

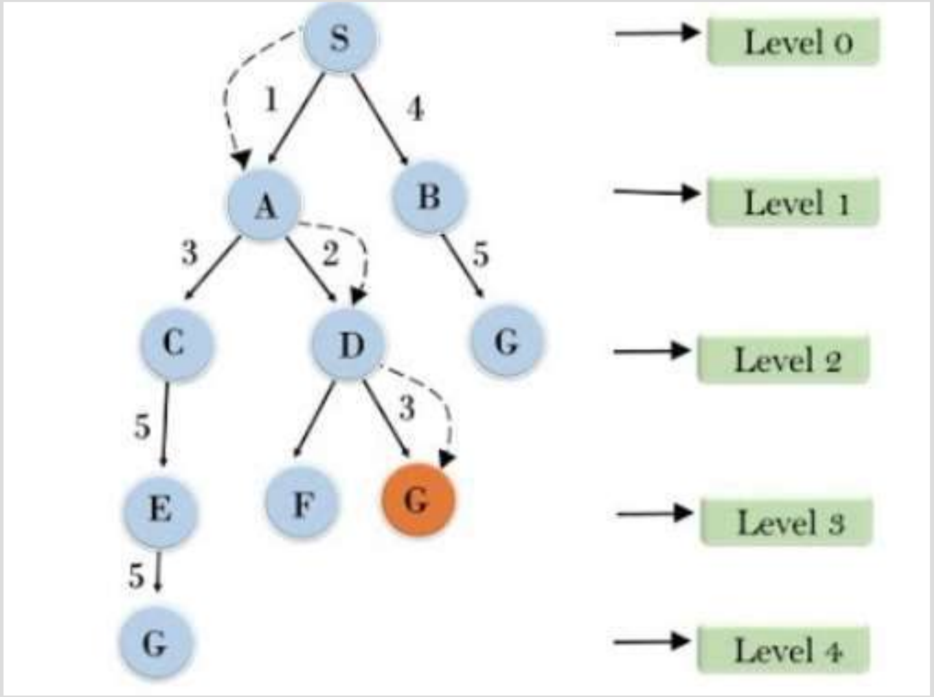
    else

      Enqueue all the children of the current node to the priority queue, with their cumulative cost from the root as priority and the current node to the visited list.

Uniform Cost Search-Example

Here, Start node = S, Goal node = G

From node S we look for a node to expand, and we have nodes A and B, but since it's a uniform cost search, it's expanding the node with the lowest step cost, so node A becomes the successor rather than B. From A we look at its children nodes C and D. Since D has the lowest step cost, it traverses through node D. Then we look at the successors of D, i.e. G. Since D has only one child G (with cost) which is our required goal state we finally reach the goal state G by implementing UCS Algorithm.



<u>Then path </u>: S→A→D→G
(Path cost=6)

Uniform Cost Search (UCS) - Advantages

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

Uniform Cost Search (UCS) - Disadvantages

- It ***does not care about the number of steps*** or finding the shortest path involved in the search problem, and it is only ***concerned about path cost***.
- This algorithm ***may be stuck in an infinite loop***.

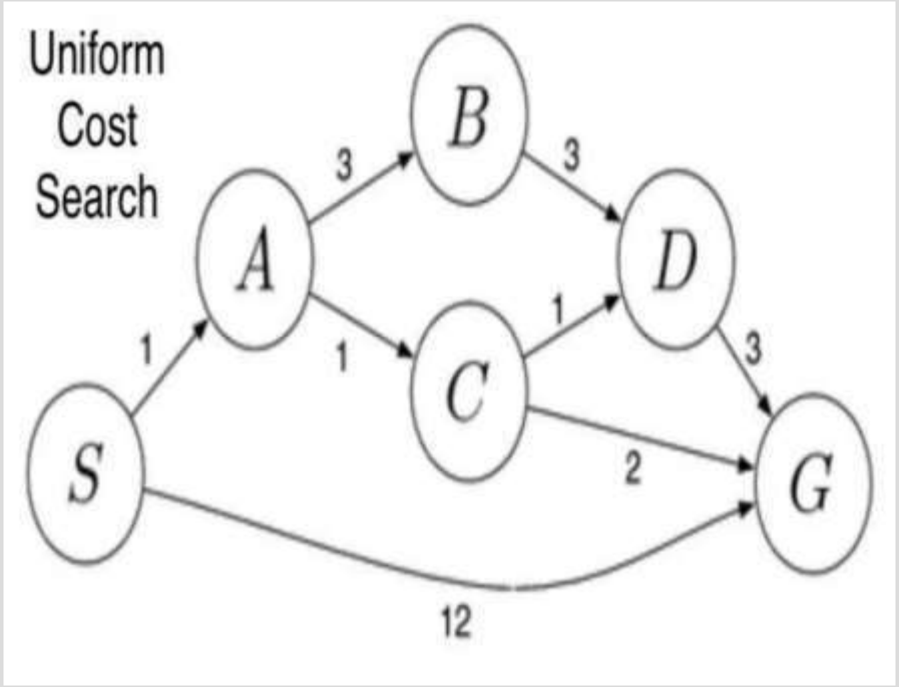
Uniform Cost Search (UCS) - Analysis

- **Complete:** Yes (if b is finite)
- **Time Complexity:** O(b(c/ϵ)) [*where, ϵ -> is the lowest cost, c -> optimal cost]*
- **Space complexity**: O(b(c/ϵ))
- **Optimal:** Yes (even for non-even cost)

Try

Here, Start node = S, Goal node = G

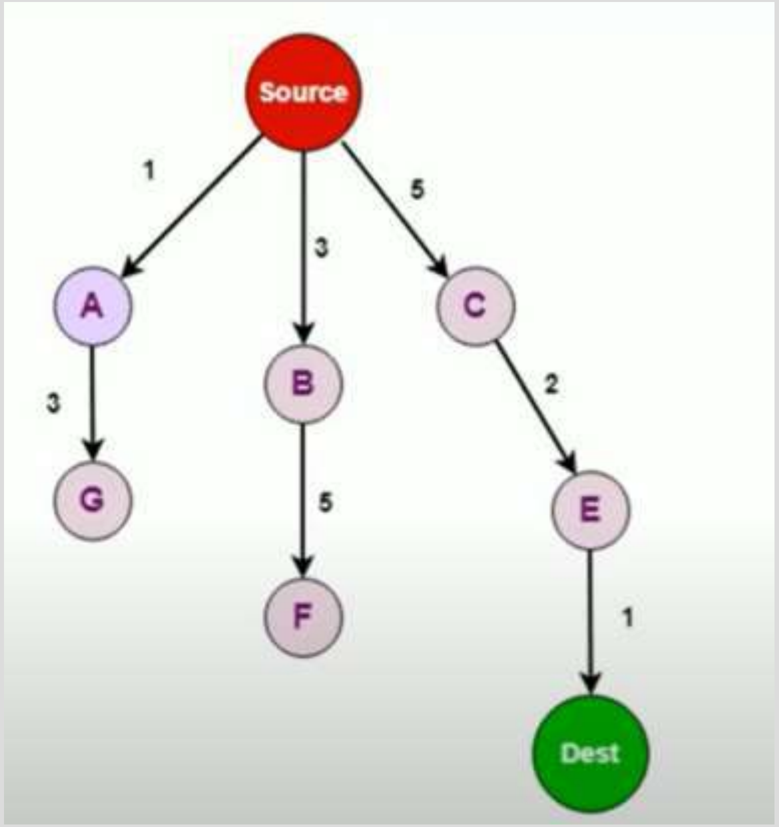
Traverse the tree from S to G using Uniform Cost Search algorithm and find the path cost for the same.



Try

Here, Start node = Source, Goal node = Dest

Traverse the tree from **Source** to **Dest** using Uniform Cost Search algorithm and find the path cost for the same.

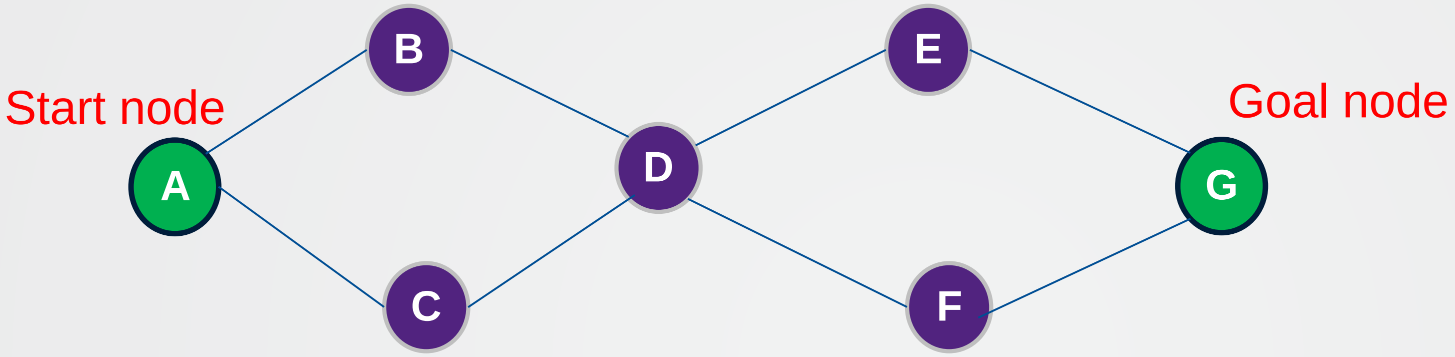


Uninformed / Exhaustive / Blind Search (Bidirectional Search(BS))

Bidirectional Search (BS)

- Bidirectional search algorithm *runs two simultaneous searches*.
    - ❖ **forward-search** : Looking from start node → end node
    - ❖ **backward-search**: Looking from end node → start 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.
- **Stopping Criteria**: ***The search stops when these two graphs intersect each other.***
- Bidirectional search ***can use search techniques such as BFS, DFS, DLS, etc.***

Bidirectional Search (BS) – Example - 1



<u>Forward BFS</u>

A

A→B→C

A→B→C→D

<u>Backward BFS</u>

G

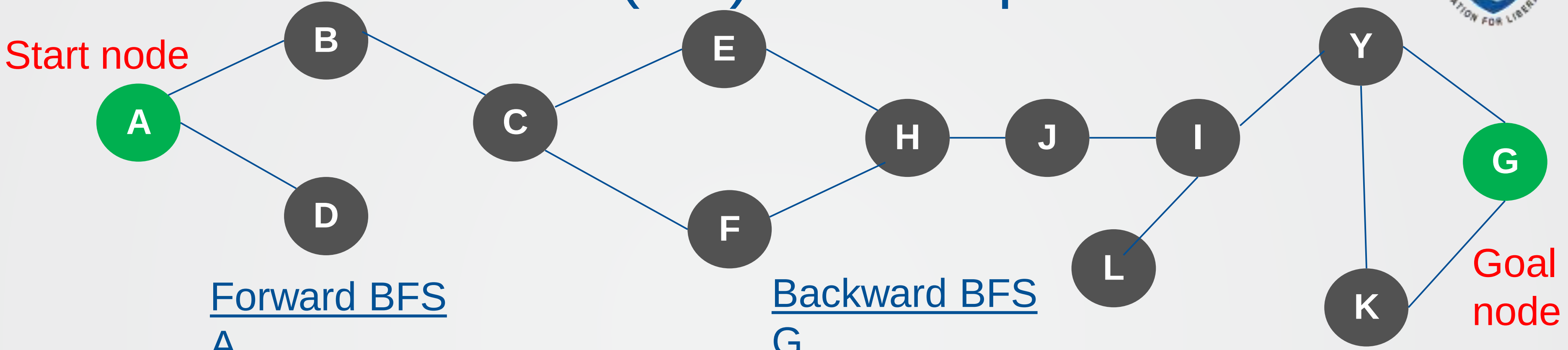
G→E→F

G→E→F→D

⬇

Hence after combining both the searches path will be: A→B→C→D→F→E→G

Bidirectional Search (BS) – Example - 2



Forward BFS

A

A→B→D

A→B→D→C

A→B→D→C→E→F

A→B→D→C→E→F→H

Backward BFS

G

G→K→Y

G→K→Y→I

G→K→Y→I→L→J

G→K→Y→I→L→J→H

After combining both searches

Then path : A→B→D→C→E→F→H→J→L→I→Y→K→G

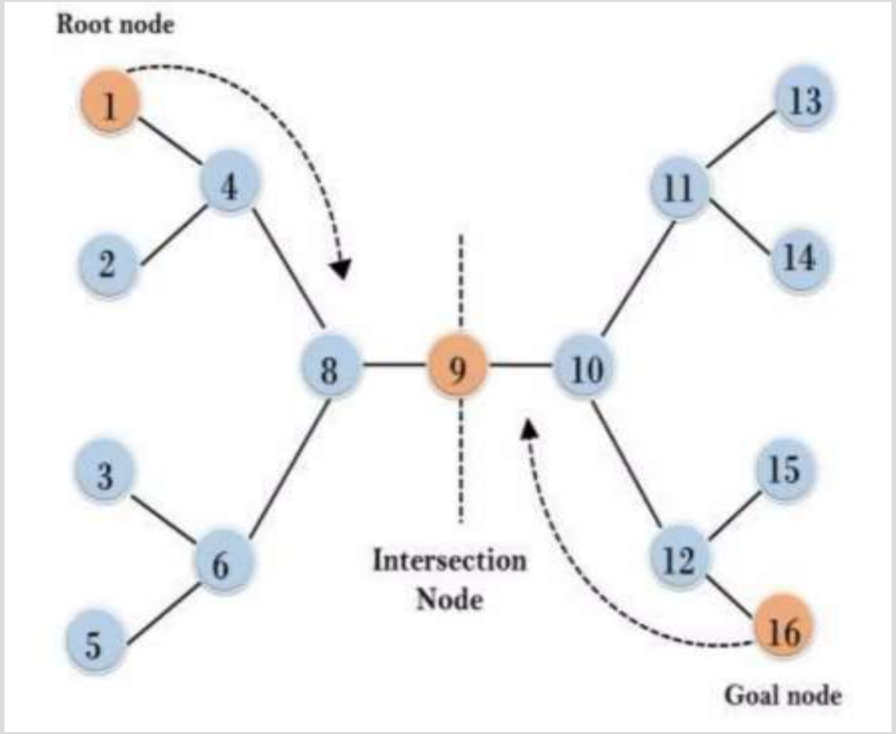
Bidirectional Search (BS) – Example - 3

Here, Start node = 1, Goal node = 16

This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.

Then path : 1→4→8→9→10→12→16



Bidirectional Search (BS) - Advantages

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

Bidirectional Search (BS) - Disadvantages

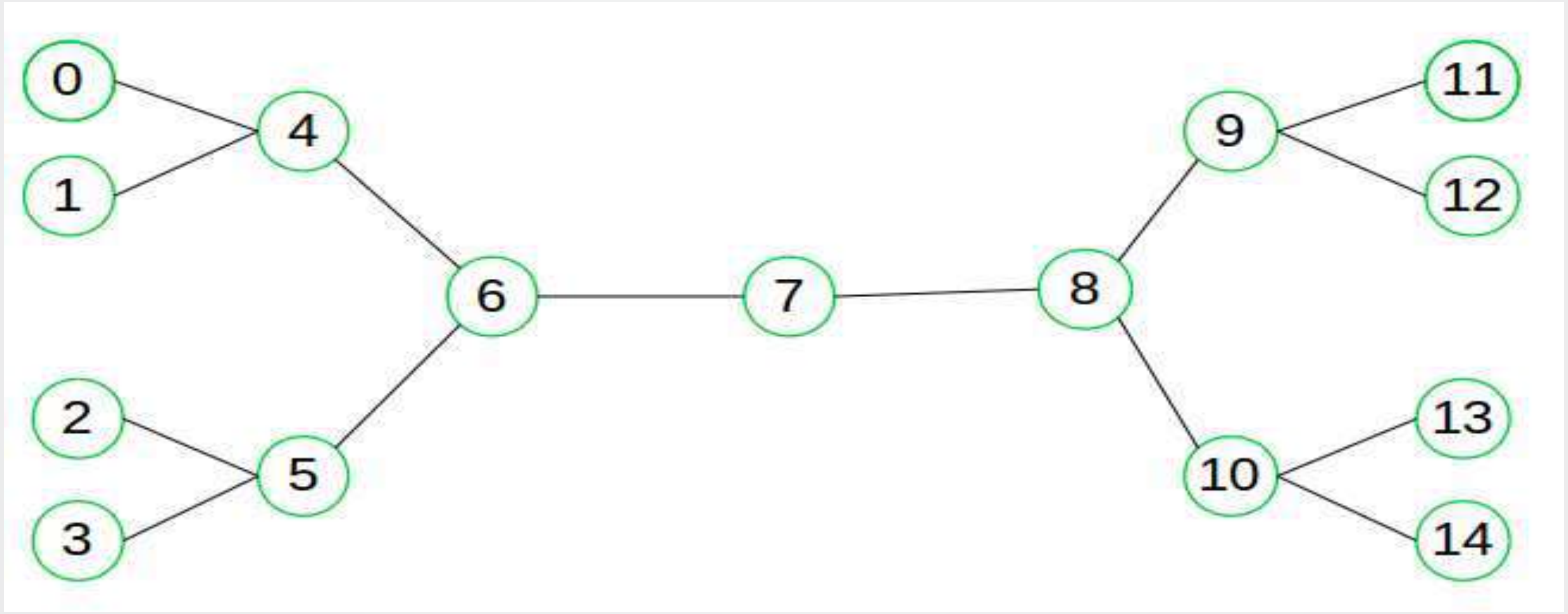
- ***Implementation*** of the bidirectional search tree ***is difficult.***
- In bidirectional search, ***one should know the goal state in advance***.

Bidirectional Search (BS) - Analysis

- **Completeness**: Bidirectional Search *is complete* if we use BFS in both searches. (No particular solution.)
- **Time Complexity**: Time complexity of bidirectional search using BFS is $O(b^{d/2})$.
- **Space Complexity**: Space complexity of bidirectional search is $O(b^{d/2})$.
- **Optimal**: Bidirectional search is Optimal.

Try

Perform Bidirectional search from 0 to 14.



References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Informed Search / Heuristic / Intelligent Search

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 04
Presentation Date – 28/09/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Understand the concept of informed or heuristic search where additional information’s are used to find the best possible solution without exhaustively searching every possible state.

Informed / Heuristic / Intelligent Search

Informed / Heuristic / Intelligent Search

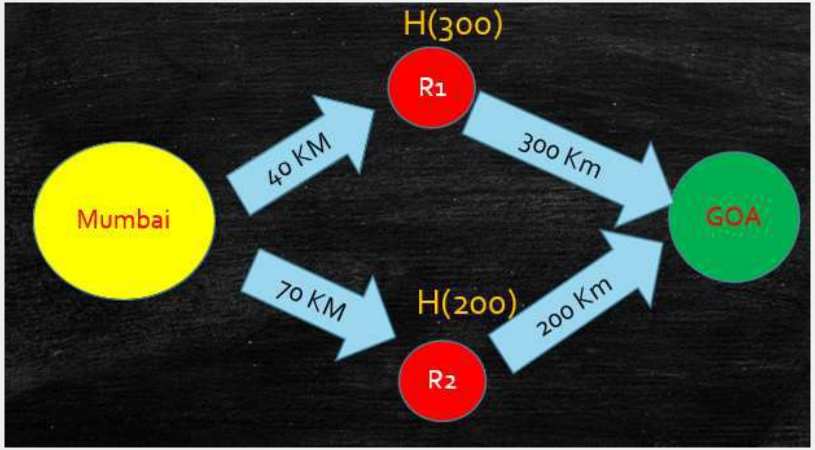
- A heuristic is a technique that is ***used to solve a problem faster than the classic methods***. These techniques are ***used to find the approximate solution of a problem*** when classical methods do not.
- Heuristics are said to be the ***problem-solving techniques that result in practical and quick solutions***.
- Heuristics are used in situations ***where there is the requirement of a short-term solution.*** On facing complex situations with limited resources and time, Heuristics can ***help the companies to make quick decisions by shortcuts and approximated calculations.***

Informed / Heuristic / Intelligent Search (Contd..)

- Heuristic search is a simple searching technique that tries to ***optimize*** a problem using ***Heuristic function.***
- ***Optimization*** means that we will ***try to solve a problem in minimum number of steps or cost.***
- This searching technique ***comes under Informed search.***

Heuristic Function : h(n)

- It is a function H(n) that gives an estimation on the cost of getting from node 'n' to the goal state.
- ***It helps in selecting optimal node for expansion.***
- ***Ex:***



R1= 340 Km

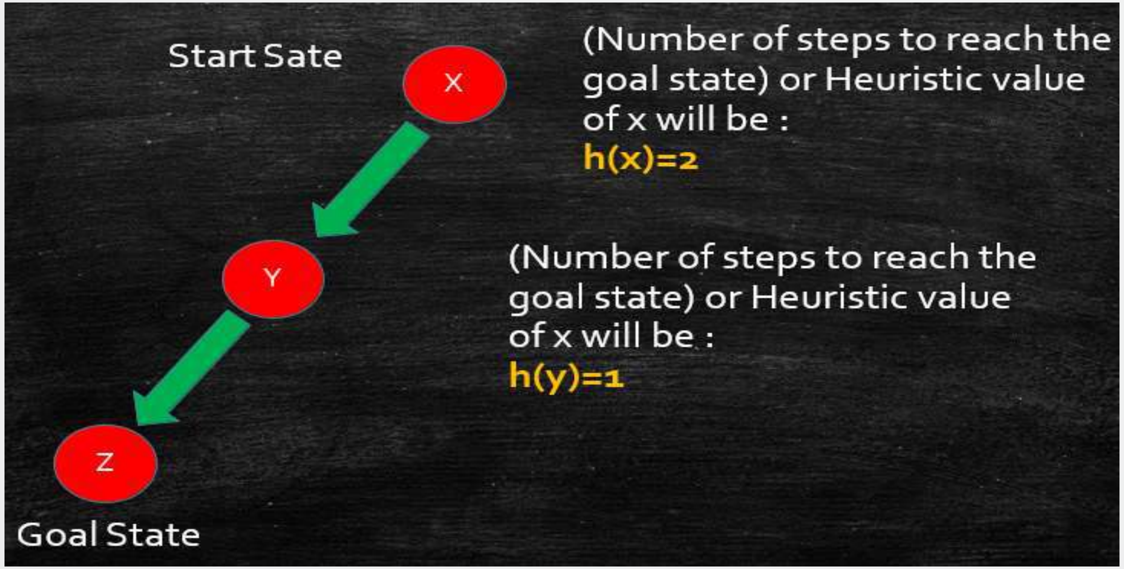
R2= 270Km

Normally, we select least route i.e., 40km from start node. But heuristic search finds the cost from each node till destination.

Here, **Start node**= Mumbai, **Goal node**= GOA

Heuristic Function : h(n)

- Heuristic search is an informed search ***as it is informing us that how far is our goal state.***
- So heuristic function can be denoted as:



(Number of steps to reach the goal state) or Heuristic value of x will be :
h(x)=2

(Number of steps to reach the goal state) or Heuristic value of x will be :
h(y)=1

Types of Heuristic Function : h(n)

- There are ***two types of Heuristic Functions:***

1. **Admissible**
2. **Non-Admissible**

Admissible Heuristic Function : h(n)

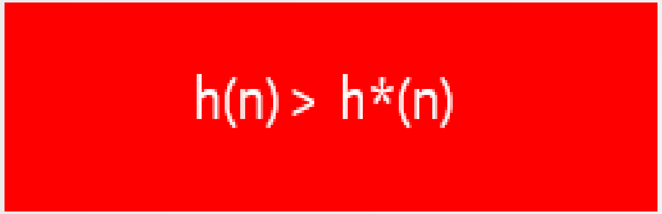
- A heuristic function is admissible if it ***never overestimates the cost of reaching the goal .***
- **i.e.,** it ***underestimates the path cost.***

$$h(n) <= h^*(n)$$

Here h(n) is heuristic cost, and h*(n) is the estimated cost / actual cost.
So ***heuristic cost should be less than or equal to the estimated cost.***

Non-Admissible Heuristic Function : h(n)

- A non-admissible heuristic ***may overestimate the cost of reaching the goal.***

- 

  $h(n) > h^*(n)$

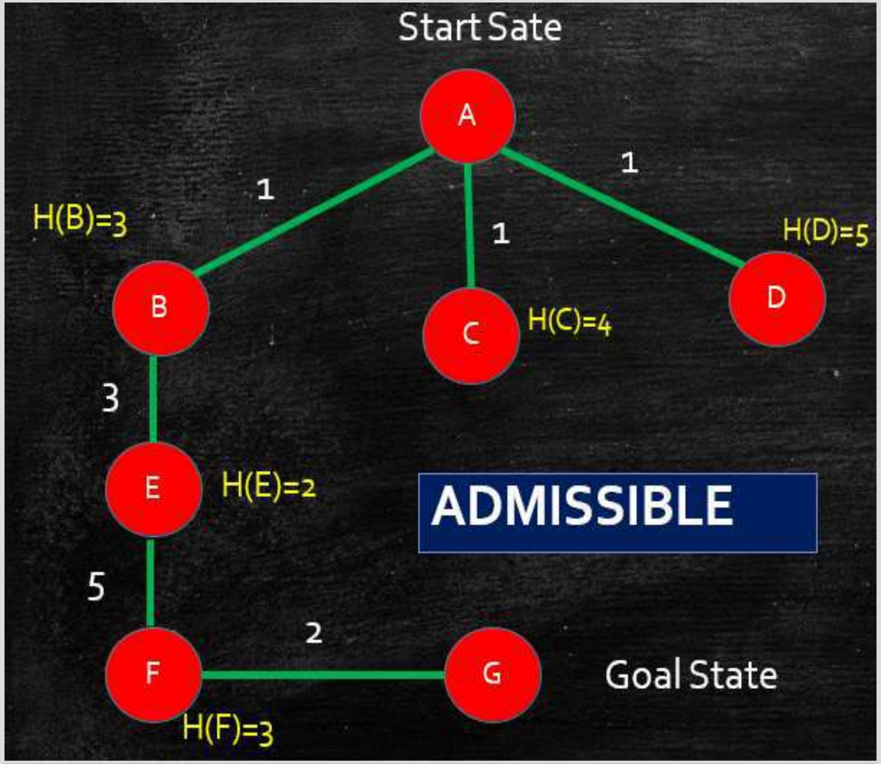
  Here h(n) is heuristic cost, and h*(n) is the estimated cost / actual cost.
  So ***heuristic cost may be greater than to the estimated cost.***

  Total Cost = Heuristic cost + Actual cost
  F(n) = G(n) + H(n)

Admissible Heuristic Function : h(n) : Example

Heuristic Cost given:
H(B)=3, H(C)=4, H(D)=5

F(n) = G(n) + H(n)
Cost = Actual cost + Heuristic cost
B= 1+3=4 (least cost)
C= 1+4=5
D= 1+5=6
Actual cost from A to G = 1+3+5+2= 11
H(B)=3 i.e., **h(n) =3 and h*(n)=11**
i.e., h(n)<=h*(n) so 3<=11
So **search is admissible.**
Same for node E, 2<=11 and node F, 3<=11



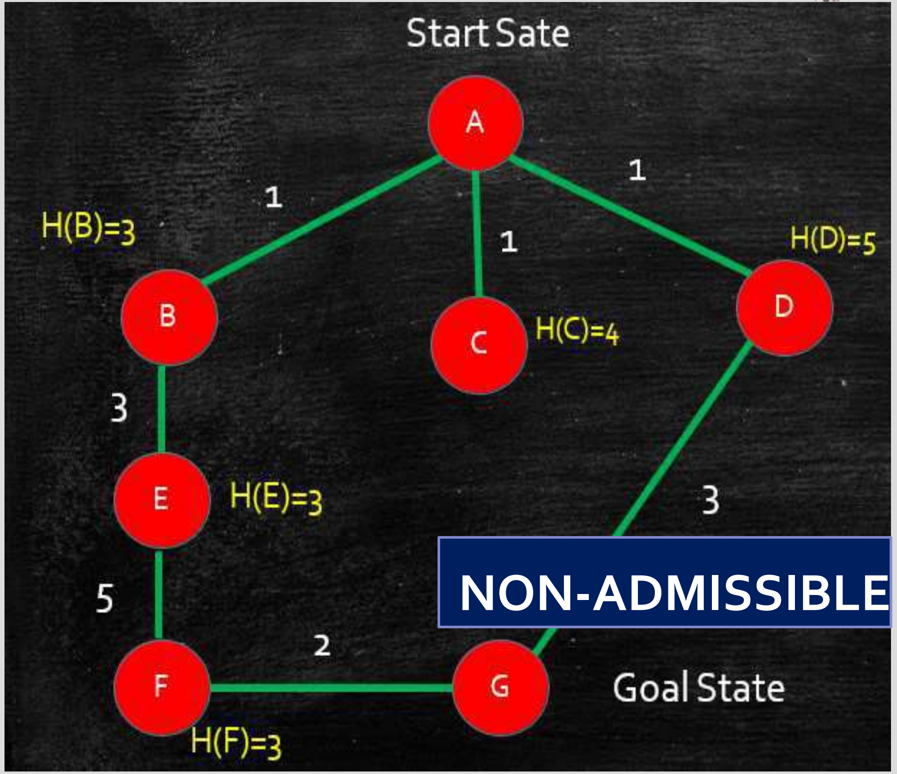
Non-admissible Heuristic Function : h(n) : Example

If we choose the node D,
Heuristic Cost : H(D)=5

F(n) = G(n) + H(n)
Cost = Actual cost + Heuristic cost

Actual cost from A to G(via D) = 1+3= 4
H(D)=5 i.e., **h(n) =5 and   h*(n)=4**

i.e., h(n)>h*(n)
5<=4
So **search is non-admissible.**



Heuristic Search Techniques - Categories

1. Direct Heuristic Search techniques
2. Weak Heuristic Search techniques

1. Direct Heuristic Search techniques

- It includes ***Blind Search, Uninformed Search, and Blind control strategy.***
- These search techniques ***are not always possible as they require much memory and time.***
- These techniques *search* ***the complete space*** for a solution and ***use the arbitrary ordering of operations.***
- The <u>**examples**</u> of Direct Heuristic search techniques include Breadth-First Search ***(BFS)*** and Depth First Search ***(DFS).***

2. Weak Heuristic Search techniques

- It includes ***Informed Search, Heuristic Search, and Heuristic control strategy.***
- These techniques are helpful when they are applied properly to the right types of tasks. They usually ***require domain-specific information.***
- The **examples** of Weak Heuristic search techniques include Best First Search ***(BFS) and A*.***

Use of Heuristic Evaluation Functions

- The following algorithms make use of heuristic evaluation functions:
  - ***Hill Climbing***
  - ***Generate-and-Test***
  - ***Best-First Search***
  - ***A* Algorithm***
  - ***AO* Algorithm***
  - ***Beam Search***
  - ***Constraint Satisfaction***

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Informed Search (Generate-and-Test, Best First Search, A* Algorithm)

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 05
Presentation Date – 28/09/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Understand the concept of informed or heuristic search where additional information’s are used to find the best possible solution without exhaustively searching every possible state.

What is Global Search?

Global search techniques ***explore the entire solution space*** to find the optimal / satisfactory solution.

Global Search Techniques:

1. **Generate and Test**

2. **Best First Search(OR graph)**

Where not only the current branch of the search space but all the so far explored nodes/states in the search space are considered in determining the next best state/node.

3. **A* Algorithm**

Which is improvised version of Best first Search.

4. **Problem Reduction and And-Or Graphs.**

AO* Algorithm.

5. **Constraint Satisfaction Problem (CSP)**

Graph Colouring Problem and Crypt Arithmetic Problems.

6. **Mean End Analysis (MEA)**

Informed / Heuristic / Intelligent Search
(1. Generate-and-Test)

Generate-and-Test

- It is a ***heuristic / informed search***.
- In this technique ***all the solutions are generated and tested for best solution.***
- It ensures that the ***best solution is checked from all possible generated solutions.***
- This approach is what is ***known as the British Museum algorithm***: finding an object in the British Museum by wandering randomly.
- It is ***like depth-first search with backtracking***, requires that complete solutions be generated for testing.
- If the solution is found, then quit.

Generate-and-Test

- The generate-and-test s***trategy is the simplest of all the approaches***. It consists of the following steps:

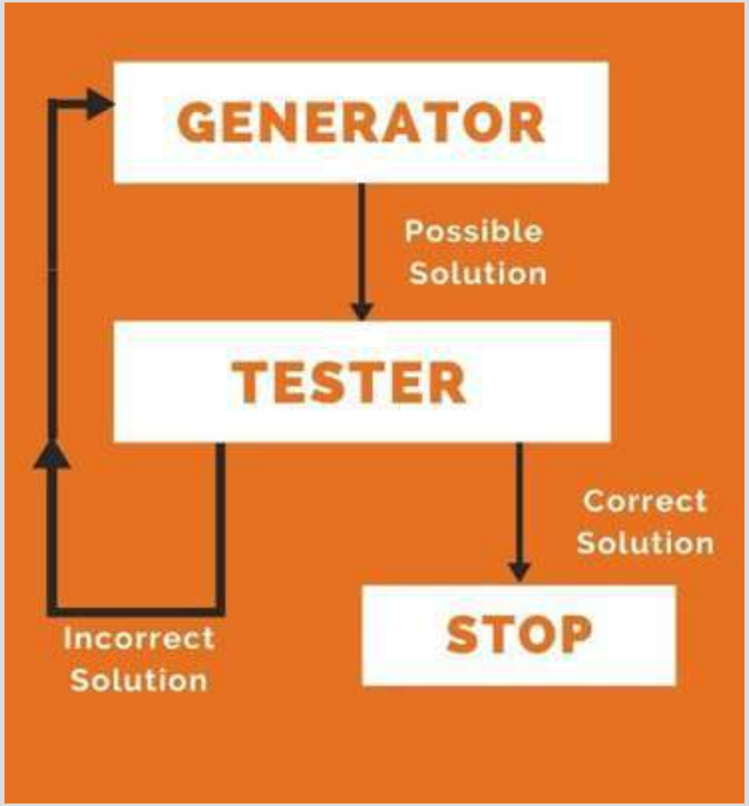
- **Algorithm**

1. Generate all possible solutions.
2. Select one solution among the possible solutions.
3. If a solution has been found and acceptable, quit. Otherwise return to step 1.

Generate-and-Test-Diagram

Two approaches are followed while generating solutions:

- Generating complete solutions &
- Generating random solutions



[Generate and Test Heuristic Search Algorithm]

Generate-and-Test – Example:

Travelling Salesman Problem (TSP)

A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow for the shortest possible round trip that both starts and finishes at any one of the cities.

Generate-and-Test – Example:

Travelling Salesman Problem (TSP)

You are given-

- A set of some cities.
- Distance between every pair of cities.

Travelling Salesman Problem states-

- A salesman has to visit every city exactly once.
- He has to come back to the city from where he starts his journey.
- What is the shortest possible route that the salesman must follow to complete his tour?

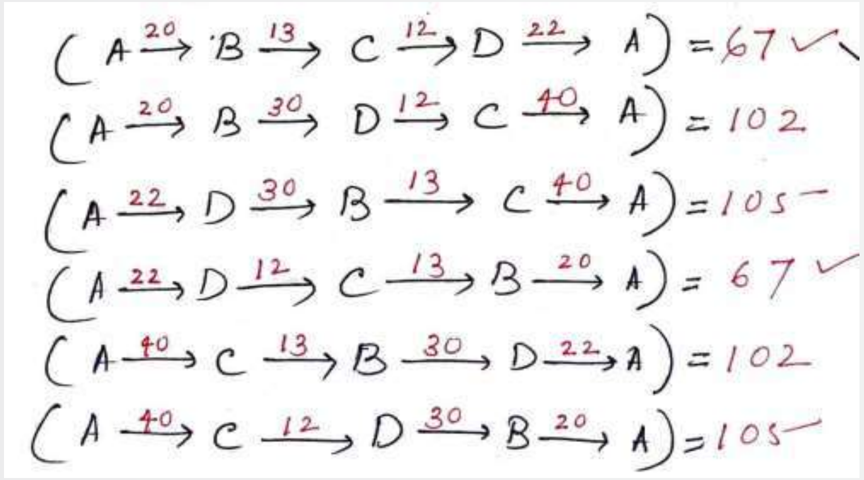
Travelling Salesman Problem (TSP)

Let the **initial state be A.**

So, all possible states from A will be:

{(A→B),(A→D),(A→C)}

i.e.,



$(A \xrightarrow{20} B \xrightarrow{13} C \xrightarrow{12} D \xrightarrow{22} A) = 67$ ✓

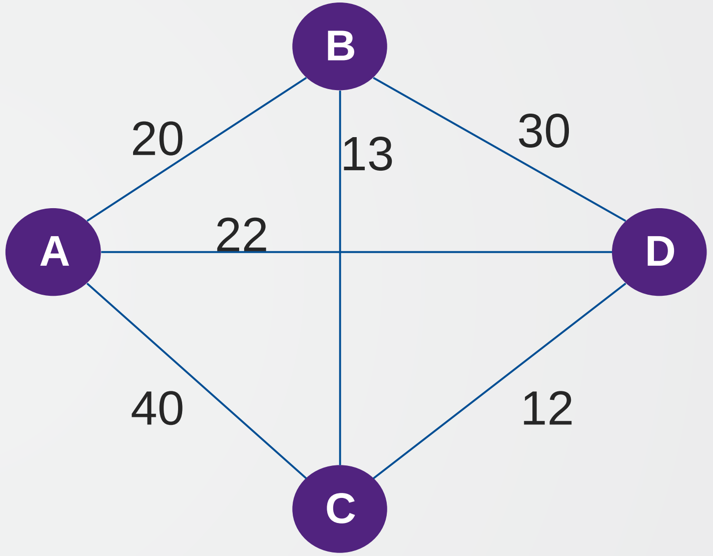
$(A \xrightarrow{20} B \xrightarrow{30} D \xrightarrow{12} C \xrightarrow{40} A) = 102$

$(A \xrightarrow{22} D \xrightarrow{30} B \xrightarrow{13} C \xrightarrow{40} A) = 105$ —

$(A \xrightarrow{22} D \xrightarrow{12} C \xrightarrow{13} B \xrightarrow{20} A) = 67$ ✓

$(A \xrightarrow{40} C \xrightarrow{13} B \xrightarrow{30} D \xrightarrow{22} A) = 102$

$(A \xrightarrow{40} C \xrightarrow{12} D \xrightarrow{30} B \xrightarrow{20} A) = 105$ —



67 is the shortest path cost if A is considered as the initial point.

Travelling Salesman Problem (TSP)

Similarly, let the **initial state be B.**

So, all possible states from B will be:

{(B→A),(B→C),(B→D)}

Similarly, let the **initial state be C.**

So, all possible states from C will be:

{(C→A),(C→B),(C→D)}

Similarly, let the **initial state be D.**

So, all possible states from D will be:

{(D→B),(D→C),(D→A)}

Find all possible routes taking into consideration as initial point as B, C, D and observe the shortest path among all. Finally, select the path whose length is less..

Informed / Heuristic / Intelligent Search
(2. Best First Search(BFS/Greedy Search/OR graph))

Best First Search (BFS)

- This is an ***informed search*** technique(Greedy Search) also called as ***HEURISTIC search.***
- This algorithm works using heuristic value.
- ***Every node*** in the search space has an ***Evaluation function*** (***heuristic function***)associated with it.
- Evaluation value= cost/distance of current node from goal node.
- For goal node evaluation function value=0
- ***Evaluation function==heuristic cost function*** (in case of minimization problem) OR ***objective function***(in case of maximization).

Best First Search (BFS)

- This algorithm uses ***evaluation function*** to decide ***which adjacent node is most promising and then explore.***
- ***i.e., the node which is having lowest evaluation is selected for the expansion because the evaluation measures distance to the goal.***
- ***Priority queue*** is used to store cost of function.
- It serves as a ***combination of both BFS and DFS.***
- This is also called as ***Greedy Search*** as it quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

Best First Search (BFS) – maintains two lists

- The implementation of Best First Search Algorithm involves *maintaining two lists*- **OPEN and CLOSED**.
- **OPEN** list contains those nodes *that have been evaluated by the heuristic function but have not been expanded* into successors yet.
- **CLOSED** list contains those nodes *that have already been visited.*

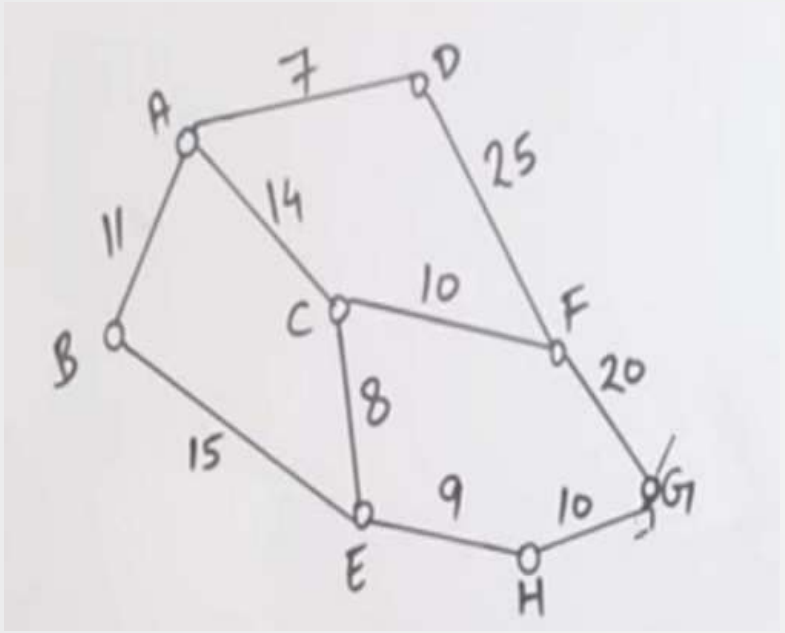
Best First Search (BFS) - Concept

- **S1**: Traverse the root node.
- **S2**: Traverse any neighbour of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue.
- **S3**: Traverse any neighbour of neighbour of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue.
- **S4:** The process will continue until we are getting the goal node.

Best First Search (BFS) - Algorithm

```
Priority queue 'PQ' containing initial states.
Loop
    If PQ=Empty Return Fail
Else
    Insert Node into PQ(Open-List)
    Remove First(PQ) ->NODE(Close-List)
    If NODE=GOAL
    Return path from initial state to NODE
Else
    Generate all successor of NODE and insert newly generated NODE into 'PQ'
    according to cost value.
End Loop
```

Best First Search (BFS) – Example



Straight-line Distance / f(n)

A→G	=40
B→G	=32
C→G	=25
D→G	=35
E→G	=19
F→G	=17
G→G	=0
H→G	=10

OPEN	CLOSED
[A]	[ ]
[C,B,D]	[A]
[F,E,B,D]	[A,C]
[G,E,B,D]	[A,C,F]
[E,B,D]	[A,C,F,G]

Here, A= Root node, G=Goal node.

Path = A→C→F→G, Cost = 44

Best First Search (BFS) - Advantages

- ***Memory efficient as compared to DFS & BFS***.
- It is ***Complete.***
- ***Time Complexity is much lesser than BFS.***

Best First Search (BFS) - Disadvantages

- It gives good solution but ***not optimal solution.***
- In worst case it may behave like unguided DFS i.e., ***sometimes it covers more distance than our consideration.***

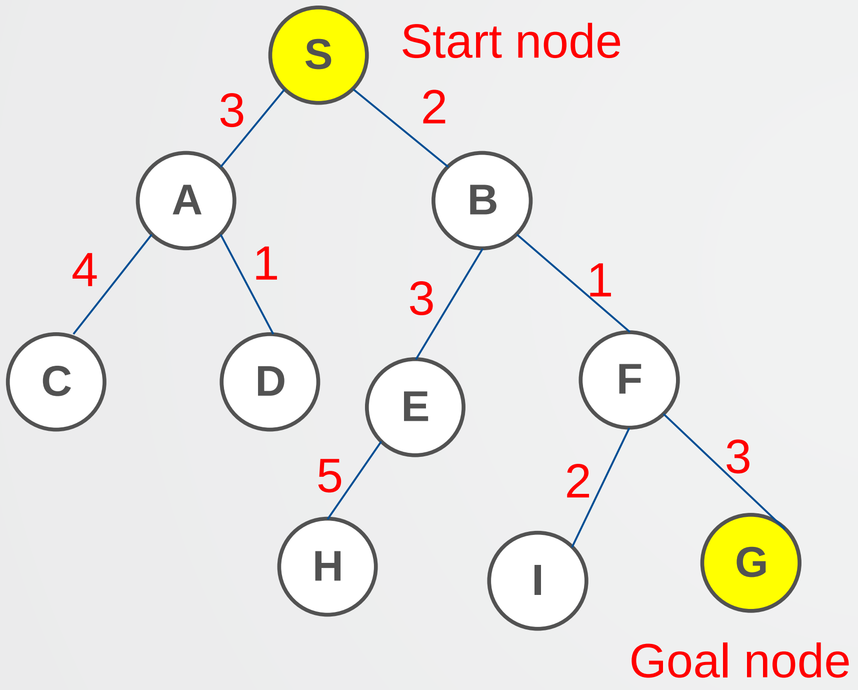
Best First Search (BFS) - Analysis

- ***Space Complexity***: O(b^d)
- ***Time Complexity***: O(b^d)

  [where, b→ branching factor, d→ depth]
- ***Solution:*** not optimal.

Try

Find the shortest path from S to G using Best First Search.



Straight-line Distance / f(n)

Node(n)	f(n)
A→G	=12
B→G	=4
C→G	=7
D→G	=3
E→G	=8
F→G	=2
G→G	=0
H→G	=4
I→G	=9
S→G	=13

Informed / Heuristic / Intelligent Search
(3. A* Algorithm)

A* Algorithm

- This is an informed search technique also called as ***HEURISTIC search.***
- It is essentially the ***extension of best first search algorithm.***
- This algorithm works using heuristic value.
- **A* uses**
    - **h(n)->**Heuristic function (cost from current node to goal node)
    - **g(n)**->Actual cost (cost from start node to current node)

A* Algorithm (Contd..)

- It ***finds shortest path*** though search space.
- It ***provides fast and optimal result.***
- A* requires the heuristic function to evaluate the cost of the path that passes through the particular state. It can be defined by the following formula.

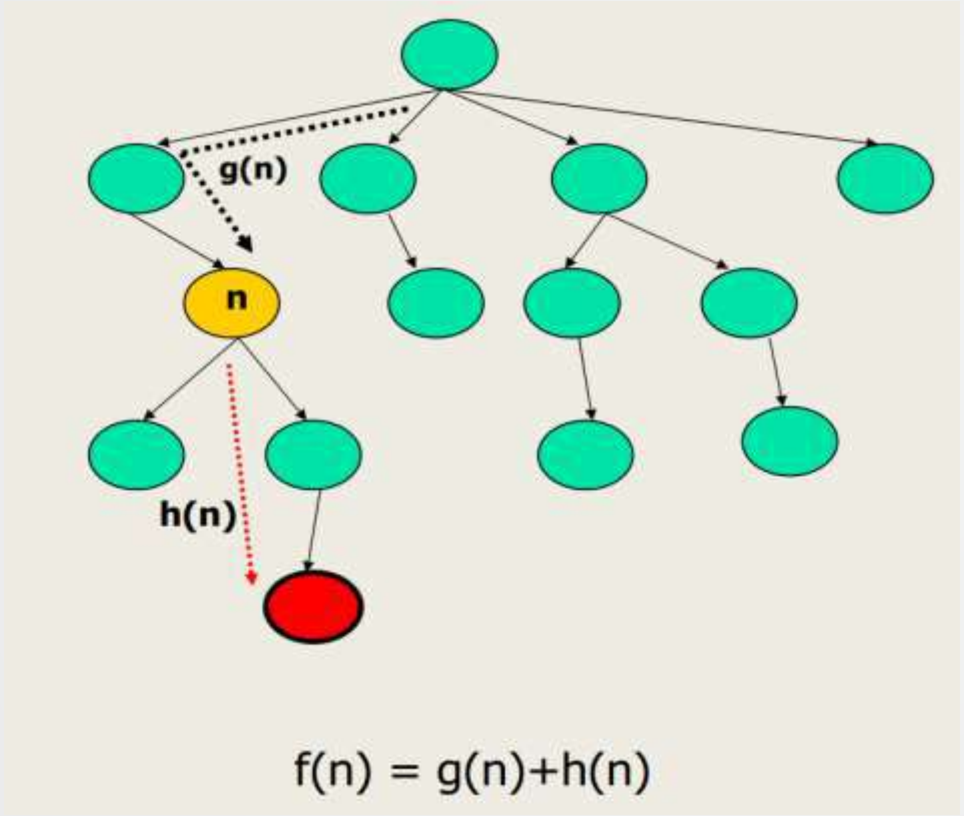
Estimated Cost : **$f(n)=g(n)+h(n)$**

[Where,f(n) = estimated cost of the node

g(n) = is the cost of the path from start node to node 'n'

h(n) = heuristic value(*of child node always considered)*]

A* Algorithm (Contd..)



A* Algorithm (Contd..) – maintains two lists

- The implementation of A* Algorithm involves maintaining two lists- ***OPEN and CLOSED.***

1. **OPEN** list contains those nodes that *have been evaluated by the heuristic function but have not been expanded* into successors yet.
2. **CLOSED** list contains those nodes *that have already been visited.*

A* Algorithm (Contd..) – Algorithm

- **Step-01:**
  - Enter starting node in OPEN list.
- **Step-02:**
  - If OPEN list is empty, return fail and exit.
- **Step-03:**

Select the node from OPEN list which has smallest value(g+h).

```
        If node=Goal,
                return SUCCESS.
```

A* Algorithm (Contd..) – Algorithm

- **<u>Step-04:</u>**
  - Expand node ‘n’ and generate all successors. Compute (g+h) for each successor node.
- **<u>Step-05:</u>**
  - If node ‘n’ is already in OPEN / CLOSED, attach to backpointer.
- **<u>Step-06:</u>**

  Go back to Step-02.

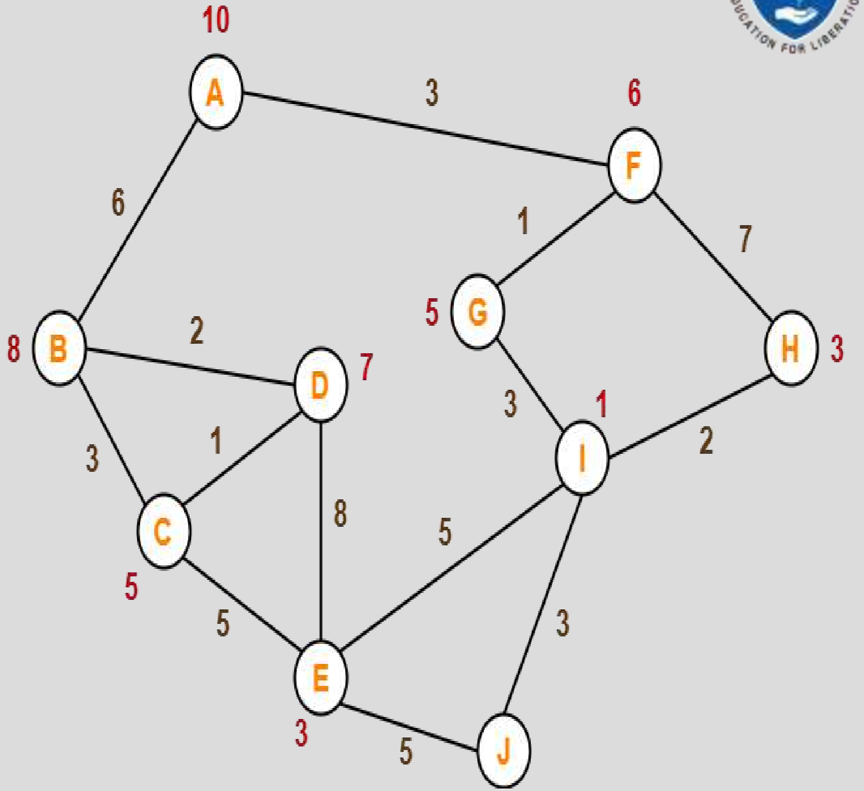
A* Algorithm (Contd..) – Example-1

Consider the following graph-

❖The numbers ***written on edges*** represent the distance between the nodes.

❖The numbers ***written on nodes*** represent the heuristic value.

❖Find the most cost-effective path to reach from ***start state A*** to ***final state J*** using A* Algorithm.

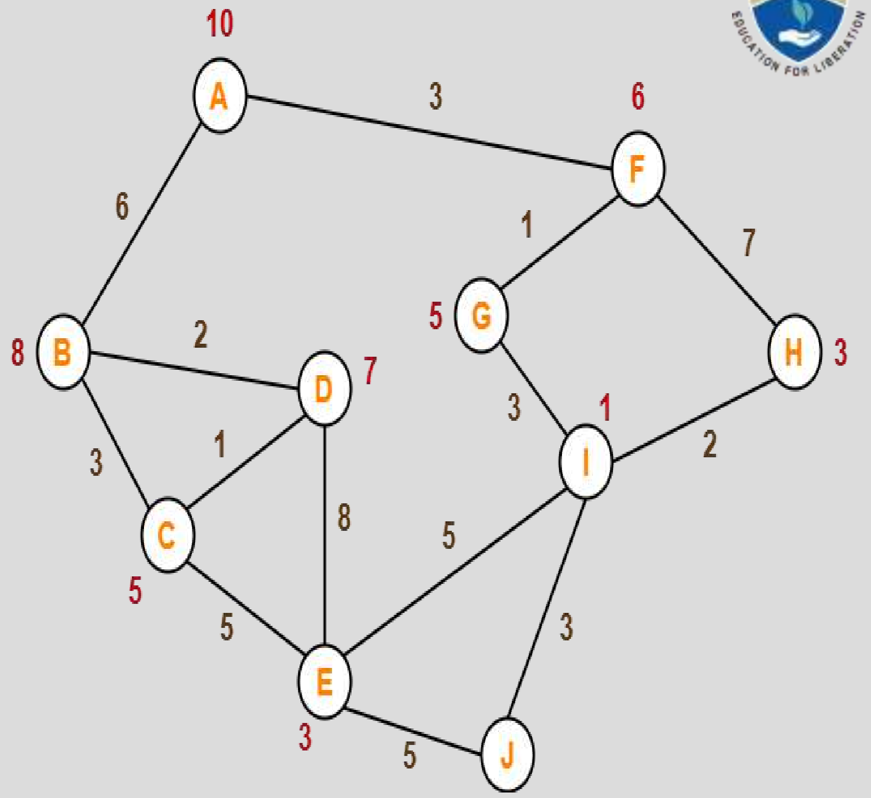


A* Algorithm (Contd..) – Example-1

Sloution:

❖**Step-01:**

- We start with node A. Node B and Node F can be reached from node A.
- A* Algorithm calculates f(B) and f(F).
- Estimated Cost **f(n)=g(n)+h(n)**
  - f(B) = g(B) + h(B) = 6 + 8 = 14
  - f(F) = g(F) + h(F) = 3 + 6 = 9
- Since f(F) < f(B), so it decides to go to node F.



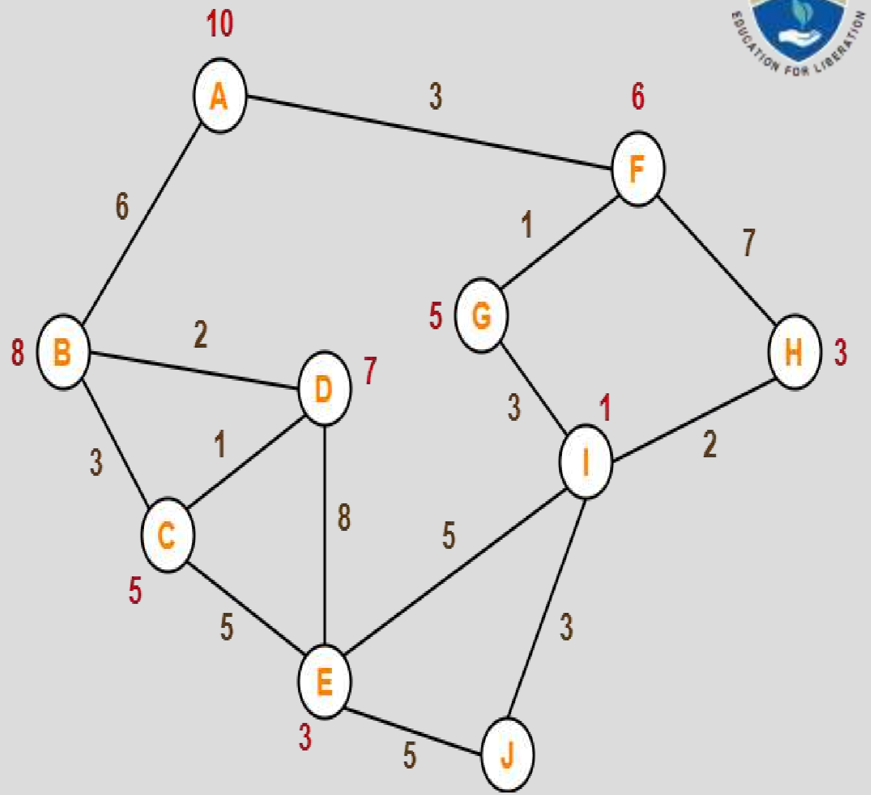
Closed list(F)
Path: A → F

A* Algorithm (Contd..) – Example-1

Sloution:

❖**Step-02:**

- Node G and Node H can be reached from node F.
- A* Algorithm calculates f(G) and f(H).
  - f(G) = g(G) + h(G) = (3+1) + 5 = 9
  - f(H) = g(H) + h(H) = (3+7) + 3 = 13
- Since f(G) < f(H), so it decides to go to node G.



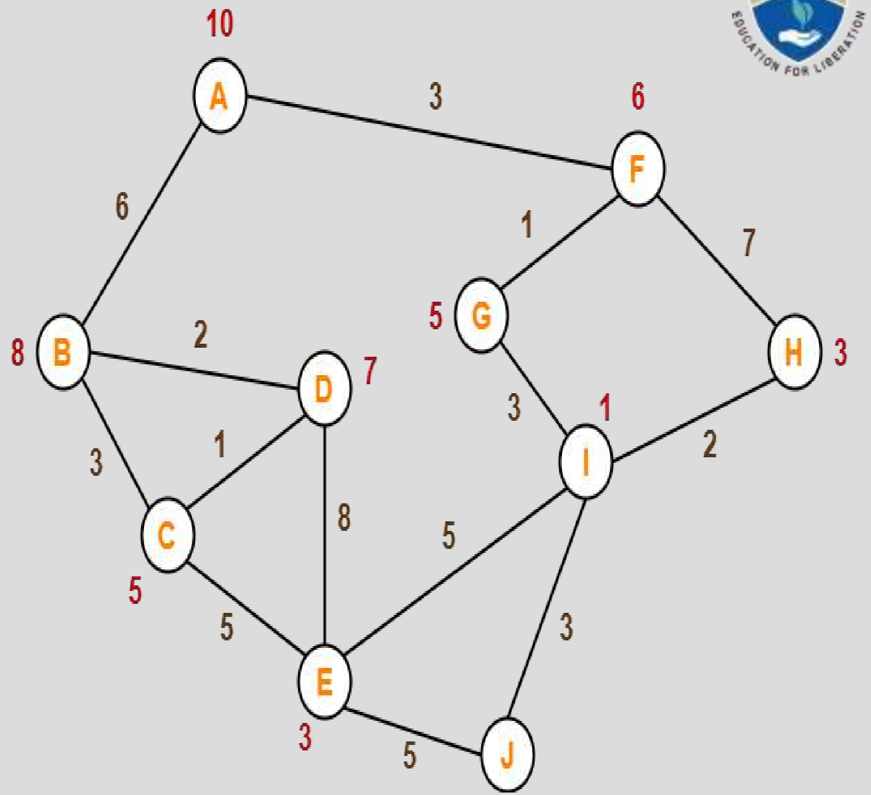
Closed list(G)
Path: A → F → G

A* Algorithm (Contd..) – Example-1

Sloution:

❖**Step-03:**

- Node I can be reached from node G.
- A* Algorithm calculates f(I).
  - f(I) = g(I) + h(I) = (3+1+3) + 1 = 8
- It decides to go to node I.



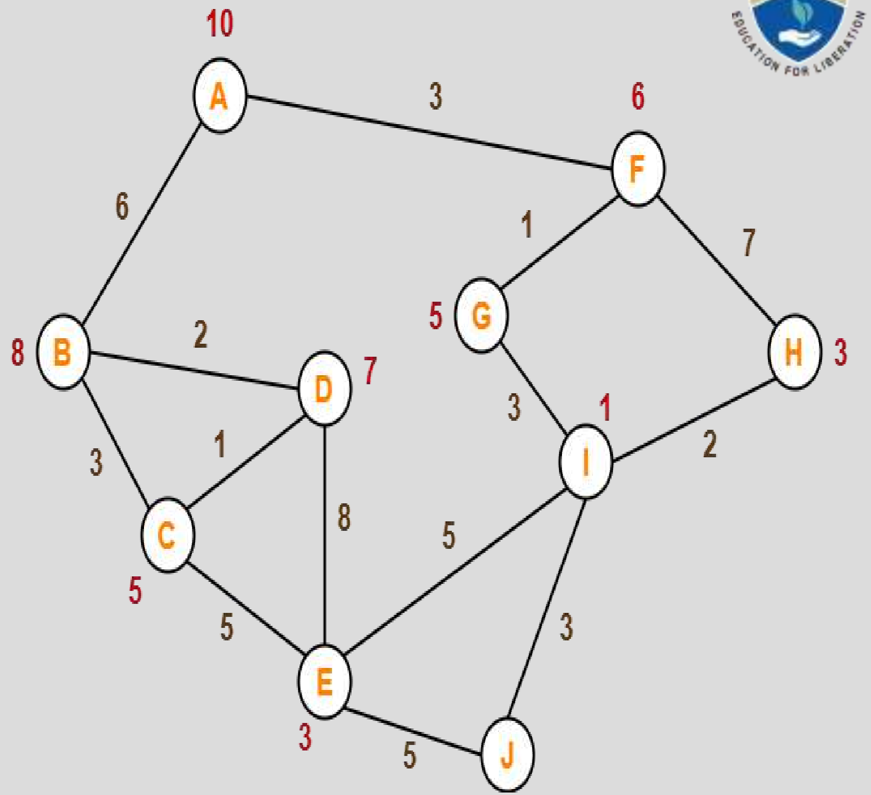
Closed list(I)
Path: A → F → G → I

A* Algorithm (Contd..) – Example-1

Sloution:

❖ **Step-04:**

- Node E, Node H and Node J can be reached from node I.
- A* Algorithm calculates f(E), f(H) and f(J).
  - f(E) = g(E) + h(E) = (3+1+3+5) + 3 = 15
  - f(H) = g(H) + h(H) = (3+1+3+2) + 3 = 12
  - f(J) = g(J) + h(J) = (3+1+3+3) + 0 = 10
- Since f(J) is least, so it decides to go to node J.



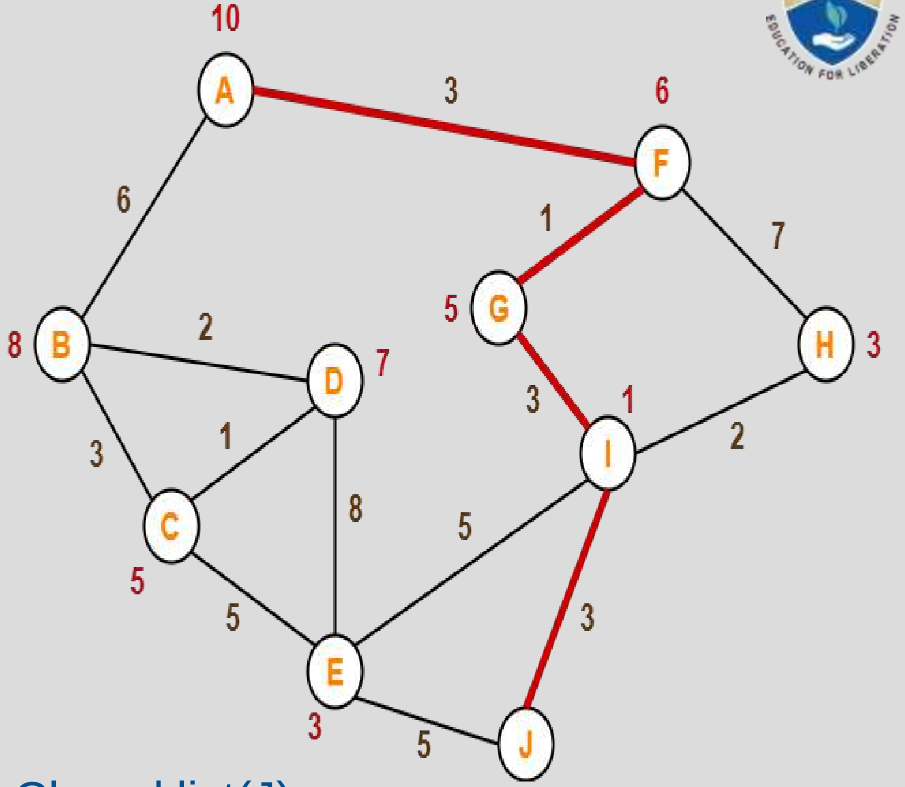
Closed list(J)
Path: A → F → G → I → J

A* Algorithm (Contd..) – Example-1

This is the required shortest path from node A to node J.

OPEN	CLOSED
[A]	[ ]
[F,B]	[A]
[G,H,B]	[A,F]
[I,H,B]	[A,F,G]
[J,H,B,E]	[A,F,G,I]
[H,B,E]	[A,F,G,I,J]

Goal node is found.

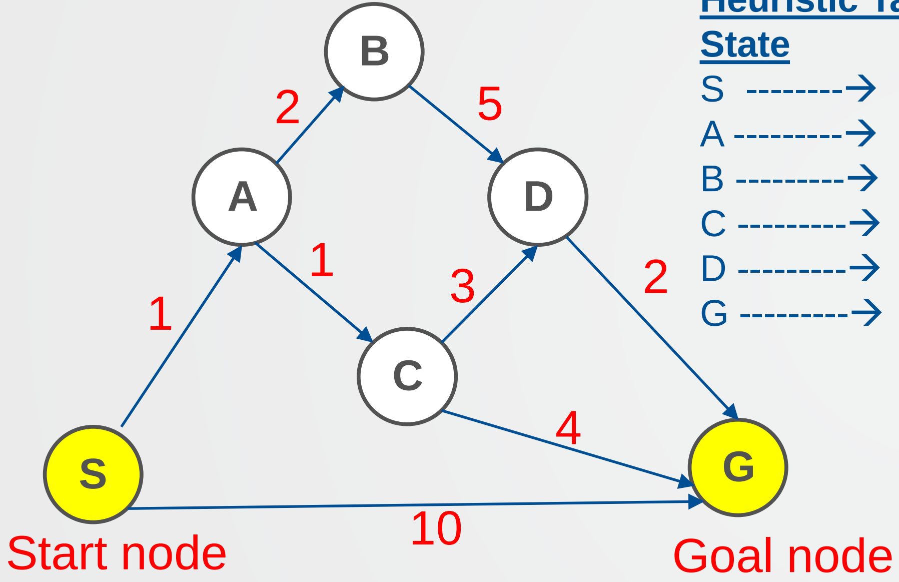


Closed list(J)

Path: A → F → G → I → J

A* Algorithm (Contd..) – Example-2

Find the shortest path from S to G using A* Algorithm by referring the given heuristic table.



Heuristic Table

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

SOLUTION:

S→A=1+3=4
S→G=10+0=10------(route1)

S→A→B=(1+2)+4=7
S→A→C=(1+1)+2=4

S→A→C→D=(1+1+3)+6=11
S→A→C→G=(1+1+4)+0=6------(route2)

S→A→B→D=(1+2+5)+6=14
S→A→C→D→G=(1+1+3+2)+0=7-----(route3)

S→A→B→D→G=(1+2+5+2)+0=10--(route4)

Path: S→A→C→G, cost=6(least cost)

A* Algorithm (Contd..) – Advantages

- A* Algorithm is ***one of the best path finding algorithms.***
- It is ***Complete & Optimal.***
- Used to ***solve complex problems***.

A* Algorithm (Contd..) – Disadvantages

- Doesn’t always produce shortest path.
- It has complexity issues.
- Requires more memory.

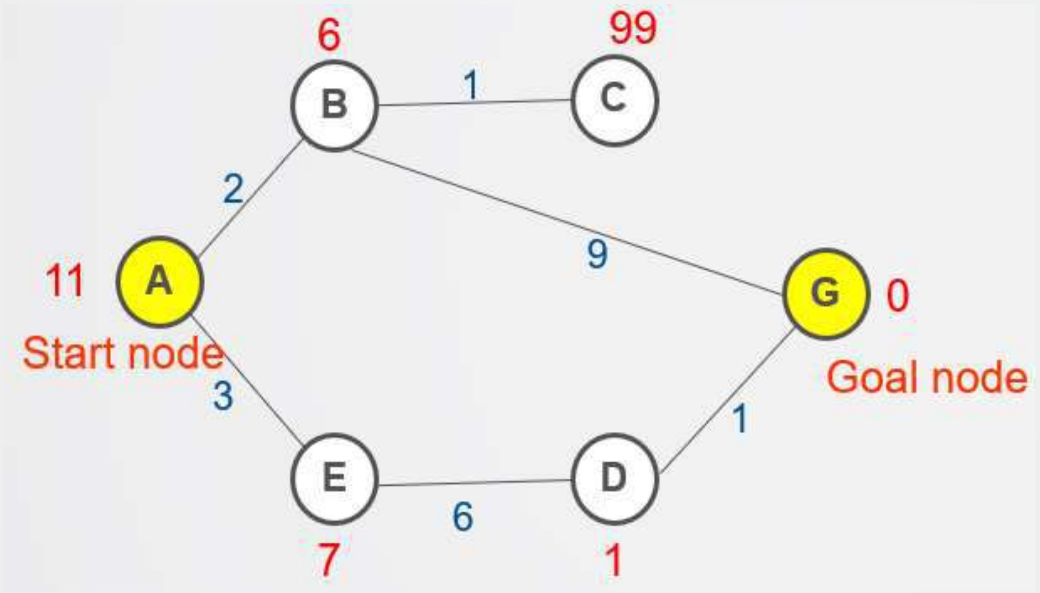
A* Algorithm (Contd..) – Analysis

- ***Space Complexity***: O(b^d)
- ***Time Complexity***: O(b^d) [depends on the heuristic.]

  [where, b→ branching factor, d→ depth]
- ***Solution:*** Optimal. [guaranteed to find a shortest path if one exists.]

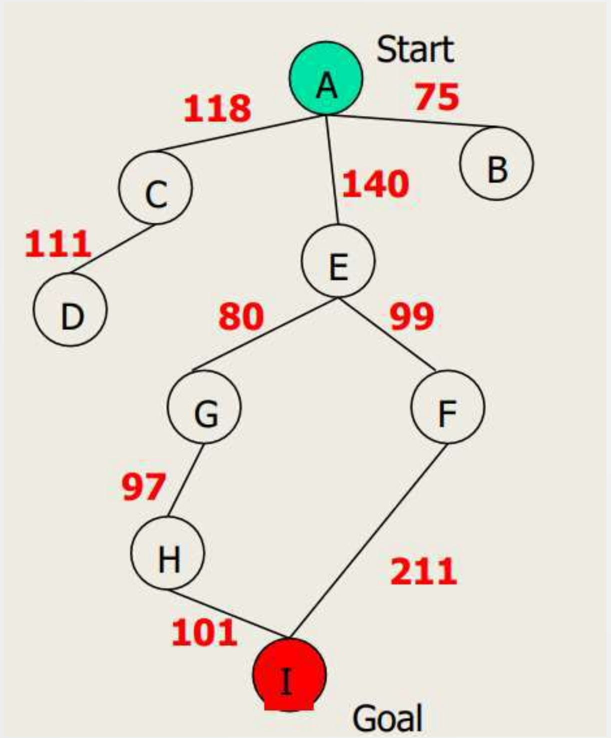
Try

- For the given graph, find the cost effective path from A to G using A* algorithm.



Try

- For the given graph, find the cost effective path from A to I using A* algorithm.



State	Heuristic: h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Informed Search (AO* Search Algorithm)

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 06
Presentation Date – 29/10/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Understand the concept of informed or heuristic search where additional information’s are used to find the best possible solution without exhaustively searching every possible state.

Informed / Heuristic / Intelligent Search
(4. Problem Reduction and And-Or Graphs)
[AO* Search Algorithm]

AO* Search Algorithm

Problem Reduction In Artificial Intelligence:

- AO* is an informed search algorithm, ***work based on heuristic.***
- We already know about the ***divide and conquer strategy***, a solution to a ***problem can be obtained by decomposing it into smaller sub-problems***.
- Each of this sub-problem can then be solved to get its sub solution.
- These sub solutions can then recombine to get a solution as a whole. That is why it is called as ***Problem Reduction***.
- ***AND-OR graphs or AND - OR trees are used*** for representing the solution.

AO* Search Algorithm (Contd..)

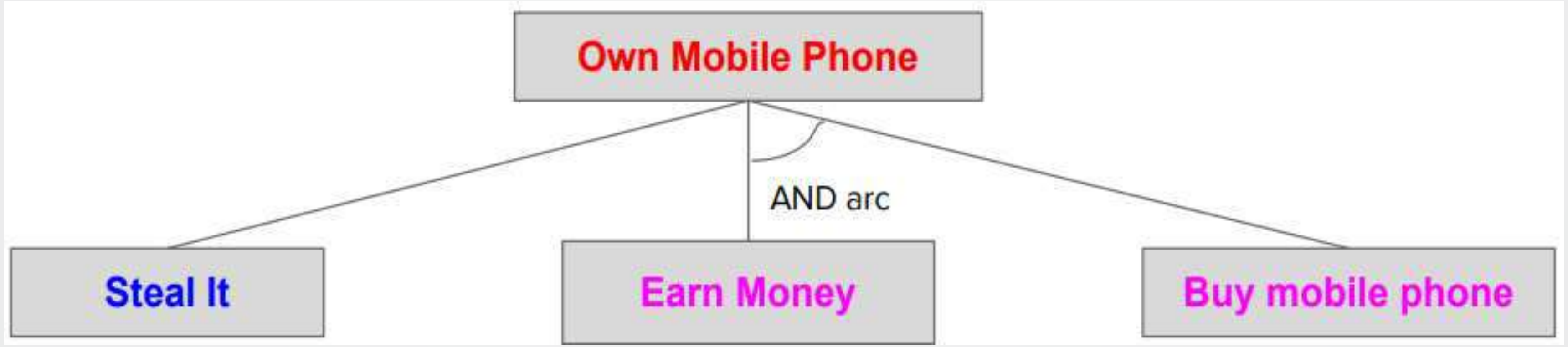
Problem Reduction In Artificial Intelligence:

- This method generates arc which is called as ***AND-OR arcs.*** One AND arc may point to any number of successor nodes, all of which must be solved in order for an arc to point to a solution.
- AND-OR graph is used to ***represent various kind of complex problem solutions.***
- ***AO* search algorithm is based on AND-OR graph so, it is called AO* search algorithm.***
- AO* Algorithm is ***based on problem decomposition*** (Breakdown problem into small pieces).

AO* Search Algorithm (Contd..)

AND-OR GRAPH [**used to find more than one solution**]

- AND-OR graph is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.



AO* Search Algorithm (Contd..)

AND-OR GRAPH

- Node in the graph will point both down to its successors and up to its parent nodes.
- Each Node in the graph will also have a heuristic value associated with it. **$f(n)=g(n)+h(n)$** **[By default $g(n)$ for any node = 1]**

  [$f(n)$: Cost function

  $g(n)$: Actual cost or Edge value

  $h(n)$: Heuristic/ Estimated value of the nodes]

AO* Search Algorithm - Steps

Step1: Initialise the graph to start node.

Step2: Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved.

Step3: Pick any of these nodes and expand it and if it has no successors call this value as **FUTILITY** otherwise calculate only f`(heuristic value) for each of the successors.

Step4: If f` is 0 then mark the node as SOLVED.

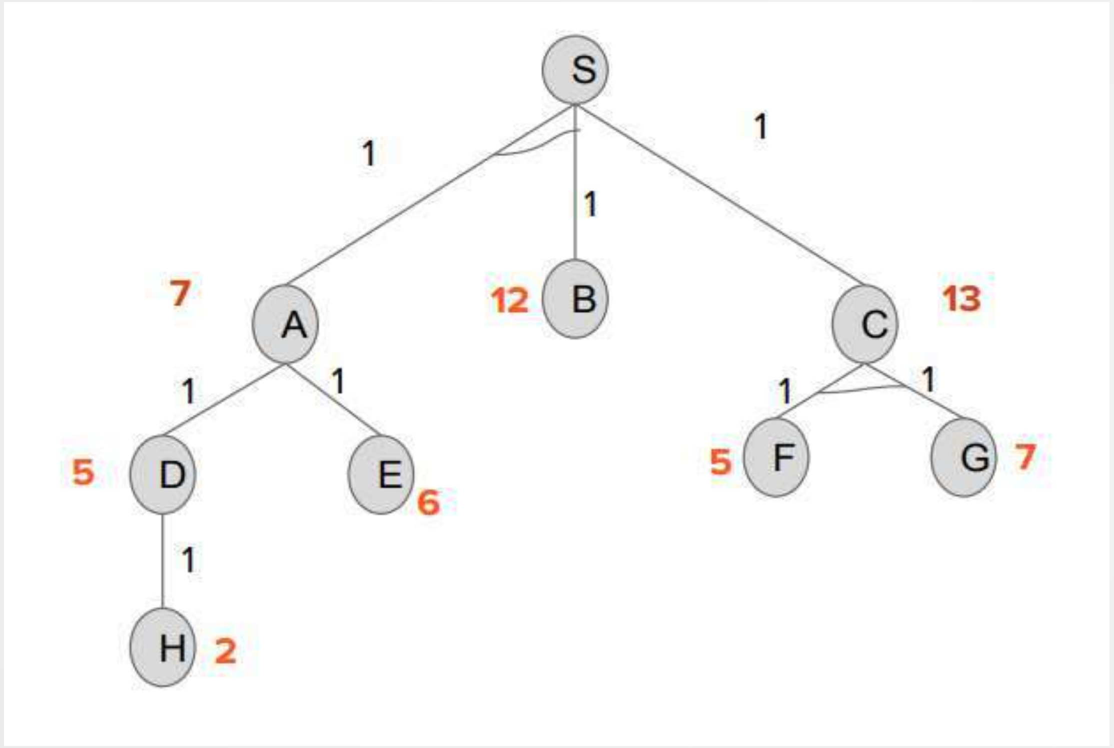
Step5: Change the value of f` for the newly created node to reflect its successors by back propagation.

AO* Search Algorithm - Steps

Step6: Wherever possible use the most promising routes and if a node is marked as SOLVED then mark the parent node as SOLVED.

Step7: If starting node is SOLVED or value greater than FUTILITY, stop, else repeat from 2.

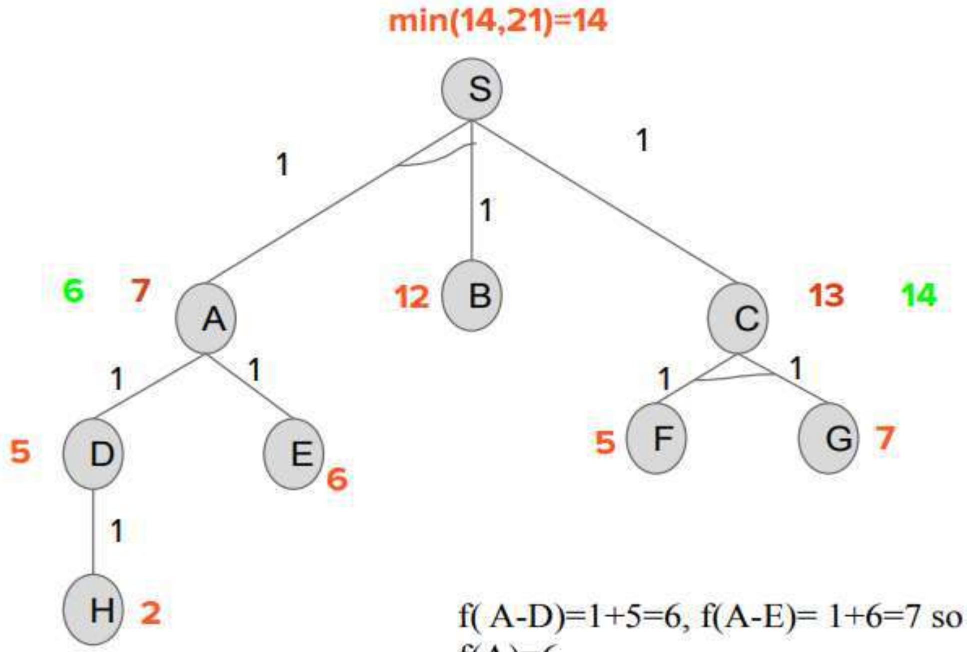
AO* Search Algorithm – Example -1



AO* Search Algorithm – Example -1

Revised cost: 15

f(n)=g(n)+h(n)
Path-1: f(S-C)= 1+13=14
Path-2: f(S-A-B)=1+1+7+12=21
f(C-F-G)=1+1+5+7=14
f(S-C)=1+14=15 **(revised)**



f(A-D)=1+5=6, f(A-E)= 1+6=7 so revised
f(A)=6
Revised (S-A-B)=1+6+1+12=20 X

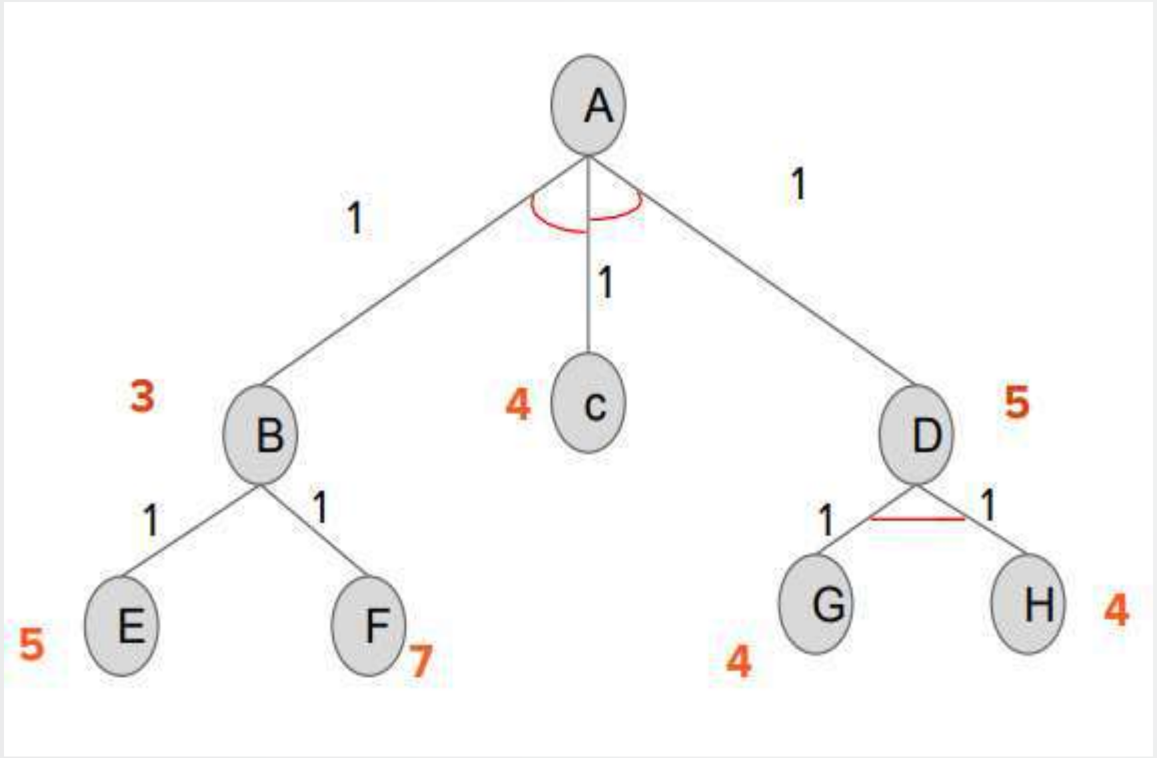
As we can see that the most promising path is f(S-C).

AO* Search Algorithm – Example -1

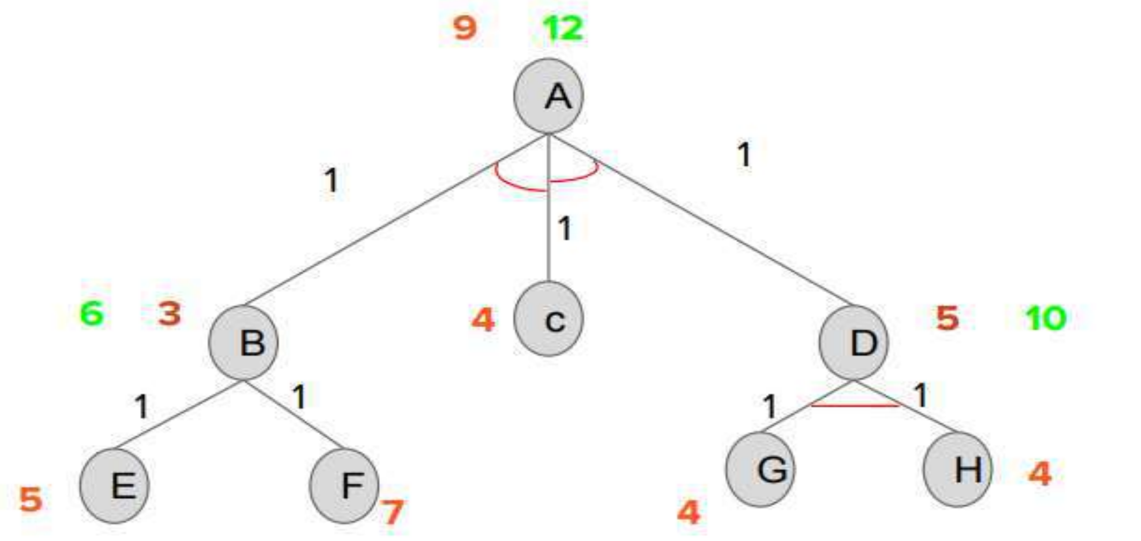
[NOTE:

Non-uniform Depth: The problem arises because, in an AND-OR graph, some parts of the solution might be deeper (require more steps or levels to solve) than others. ***When different subproblems have different depths (levels), the AO* algorithm prioritizes the most promising path based on the evaluation function*** (cost and heuristic estimates). This evaluation can lead the algorithm to focus on subtrees that look more promising at the time, ***without resolving others that are at a different depth or level.***]

AO* Search Algorithm – Example -2



AO* Search Algorithm – Example -2

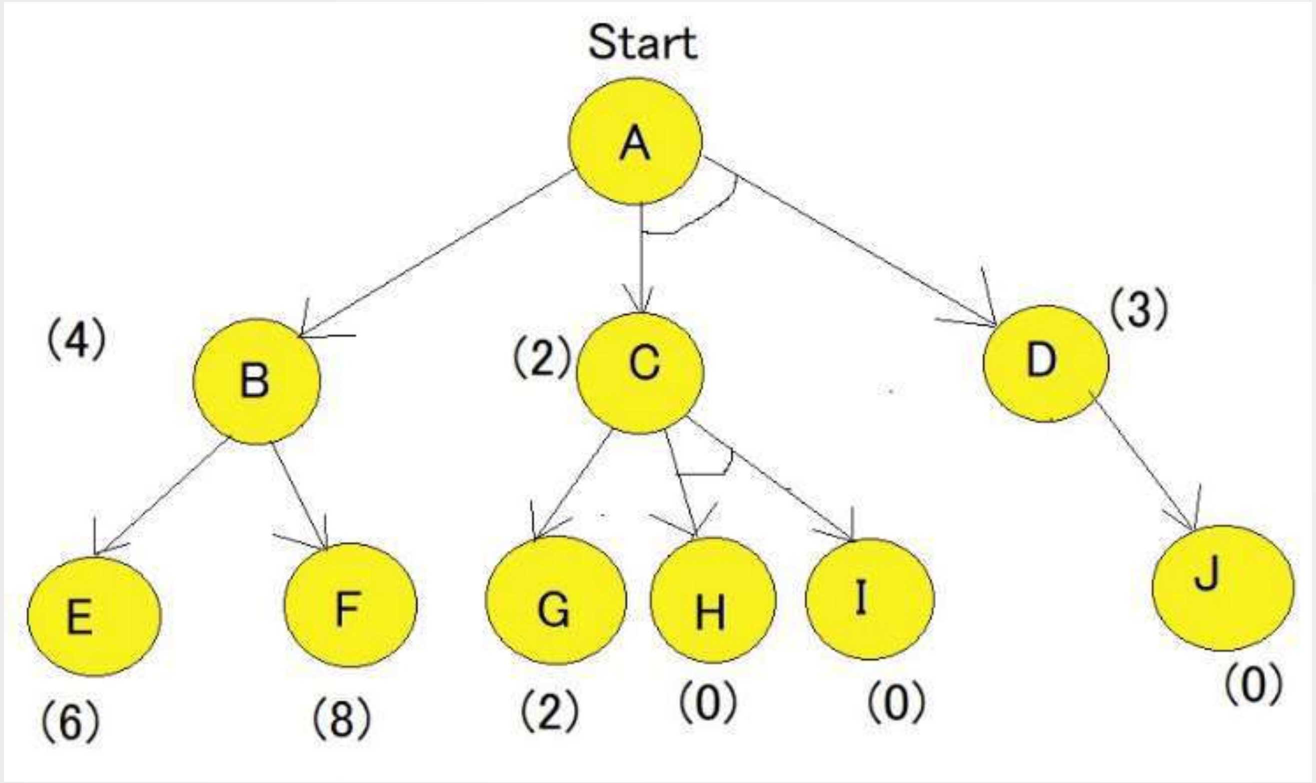


Path -1: f(A-B-C)= 1+1+3+4= 9
f(B-E)=1+5=6
f(B-F)=1+7=8
f(A-B-C)= 1+1+6+4= 12

Path-2: f(A-C-D)=1+1+4+5=11
f(D-G-H)= 1+1+4+4 =10
f(A-C-D)= 1+1+4+10= 16

As we can see that the most promising path is f(A-B-C).

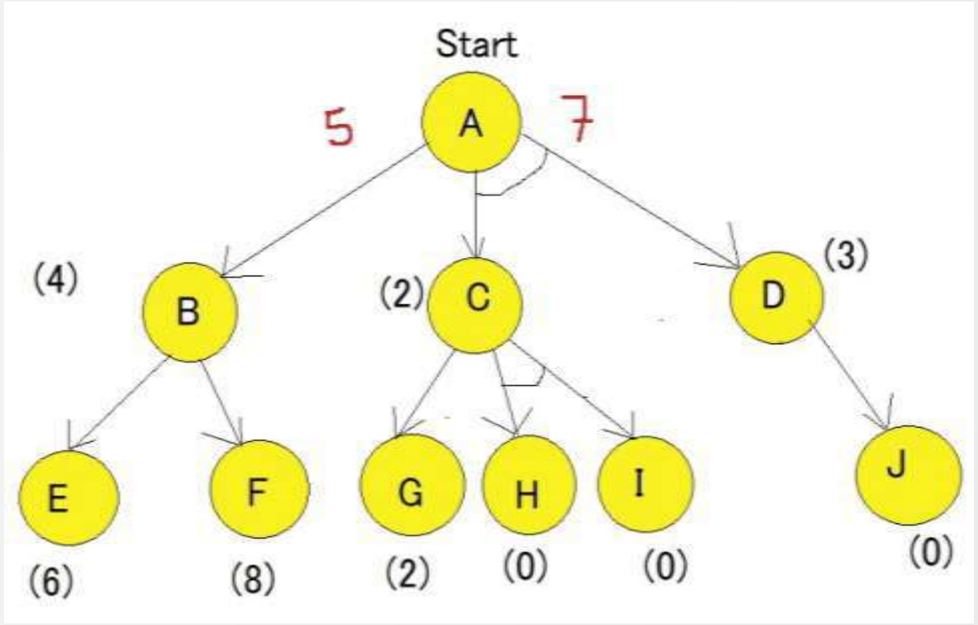
AO* Search Algorithm – Example -3



AO* Search Algorithm – Example -3

f(n)=g(n)+h(n)
Path1: f(A-B) = 4+1 = 5
Path2: f(A-C-D) = 2+1+3+1 = 7

The minimum cost path is chosen i.e A-B.



AO* Search Algorithm – Example -3

Explore path A-B

f(B-E) = 6+1 = 7

f(B-F) = 8+1 = 9

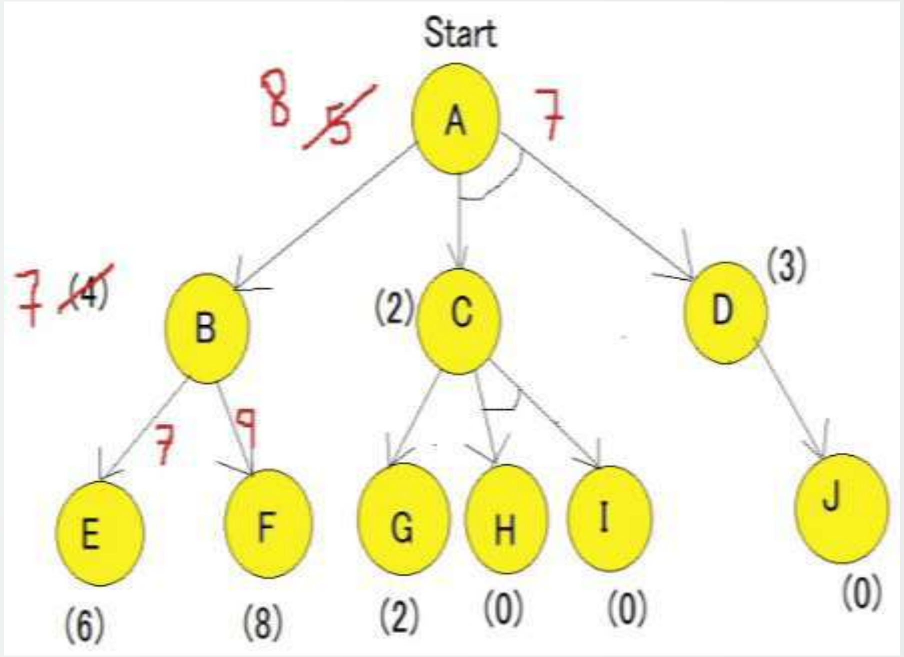
f(B) = 4 (replaced with 7 as between 7 and 9, least cost is 7)

Now, update the path cost f(A-B) by applying backtracking.

f(A-B) = 7+1 = 8

As f(A-C-D) = 7 < f(A-B)

Explore f(A-C-D) path.



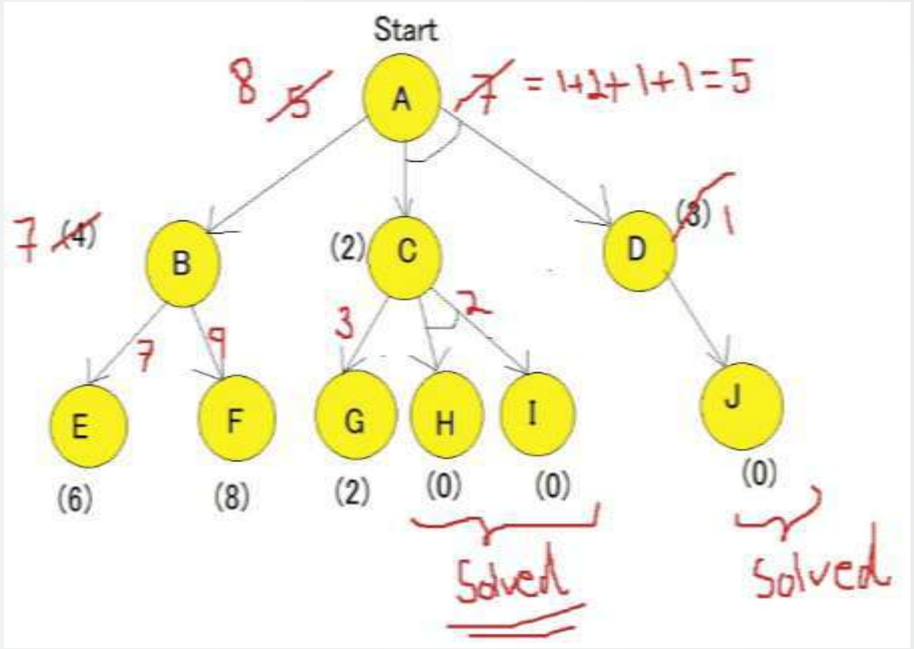
AO* Search Algorithm – Example -3

Explore A-C-D path

f(C-G) = 2+1 = 3
f(C-H-I) = 0+1+0+1 = 2
f(C) = 2 (given)
So, No changes happened.

Now, f(D-J) = 0+1 = 1
f(D) = 3 (replaced with 1)

Again, recalculate the updated path cost for
f(A-C-D) = 2+1+1+1 = **5**



Hence the most promising path is f(A-C-D) as it has the less cost.

AO* Search Algorithm

NOTE: The algorithm does not explore all the solution path once it find a solution.

AO* Search Algorithm - Advantages

- It is ***Complete.***
- Will ***not go in infinite loop.***
- ***Less Memory*** Required.
- If traverse according to the ordering of nodes.
- It can be ***used for both OR and AND graph***.

AO* Search Algorithm - Disadvantages

- It is ***not optimal*** as it does not explore all the path once it find a solution.

AO* Search Algorithm - Applications

- AO* is often ***used for the common pathfinding problem*** in applications such as ***video games***, but was originally designed as a general graph traversal algorithm.
- It finds applications in diverse problems, including the ***problem of parsing using stochastic grammars in NLP.***
- Other cases include an ***Informational search with online learning.***
- It is useful for ***searching game trees, problem solving etc.***

Difference Between A* and AO*

A*

- It represents an ***OR graph algorithm*** that is used to ***find a single solution*** (either this or that).
- It is a computer algorithm which is ***used in path-finding and graph traversal***. It is used in the process of plotting an efficiently directed path between a number of points called nodes.

AO*

- It represents ***an AND-OR graph algorithm*** that is used to ***find more than one solution*** by ANDing more than one branch.
- In this algorithm you follow a similar procedure but there are constraints traversing specific paths.

Difference Between A* and AO*

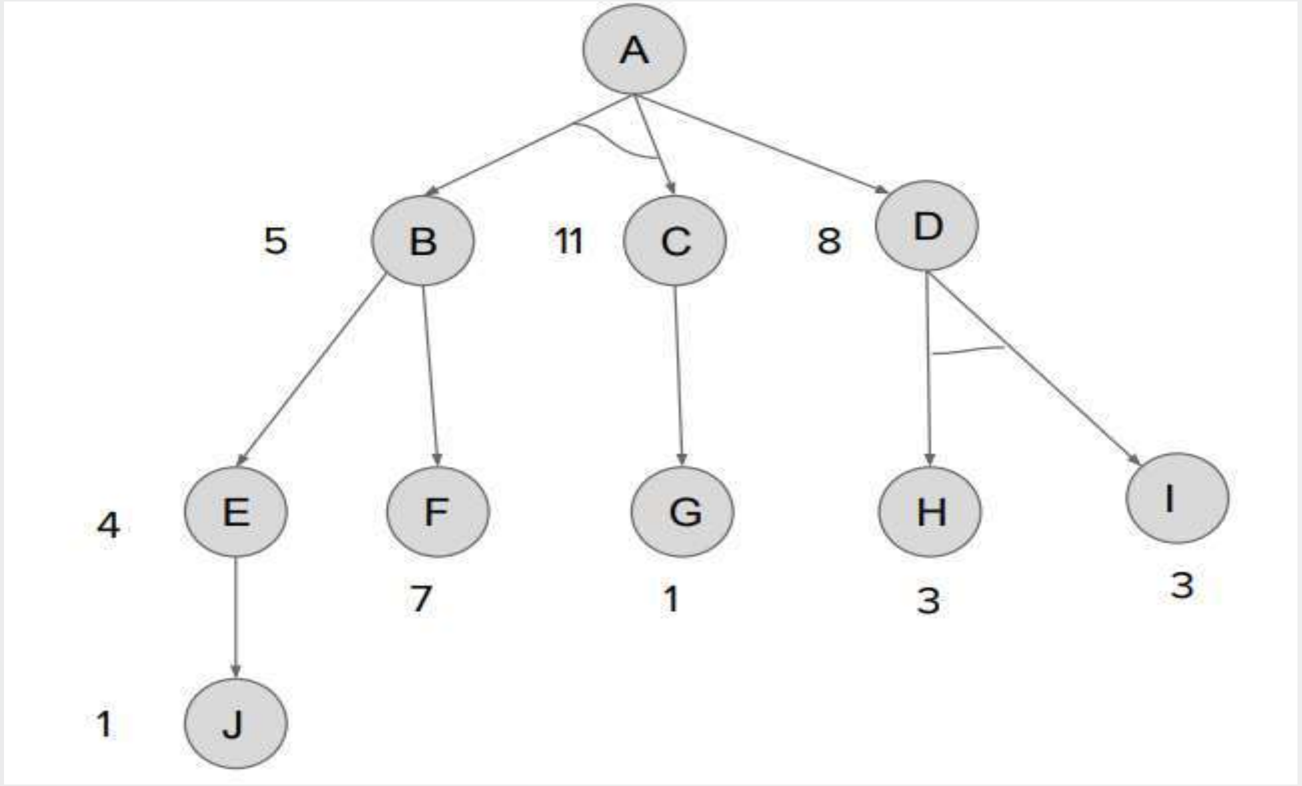
A*

- In this algorithm you traverse the tree in depth and keep moving and adding up the total cost of reaching the cost from the current state to the goal state and add it to the cost of reaching the current state.

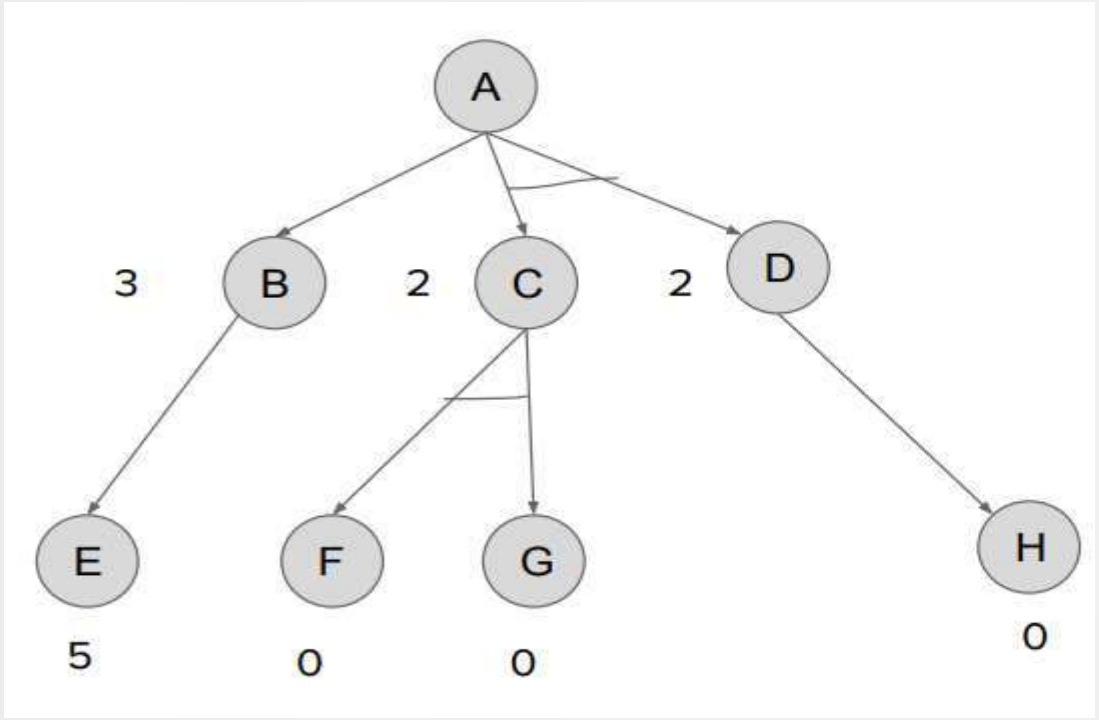
AO*

- When you traverse those paths, cost of all the paths which originate from the preceding node are added till that level, where you find the goal state regardless of the fact whether they take you to the goal state or not.

Try : AO* Search Algorithm



Try : AO* Search Algorithm



References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Informed Search (CSP, MEA)

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 07
Presentation Date – 30/10/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Understand the concept of informed or heuristic search where additional information’s are used to find the best possible solution without exhaustively searching every possible state.

Informed / Heuristic / Intelligent Search (5. Constraint Satisfaction Problem (CSP))

[Graph Colouring Problem and Crypt Arithmetic Problems.]

Constraint Satisfaction

Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.

Constraint Satisfaction Problem

- It is a search procedure that ***operates in a space of constraints***.
- Any problem in the world can mathematically be represented as CSP.
- The solution is typically a state that can satisfy all the constraints.

Constraint Satisfaction Problems - Goal

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue.

For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

Constraint Satisfaction Problem - Process

- Constraints are discovered and propagated throughout the system.
- If still there is no solution, search begins.

  i.e., a guess is made about something and added as new constraints.

Constraint Satisfaction Problem - Components

There are mainly ***three basic components*** in the constraint satisfaction problem:

- **Variables**: {V1, V2, V3 … Vn}
- **Set of Domains**: {D1, D2, D3, … Dn} for each variable.
- **Constraints**: The guidelines that control how variables relate to one another are known as constraints. (i.e., it specify the allowable combination of values.)

Constraint Satisfaction Problem - Representation

Constraints can be represented as an order pair of :

Ci = {Scope, Rel}

[Where,

Scope is set of variables that participate in constraint.

Rel is relation that defines the values that variable can take.]

Constraint Satisfaction Problem - Representation

Example:

$$\begin{tikzcd}
\mathbf{V1} \arrow[d] & \text{and} & \mathbf{V2} \text{ ------------variables} \arrow[d] \\
\mathbf{A} & & \mathbf{B} \text{ -------------Domains}
\end{tikzcd}$$

Here the values of V1 and V2 can’t be same.
Then the Constraints can be represented as :

$$\begin{tikzcd}
\mathbf{C1 = \{(V1, V2),} \arrow[d] & \mathbf{V1!=V2\}} \arrow[d] \\
\text{Scope} & \text{Relation}
\end{tikzcd}$$

Constraint Satisfaction Problem - Example

1. Real world problems.
2. Problems solved using intelligent backtracking. (Node Colouring , Crypt Arithmetic Problem)

Real World CSP’S

- ***Assignment problems***

  e.g., who teaches what class

- ***Timetable problems***

  e.g., which class is offered when and where?

- ***Transportation scheduling***
- ***Factory scheduling***

Intelligent Backtracking Method

- Intelligent Backtracking method is used to solve CSP.
- Here, Intelligent Backtracking means, when ***conflict occurs it backtrack.***

Problem1 – Node Colouring

- It is one of the classical example of CSP.
- Here

V = {1, 2, 3, 4}

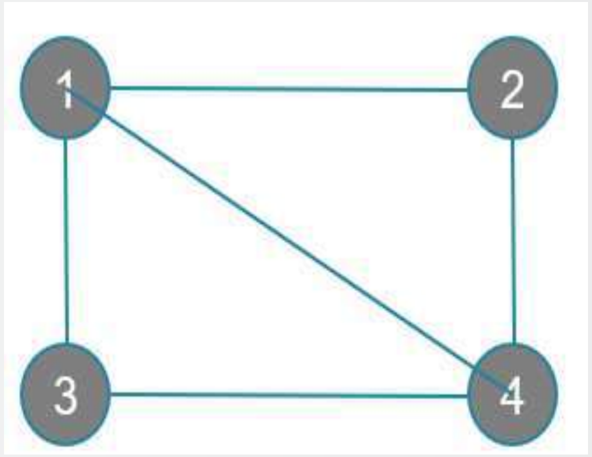
D = {Red, Green, Blue}

C = {1≠2, 1≠3, 1≠4, 2≠4, 3≠4} i.e., adjacent nodes should not have same color.

(Where, V → Set of variables,

D → Set of domains,

C → Constraints)

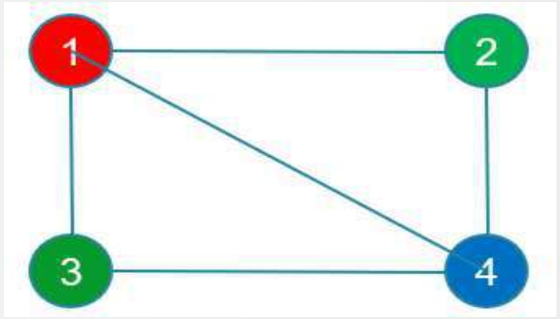


Problem1 – Node Colouring

- **Solution:**

Let,

	1	2	3	4
Initial	R, G, B	R, G, B	R, G, B	R, G, B
1 = R	R	G, B	G, B	G, B
2 = G	R	G	G, B	B
3 = B	R	G	B	🟥
3 = G	R	G	G	B



Error has occurred, so backtrack the node where error was occurred.

Problem2 – Cryptarithmetic Problem

It is a type of Constraint Satisfaction Problem.

Rules / Constraints :

1. Variables: can take values from 0-9.
2. No two variables should take same value.
3. The values should be selected in such a way that it should comply with arithmetic properties.
4. There should be only one carry forward, while performing the addition operation on a problem.

Problem2 – Cryptarithmetic Problem

Given a cryptarithmetic problem,

i.e., **S E N D + M O R E = M O N E Y**

In this example, add both terms S E N D and M O R E to bring M O N E Y as a result. i.e.,

```
      S E N D
+     M O R E
-------------
    M O N E Y
```

Problem2 – Cryptarithmetic Problem

Solution:

Step1:

Starting from the left hand side (L.H.S) , the terms are S and M. Assign a digit which could give a satisfactory result.

$$\begin{array}{r} S \\ +M \\ \hline MO \\ \hline \end{array} \longrightarrow \begin{array}{r} 9 \\ +1 \\ \hline 10 \\ \hline \end{array}$$

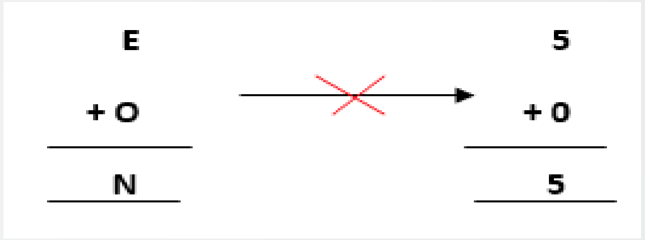
Let’s assign **S->9 and M->1.**

Hence, we get a satisfactory result by adding up the terms and got an assignment for **O** as **O->0** as well.

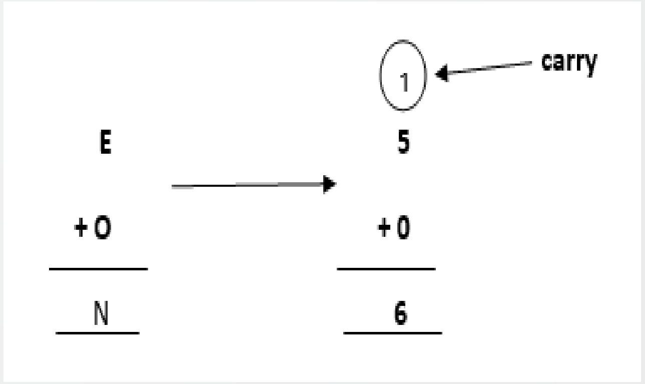
Problem2 – Cryptarithmetic Problem

Step2:

Now, move ahead to the next terms E and O to get N as its output.



Adding E and O, which means 5+0=0, which is not possible because according to cryptarithmetic constraints, we cannot assign the same digit to two letters. So, we need to think more and assign some other value.



Problem2 – Cryptarithmetic Problem

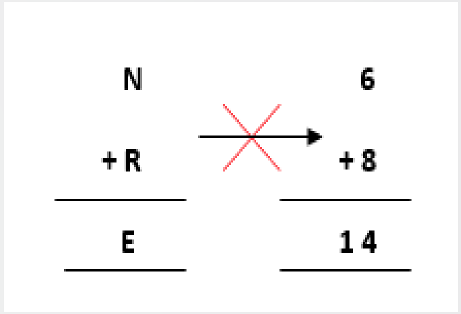
Step3:

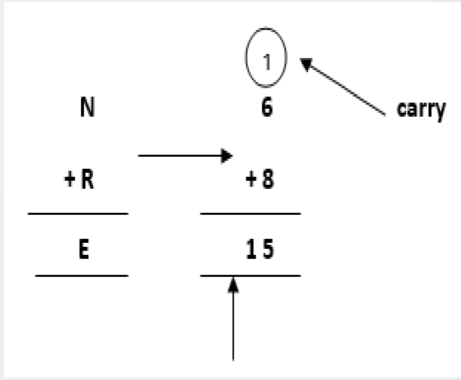
Further, adding the next two terms **N and R** we get,

But, we have already assigned **E->5.** Thus, the above result does not satisfy the values

because we are getting a different value for **E**. So, we need to think more.

Again, after solving the whole problem, we will get a carryover on this term, so our answer will be satisfied.





Problem2 – Cryptarithmetic Problem

Step4:

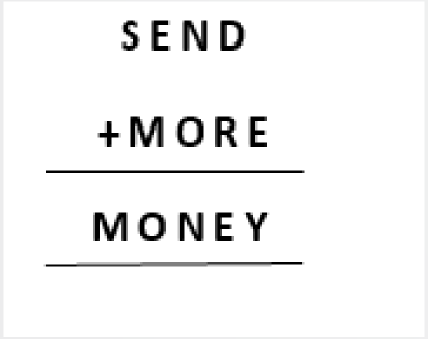
Again, on adding the last two terms, i.e., the rightmost terms D and E, we get Y as its result.

$$\begin{array}{r} D \\ +E \\ \hline Y \\ \hline \end{array} \qquad\qquad \begin{array}{r} 7 \\ +5 \\ \hline 12 \\ \hline \uparrow \end{array}$$

Problem2 – Cryptarithmetic Problem

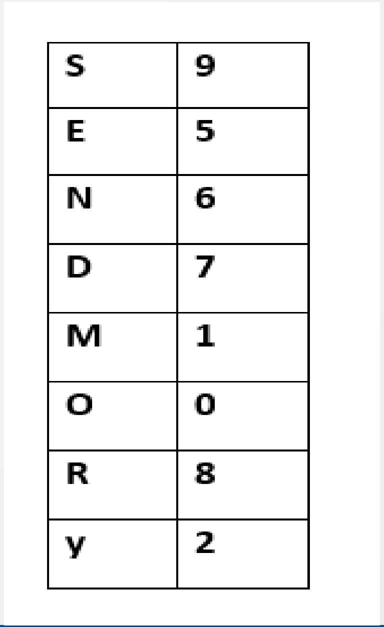
Result:

Final result



```
  SEND
 +MORE
 -----
 MONEY
```

Representation of the assignment of the digits to the alphabets



S	9
E	5
N	6
D	7
M	1
O	0
R	8
y	2

Problem3 – Cryptarithmetic Problem

$$\begin{array}{rccc} & & T & O \\ + & & G & O \\ \hline & O & U & T \end{array}$$

Left most digit should be = 1
!=0

Now
O = 1 (default)

T + G = U + 10
Let T=2, G=9
2 + 9 = U + 10
U = 1 (Not possible as O=1)

Let T=2, G=8
2 + 8 = U + 10
U = 0 (Correct)

Representation of the assignment of the digits to the alphabets

Letters	Digit
T	2
O	1
G	8
U	0

Try

1.

```
  BASE
 +BALL
 -----
 GAMES
 -----
```

2.

```
  YOUR
  +YOU
  ----
 HEART
 -----
```

Informed / Heuristic / Intelligent Search (6. Means-Ends Analysis (MEA))

Means-Ends Analysis (MEA) - Introduction

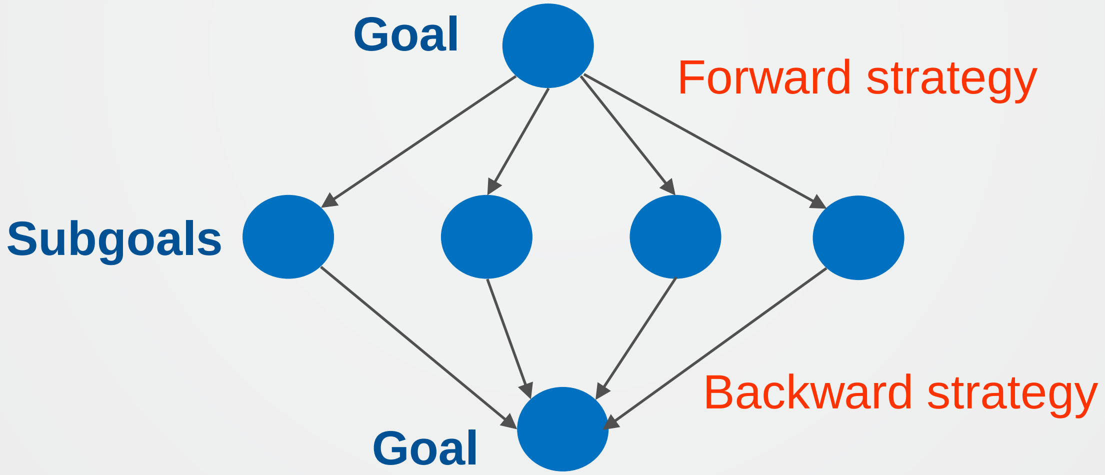
- Means End Analysis (MEA) is a ***problem-solving technique*** that has been used ***since the 1950s*** to ***stimulate creativity***.
- The MEA technique as a problem-solving strategy was first introduced in ***1961*** by ***Allen Newell*** and ***Herbert A. Simon*** in their computer problem solving program ***General Problem Solver (GPS).***
- Means-end analysis (MEA) is a The approach involves identifying actions or "***problem-solving technique used in AI to reduce the difference between the current state and the goal state by breaking down the problem into smaller, manageable sub-problems,*** means that will help achieve specific "ends" or objectives, making progress towards the goal state step-by-step.

What is Means-Ends Analysis (MEA)

- *MEA is a problem-solving technique that identifies the* ***current state****, defines the* ***end goal*** *and determines the* ***action plan*** *to reach the end state in a* ***modular way.***
- ***End Goals*** are splited into sub-goals and sub-sub goals and then action plans are drawn to achieve sub-goals first and then move towards achieving the main goal progressively.
- ***Most of the problem-solving strategies will have either forward actions or backward actions.***
- But this technique ***combines both forward and backward strategies to solve complex problems.***

What is Means-Ends Analysis (MEA)

- That means, MEA will have a ***mixture of action plans in either of the directions to solve the problems*** in a modular way, in the sense that it attempts to solve the ***major problems*** first and get back to ***minor problems*** later or vice versa.

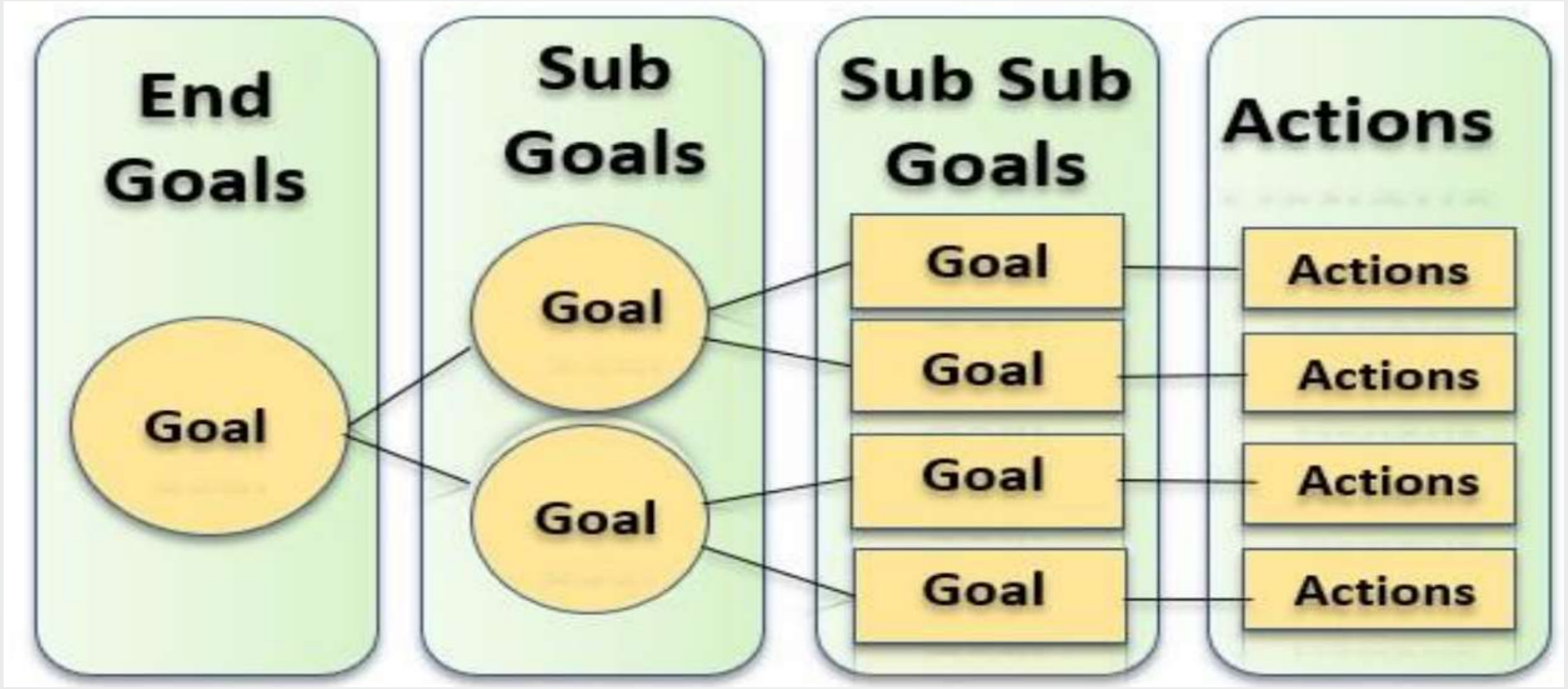


How MEA Works ?

The means-ends analysis process ***can be applied recursively*** for a problem. It is a strategy to control search in problem-solving. ***Following are the main Steps*** which describes the ***working of MEA technique for solving a problem.***

1. *First, evaluate the* ***difference between Initial State and final State.***
2. *Select the* ***various operators*** *which can be applied for each difference.*
3. ***Apply the operator at each difference****, to get closer to the goal.*
4. *This process of identifying differences and selecting operations is* ***repeated*** *until the goal state is reached or no further operations are possible.*

How MEA Works ?



Operator Subgoaling

- In the MEA process, we detect the differences between the current state and goal state.
- Once these differences occur, then we can apply an operator to reduce the differences.
- But ***sometimes it is possible that an operator cannot be applied to the current state.***
- Hence, ***we create the subproblem of the current state***, in which operator can be applied, such type of ***backward chaining*** in which ***operators are selected, and then sub goals are set up*** to establish the preconditions of the operator is called ***Operator Subgoaling.***

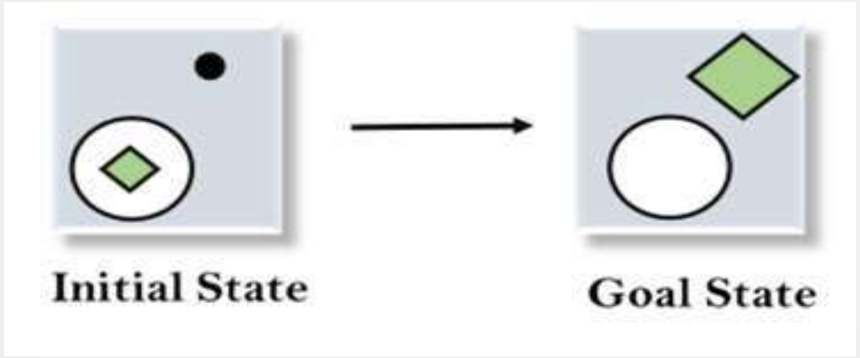
MEA – Example-1

If a robot needs to navigate through a maze, it will:

1. Identify its current position and the goal position.
2. Choose a movement action that will bring it closer to the goal (e.g., move forward, turn left).
3. Execute the movement.
4. Reassess the new difference between its current position and the goal, and repeat the process.

MEA – Example-2

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



MEA – Example (Contd..)

Solution:

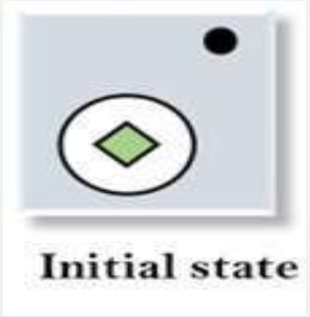
- To solve the above problem, we will ***first find the differences between initial states and goal states***, and for each difference, we will generate a new state and will apply the operators.
- The ***operators*** we have for this problem are:

1. Move – Move the object diamond outside the circle.
2. Delete – Delete the black circle.
3. Expand – Expand or increase the size of the diamond.

MEA – Example (Contd..)

Step1: Evaluating the initial state:

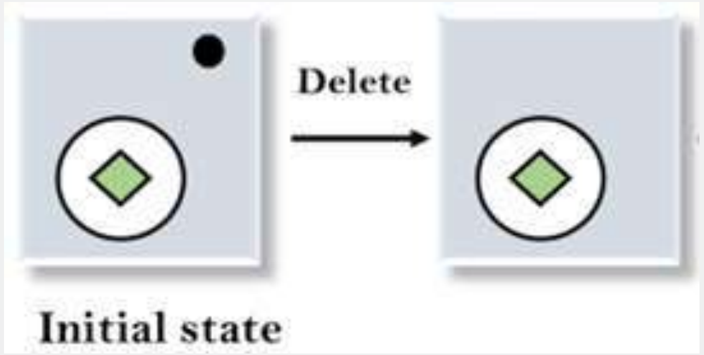
In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



MEA – Example (Contd..)

Step2: Applying Delete operator:

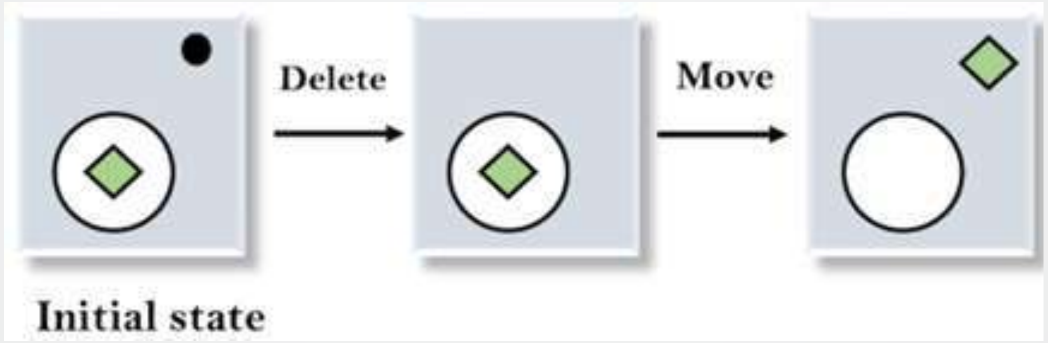
As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the Delete operator to remove this dot.



MEA – Example (Contd..)

Step3: Applying Move Operator:

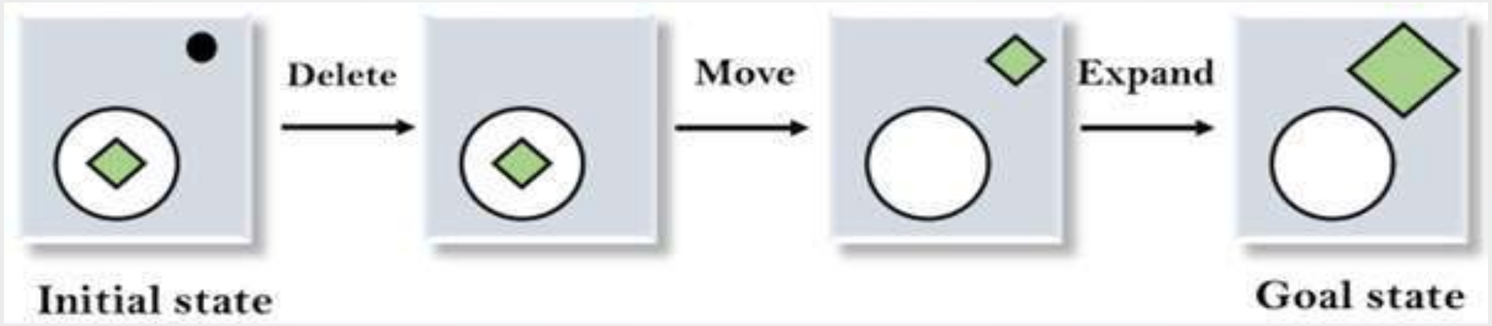
After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the ***Move Operator.***



MEA – Example (Contd..)

Step4: Applying Expand Operator:

Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply ***Expand operator***, and finally, it will generate the goal state.



MEA - Applications

Means-End Analysis is used in the following disciplines:

1. ***Artificial Intelligence application***
2. ***General Management area***
3. ***implementing Business transformation projects***
4. ***personal life***

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Local Search Algorithms and Optimization Problems (Hill Climbing Algorithm)

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 08
Presentation Date – 01/11/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Apply local search algorithms and optimization problems in large-scale problems to get high-quality solutions.

Local Search Algorithms and Optimization Problems
(Beyond Classical Search)

Local Search Algorithms and Optimization Problems

- The search algorithms we have seen so far, more often ***concentrate on path through which the goal is reached.***
- But ***if the problem does not demand the path of the solution and it expects only the final configuration*** of the solution then we have different types of problem to solve.

Local Search

- They ***operate using single state***. (rather than multiple paths).
- They generally ***move to neighbours of the current state.***
- There is ***no such requirement of maintaining paths in memory.***
- They are ***not "Systematic***" algorithm procedure.

Local Search - Advantages

The main ***advantages of local search algorithm are:***

*1) They **use very little and constant amount of memory**.*

*2) They have ability to **find reasonable solution for infinite state spaces** (for which systematic algorithms are unsuitable).*

Optimization Problem

Local search algorithms are ***useful for solving pure optimization problems.*** In pure optimization problems main ***aim is to find the best state*** according to required ***objective function.***

Local Search : State Space Landscape

Local search algorithm ***make use of concept called as state space landscape.*** This ***landscape has two structures*** –

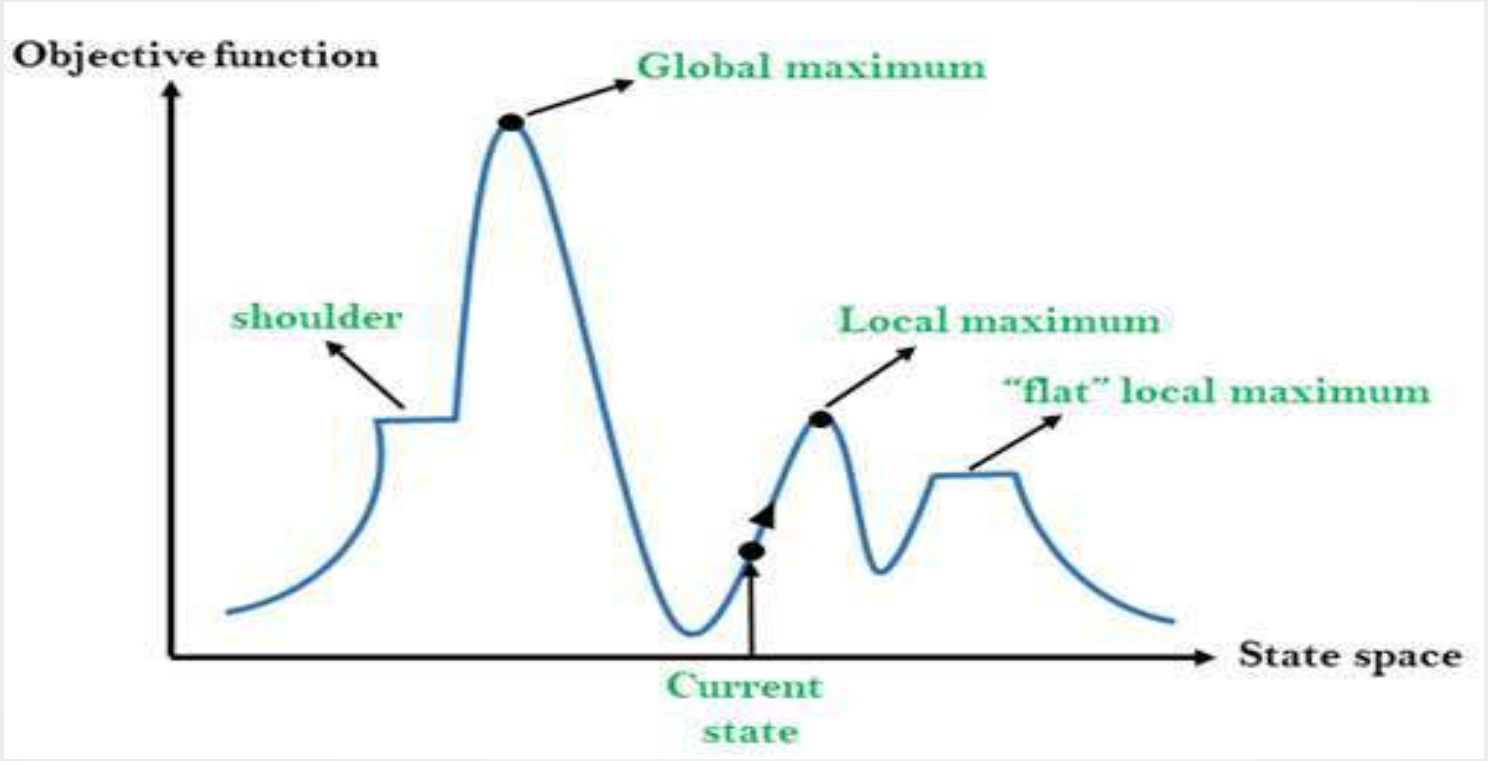
- ❖**Location** (defined by the state)
- ❖**Elevation** (defined by the value of the heuristic cost function or objective function).

✓If elevation corresponds to the cost, then the aim is to find the lowest valley - (a global minimum).

✓If elevation corresponds to the objective function, then aim is to find the highest peak - (a global maximum).

✓Local search algorithms explore the landscape.

Local Search : State Space Landscape



[Local search representation]

Local Search Algorithms and Optimization Problems - Examples

Following are the problems where only ***solution state configuration is important*** and ***not the path*** which has arrived at solution.

1) 8-queen (where only solution state configuration is expected).

2) Integrated circuit design.

3) Factory-floor layout.

4) Job-shop scheduling.

5) Automatic programming.

6) Telecommunication network optimization.

7) Vehicle routing.

Types of Local Search Algorithms

Several local search algorithms are commonly used in AI and optimization problems. Let's explore a few of them:

i. Hill Climbing

ii. Simulated Annealing

iii. Local Beam Search

Local Search : Performance Measurement

- Local search is ***complete*** i.e. it surely finds goal if one exists.
- Local search is ***optimal*** as it always ***find global minimum or maximum.***

Local Search Algorithms

(Hill Climbing)

Hill Climbing - Introduction

- Hill climbing algorithm is a ***local search algorithm.***
- It continuously ***moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem***.
- It ***terminates*** ***when it reaches a peak value where no neighbour has a higher value.***
- It is a ***heuristic search technique.***
- It is also called ***greedy local search*** as ***it only looks to its good immediate neighbour state*** and not beyond that.

Hill Climbing - Introduction

- This ***algorithm has a node that comprises two parts***: ***state*** and ***value.***
- It ***begins with a non-optimal state (the hill's base)*** and upgrades this state until a certain precondition is met.
- The ***heuristic function*** is used as the basis for this precondition.
- The process of continuous improvement of the current state of iteration can be termed as ***climbing***. This explains why the algorithm is termed as a ***hill-climbing algorithm.***

Hill Climbing - Objective

A hill-climbing algorithm’s ***objective is to attain an optimal state that is an upgrade of the existing state.*** When the current state is improved, the algorithm will perform further incremental changes to the improved state. This process will continue until a peak solution is achieved. The peak state cannot undergo further improvements.

Hill Climbing - Features

- **Generate and Test Approach:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method ***produce feedback which helps to decide which direction to move*** in the search space.
- **Greedy Local Search**: The algorithm uses a cheap strategy, ***opting for immediate beneficial moves*** that promise local improvements. i.e., Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No Backtracking:** A hill-climbing algorithm ***only works on the current state and succeeding states (future).*** It does not look at the previous states.

Hill Climbing - Algorithm

- **S1:** Evaluate the starting state. If it is a goal state, then stop and return success.
- **S2**: Else continue with the starting state as considering it as a current state.
- **S3**: Continue with (S4) until a solution is found. i.e., until there are no new states left to be applied in the current state.
- **S4**:

    i) Select a state that has not been yet applied to the current state and apply it to produce a new state.

    ii) Procedure to evaluate a new state.

Hill Climbing - Algorithm

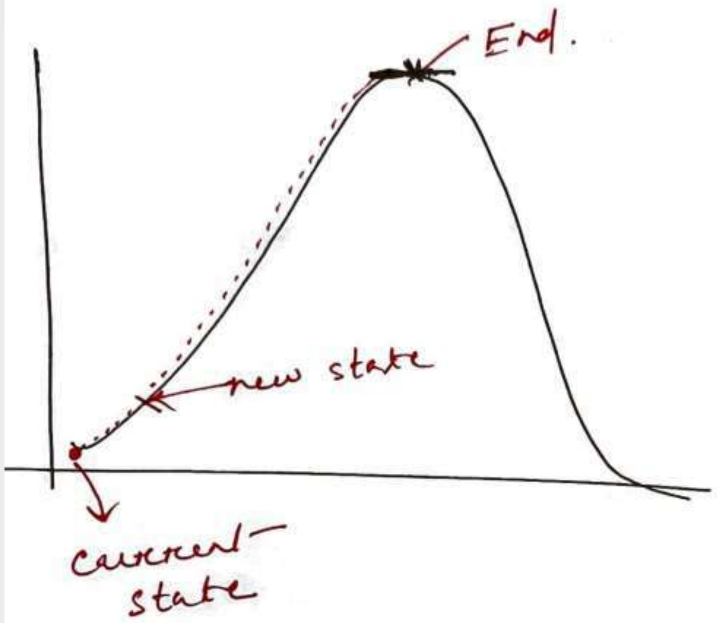
-If the current state is a goal state, then stop and return success.

-if it is better than the current state, then make it current state and proceed further.

-if it is not better than current state, then continue in the loop until a solution is found.

- **S5**: Exit.

Hill Climbing - Example



If new state is better than current state, ⇒ newstate = current state.

Hill Climbing - Types

1. Simple Hill Climbing
2. Steepest-Ascent Hill Climbing
3. Stochastic hill climbing

1. Simple Hill Climbing

- Simple hill climbing is the ***simplest way to implement a hill climbing algorithm.***
- It only evaluates the neighbour node state at a time and selects the first one which optimizes current cost and set it as a current state. ***It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state.***
- This algorithm has the following ***features:***
  - Less time consuming.
  - Less optimal solution and the solution is not guaranteed.

2. Steepest-Ascent Hill Climbing

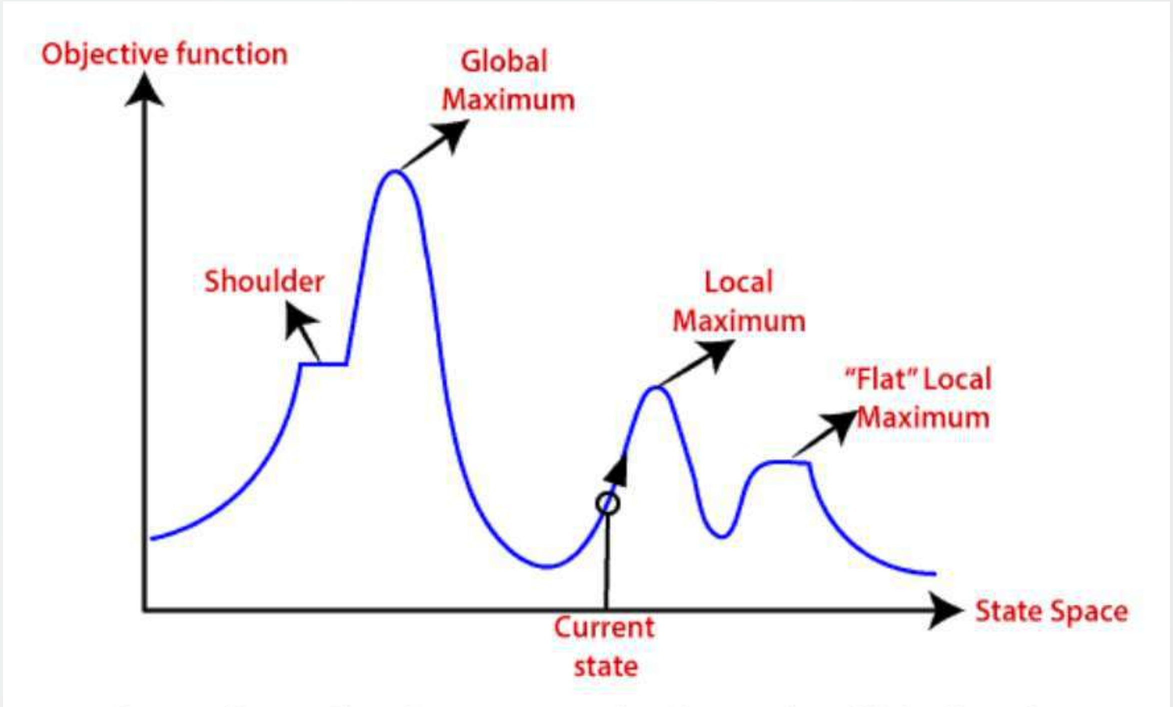
- The steepest-Ascent algorithm is a ***variation of simple hill climbing algorithm.***
- This algorithm ***examines all the neighbouring nodes of the current state and selects one neighbour node which is closest to the goal state.***
- This algorithm consumes more time as it searches for multiple neighbours.

3. Stochastic hill climbing

- Stochastic Hill Climbing ***doesn’t look at all its neighbouring nodes*** to check if it is better than the current node instead, it ***randomly selects one neighbouring node***, and based on the pre-defined criteria it decides whether to go to the neighbouring node or select an alternate node.

Hill Climbing – State Space Diagram and Analysis

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a ***graph between various states of algorithm and Objective function/Cost***.



A one-dimensional state-space landscape in which elevation corresponds to the objective function

Different Regions in the State Space Diagram

Local Maximum: Local maximum is a *state which is better than its neighbour states, but there is also another state which is higher than it.*

Global Maximum: Global maximum *is the best possible state of state space landscape*. It has the highest value of objective function.

Current state: It is a state in a landscape diagram *where an agent is currently present.*

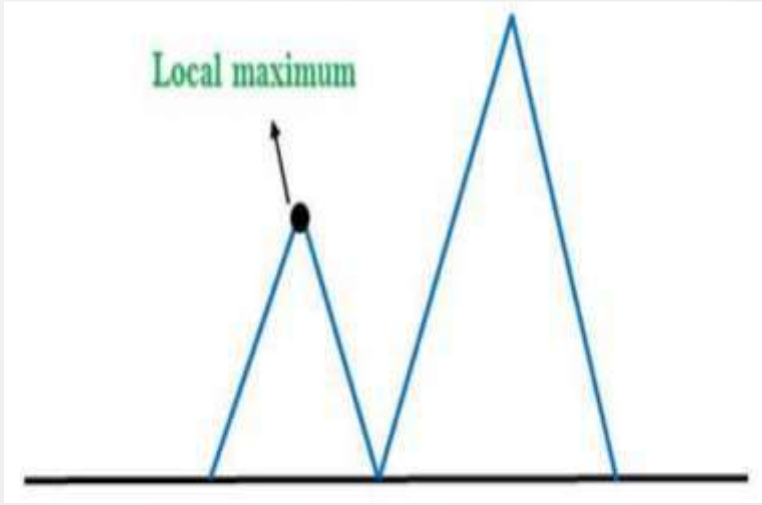
Flat local maximum: It is a flat space in the landscape where *all the neighbour states of current states have the same value.*

Shoulder: It is a *plateau region which has an uphill edge.*

Hill climbing - Problems in Different Regions

1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighbouring states, but there is another state also present which is higher than the local maximum.

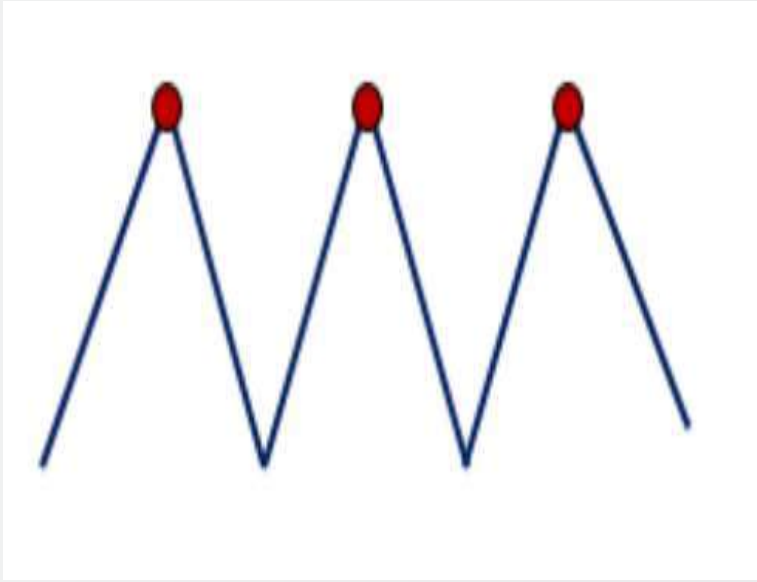
Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



Hill climbing - Problems in Different Regions

2. Ridges: Ridge occurs when there are multiple peaks and all have the same value or in other words, there are multiple local maxima which are same as global maxima.

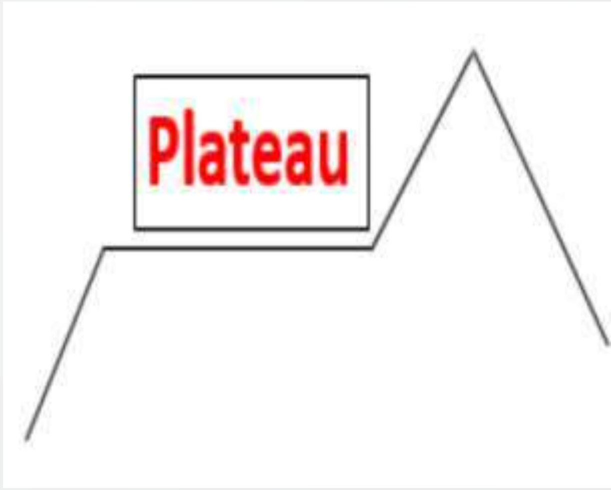
Solution: Ridge obstacle can be solved by moving in several directions at the same time.



Hill climbing - Problems in Different Regions

3. Plateau: A plateau is the flat area of the search space in which all the neighbour states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find nonplateau region.



Hill climbing - Advantages

1. Hill Climbing is very useful in ***routing-related problems*** like ***Travelling Salesmen Problem***, ***Job Scheduling, Chip Designing, and Portfolio Management.***
2. It is good in solving ***the optimization problem*** while using only limited computation power.
3. It is more ***efficient than other search*** algorithms.
4. This algorithm is straight forward to understand and implement.
5. It's memory efficient, maintaining only the current state's data.

Hill climbing - Disadvantages

1. **Susceptibility to Local Optima**: The algorithm can become stuck at locally optima solutions that are not the best overall.
2. **Limited Exploration**: Its tendency to focus on the immediate vicinity limits its exploration, potentially overlooking globally optimal solutions.
3. **Dependence on initial state:** The quality and effectiveness of the solution found heavily depend on the straight point.

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Local Search Algorithms and Optimization Problems (Simulated Annealing, Local Beam Search)

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 09
Presentation Date – 02/11/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Apply local search algorithms and optimization problems in large-scale problems to get high-quality solutions.

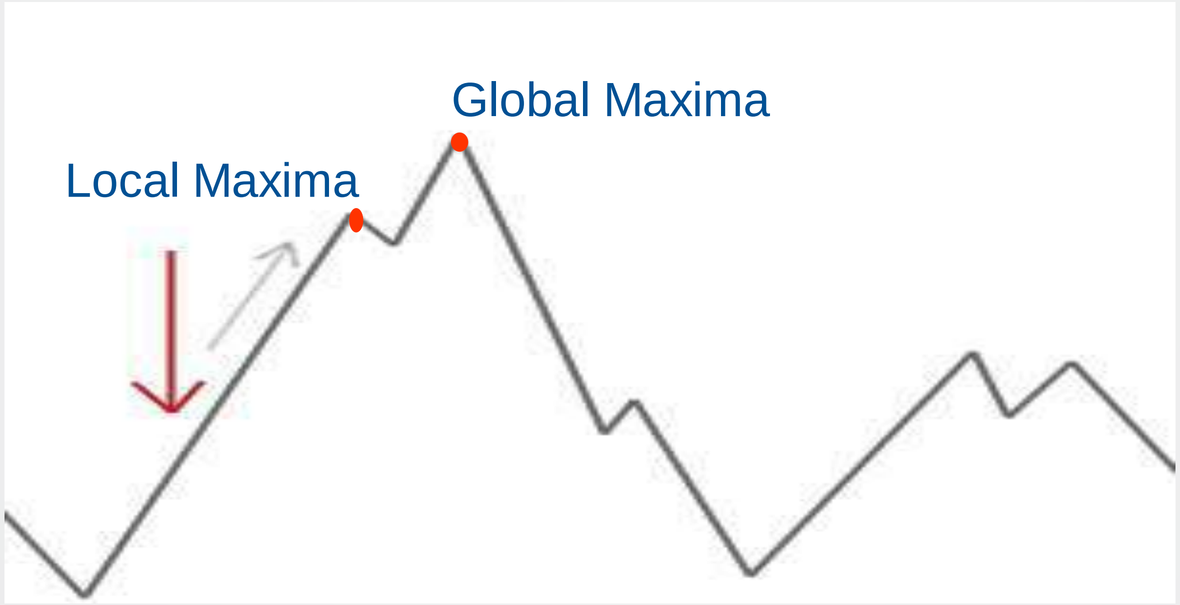
Local Search Algorithms
(Simulated Annealing)

Simulated Annealing

- ***Annealing*** is the process *used to temper or harden metals and glass by heating them to a high temperature* and then gradually cooling them, thus allowing the material to reach a low energy crystalline state.
- The simulated annealing algorithm is ***quite similar to hill-climbing.***
- Instead of picking the best move, however, ***it picks a random move.*** If the move improves the situation , it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1 or it moves downhill and chooses another path.

Simulated Annealing

- ***Checks all the neighbours.***
- Moves to ***worst state may be accepted***. (i.e., it allows downward steps)



Simulated Annealing

Advantages

- Easy to code for complex problem.
- Gives good solution.

Disadvantages

- Slow process.
- Can’t tell whether an optimal solution is found.

Difference Between Simulated Annealing & Hill Climbing

Simulated Annealing

- Annealing schedule is maintained.
- Moves to worst step may be accepted.

Hill Climbing

- No schedule is maintained.
- Worst step not possible here. Always moves in upward direction.

Local Search Algorithms

(Local Beam Search)

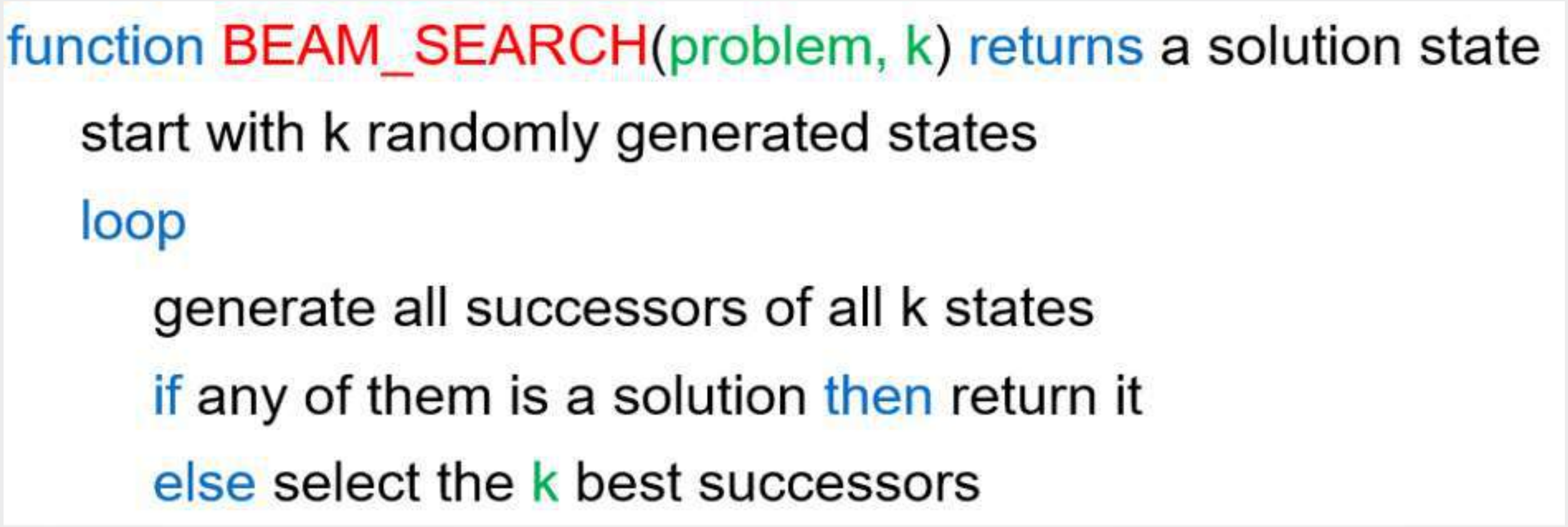
Local Beam Search

- **Local Beam Search** is a heuristic search algorithm used to solve optimization and search problems.
- It ***improves upon hill-climbing*** by ***keeping track of multiple states instead of just one.***

Local Beam Search - How it works?

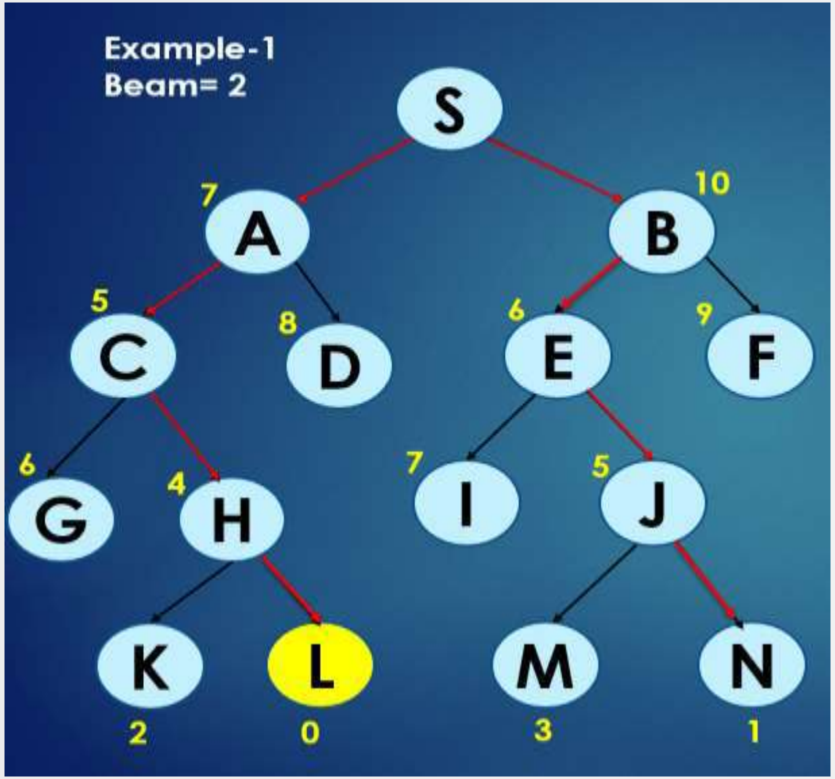
- Begin with **k randomly chosen initial states**.
- In each iteration, generate the successors of all current states.
- Out of all the successors, select the **top k** based on the evaluation function.
- Continue this process until one of the states reaches the goal or a stopping criterion is met.

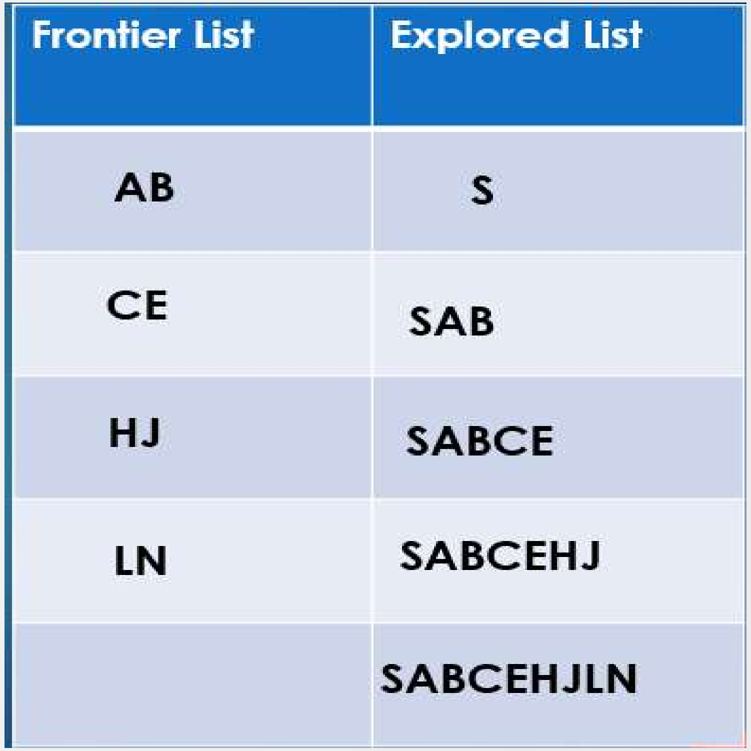
Local Beam Search - Algorithm



```
function BEAM_SEARCH(problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

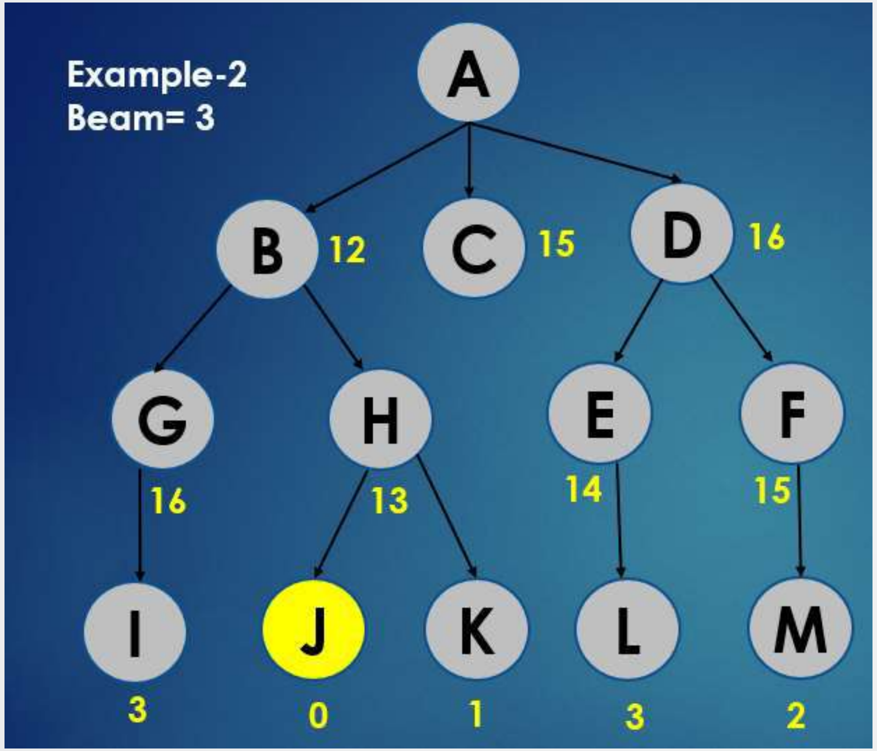
Local Beam Search – Example1





Frontier List	Explored List
AB	S
CE	SAB
HJ	SABCE
LN	SABCEHJ
	SABCEHJLN

Local Beam Search – Example2



Frontier List	Explore List
BCD	A
HEF	ABCD
JKM	ABCDHEF
	ABCDHEFJKM

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Searching with Partial Observations

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 10
Presentation Date – 02/11/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

Topic Learning Outcome

Utilize searching mechanism with Partial Observations to improve the robustness of AI systems.

Beyond Classical Search
(Searching with Partial Observations)

Searching with Partial Observations

- Search problems where ***agents do not have complete information about the environment.***
- **<u>Example</u>**: A robot navigating a maze without knowing the full layout.
- In this, ***the agent knows it's actions and goal state.***
- **<u>More Ex:</u>**
    - Maze game
    - Automatic Taxi
    - Hill climbing etc.

Challenges of Partial Observations

- **Uncertainty**: ***Limited data and missing information***. (leads to less reliable decisions). For instance, a robot navigating through a foggy landscape may not "see" all obstacles, making safe navigation difficult.
- **Ambiguity**: ***Multiple interpretations of available data***. For example, if a self-driving car detects an object at the roadside, it could be a stationary vehicle or a pedestrian; without further data, it must choose the safest interpretation.

Challenges of Partial Observations

- **Dynamic Environments**: ***The world might change unpredictably***. For instance, in a crowded area, people and vehicles constantly move, so a surveillance AI must account for such shifts without having a clear view of the entire scene.
- **Sensor Limitations**: ***Incomplete or noisy sensor inputs***. (leading to incomplete or inaccurate information). For example, a drone using a camera for navigation may suffer from low visibility in poor weather conditions, reducing the quality of the data it can process.

Examples of Partial Observation Scenarios

- **Robotics**: Navigating an unknown environment.
- **Games**: Playing chess without full visibility of the opponent's moves.
- **Medical Diagnosis**: Symptoms are incomplete, requiring intelligent guessing.

Key Techniques for Searching with Partial Observations

- **Belief States**: Representing the agent’s knowledge as a set of possible states.
- **Conditional Plans**: Creating plans that account for possible scenarios.
- **Probabilistic Reasoning**: Using probability to handle uncertainty.
- **Heuristic Search Methods**: Adapting heuristics for environments with limited visibility.

Belief State Representation

- A set of all possible states the agent might be in, based on its observations.
- **Example**: A robot in a grid world knowing only some obstacles.

Key Algorithms for Partial Observation Search

- **Heuristic Search**:
  Adapting traditional search algorithms (like A*) for environments where the full state isn't known.
- **POMDPs (Partially Observable Markov Decision Processes)**
  - Combines Markov Decision Processes with incomplete information.
  - Accounts for uncertainty in both states and actions.
- **Information-Gathering Search**: Focuses on acquiring more information before committing to decisions.

Applications

- **Robotics**: Autonomous navigation with incomplete maps.
- **AI in Games**: Handling uncertainty in adversarial environments.
- **Healthcare**: Decision-making with incomplete patient data.

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Backtracking Search

Presenter’s Name – Ms. Bidyutlata Sahoo
Presenter’s ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 11
Presentation Date – 02/11/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

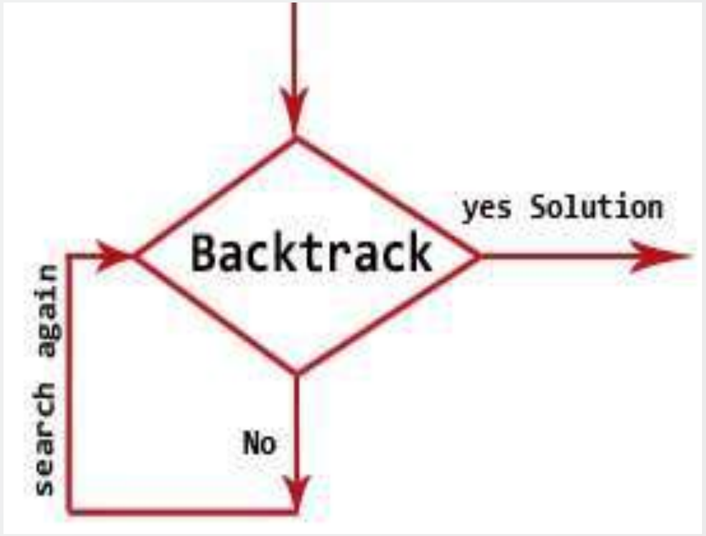
Topic Learning Outcome

Make use of backtracking search to prune infeasible paths and handle complex decision making for solving a wide range of problems.

Backtracking Search

Backtracking Search

- ***Backtracking is nothing but the modified process of the brute force approach, where the technique systematically searches for a solution to a problem among all available options. It does so by assuming that the solutions are represented by vectors (v1, ..., in) of values and by traversing through the domains of the vectors until the solutions is found.***



Backtracking Search

- Backtracking is a modified depth-first search of a tree.
- It is the procedure whereby, after determining that a node can lead to nothing but dead nodes, we go back (“backtrack”) to the node’s parent and proceed with the search on the next child.

Backtracking Search - Algorithm

- Based on depth-first recursive search
- Approach
  1. Tests whether solution has been found
  2. If found solution, return it
  3. Else for each choice that can be made
     a) Make that choice
     b) Recursive
     c) If recursion returns a solution, return it
  4. If no choices remain, return failure
- Some times called “search tree”

Backtracking Search - Advantages

- Comparison with the Dynamic Programming, Backtracking Approach is ***more effective in some cases.***
- Backtracking Algorithm ***is the best option for solving tactical problem.***
- Also Backtracking is ***effective for constraint satisfaction problem***.
- In greedy Algorithm, getting the Global Optimal Solution is a long procedure and depends on user statements but in Backtracking It Can Easily getable.

Backtracking Search - Advantages

- Backtracking technique is ***simple to implement and easy to code.***
- Different states are stored into stack so that the data or Info can be usable anytime.
- The ***accuracy is granted.***

Backtracking Search - Disadvantages

- Backtracking Approach ***is not efficient for solving strategic Problem.***
- The ***overall runtime*** of Backtracking Algorithm ***is normally slow.***
- ***Need Large amount of memory space*** for storing different state function in the stack for big problem.
- The Basic Approach ***Detects the conflicts too late.***

Backtracking Search - Applications

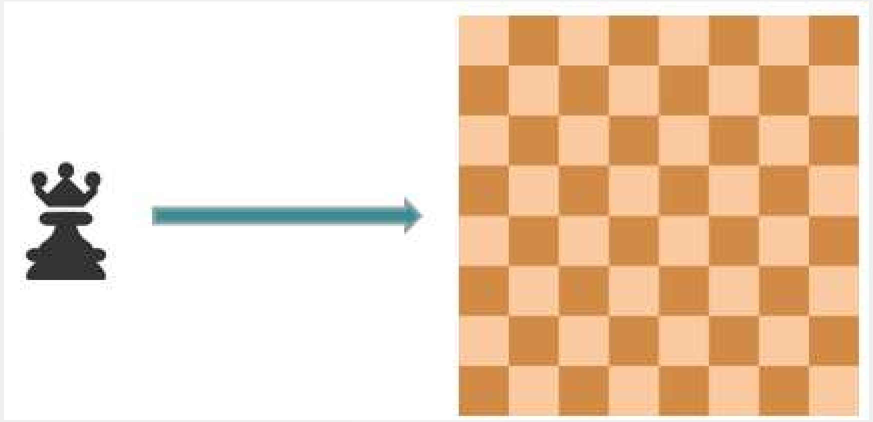
- Optimization and tactical problems
- Constraints Satisfaction Problem
- Electrical Engineering
- Robotics
- Artificial Intelligence
- Genetic and bioinformatics Algorithm
- Materials Engineering
- Network Communication
- Solving puzzles and path

Backtracking Search - Example

- ***N- Queens Problem***
- ***Hamiltonian Cycle***
- ***Sudoku Puzzle***
- ***Maze Generation***

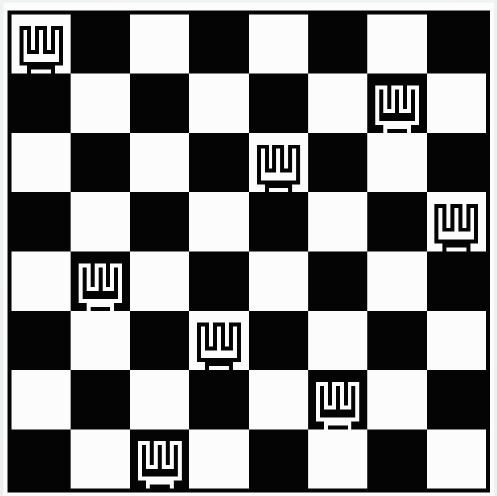
Example1: 8 Queens Problem

- The 8-queens problem is a classical combinatorial problem in which it is ***required to place eight queens on an 8 x 8 chessboard so no two queens can attack each other.***
- A ***queen can attack another queen*** if it exists ***in the same row, column or diagonal as the queen.***



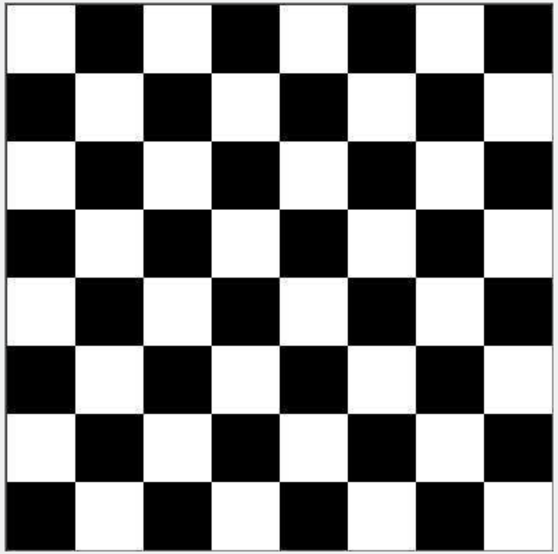
Example1: 8 Queens Problem (Contd..)

- This problem can be solved by trying to place the first queen, then the second queen so that it cannot attack the first, and then the third so that it is not conflicting with previously placed queens.



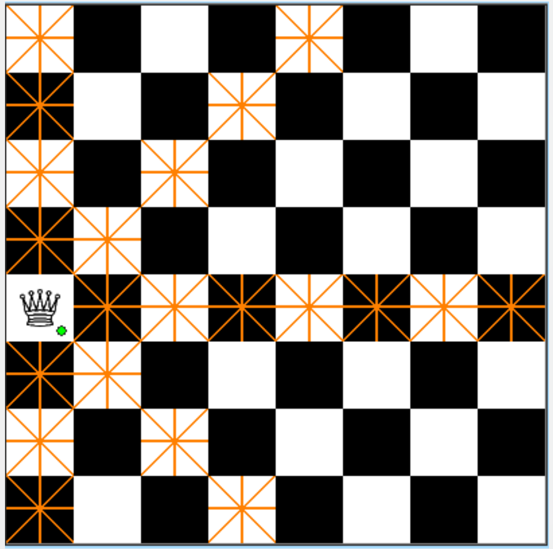
Example1: 8 Queens Problem (Contd..)

- It is an ***empty 8 x 8 chess board***. We have to place the queens in this board.



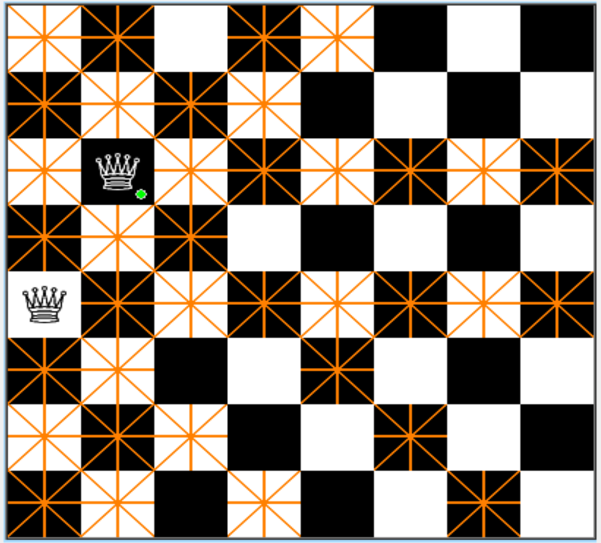
Example1: 8 Queens Problem (Contd..)

- We have ***placed the first queen on the chess board.***



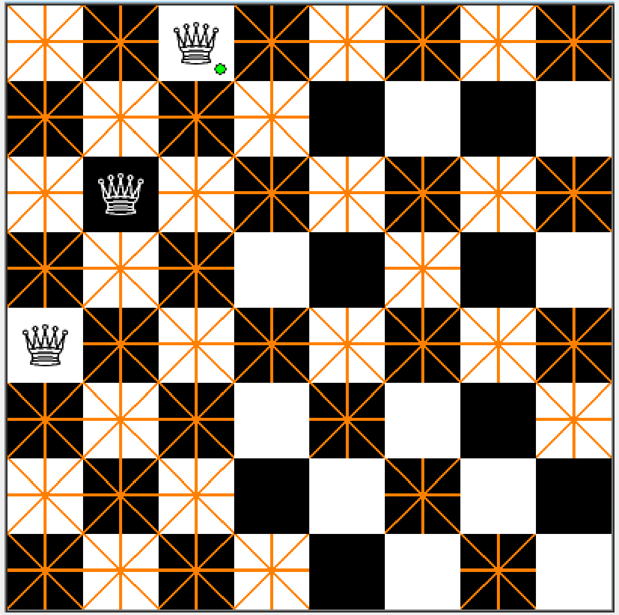
Example1: 8 Queens Problem (Contd..)

- Then we have ***placed the second queen on the board.***
- The darken place should not have the queens because they are horizontal, vertical, diagonal to the placed queens.



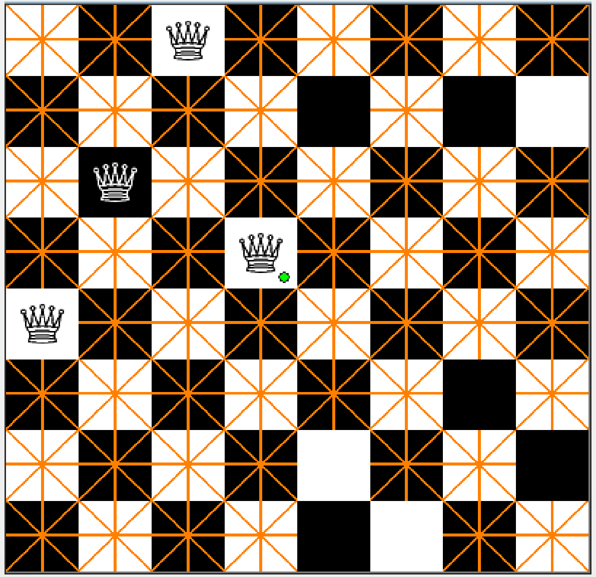
Example1: 8 Queens Problem (Contd..)

- We have ***placed the third queen on board.***



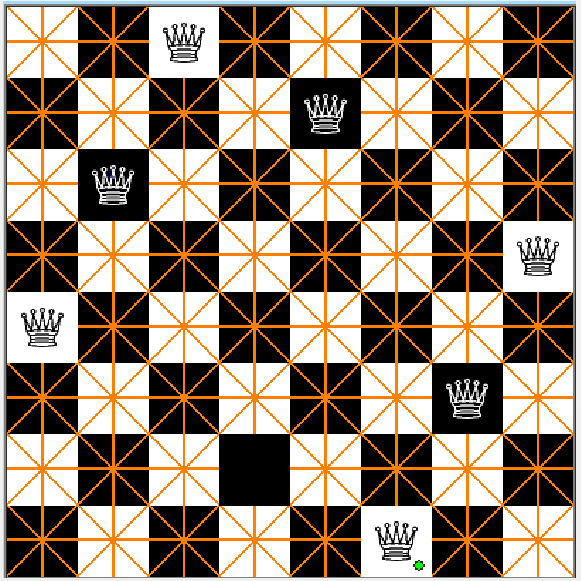
Example1: 8 Queens Problem (Contd..)

- We have ***placed the 4th queen on the board.***
- We have ***placed that in the wrong spot, so we backtrack and change the place of that one.***



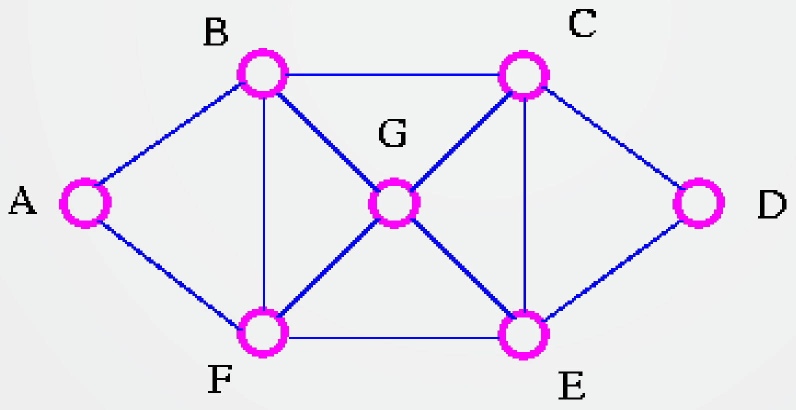
Example1: 8 Queens Problem (Contd..)

- In this way, we have to continue the process until our goal is reached i.e., we must place 8 queens on the board.



Example2: Hamiltonian Cycle

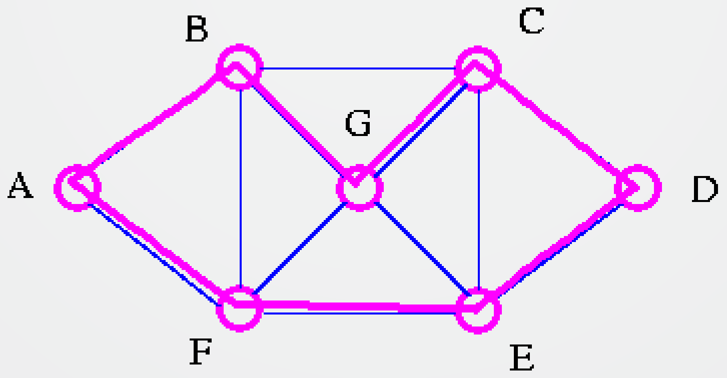
- Hamiltonian Cycle is a graph theory problem where the ***graph cycle through a graph can visit each node only once.***
- The puzzle was first devised by ***Sir William Rowan Hamilton*** and the Problem is named after Him.



Condition: The Cycle Started with a Starting Node, and visit all the Nodes in the Graph. (Not necessary to visit in sequential Order and not creating edge that is not given) and Stop at The Starting Point / Node.

Example2: Hamiltonian Cycle (Contd..)

- The algorithm first check the starting node, if there is any edge to the next node. If yes, then the algorithm will check that node for the edge to the next node. It will also check if any node is visited twice by the previous node. If there is any then the algorithm will ignore one and choose the optimal one for the solution.
- The Important thing is the tour must finish at the starting point.



References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE

INSTITUTE OF AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Performance of Search Algorithms

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 12
Presentation Date – 02/11/2024

Course Outcome

At the end of the course, students should be able to:

CO3: Interpret uninformed and informed search strategies, and select the appropriate approach for different AI problems.

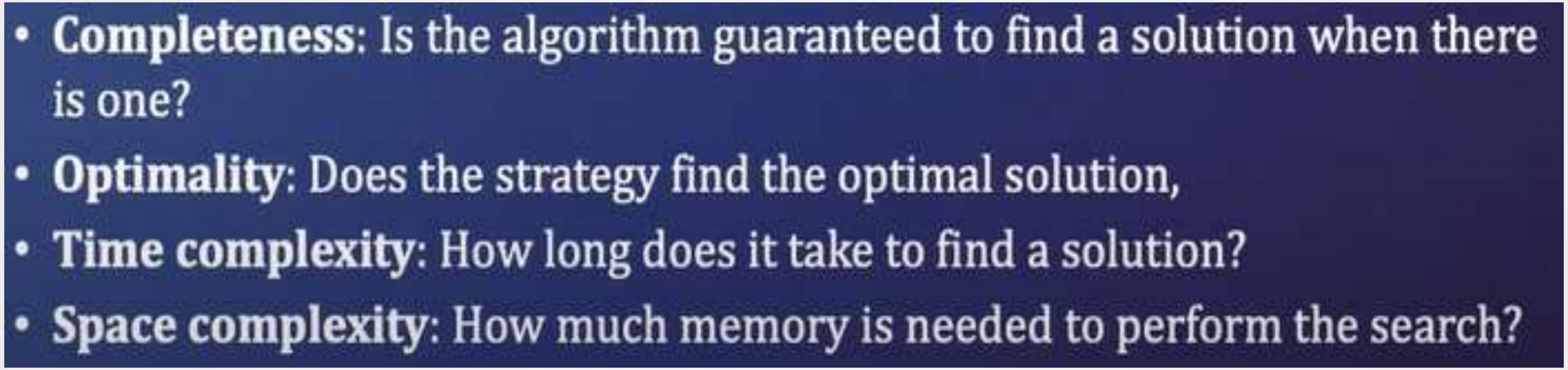
Topic Learning Outcome

Understand the performance of search algorithms in terms of efficiency and effectiveness, analysing their characteristics, comparing strategies, and applying performance metrics to real-world problems.

Performance of Search Algorithms

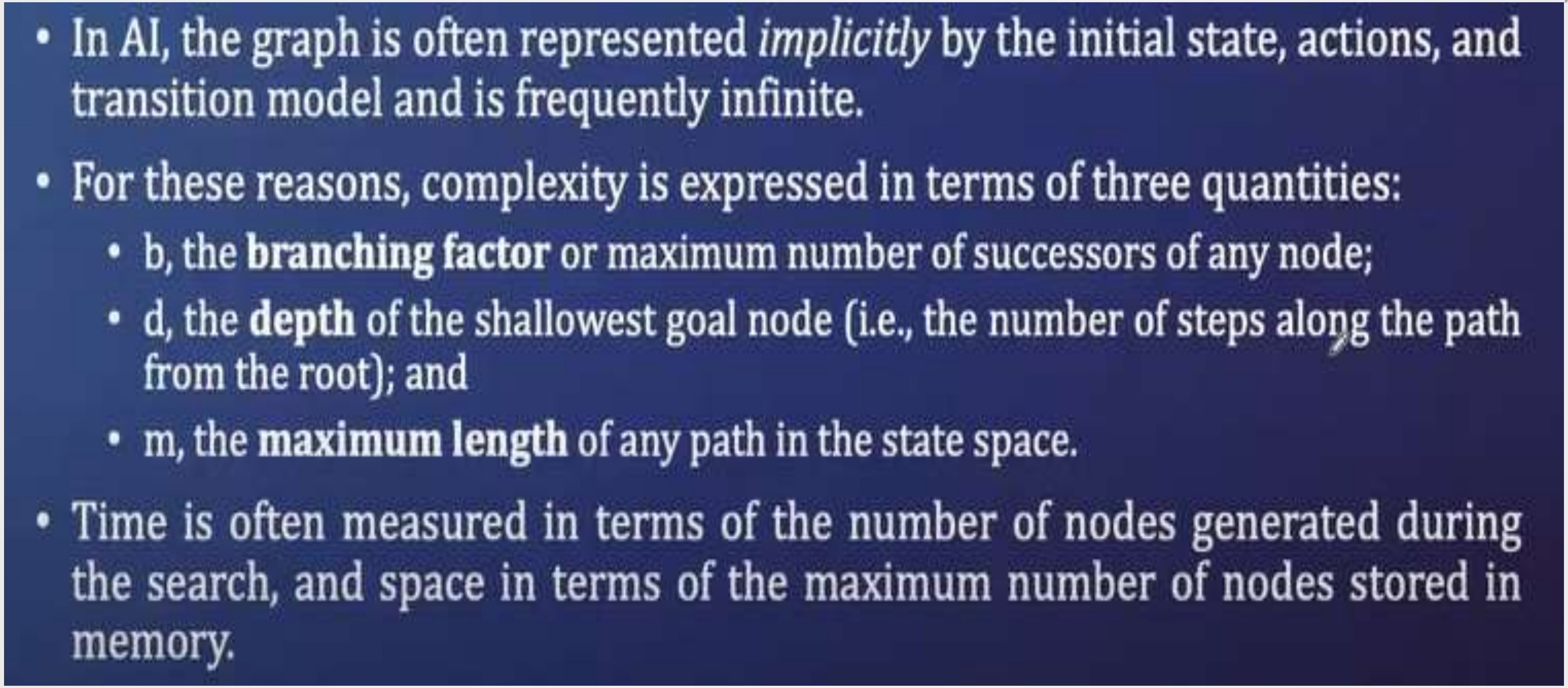
Performance of Search Algorithms

Before we get into the design of specific search algorithms, we need to consider the criteria that might be used to choose among them. We can **evaluate the algorithm performance in four ways**:



- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution,
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?

Performance of Search Algorithms



- In AI, the graph is often represented *implicitly* by the initial state, actions, and transition model and is frequently infinite.
- For these reasons, complexity is expressed in terms of three quantities:
  - b, the **branching factor** or maximum number of successors of any node;
  - d, the **depth** of the shallowest goal node (i.e., the number of steps along the path from the root); and
  - m, the **maximum length** of any path in the state space.
- Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory.

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE

INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Knowledge Representation and Propositional Logic

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 13
Presentation Date – 05/11/2024

Course Outcome

At the end of the course, students should be able to:

CO4: Demonstrate computable functions and predicates in computational system to construct logical expressions.

Topic Learning Outcome

Demonstrate an understanding of propositional logic by representing and manipulating logical expressions, applying truth tables, and evaluating the validity of arguments in AI systems.

Knowledge Representation

Knowledge

- *Facts, information, and skills acquired through experience or education*; the theoretical or practical understanding of a subject.
- ***Knowledge = information + rules***
- ***Knowledge*** is the information about a domain that can be used to solve problems in that domain. But the ability to use this knowledge is called as ***intelligence*****.**
- ***To solve any problem requires knowledge, and this knowledge must be represented in the computer.***

Knowledge Representation

- If we want to make machines interpret and solve problem as humans, we need to make machines intelligent.
- So, ***knowledge*** is the most important factor to make the machine intelligent.

Knowledge Base

- A ***representation scheme*** is the form of the knowledge that is used by an agent (knowledge agent).
- A ***Knowledge base*** is the representation of all the knowledge that is stored by an agent.
- ***Knowledge base is required*** for updating knowledge for an agent to learn with experiences and take action as per the knowledge.
- Ex: Chatbot

Knowledge Based Agent (KBA)

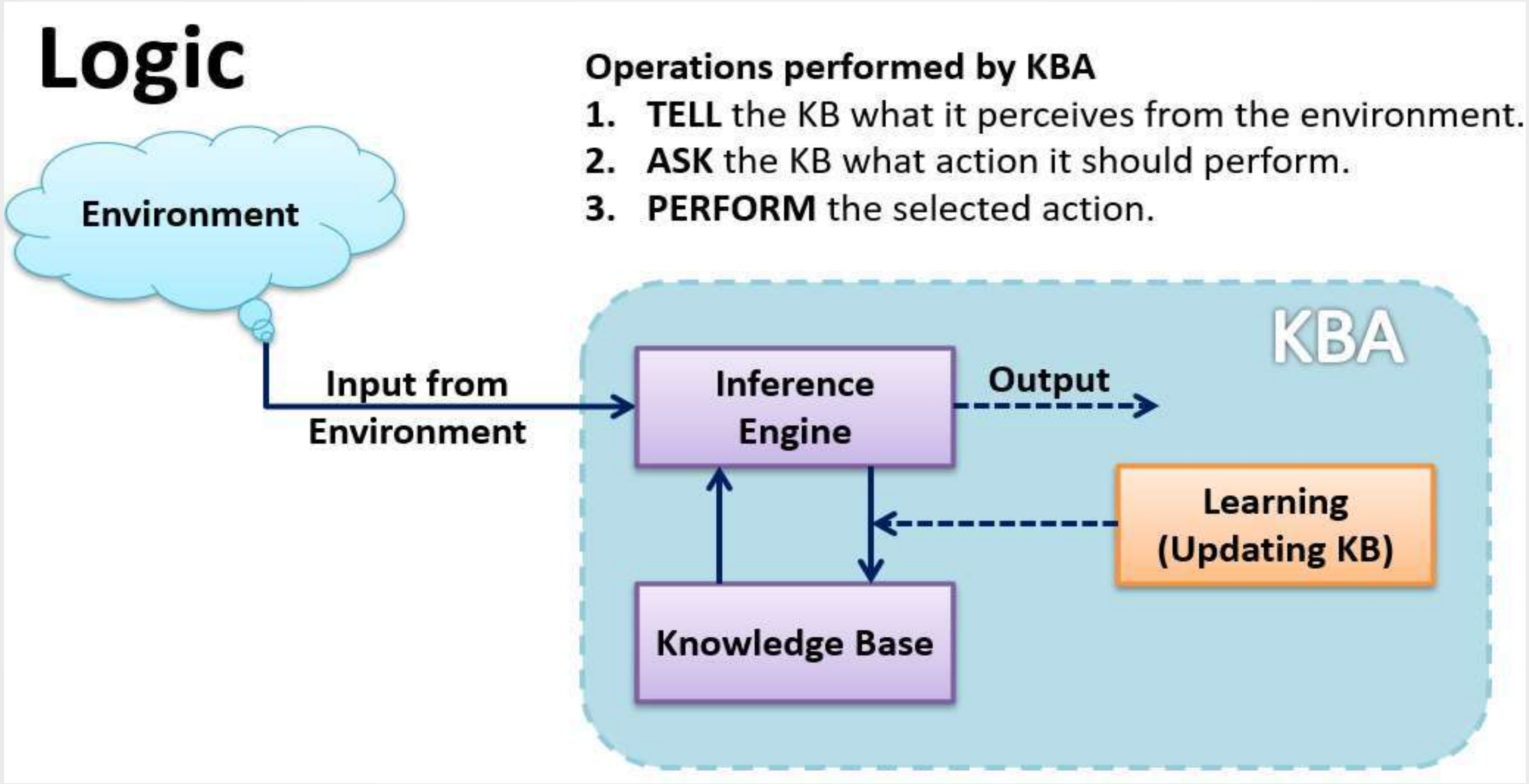
- It consists of 2 things:

  Knowledge Base and an Inference System

- ***Knowledge Base*** is the set of sentences. (Here sentence is not identical to sentence in English) . These sentences are expressed in a language which is called as knowledge representation language. Basically, Knowledge base stores all facts about the world.

- ***Inference System*** derives new sentences from the input and KB. It allows us to add new sentence and applies logical rules to KB to deduce new information. It generates new facts so that agent can update its KB. So, it depends upon the representation of KB.

Knowledge Based Agent (KBA)


Logic

Operations performed by KBA

1. **TELL** the KB what it perceives from the environment.
2. **ASK** the KB what action it should perform.
3. **PERFORM** the selected action.

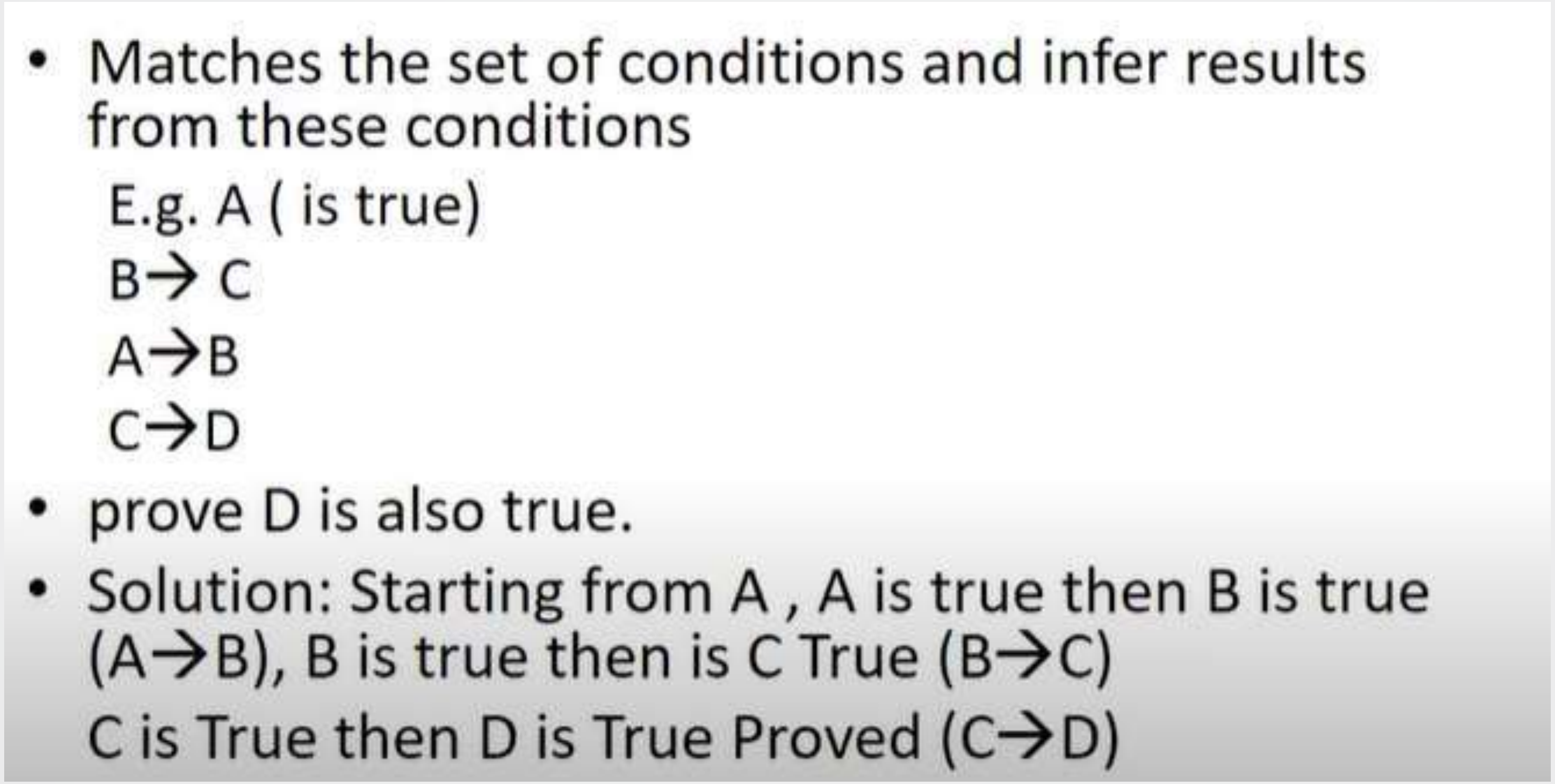
Knowledge Representation Techniques

- Propositional Logic
- Predicate Logic or First Order Logic
- Semantic Net
- Scripts
- Frames
- Conceptual Dependency

Inference Techniques

- Forward Chaining
- Backward Chaining
- Proof by Resolution

Forward Chaining



- Matches the set of conditions and infer results from these conditions

  E.g. A (is true)

  B→ C

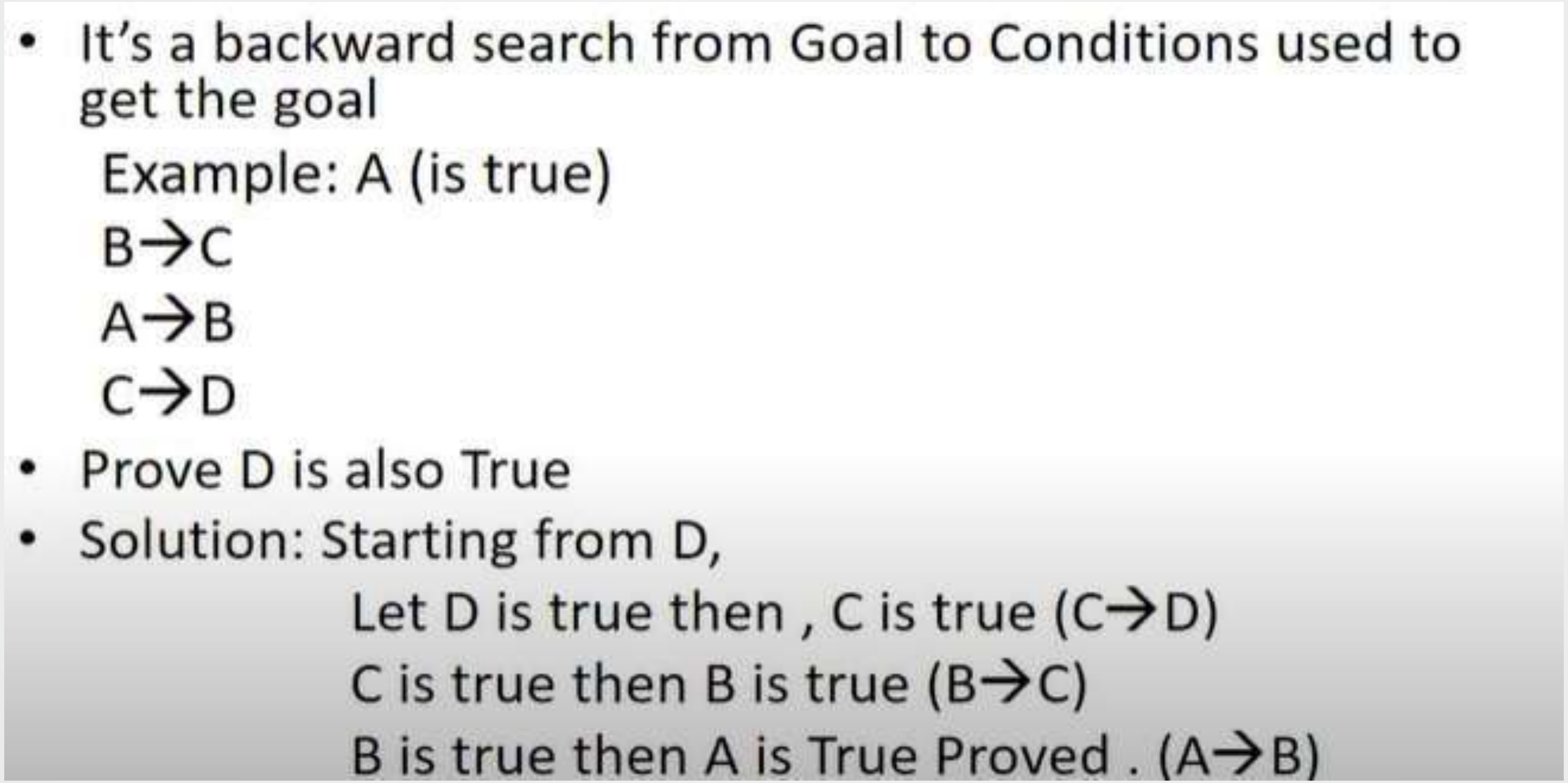
  A→B

  C→D

- prove D is also true.
- Solution: Starting from A , A is true then B is true (A→B), B is true then is C True (B→C)

  C is True then D is True Proved (C→D)

Backward Chaining



- It’s a backward search from Goal to Conditions used to get the goal

  Example: A (is true)

  B→C

  A→B

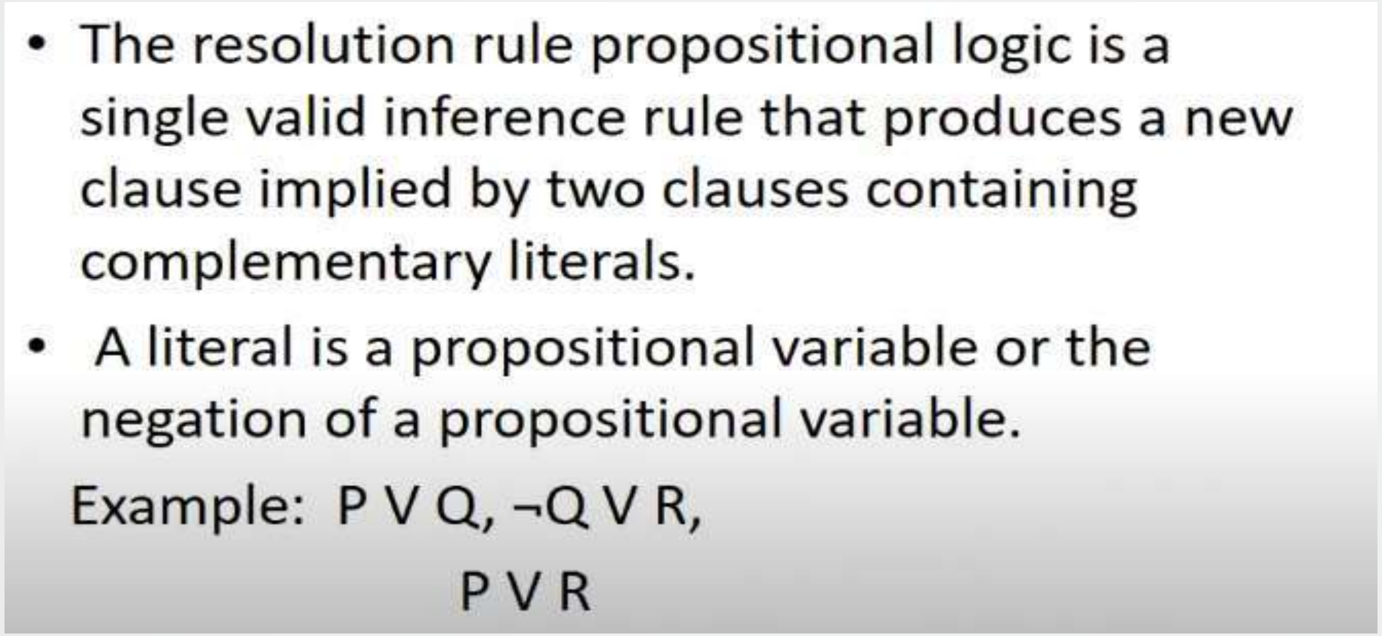
  C→D
- Prove D is also True
- Solution: Starting from D,

  Let D is true then , C is true (C→D)

  C is true then B is true (B→C)

  B is true then A is True Proved . (A→B)

Proof by Resolution



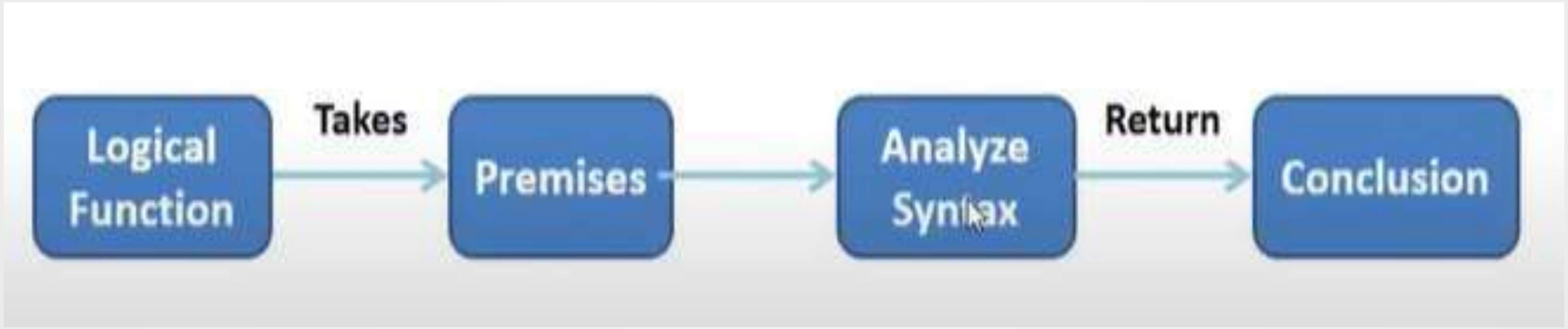
- The resolution rule propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals.
- A literal is a propositional variable or the negation of a propositional variable.

Example: $P \lor Q$, $\neg Q \lor R$,

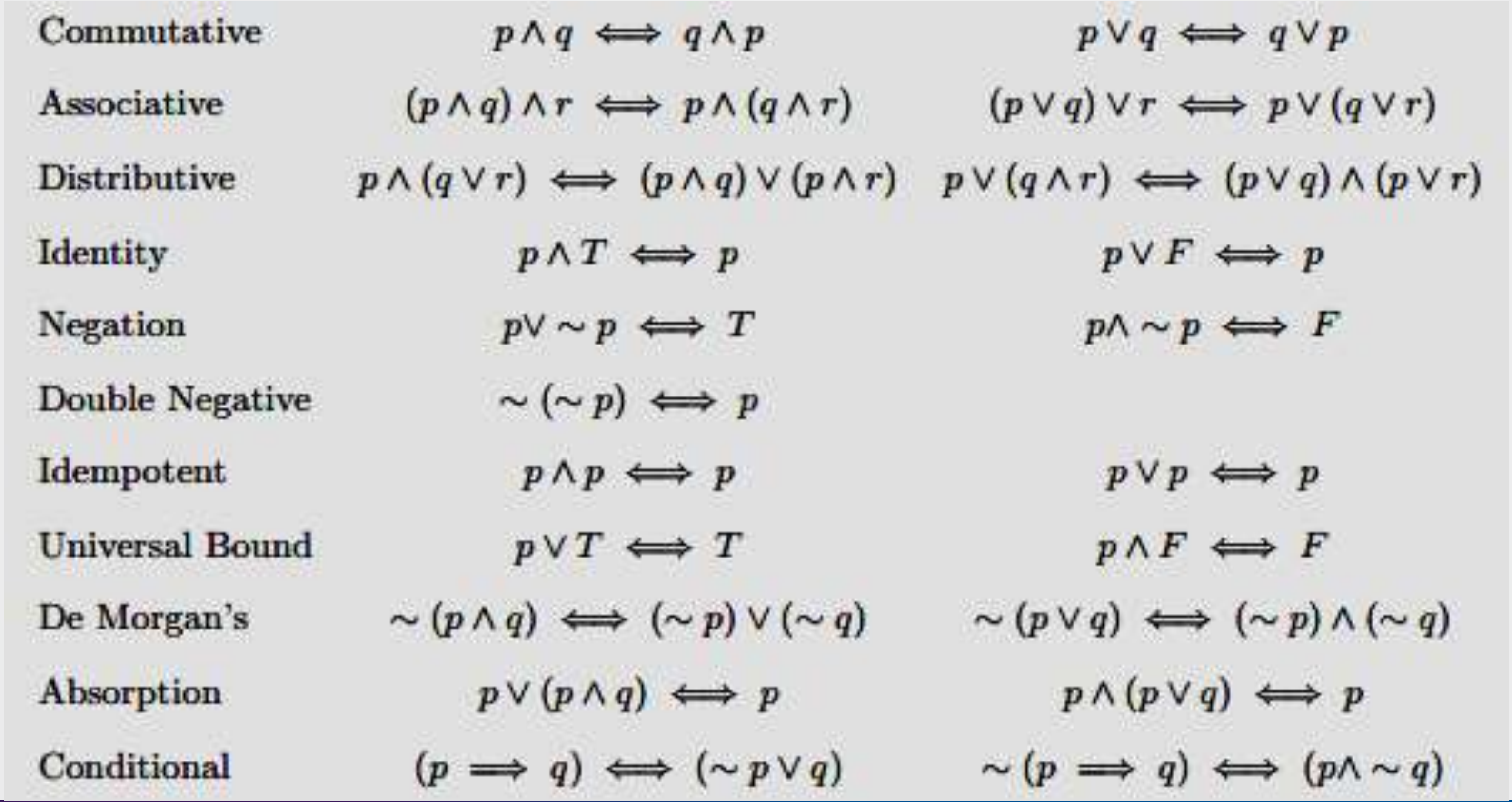
$$P \lor R$$

Inference Rules

- A chain of conclusions that leads to the desired goal.
- Logical function takes premises i.e., checks P and Q and analyzes the syntax and returns conclusions.

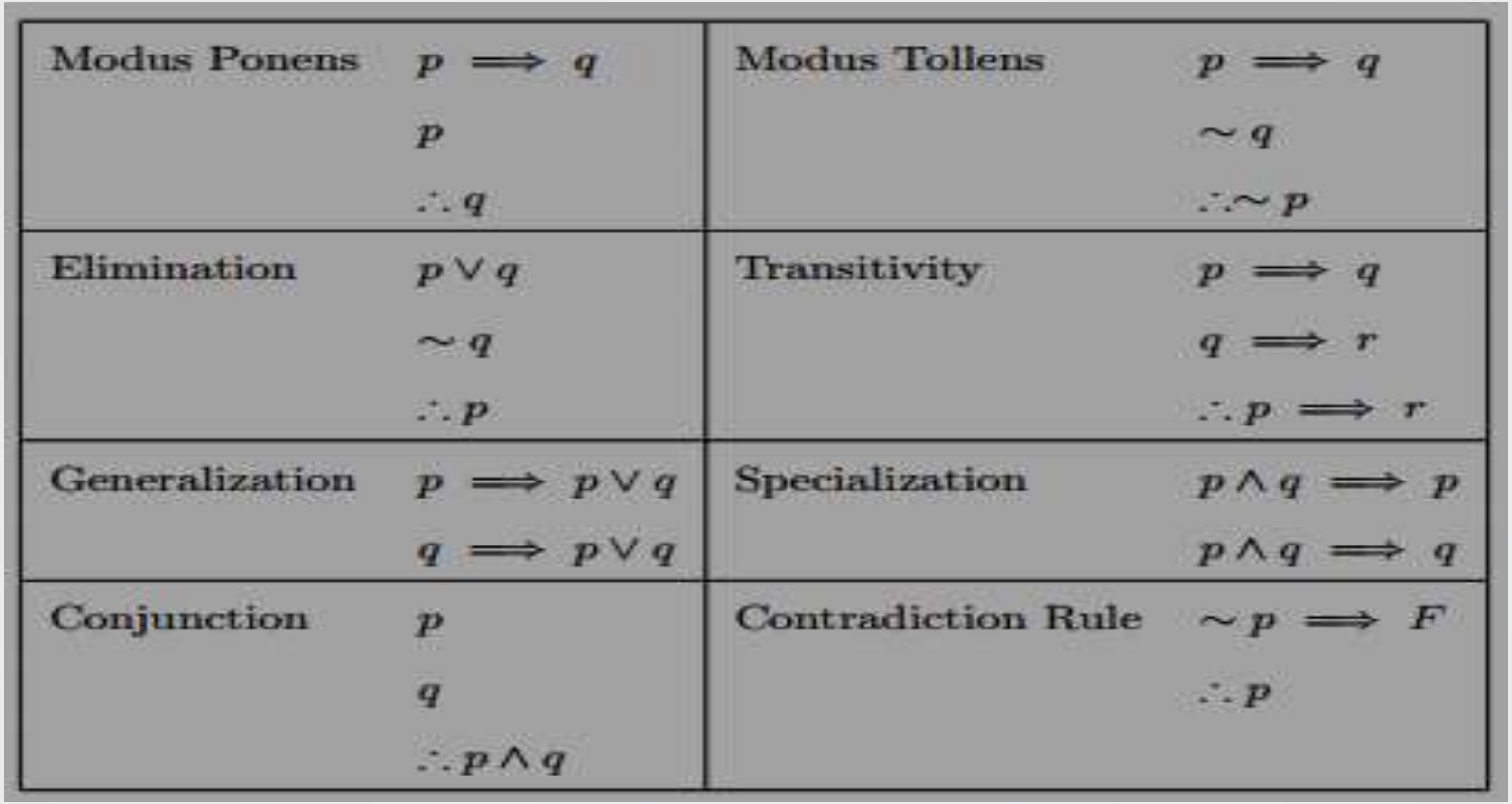


Some Inference Rules



Commutative	$p \wedge q \iff q \wedge p$	$p \vee q \iff q \vee p$
Associative	$(p \wedge q) \wedge r \iff p \wedge (q \wedge r)$	$(p \vee q) \vee r \iff p \vee (q \vee r)$
Distributive	$p \wedge (q \vee r) \iff (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) \iff (p \vee q) \wedge (p \vee r)$
Identity	$p \wedge T \iff p$	$p \vee F \iff p$
Negation	$p \vee \sim p \iff T$	$p \wedge \sim p \iff F$
Double Negative	$\sim (\sim p) \iff p$	
Idempotent	$p \wedge p \iff p$	$p \vee p \iff p$
Universal Bound	$p \vee T \iff T$	$p \wedge F \iff F$
De Morgan's	$\sim (p \wedge q) \iff (\sim p) \vee (\sim q)$	$\sim (p \vee q) \iff (\sim p) \wedge (\sim q)$
Absorption	$p \vee (p \wedge q) \iff p$	$p \wedge (p \vee q) \iff p$
Conditional	$(p \implies q) \iff (\sim p \vee q)$	$\sim (p \implies q) \iff (p \wedge \sim q)$

Some Inference Rules



Modus Ponens	$p \implies q$	Modus Tollens	$p \implies q$
	p		$\sim q$
	$\therefore q$		$\therefore \sim p$
Elimination	$p \vee q$	Transitivity	$p \implies q$
	$\sim q$		$q \implies r$
	$\therefore p$		$\therefore p \implies r$
Generalization	$p \implies p \vee q$	Specialization	$p \wedge q \implies p$
	$q \implies p \vee q$		$p \wedge q \implies q$
Conjunction	p	Contradiction Rule	$\sim p \implies F$
	q		$\therefore p$
	$\therefore p \wedge q$		

Propositional Logic

Propositional Logic

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

  a) It is Sunday.

  b) The Sun rises from West (False proposition)

  c) 3+3= 7(False proposition)

  d) 5 is a prime number.

Facts about Propositional Logic

- Propositional logic is ***also called*** ***Boolean logic*** as it works on 0 and 1.
- Propositions ***can be either true or false, but it cannot be both.***
- 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 or Unsatisfiable .***

Facts about Propositional Logic

- ***A statement is satisfiable*** if there is some interpretation for which it is true.
- ***Statements which are questions, commands, or opinions are not propositions*** such as "Where is Rohini", "How are you", "What is your name", are not propositions.

Syntax of Propositional Logic

- There are ***two types of Propositions***:

1. Atomic Propositions
2. Compound proposition

Atomic Propositions

- Atomic propositions *are the simple propositions*. It *consists of a single proposition symbol.* These are the sentences which must be either true or false.
- **Example:**

  a) 2+2 is 4, it is an atomic proposition as it is a true fact.

  b) "The Sun is cold" is also a proposition as it is a false fact.

Compound Propositions

- Compound propositions *are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.*
- **Example:**

  a) "It is raining today and street is wet."

  b) "Ankit is a doctor and his clinic is in Mumbai."

Logical Connectives

Word	Symbol	Example	Meaning of Example	Terminus Techniques
not	¬	¬A	not A	negation
and	∧	A ∧ B	A and B	conjunction
or	∨	A ∨ B	A or B	disjunction
implies	→	A → B	A implies B	implication
if and only if	↔	A ↔ B	A if and only if B	biconditional

Logical Connectives - Examples

- **Negation**:
  - A sentence such as ¬ P is called ***negation*** of P. A literal can be either Positive literal or negative literal.
  - Example: “ Vivin can play tennis.”

    Let P= Vivin can paly tennis

    so, we can write it as P.
  - Example: “ Vivin can’t play tennis.”

    Let P= Vivin can paly tennis

    Vivin can’t paly tennis can be represented as ¬ P.

Logical Connectives - Examples

- **Conjunction**:
  - A sentence which has ∧ connective such as, P ∧ Q is called ***conjunction.***
  - Example: Rohan is intelligent and hardworking.
    Let P= Rohan is intelligent, Q= Rohan is hardworking.

    so, we can write it as P∧ Q.

Logical Connectives - Examples

- **Disjunction:**
  - A sentence which has ∨ connective, such as P ∨ Q. is called ***disjunction***, where P and Q are the propositions.
  - Example: "Ritika is a doctor or Engineer",
    Let P= Ritika is Doctor. Q= Ritika is Engineer,

    so, we can write it as P ∨ Q

Logical Connectives - Examples

- **Implication:**
  - A sentence such as $P \rightarrow Q$, is called an implication. ***Implications*** are also known as ***if-then rules***.
  - Example: If it is raining, then the street is wet.

    Let P= It is raining, and Q= Street is wet,

    So, it is represented as $P \rightarrow Q$

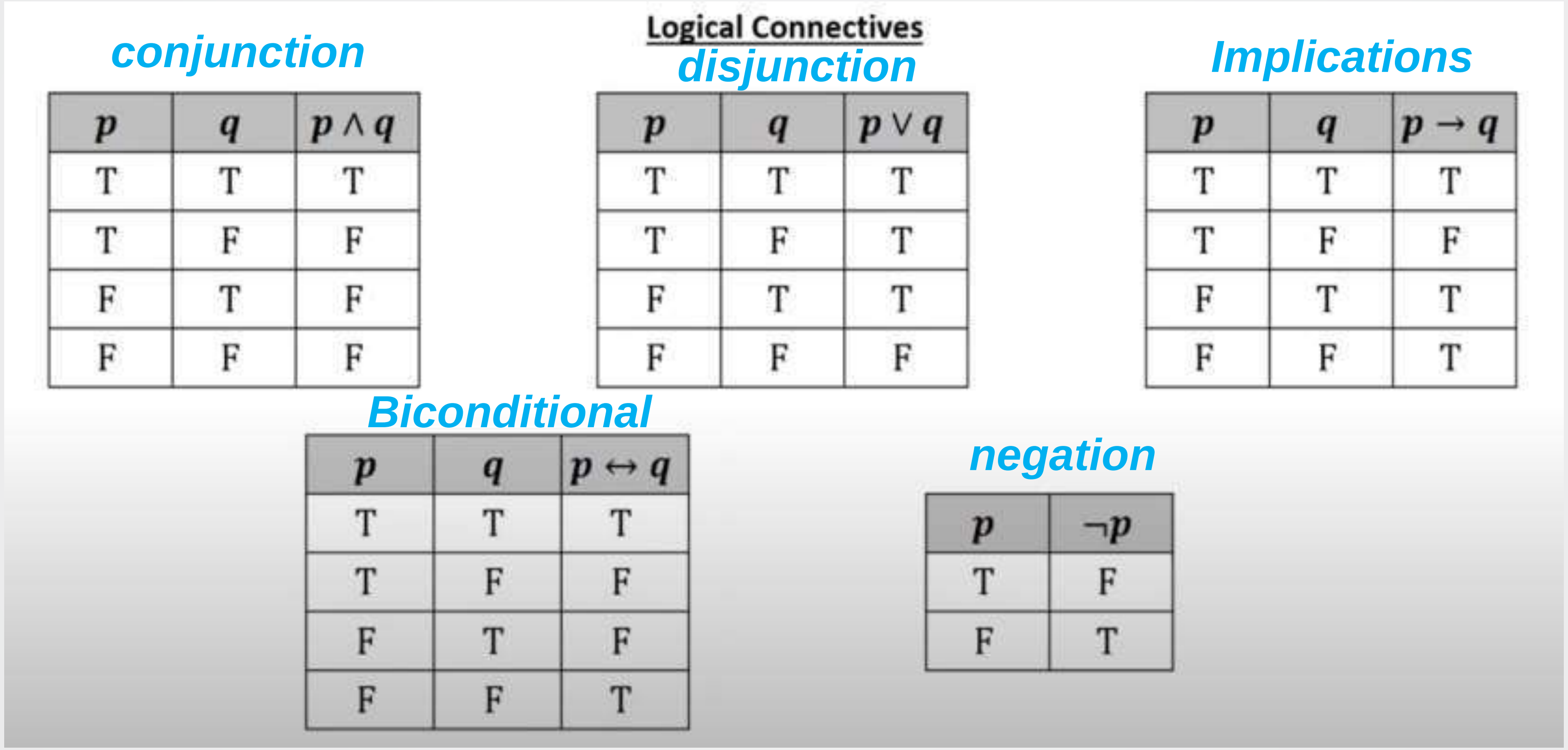
Logical Connectives - Examples

- **Biconditional:**
  - A sentence such as P⇔ Q is a ***Biconditional*** sentence.
  - Example: If I am breathing, then I am alive.
    Let P= I am breathing, Q= I am alive,

    it can be represented as P ⇔ Q
  - Example: If I have 1000 Rupees then only, I will go for shopping. Let P= I have 1000 Rupees , Q= I will go for shopping,

    it can be represented as P ⇔ Q i.e., I will go for shopping if and only if I have 1000 Rupees.

Truth Table



Logical Connectives

conjunction

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

disjunction

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

Implications

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

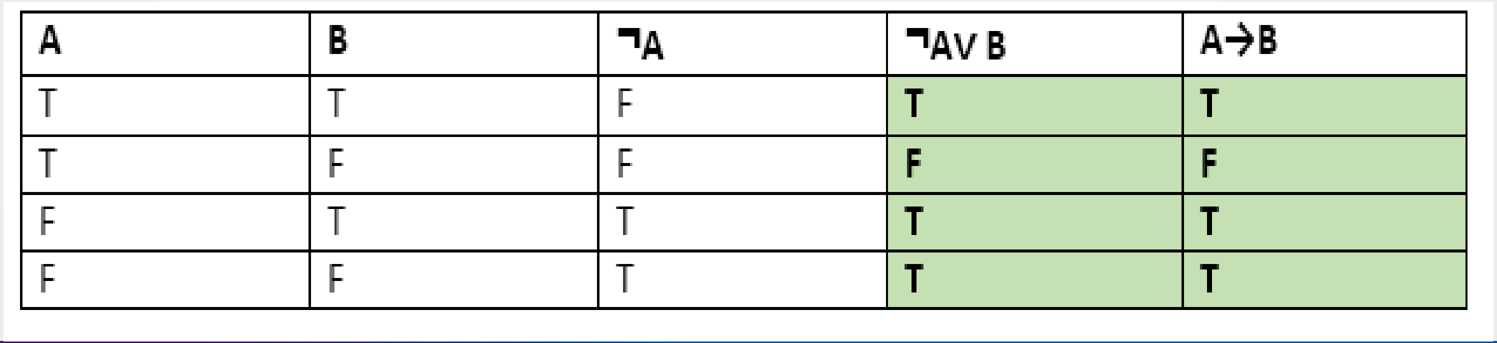
negation

p	$\neg p$
T	F
F	T

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⇔B. In below truth table we can see that column for ¬A∨ B and A→B are identical hence ¬A∨ B ⇔ A→B.



A	B	¬A	¬A∨ B	A→B
T	T	F	**T**	**T**
T	F	F	**F**	**F**
F	T	T	**T**	**T**
F	F	T	**T**	**T**

Inference Rules in Propositional Logic

1. Idempotent rule:

$$P \wedge P \implies P$$

$$P \vee P \implies P$$

2. Commutative rule:

$$P \wedge Q \implies Q \wedge P$$

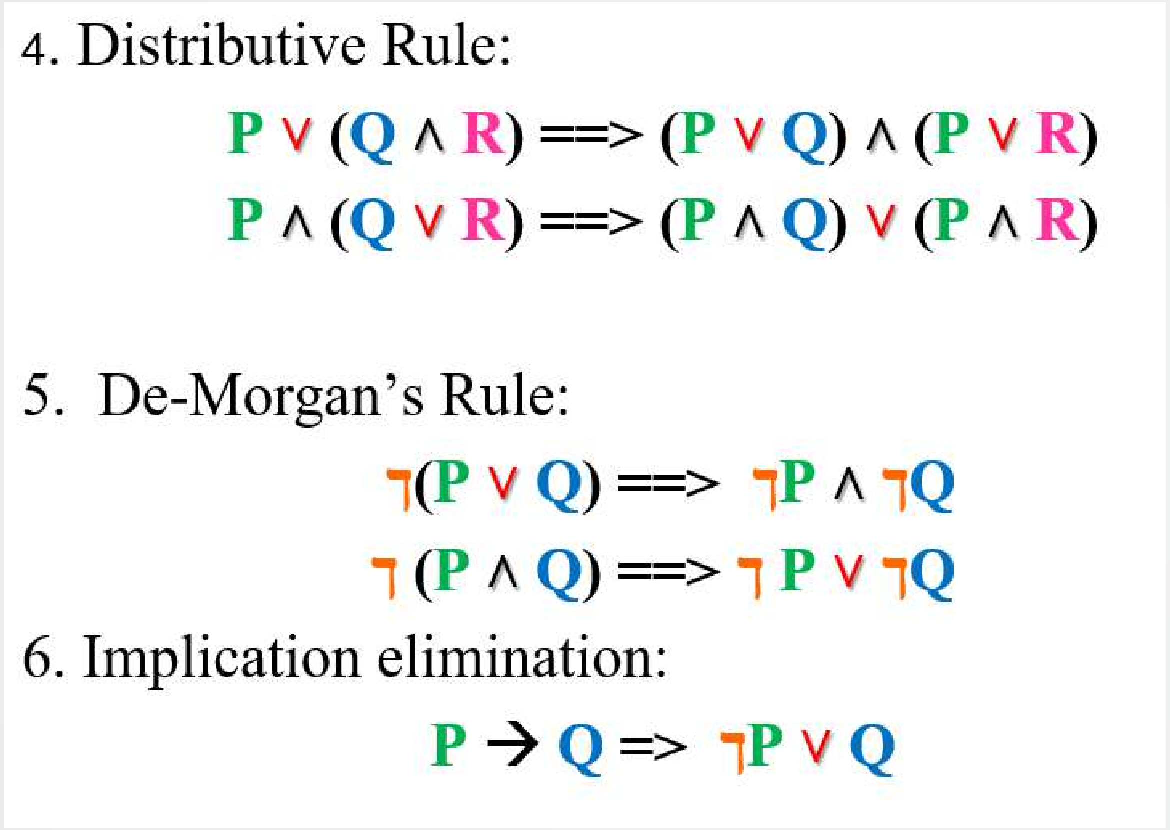
$$P \vee Q \implies Q \vee P$$

3. Associative rule:

$$P \wedge (Q \wedge R) \implies (P \wedge Q) \wedge R$$

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

Inference Rules in Propositional Logic



4. Distributive Rule:

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

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

5. De-Morgan's Rule:

$$\neg(P \vee Q) ==> \neg P \wedge \neg Q$$

$$\neg (P \wedge Q) ==> \neg P \vee \neg Q$$

6. Implication elimination:

$$P \rightarrow Q => \neg P \vee Q$$

Inference Rules in Propositional Logic

7. Bidirectional Implication elimination:

$$(P \Leftrightarrow Q) \Longrightarrow (P \rightarrow Q) \land (Q \rightarrow P)$$

8. Contrapositive rule:

$$P \rightarrow Q \Rightarrow \neg P \rightarrow \neg Q$$

9. Double Negation rule:

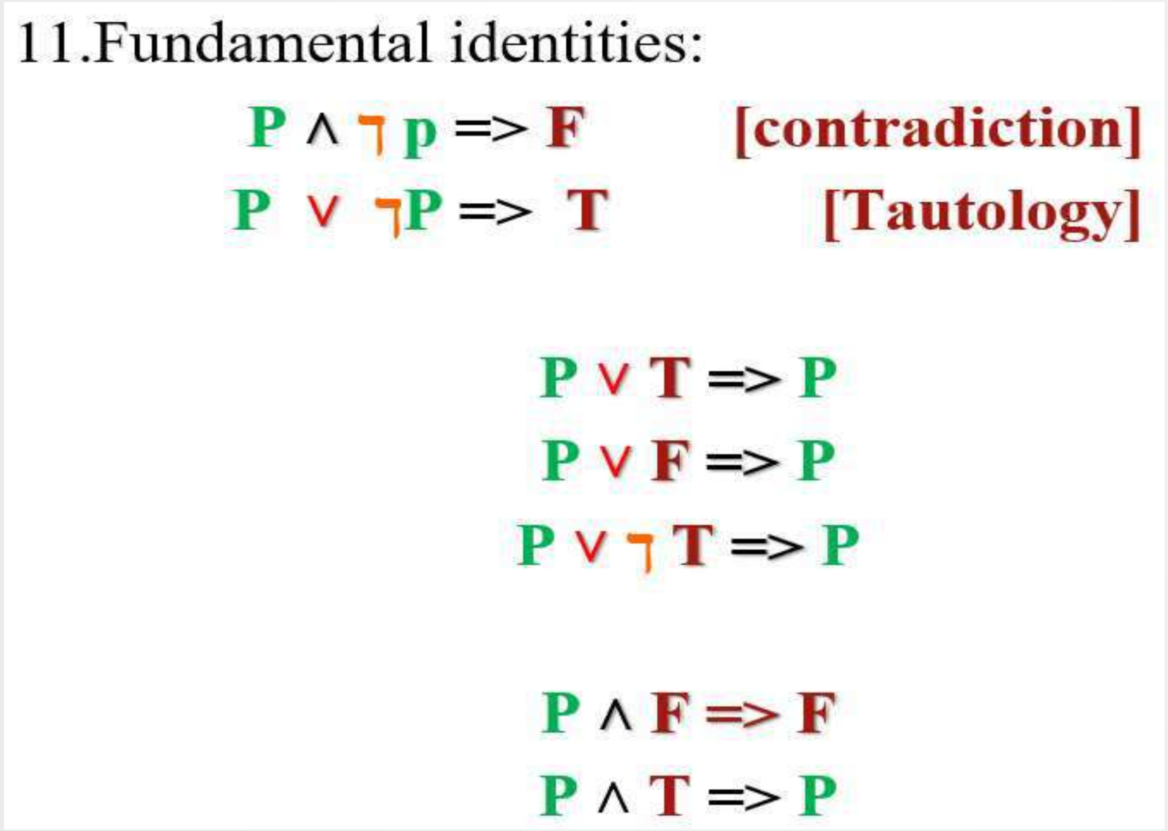
$$\neg(\neg P) \Rightarrow P$$

10. Absorption Rule:

$$\underline{P} \lor (\underline{P} \land Q) \Rightarrow \underline{P}$$

$$\underline{P} \land (\underline{P} \lor Q) \Rightarrow \underline{P}$$

Inference Rules in Propositional Logic



11.Fundamental identities:

$P \land \neg p \Rightarrow F$ [contradiction]

$P \lor \neg P \Rightarrow T$ [Tautology]

$P \lor T \Rightarrow P$

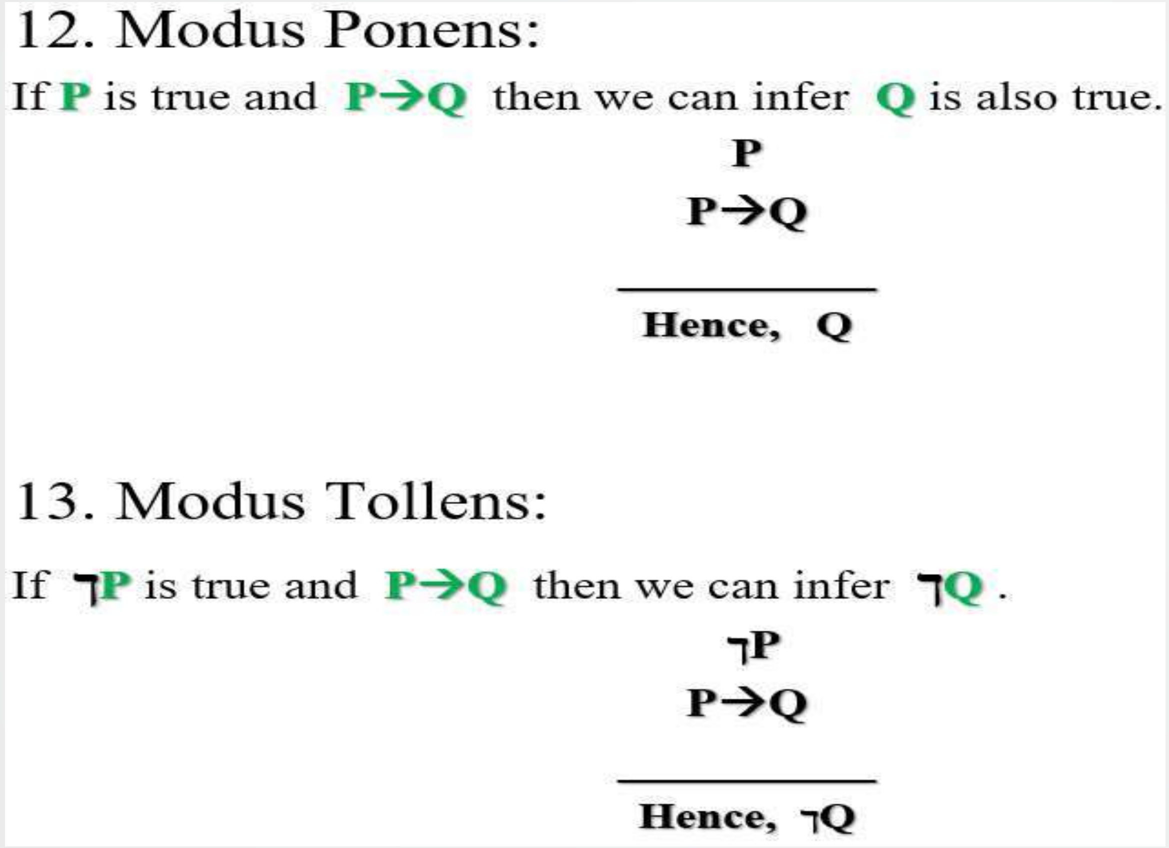
$P \lor F \Rightarrow P$

$P \lor \neg T \Rightarrow P$

$P \land F \Rightarrow F$

$P \land T \Rightarrow P$

Inference Rules in Propositional Logic



12. Modus Ponens:

If **P** is true and **P→Q** then we can infer **Q** is also true.

$$\frac{\begin{array}{c} P \\ P \to Q \end{array}}{\text{Hence, } Q}$$

13. Modus Tollens:

If ¬**P** is true and **P→Q** then we can infer ¬**Q** .

$$\frac{\begin{array}{c} \neg P \\ P \to Q \end{array}}{\text{Hence, } \neg Q}$$

Inference Rules in Propositional Logic

14. Chain rule:

If $p \rightarrow q$ and $q \rightarrow r$ then $p \rightarrow r$

15. Disjunctive Syllogism:

If **¬p** and **p∨q** we can infer **q** is true.

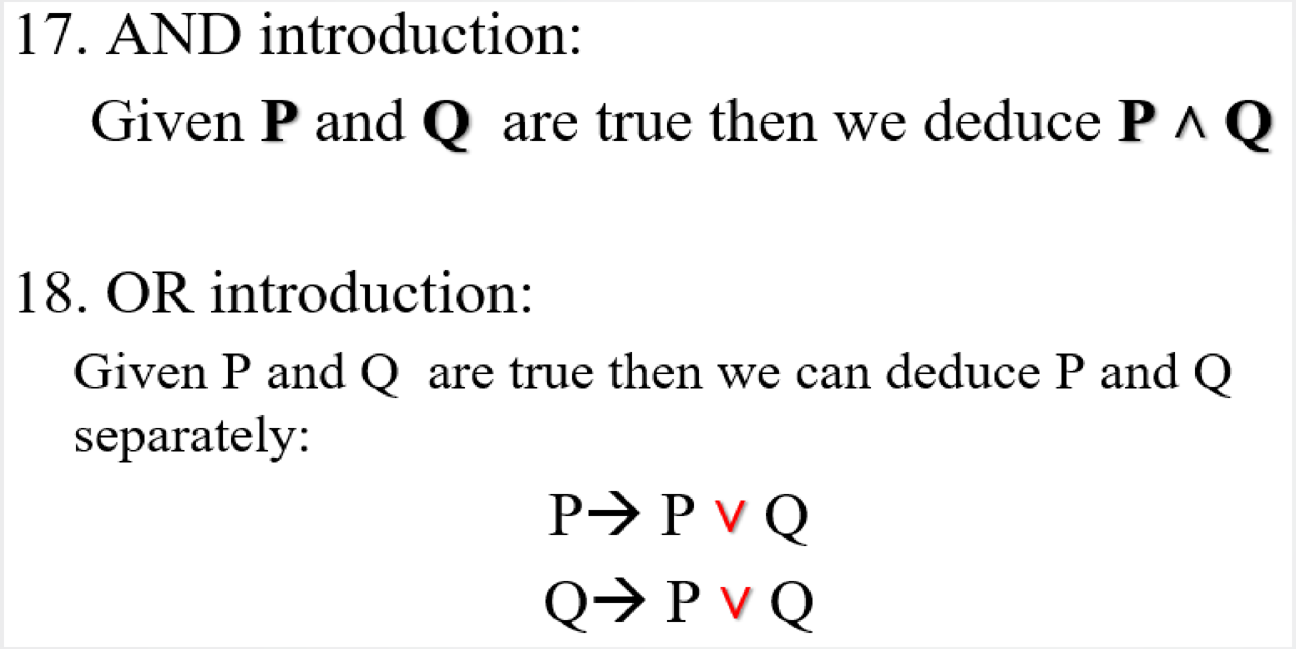
16. AND elimination:

Given P and Q are true then we can deduce P and Q separately:

$$P \wedge Q \rightarrow P$$

$$P \wedge Q \rightarrow Q$$

Inference Rules in Propositional Logic



17. AND introduction:

Given **P** and **Q** are true then we deduce **P ∧ Q**

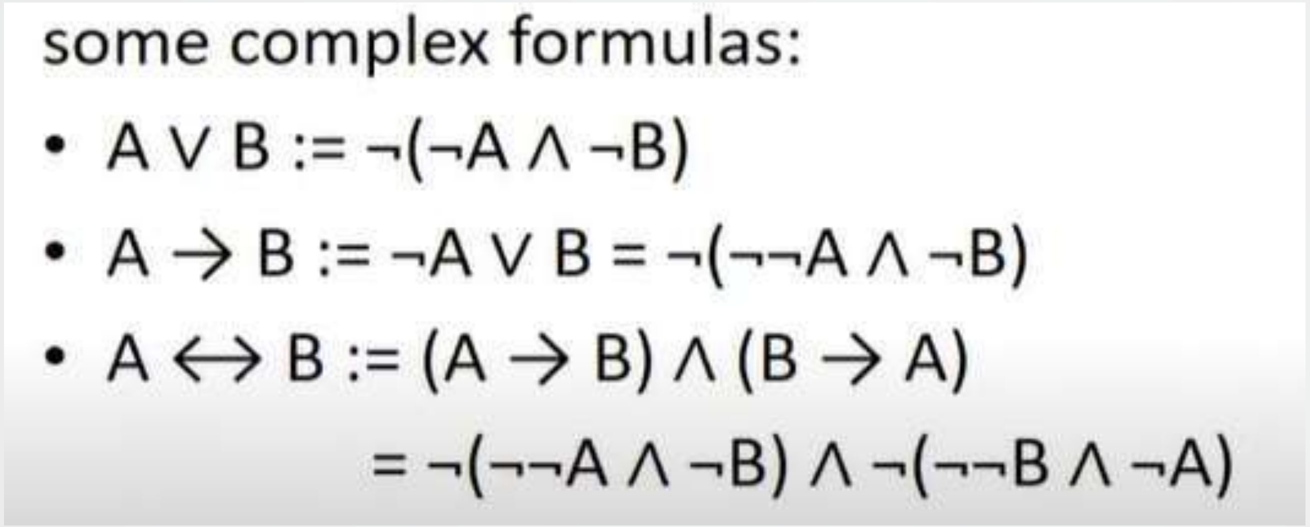
18. OR introduction:

Given P and Q are true then we can deduce P and Q separately:

$$P \rightarrow P \vee Q$$

$$Q \rightarrow P \vee Q$$

Complex formulae in Propositional Logic

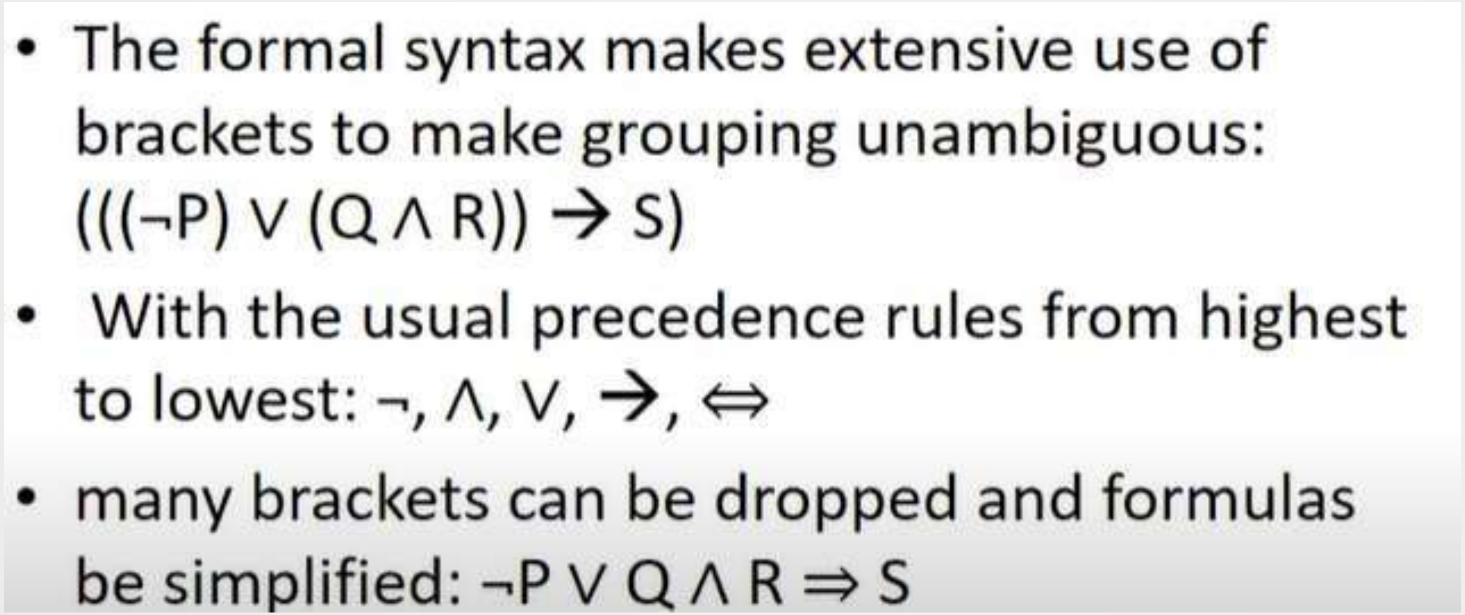


some complex formulas:

- $A \vee B := \neg(\neg A \wedge \neg B)$
- $A \rightarrow B := \neg A \vee B = \neg(\neg\neg A \wedge \neg B)$
- $A \leftrightarrow B := (A \rightarrow B) \wedge (B \rightarrow A)$

$$= \neg(\neg\neg A \wedge \neg B) \wedge \neg(\neg\neg B \wedge \neg A)$$

Precedence of Rules from Highest to Lowest



- The formal syntax makes extensive use of brackets to make grouping unambiguous: $(((\neg P) \vee (Q \wedge R)) \rightarrow S)$
- With the usual precedence rules from highest to lowest: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\Leftrightarrow$
- many brackets can be dropped and formulas be simplified: $\neg P \vee Q \wedge R \Rightarrow S$

Limitations of Propositional Logic

- We ***cannot represent relations like ALL, some, or none*** with propositional logic.
- **Example:**
  - ❖ ***All the girls are intelligent.***
  - ❖ ***Some apples are sweet.***
  - ❖ ***All students are present in the class.***
  - ❖ ***Some students are absent in the class.***
- Propositional logic has limited expressive power.
- All **(∀)** and Some **(∃)** can be represented only by **quantifiers**.

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Predicate Logic : Representing Simple Facts in Logic

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 14
Presentation Date – 11/11/2024

Course Outcome

At the end of the course, students should be able to:

CO4: Demonstrate computable functions and predicates in computational system to construct logical expressions.

Topic Learning Outcome

Interpret and analyze predicate logic expressions to understand their meaning and implications.

Predicate Logic

Predicate Logic - Introduction

- ***First-order logic*** is another way of knowledge representation in AI.
- It is an ***extension to propositional logic***.
- ***FOPL*** 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 an easier way and can also express the relationship between those objects.

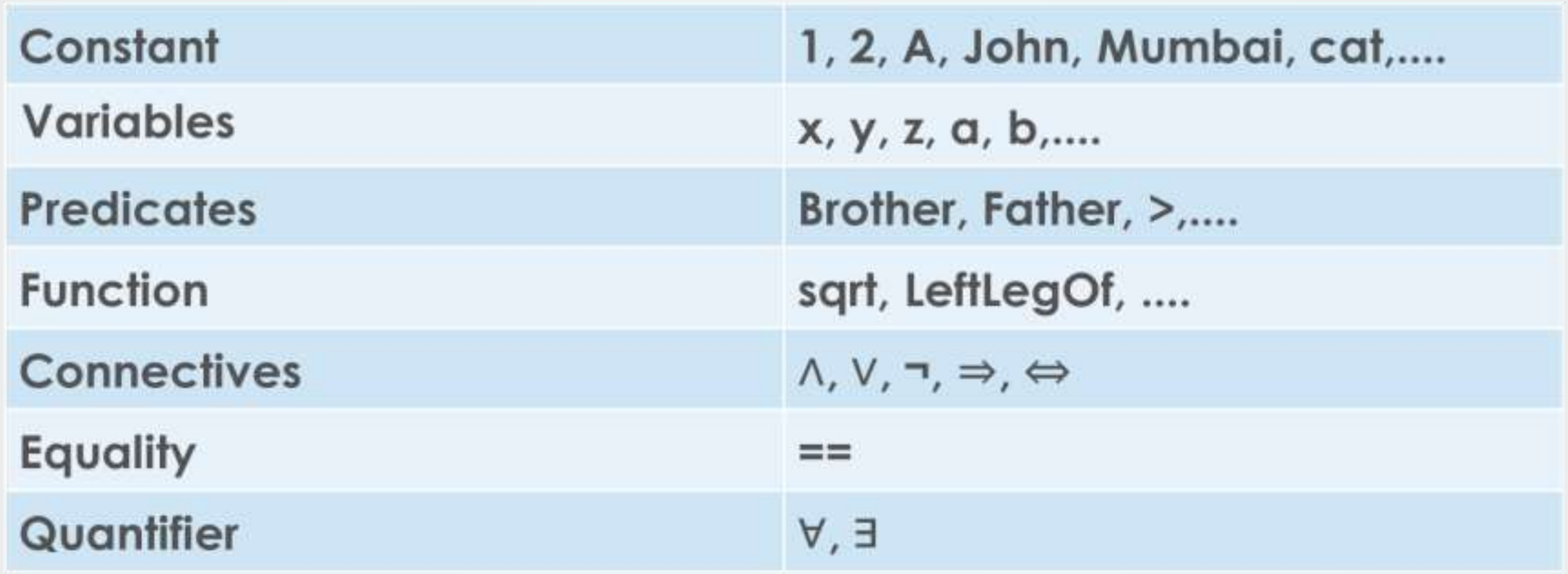
Predicate Logic - Introduction

- 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, wumpus, ...
  - **Relations**: It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between, bigger than….
  - **Facts/Function**: is Father of, is best friend, third inning of, end of, can swim......

Predicate Logic - Introduction

- As a natural language, ***first-order logic also has two main parts:***
  - ***Syntax***
  - ***Semantics***

Syntax of FOL – Basic Elements



Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic Sentences

- Atomic sentences are the ***most basic sentences of first-order logic.*** These sentences ***are formed from a predicate symbol followed by a parenthesis with a sequence of terms***.
- We can represent atomic sentences as

  Predicate (term1, term2, ..., term n).

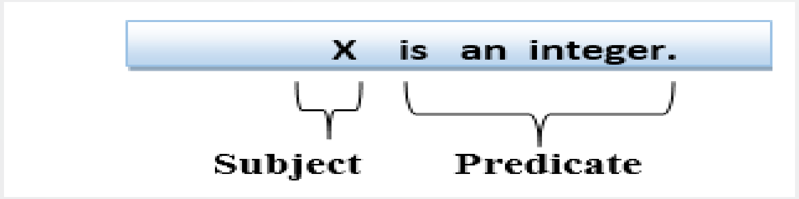
- **Example:**

  Ravi and Ajay are brothers: => Brothers(Ravi, Ajay)

  Tom is a cat: => cat (Tom)

Complex Sentences

- Complex sentences are made by ***combining atomic sentences using connectives.***
- ***First-order logic statements can be divided into two parts:***
  - ***Subject***: Subject is the main part of the statement.
  - ***Predicate:*** A predicate can be defined as a relation, which binds two atoms together in a statement.
- **Example:** "x is an integer."



Quantifiers in FOL

- There are two types of quantifier:

1. **Universal Quantifier(∀),** (for all, for each, for every, everyone, everything)
2. **Existential quantifier(∃),** (for some, at least one)

Universal Quantifier

- Universal quantifier is a symbol of logical representation, which specifies that the ***statement within its range is true for everything or every instance of a particular thing.***
- The Universal quantifier is ***represented by a symbol*** ∀ ***.***
- In universal quantifier ***we use implication "→".***
- ***If x is a variable, then for ∀x is read as:***
  - For all x
  - For each x
  - For every x

Universal Quantifier - Example

- **Every man respects his parent.**

  predicate is "respects(x, y)," where x=man, and y= parent

  $\forall x$: man(x) $\rightarrow$ respects (x, parent)

- **All man drink coffee.**

  $\forall x$: man(x) $\rightarrow$ drink (x, coffee)

  i.e., there are all x where x is a man who drink coffee.

- **All birds fly.** $\forall x$: bird(x) $\rightarrow$ fly(x)
- **All graduates are unemployed.** $\forall x$: graduate(x) $\rightarrow$ unemployed(x)

Existential Quantifier

- Existential quantifiers are the type of quantifiers, which express that ***the statement within its scope is true for at least one instance of something***.
- It is ***denoted by the logical operator* ∃.**
- In Existential quantifier ***we always use AND or Conjunction symbol* (∧).**
- ***If x is a variable, then for ∃x is read as:***
  - There exists a ‘x’
  - For some ‘x’
  - For at least one ‘x’

Existential Quantifier - Example

- **Some boys play cricket.**

  predicate is "play(x, y)," where x= boys, and y= game

  ∃x: boys(x) ∧ play(x, cricket).

- **Some boys are intelligent.**

  ∃x: boys(x) ∧ intelligent(x)

  i.e., there are some x where x is a boy who is intelligent.

- **Someone is crying.**

  ∃x: crying(x)

Properties of Quantifiers

- **In universal Quantifiers**
  - $\forall x \, \forall y$ is similar to $\forall y \, \forall x$
- **In Existential Quantifiers**
  - $\exists x \, \exists y$ is similar to $\exists y \, \exists x$
  - $\exists x \, \forall y$ is not similar to $\forall y \, \exists x$

Representing Simple Facts in Logic

Representing Simple Facts in Logic

- **The ball colour is red.**

  Color(Ball, Red)

- **Rishi likes Mango.**

  Likes(Rishi, Mango)

- **Mihir is a man.**

  Man(Mihir)

- **Mihir is a Mumbaikar.**

  Mumbaikar(Mihir)

Representing Simple Facts in Logic

- **Everybody loves somebody.**

  $\forall(x)\ \exists y : loves(x, y)$

- **Mihir is a Mumbaikar.**

  Mumbaikar(Mihir)

- **All Mumbaikars are Indian.**

  $\forall(x) : Mumbaikars(x) \rightarrow Indian(x)$

- **Every Mumbaikar likes Mumbai.**

  $\forall(x) : Mumbaikar(x) \rightarrow likes(x, Mumbai)$

Representing Simple Facts in Logic

- **All red flowers are beautiful.**

  $\forall(x) : flower(x) \wedge red(x) \rightarrow beautiful(x)$

- **Everyone is loyal to someone.**

  $\forall(x)\ \exists y : loyal(x, y)$

- **Everyone loves everyone.**

  $\forall x\ \forall y : loves(x,y)$

- **Everyone loves everyone except himself.**

  $\forall x\ \forall y : loves(x,y) \wedge \sim loves(x, x)$

Representing Simple Facts in Logic

- **Ram is tall.**

  tall(Ram)

- **Ram loves Sita.**

  loves(Ram, Sita)

- **Ram teaches either maths or java.**

  Teaches(Ram, Math) V Teaches(Ram, Java)

- **Ram teaches math if and only if Ram doesn’t teach java.**

  Teaches(Ram, Math) ↔ ~ Teaches(Ram, Java)

Representing Simple Facts in Logic

- **All students like football.**

  $\forall x : Student(x) \rightarrow like(x, football)$

- **Some student like football.**

  $\exists x : Student(x) \wedge like(x, football)$

- **Not all students like both Mathematics and Science.**

  $\sim\forall(x)\ [student(x) \rightarrow like(x, Mathematics) \wedge like(x, Science)]$

Representing Simple Facts in Logic

- **Bill is a student.**

  Student(Bill)

- **Bill takes either Analysis or Geometry (but not both).**

  Takes(Bill, Analysis) ⇔ ¬Takes(Bill, Geometry)

- **Bill takes Analysis or Geometry (or both).**

  Takes(Bill, Analysis) ∨ Takes(Bill, Geometry)

- **Bill takes Analysis and Geometry.**

  Takes(Bill, Analysis) ∧ Takes(Bill, Geometry)

- **Bill does not take Analysis.** ¬Takes(Bill, Analysis)

Representing Simple Facts in Logic

- **No student loves Bill.**

  $\neg\exists x\ (Student(x) \wedge Loves(x, Bill))$

- **Bill has at least one sister.**

  $\exists x\ SisterOf(x, Bill)$

- **Bill has no sister.**

  $\neg\exists x\ SisterOf(x, Bill)$

Comparison between PL and FOL

Propositional Logic (PL)	**First Order Logic (FOL)**
PL can not represent small worlds like vacuum cleaner world.	FOL can very well represent small world's problems.
PL is a weak knowledge representation language.	FOL is a strong way of representing language.
PL uses propositions in which the complete sentence is denoted by a symbol.	FOL uses predicated which involve constants, variables, functions, relations.
PL can not directly represent properties of individual entities or relation between entities. i.e., Hiral is short.	FOL can directly represent properties of individual entities or relation between entities. i.e., Hiral is short. Ans is: short (Hiral)

Try

Write the following sentences using predicate logic.

1. Anyone whom Mary loves is a football star.
2. Any student who does not pass does not play.
3. John is a student.
4. Any student who does not study does not pass.
5. Anyone who does not play is not a football star.
6. If John does not study, then Mary does not love John.

Try

Write the following sentences using predicate logic.

1. Every child loves Santa.
2. Everyone who loves Santa loves any reindeer.
3. Rudolph is a reindeer, and Rudolph has a red nose.
4. Anything which has a red nose is weird or is a clown.
5. No reindeer is a clown.
6. Scrooge does not love anything which is weird.
7. Scrooge is not a child.

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE

INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE

INSTITUTE OF AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Predicate Logic : Representing Instance and ISA Relationships, Computable Functions and Predicates, wff

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 15
Presentation Date – 11/11/2024

Course Outcome

At the end of the course, students should be able to:

CO4: Demonstrate computable functions and predicates in computational system to construct logical expressions.

Topic Learning Outcome

1. **Make use of** instance and ISA relationships to build and interpret class hierarchies, object-oriented models, or ontologies.
2. **Apply** computable functions and predicates to solve problems in various domains, such as algorithms, databases, or formal verification.
3. **Interpret** the properties of well-formed formula to manipulate formulas, providing theorems and validating logical arguments.

Representing Instance and ISA Relationships

Representing Instance and ISA Relationships

- Two attributes **isa** and **instance** play an important role in many aspects of knowledge representation. The reason for this is that ***they support property of inheritance.***
- **isa**
  - Used to show class inclusion,
  - e.g. isa (mega_star,rich).
- **Instance**
  - Used to show class membership,
  - e.g. instance(prince,mega_star).

Representing Instance and ISA Relationships

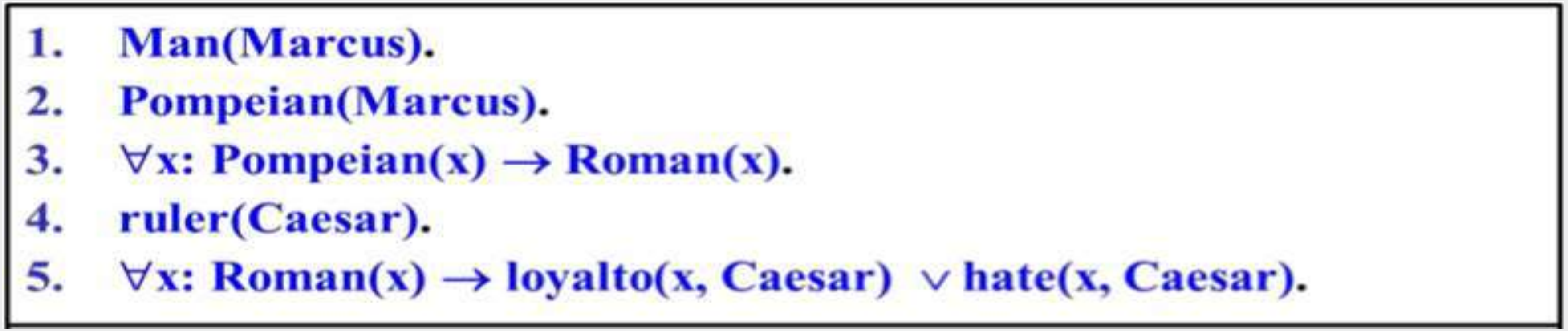
Example :

Consider the following five sentences:

1. **Marcus was a man.**
2. **Marcus was a Pompeian.**
3. **All Pompeians were Romans.**
4. **Caesar was a ruler.**
5. **All Pompeian's were either loyal to Caesar or hated him.**

Representing Instance and ISA Relationships

Representation using Pure Predicate Logic

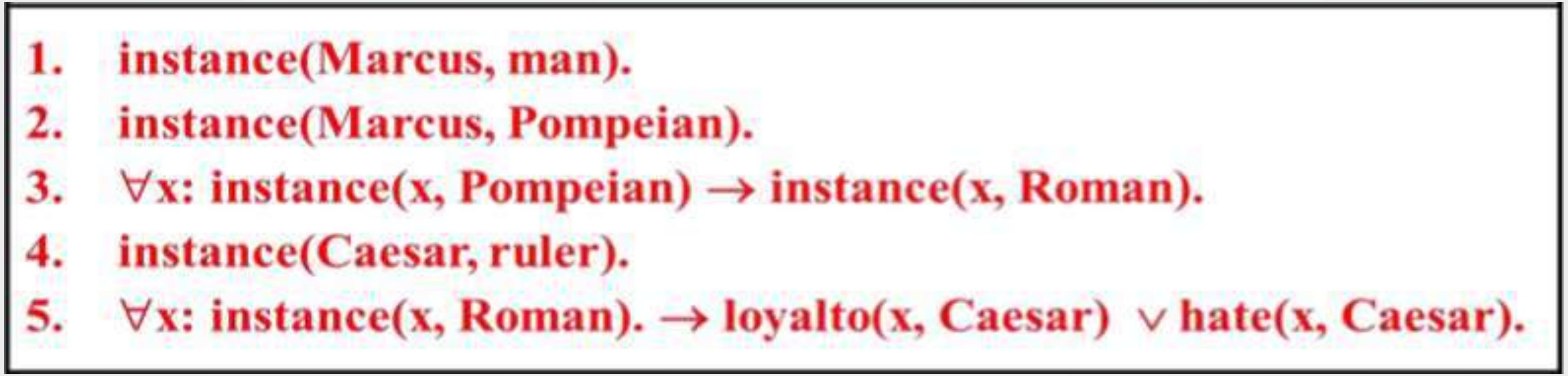


1. Man(Marcus).
2. Pompeian(Marcus).
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x).
4. ruler(Caesar).
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

Here the above five sentences are represented using pure predicate logic. In these representations, class ***membership is represented with unary predicates (***such as Roman), each of which corresponds to a class. Asserting that P(x) is true is equivalent to asserting that x is an instance of P.

Representing Instance and ISA Relationships

Representation using Instance Relationship

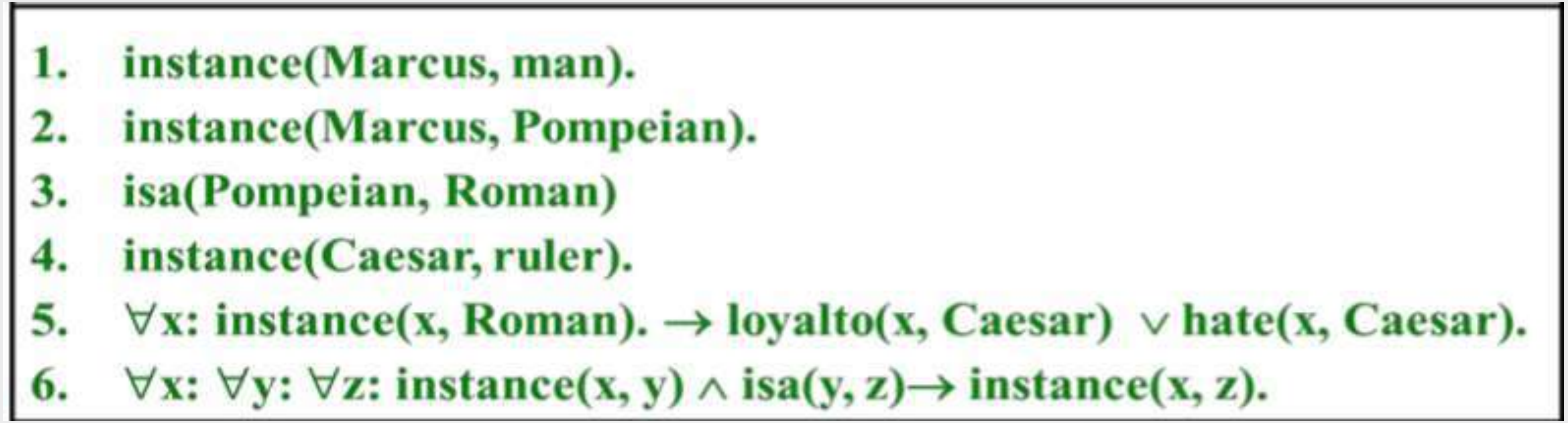


1. instance(Marcus, man).
2. instance(Marcus, Pompeian).
3. $\forall$x: instance(x, Pompeian) $\rightarrow$ instance(x, Roman).
4. instance(Caesar, ruler).
5. $\forall$x: instance(x, Roman). $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar).

Here the predicate **instance** is a binary one, whose ***first argument is an object*** and whose ***second argument is a class*** to which the object belongs.

Representing Instance and ISA Relationships

Representation using Instance and isa Relationship



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

For 3rd statement the implication rule states that ***if an object is an instance of the subclass Pompeian then it is an instance of the superclass Roman.***

Computable Functions and Predicates

Computable Functions and Predicates

- Simple facts like "**Marcus tried to assassinate Caesar**" can be represented in predicate logic as :

  tryassassinate(Marcus, Caesar)

- This is fine if the number of facts is not very large or if facts are not sufficiently structured. But suppose if we want to express simple facts such as "**greater-than**" and "**less-than**" relationships:

- **Computable predicates**

• greater-than(1, 0)	less-than(0, 1)
• greater-than(2, 1)	less-than(1, 2)
• greater-than(3, 2)	less-than(2, 3)

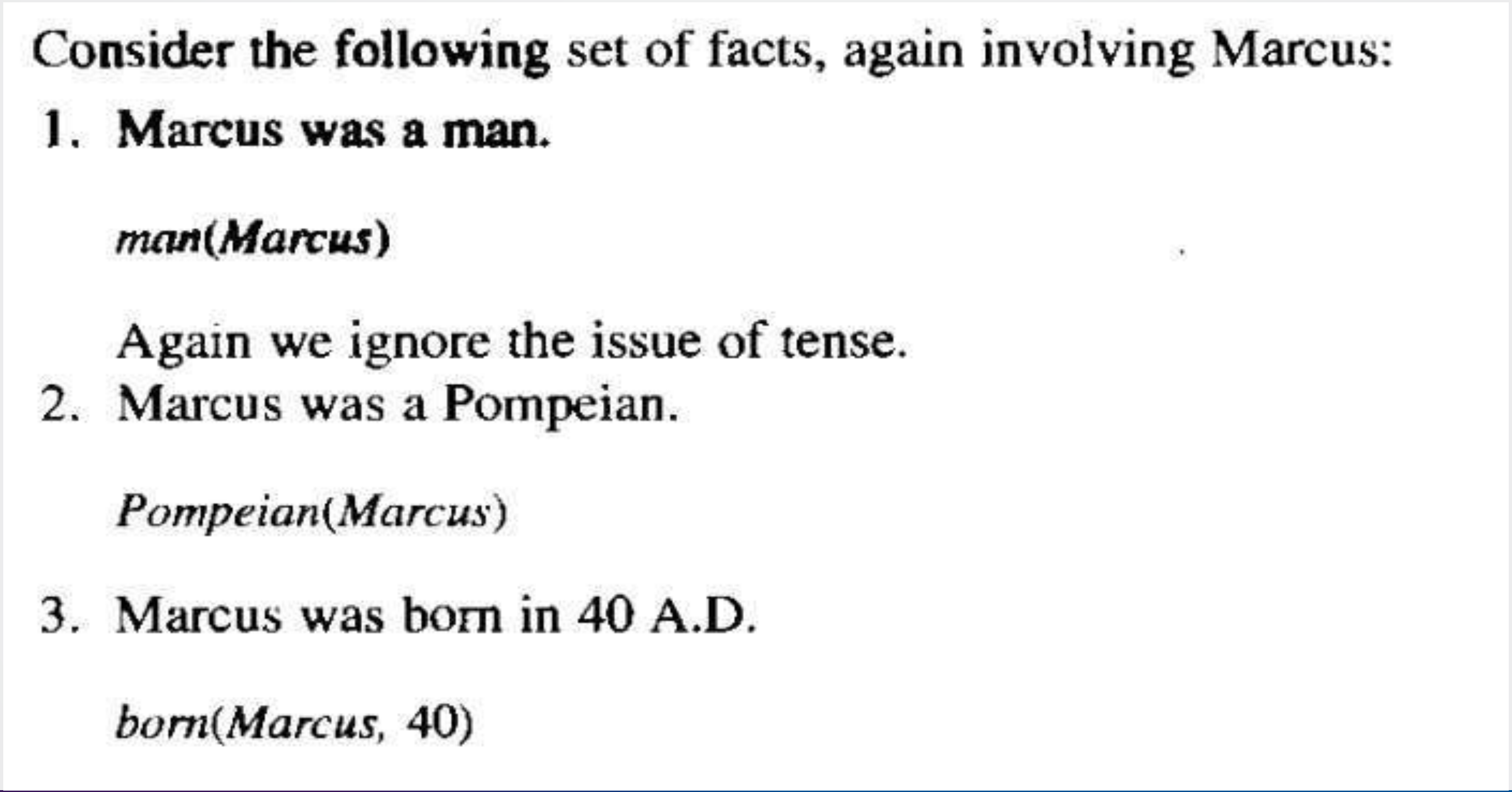
Computable Functions and Predicates

- It is often useful to have computable functions as well as computable predicates.
- **Example:**

  gt(2+3,1)

  the value of the plus function is computed given the arguments 2 and 3 , and then send the arguments 5 and 1 to gt

Computable Functions and Predicates



Consider the following set of facts, again involving Marcus:

1. **Marcus was a man.**

   man(*Marcus*)

   Again we ignore the issue of tense.
2. Marcus was a Pompeian.

   Pompeian(*Marcus*)

3. Marcus was born in 40 A.D.

   born(*Marcus*, 40)

Computable Functions and Predicates

4. All men are mortal.

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

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

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

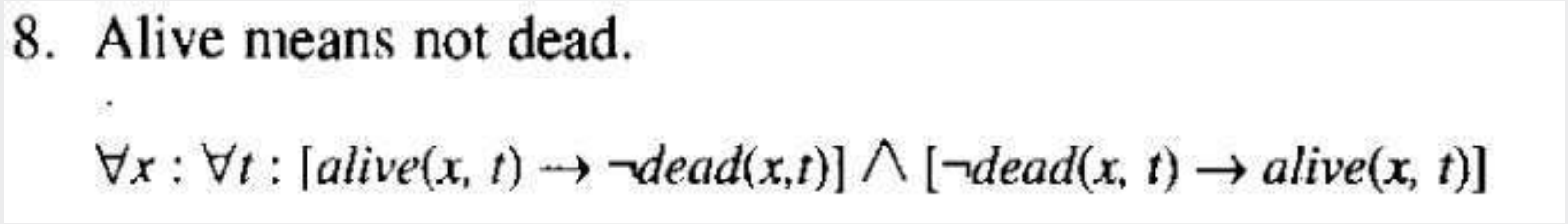
6. No mortal lives longer than 150 years.

$\forall x : \forall t_1 : \forall t_2 : mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \rightarrow dead(x, t_2)$

7. It is now 1991.

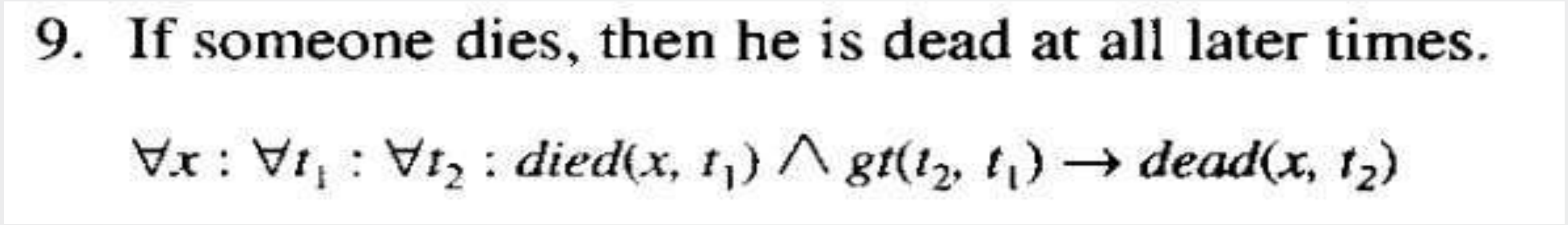
$now = 1991$

Computable Functions and Predicates



8. Alive means not dead.

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



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

$$\forall x : \forall t_1 : \forall t_2 : died(x, t_1) \wedge gt(t_2, t_1) \rightarrow dead(x, t_2)$$

Well Formed Formula (wff) and Its Properties

Well Formed Formula (wff) and Its Properties

- A well formed formula is just a formula in FOPL (first order predicate logic) is defined recursively as follows:

  I. An **atom or atomic function** is a wff. e.g., **E, G** etc.

  II. If **E** and **G** are wff, then each of **¬(E), ¬(G), (E ∧ G), (E ⇒ G), (E⇔G)** is a wff.

  III. If **E** is a wff and **x** is a free variable in E, then **∀x (E)** is a wff. [but (∀x)E   is wrong.]

- A wff can be obtained by the application of i), ii) and iii).

Well Formed Formula (wff) and Its Properties

- **(P)**, ‘P’ is valid by Rule 1 but *‘P’ inside parenthesis is not valid.*
- **¬P ∧ Q**, this can be either (¬P∧Q) or ¬(P∧Q) not valid because of ambiguity.

Parentheses are mandatory to be included in Composite Statements.

- **((P ⇒ Q)),** We can say (P⇒Q) is valid and (P⇒Q) = A is not.
- **(P ⇒⇒ Q),** is not valid because connective symbol right after a connective symbol is not allowed.
- **((P ∧ Q) ∧)Q),** conjunction operator after (P∧Q) is not valid.
- **((P ∧ Q) ∧ PQ),** invalid placement of variables (PQ).
- **(P ∨ Q) ⇒ (∧ Q),** is not valid because inside a parentheses minimum of 2 variables are required.

Well Formed Formula (wff) and Its Properties

Example of wff

- **Every person has a father.**

  $\forall x\ [\text{Person}(x) \Rightarrow \text{Father}(x)]$

  (Where, Person p(x) : x is a person.

  Father (x) : x has a father.)

- **There is a man and he is the father of Ram.**

  $\exists x\ [\text{Man}(x) \wedge \text{Father}(x, \text{Ram})]$

  (Where, Man(x) : x is a man

  Father(x, Ram) : x is the father of Ram.)

Well Formed Formula (wff) and Its Properties

Example of wff

- **All dancers love to dance.**

  $\forall x\ [\text{dancer}(x) \Rightarrow \text{love}(x, \text{dance})]$

  (Where, dancer(x) : x is a dancer.

  love(x, dance) : x loves dance.)

- **Everyone who sings and plays an instrument loves to dance.**

  $\forall x\ [\text{sings}(x) \wedge \text{plays}(x)] \Rightarrow \text{loves}(x, \text{dance})$

  (Where, sings(x) : x who sings, plays(x) : x who plays

  love(x, dance) : x loves dance)

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Course Title - Logic Programming for Artificial Intelligence

Topic Title – Predicate Logic : Clausal Forms, Conversion to clausal forms, Resolution

Presenter's Name – Ms. Bidyutlata Sahoo
Presenter's ID – IARE11028
Department Name – CSE (AI & ML)
Lecture Number - 16
Presentation Date – 12/11/2024

Course Outcome

At the end of the course, students should be able to:

CO4: Demonstrate computable functions and predicates in computational system to construct logical expressions.

Topic Learning Outcome

1. **Utilize** causal forms in AI systems to make informed decisions by evaluating the potential impacts of different actions.
2. **Demonstrate** the conversion to casual forms to standardize and simplify logical expressions.
3. **Apply** resolution method for deriving conclusions from a set of premises.

Clausal Form

Clausal Form

- Clausal form is essential in predicate logic *for simplifying logical expressions*, *making them suitable for automated reasoning methods like* ***resolution***, which rely on finding contradictions by analysing clauses.
- In clausal form, *the formula is made up of a number of clauses, where each clause is composed of a number of literals connected by* ***OR logical connectives*** *only*. **(CNF)**
- A formula can have the following quantifiers:
  - **Universal quantifier (∀)**
  - **Existential quantifier (∃)**

Clausal Form

Universal quantifier –

- It can be understood as – “For all x, P(x) holds”, meaning P(x) is true for every object x in the universe.
- Example: All trucks has wheels. $\forall x\ [\text{Truck}(x) \Rightarrow \text{wheel}(x)]$

Existential quantifier –

- It can be understood as – “There exists an x such that P(x)”, meaning P(x) is true for at least one object x of the universe.
- Example: Someone cares for you.

$$\exists x\ [\text{Someone}(x) \wedge \text{cares}(x, \text{you})]$$

Resolution Refutation in Propositional Logic (Conversion to CNF and DNF in Propositional logic)

Resolution Refutation in Propositional Logic

- This is an important method used in propositional logic *to prove a formula or derive a goal from a given set of cluses by contradictions*.
- Here the meaning of ***Clause*** *is a propositional formula formed from a finite collection of literals* (atoms or their negations) and logical connectives.
- In this method ***only two operations are used.***

(¬) and (∨)

Resolution Refutation in Propositional Logic

- In propositional logic there are 2 normal forms
  1. **Conjunctive Normal Form (CNF)**
  2. **Disjunctive Normal Form (DNF)**
- A formula is said to be in its **normal form**, if it is constructed using only natural connectives (**¬, ∨**)

Resolution Refutation in Propositional Logic

- In **CNF**, the formula is represented as ***Conjunction of Disjunction***.
- i.e., (A ∨ B) ∧ (A ∨ ¬B)

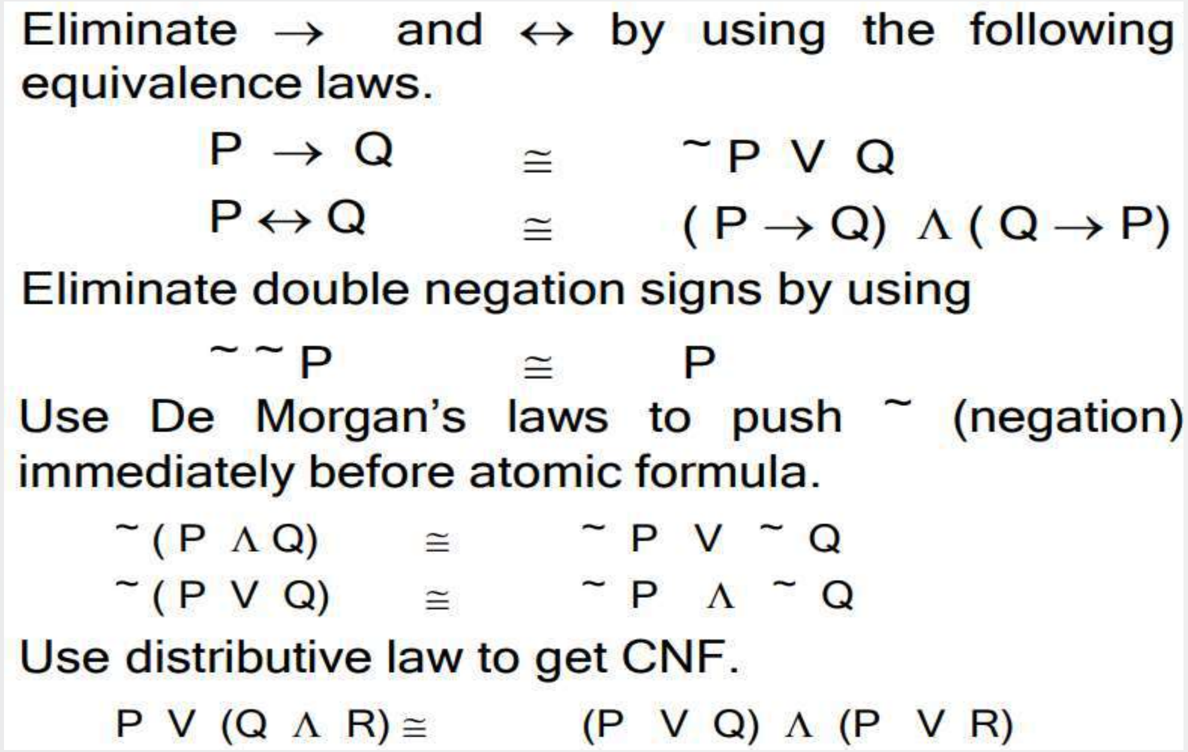
Must be disjunction

- In **DNF**, the formula is represented as ***Disjunction of Conjunction.***
- i.e., (A ∧ B) ∨ (A ∧ ¬B)

Must be conjunction

Resolution Refutation in Propositional Logic

Rules / Steps for the conversion of a formula to its CNF / DNF



1 Eliminate $\rightarrow$ and $\leftrightarrow$ by using the following equivalence laws.

$$P \rightarrow Q \cong \sim P \vee Q$$

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

2 Eliminate double negation signs by using

$$\sim\sim P \cong P$$

3 Use De Morgan's laws to push ~ (negation) immediately before atomic formula.

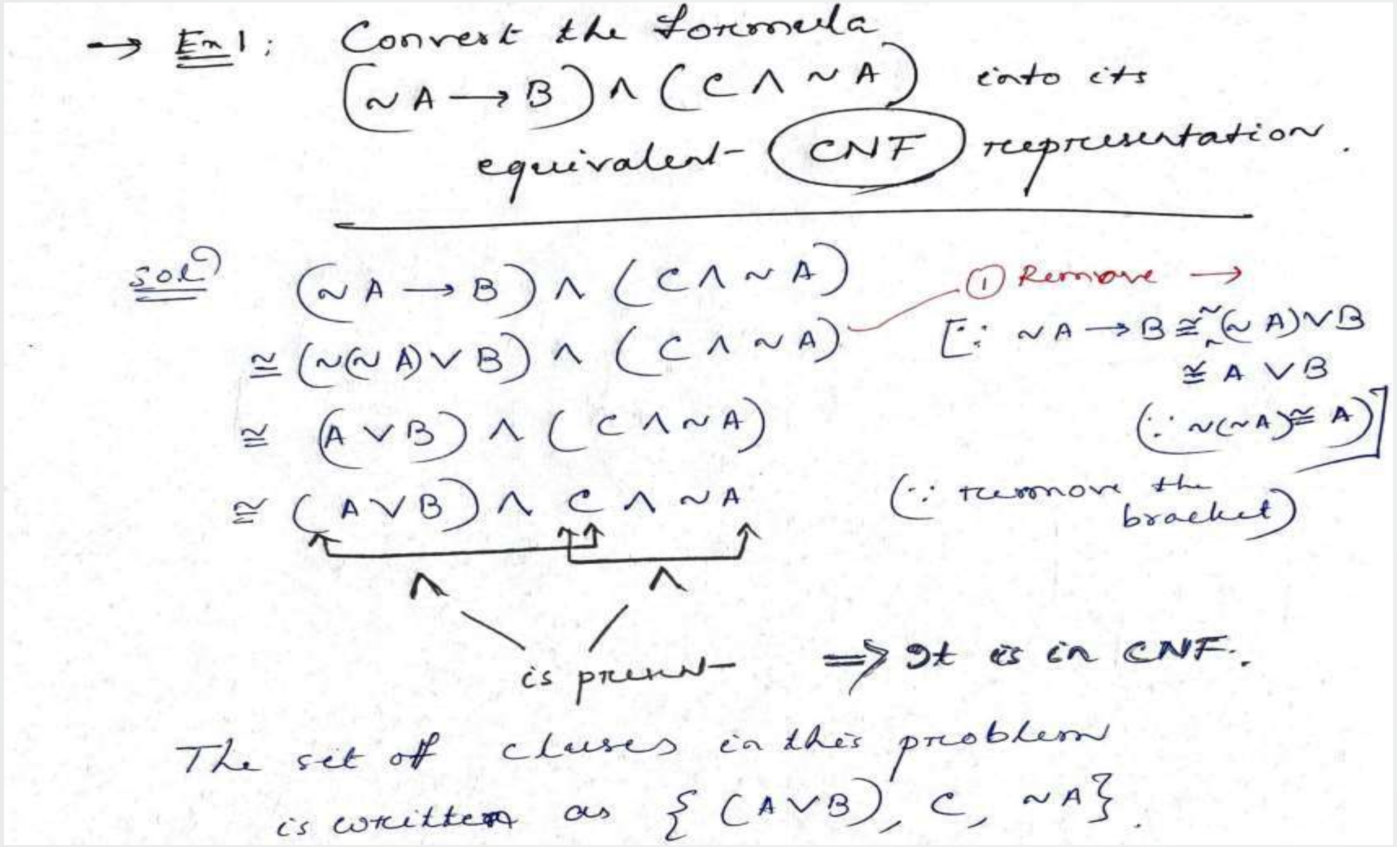
$$\sim(P \wedge Q) \cong \sim P \vee \sim Q$$

$$\sim(P \vee Q) \cong \sim P \wedge \sim Q$$

4 Use distributive law to get CNF.

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

Example-1



→ Ex 1: Convert the formula $(\sim A \rightarrow B) \wedge (C \wedge \sim A)$ into its equivalent CNF representation.

Sol:

$(\sim A \rightarrow B) \wedge (C \wedge \sim A)$ ① Remove →

$\cong (\sim(\sim A) \vee B) \wedge (C \wedge \sim A)$ $[\because \sim A \rightarrow B \cong \sim(\sim A) \vee B \cong A \vee B$ $(\because \sim(\sim A) \cong A)]$

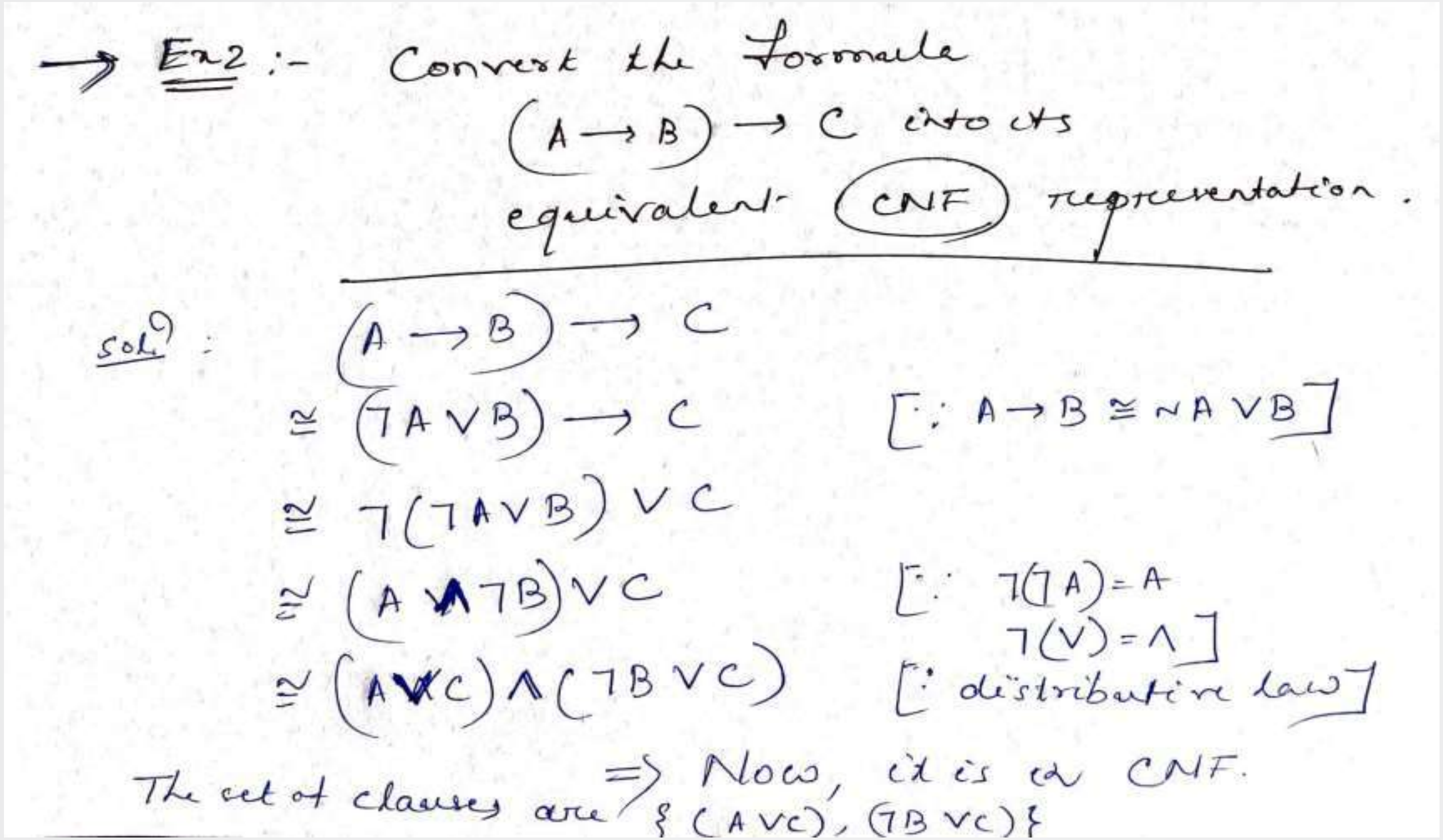
$\cong (A \vee B) \wedge (C \wedge \sim A)$

$\cong (A \vee B) \wedge C \wedge \sim A$ (∵ remove the bracket)

$\wedge$ $\wedge$ is present ⇒ It is in CNF.

The set of clauses in this problem is written as $\{(A \vee B), C, \sim A\}$.

Example-2



→ Ex2 :- Convert the formula $(A \to B) \to C$ into its equivalent CNF representation.

sol: $(A \to B) \to C$

$\cong (\neg A \vee B) \to C$ [∵ $A \to B \cong \sim A \vee B$]

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

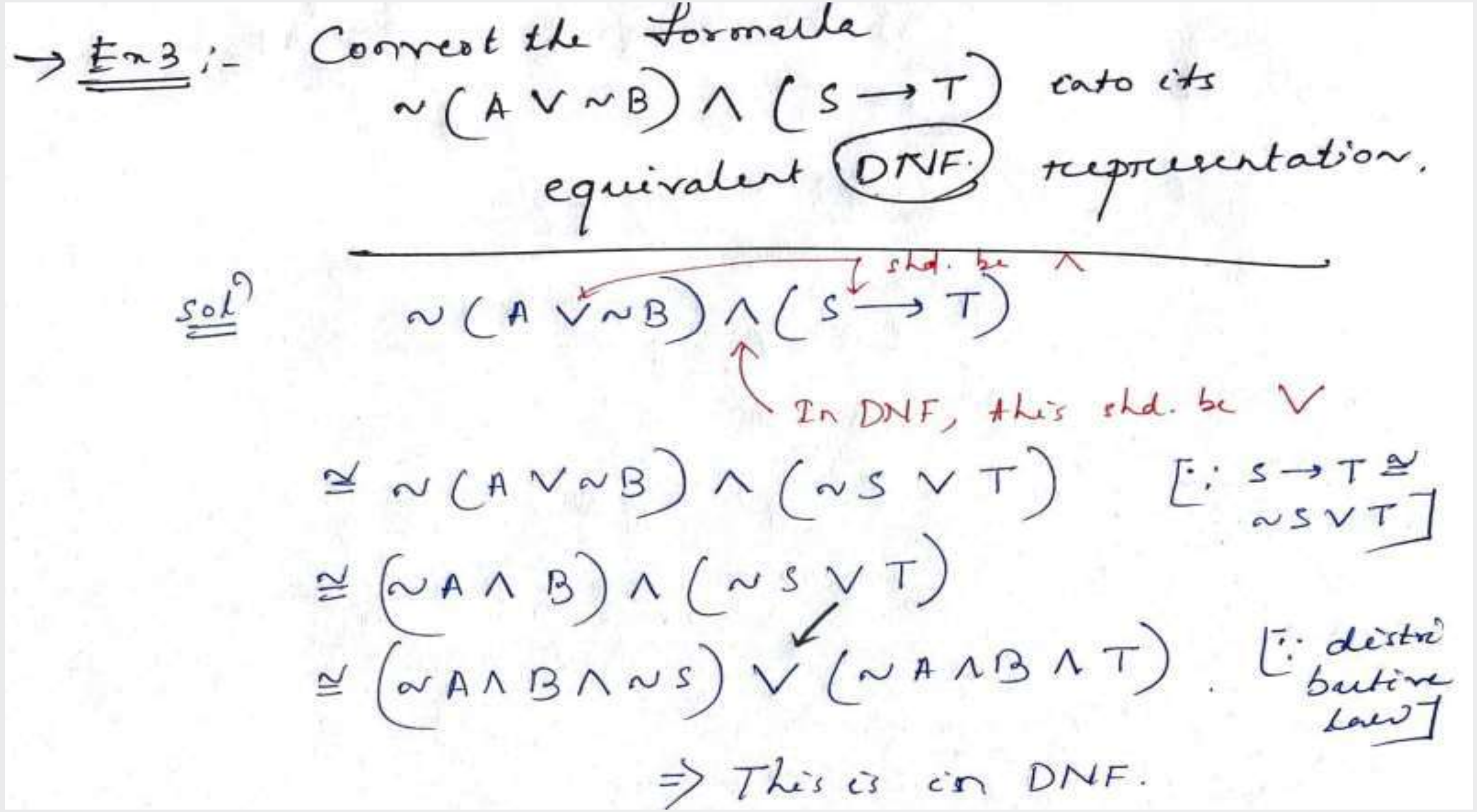
$\cong (A \wedge \neg B) \vee C$ [∵ $\neg(\neg A) = A$, $\neg(\vee) = \wedge$]

$\cong (A \vee C) \wedge (\neg B \vee C)$ [∵ distributive law]

⇒ Now, it is a CNF.

The set of clauses are $\{(A \vee C), (\neg B \vee C)\}$

Example-3



→ Ex 3 :- Convert the formula $\sim(A \vee \sim B) \wedge (S \rightarrow T)$ into its equivalent DNF representation.

Sol: $\sim(A \vee \sim B) \wedge (S \rightarrow T)$

shd. be $\wedge$

In DNF, this shd. be $\vee$

$\cong \sim(A \vee \sim B) \wedge (\sim S \vee T)$ $[\because S \rightarrow T \cong \sim S \vee T]$

$\cong (\sim A \wedge B) \wedge (\sim S \vee T)$

$\cong (\sim A \wedge B \wedge \sim S) \vee (\sim A \wedge B \wedge T)$. [∵ distributive law]

$\Rightarrow$ This is in DNF.

Conversion to Clausal Forms
(Conversion to CNF and DNF in Predicate Logic)

Conversion to Clausal Forms

- In **predicate logic**, CNF and DNF also apply but require additional steps due to the presence of **quantifiers** (like ∀ and ∃) and **predicates** (such as P(x) or Q(x,y)) that describe relationships among objects.

Conversion to Clausal Forms

- **CNF in Predicate Logic**: Transforming a predicate logic expression to CNF involves additional **steps** like *eliminating quantifiers*, *Skolemization* (replacing existential quantifiers with Skolem functions or constants), and *converting the expression to a conjunction of disjunctions*. This is crucial for inference methods such as **resolution**.
- **DNF in Predicate Logic**: Similarly, DNF in predicate logic involves *rewriting the expression as a disjunction of conjunctions of literals*, after handling quantifiers. However, *DNF is less commonly used in automated reasoning* for predicate logic since *CNF is more suited to proof techniques like resolution.*

Conversion to Clausal Forms - Rules

1. Eliminate $\rightarrow$.

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

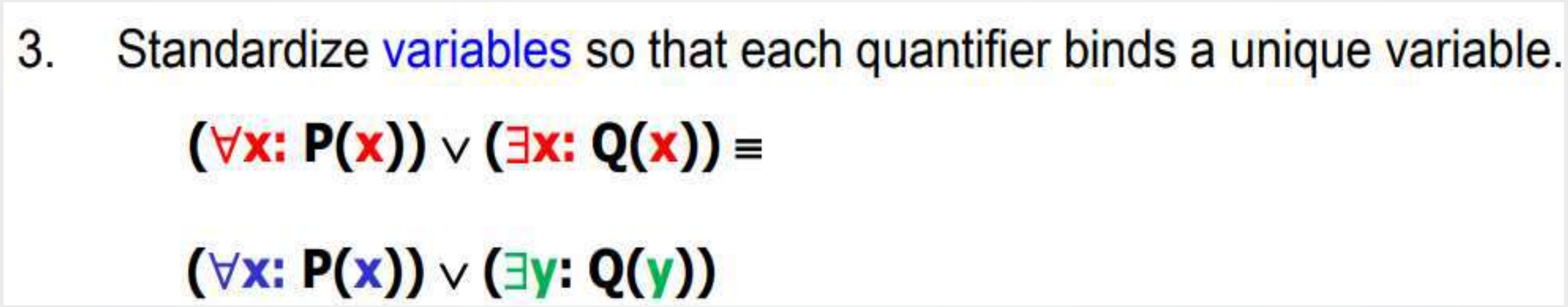
2. Reduce the scope of each $\neg$ to a single term.

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

$$\neg\forall x\text{:}\ P \equiv \exists x\text{:}\ \neg P$$
$$\neg\exists x\text{:}\ p \equiv \forall x\text{:}\ \neg P$$

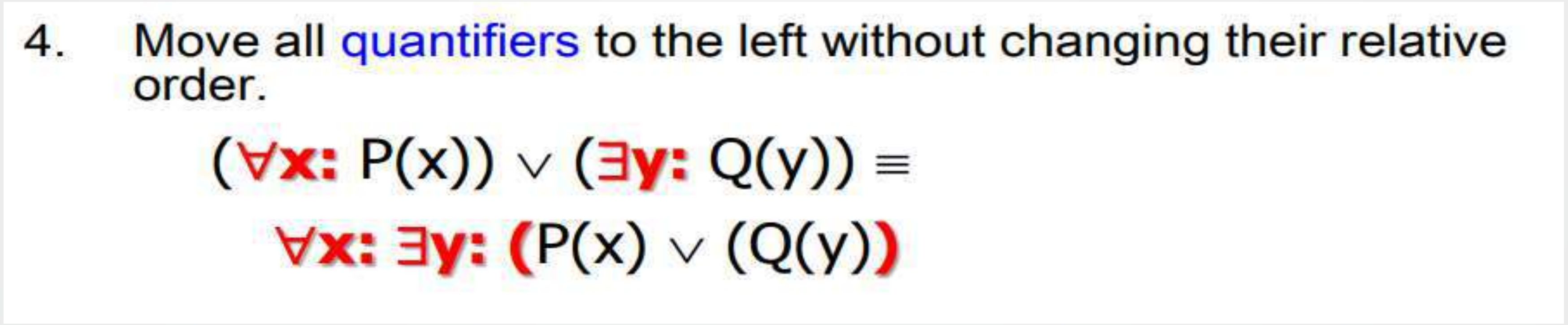
$$\neg\neg\, P \equiv P$$

Conversion to Clausal Forms - Rules


3. Standardize variables so that each quantifier binds a unique variable.

$$(\forall x: P(x)) \vee (\exists x: Q(x)) \equiv$$

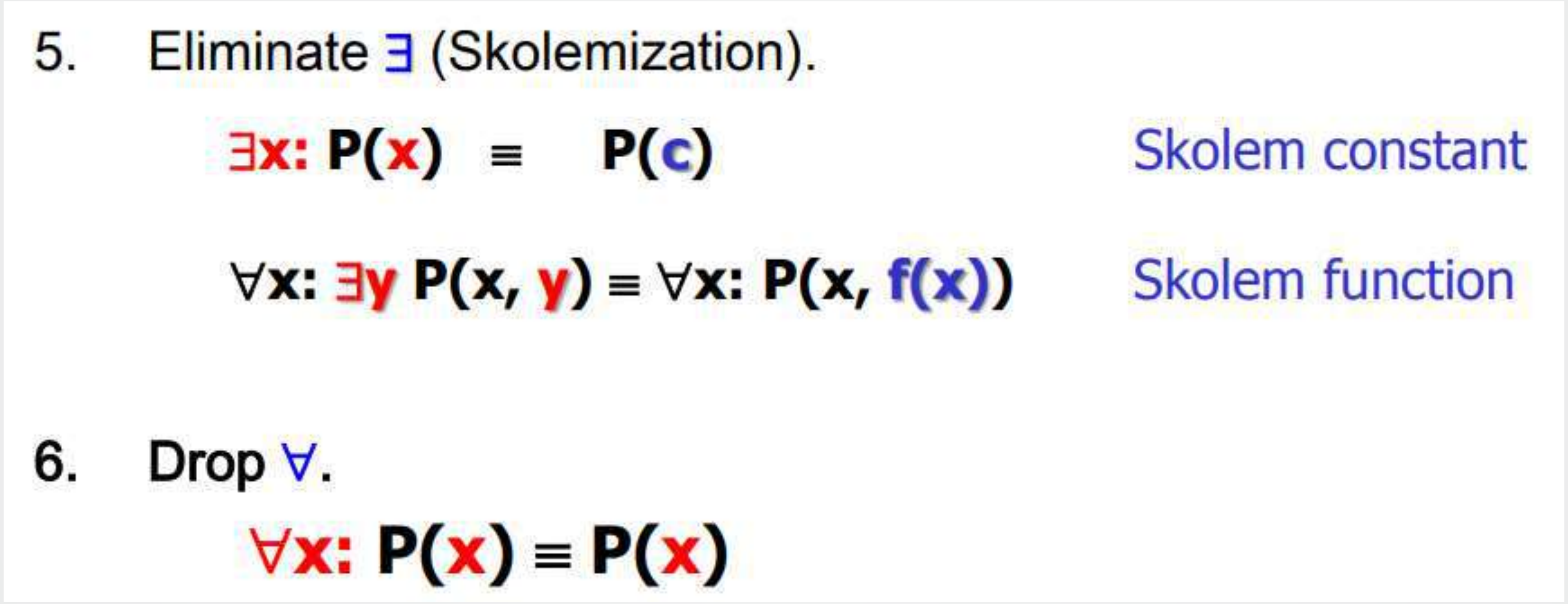
$$(\forall x: P(x)) \vee (\exists y: Q(y))$$


4. Move all quantifiers to the left without changing their relative order.

$$(\forall x: P(x)) \vee (\exists y: Q(y)) \equiv$$

$$\forall x: \exists y: (P(x) \vee (Q(y))$$

Conversion to Clausal Forms - Rules



5. Eliminate $\exists$ (Skolemization).

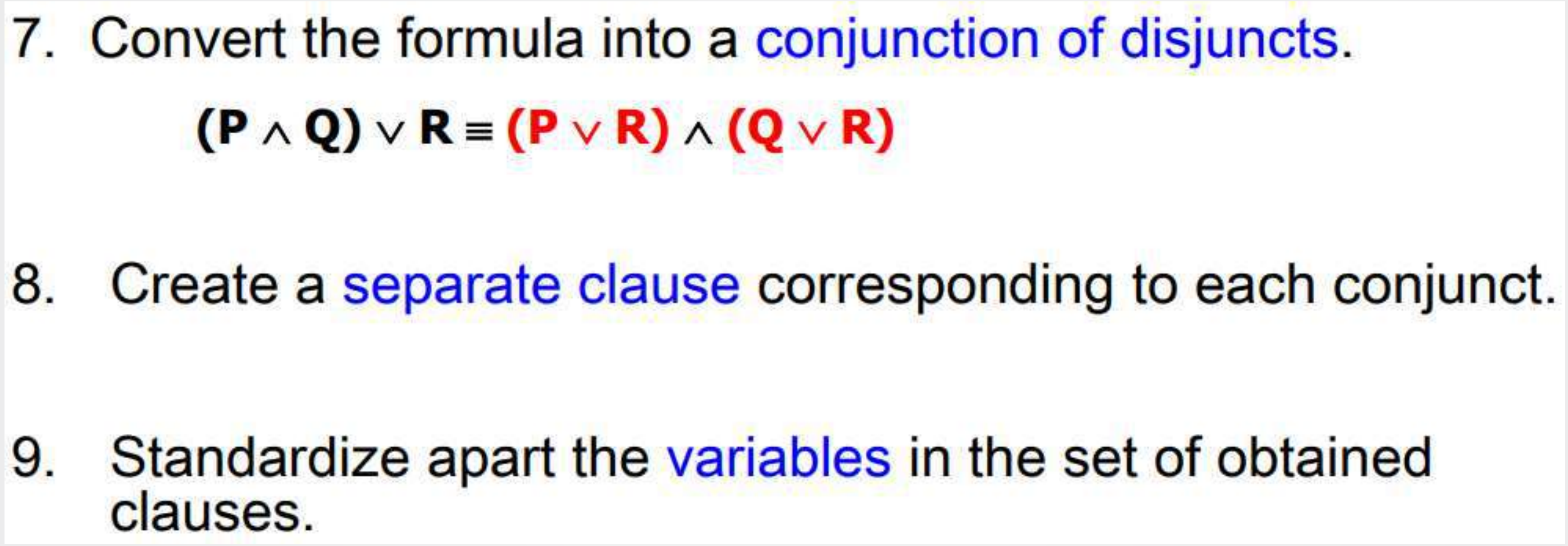
$\exists x\colon P(x) \equiv P(c)$ Skolem constant

$\forall x\colon \exists y\ P(x, y) \equiv \forall x\colon P(x, f(x))$ Skolem function

6. Drop $\forall$.

$\forall x\colon P(x) \equiv P(x)$

Conversion to Clausal Forms - Rules



7. Convert the formula into a conjunction of disjuncts.

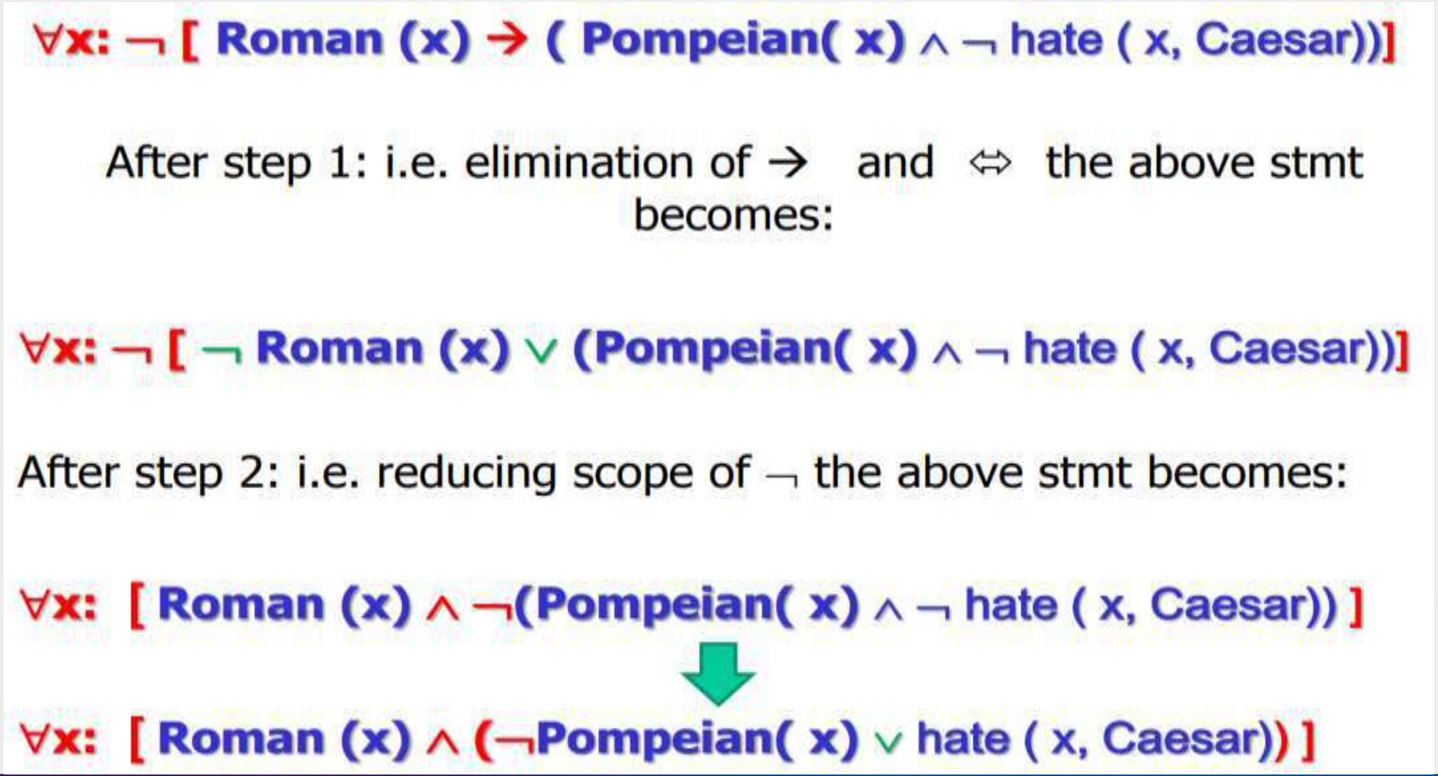
$$(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$$

8. Create a separate clause corresponding to each conjunct.

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

Conversion to Clausal Forms – Example

Example to demonstrate Step – 1 & 2



$\forall x: \neg [\text{ Roman } (x) \rightarrow (\text{ Pompeian}(x) \wedge \neg \text{ hate } (x, \text{ Caesar}))]$

After step 1: i.e. elimination of → and ⇔ the above stmt becomes:

$\forall x: \neg [\neg \text{ Roman } (x) \vee (\text{Pompeian}(x) \wedge \neg \text{ hate } (x, \text{ Caesar}))]$

After step 2: i.e. reducing scope of ¬ the above stmt becomes:

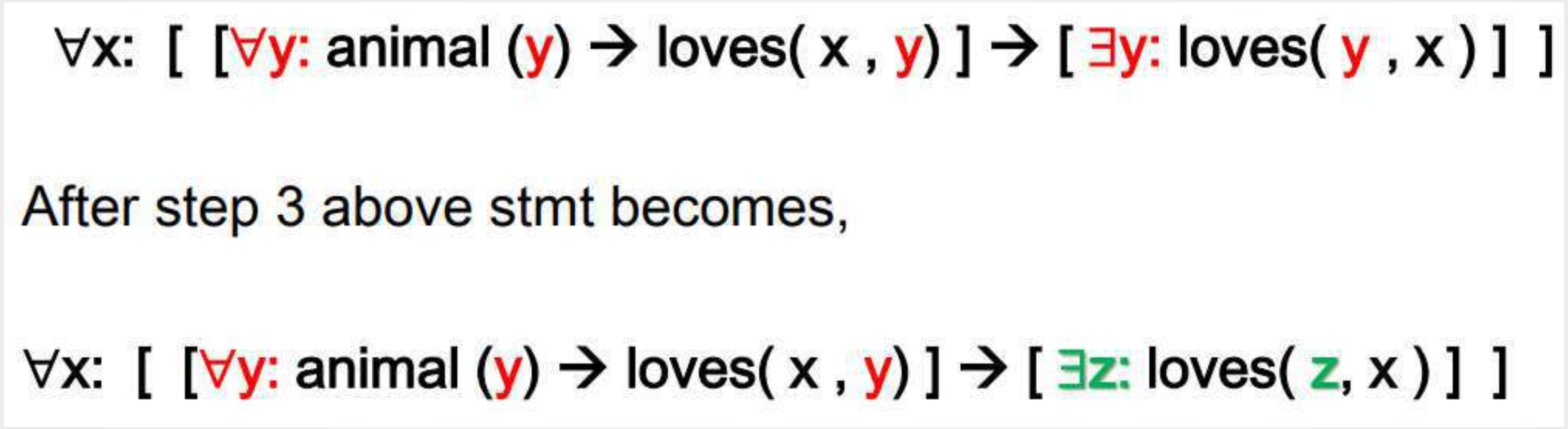
$\forall x: [\text{ Roman } (x) \wedge \neg(\text{Pompeian}(x) \wedge \neg \text{ hate } (x, \text{ Caesar}))]$

$\downarrow$

$\forall x: [\text{ Roman } (x) \wedge (\neg\text{Pompeian}(x) \vee \text{ hate } (x, \text{ Caesar}))]$

Conversion to Clausal Forms – Example

Example to demonstrate Step – 3 i.e., standardization of variables



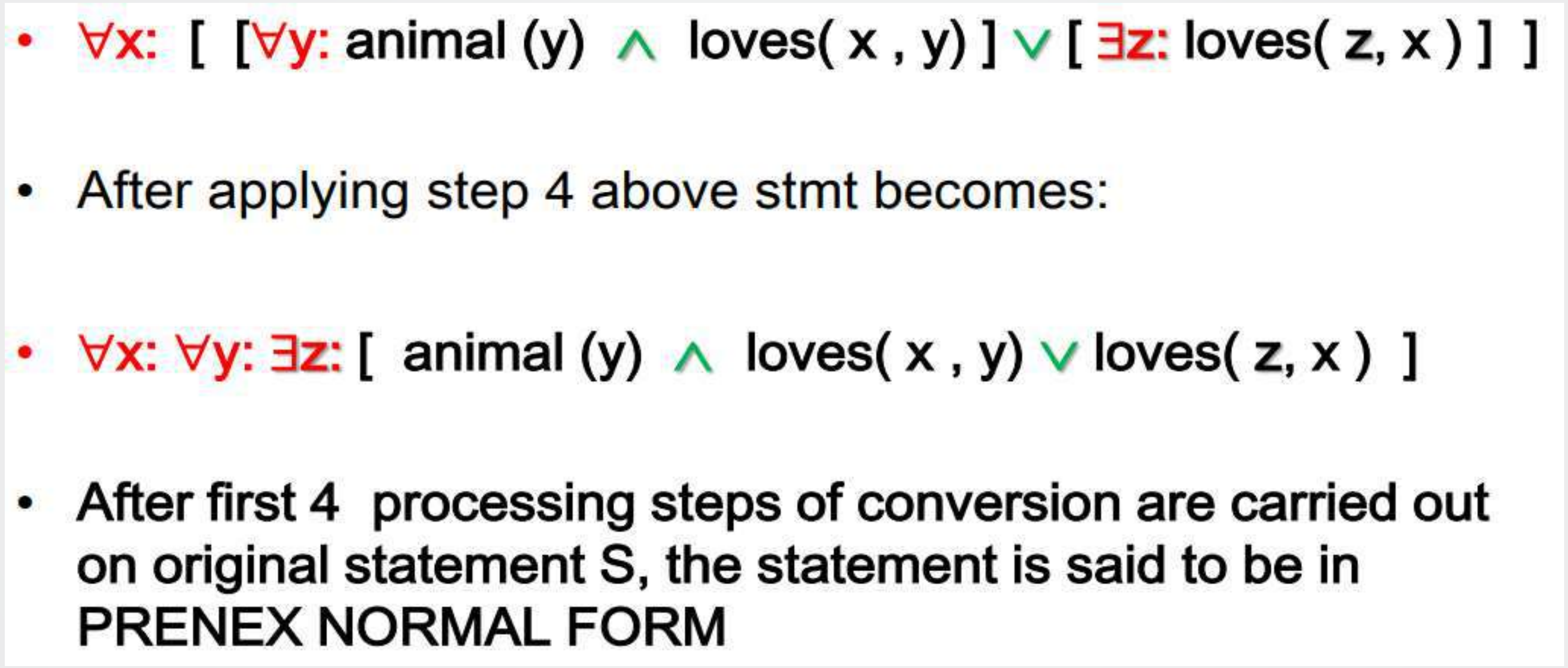
$\forall x: [\ [\forall y: \text{animal}(y) \rightarrow \text{loves}(x, y)\ ] \rightarrow [\ \exists y: \text{loves}(y, x)\ ]\ ]$

After step 3 above stmt becomes,

$\forall x: [\ [\forall y: \text{animal}(y) \rightarrow \text{loves}(x, y)\ ] \rightarrow [\ \exists z: \text{loves}(z, x)\ ]\ ]$

Conversion to Clausal Forms – Example

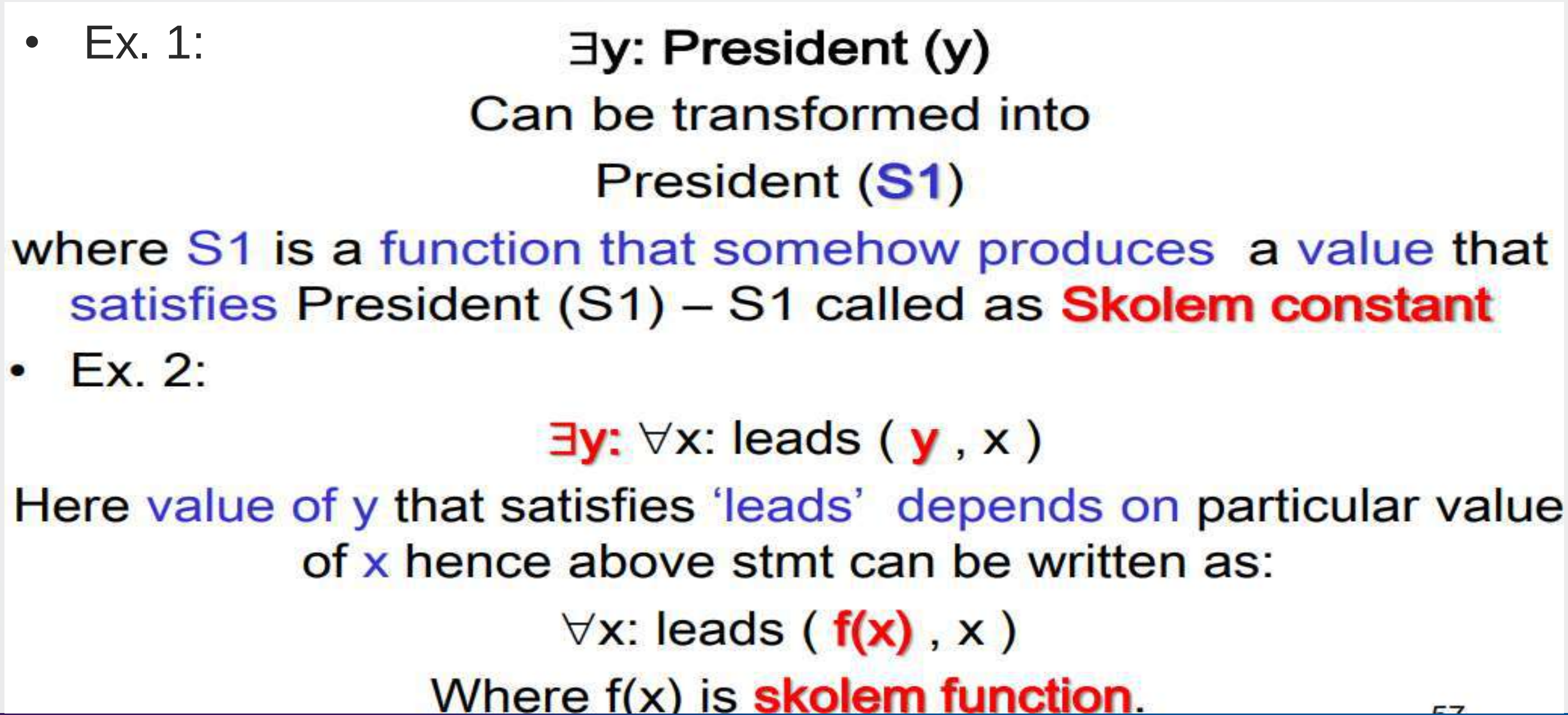
Example to demonstrate Step – 4 i.e., move all quantifiers to the left without changing their relative order.



- $\forall x$: [[$\forall y$: animal (y) $\wedge$ loves(x , y)] $\vee$ [$\exists z$: loves(z, x)]]
- After applying step 4 above stmt becomes:
- $\forall x$: $\forall y$: $\exists z$: [animal (y) $\wedge$ loves(x , y) $\vee$ loves(z, x)]
- After first 4 processing steps of conversion are carried out on original statement S, the statement is said to be in PRENEX NORMAL FORM

Conversion to Clausal Forms – Example

Example to demonstrate Step – 5 i.e., Skolemization (i.e., elimination of ∃ quantifier)



- Ex. 1:

$\exists y$: President (y)

Can be transformed into

President (S1)

where S1 is a function that somehow produces a value that satisfies President (S1) – S1 called as **Skolem constant**

- Ex. 2:

$\exists y$: $\forall x$: leads (y , x)

Here value of y that satisfies 'leads' depends on particular value of x hence above stmt can be written as:

$\forall x$: leads (f(x) , x)

Where f(x) is **skolem function**.

Conversion to Clausal Forms – Example

Example to demonstrate Step – 6 i.e., dropping prefix $\forall$

$$\forall x: \forall y: \forall z: [\neg \text{Roman}(x) \vee \neg \text{know}(x, y) \vee \text{hate}(y, z)]$$

- After prefix dropped becomes,

$$[\neg \text{Roman}(x) \vee \neg \text{know}(x, y) \vee \text{hate}(y, z)]$$

Conversion to Clausal Forms – Example

Example to demonstrate Step – 7 i.e., convert the formula into a conjunction of disjuncts(CNF)

- $\text{Roman}(x) \vee ((\text{hate}(x, \text{caesar}) \wedge \neg\text{loyalto}(x, \text{caesar}))$
- $\underbrace{\text{Roman}(x)}_{P} \vee ((\underbrace{\text{hate}(x, \text{caesar})}_{Q} \wedge \underbrace{\neg\text{loyalto}(x, \text{caesar})}_{R})$
- $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

CLAUSE 1 $\longrightarrow (\text{Roman}(x) \vee (\text{hate}(x, \text{caesar})) \quad \wedge$

CLAUSE 2 $\longrightarrow (\text{Roman}(x) \vee \neg\text{loyalto}(x, \text{caesar}))$

Resolution in FOPL

Resolution Predicate Logic(Resolution in FOPL)

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

Steps for Resolution

- **1. Conversion of facts into FOL.**
- **2. Convert FOL to CNF.**
  - A. Elimination of implication
    - ✓Eliminate all ”→” sign
    - ✓Replace P → Q with ~P V Q
  - B. Distribute negations
    - ✓Replace ~~P with P
    - ✓Replace ~(P V Q) with ~P ∧ ~Q

Steps for Resolution

- C. Eliminate existential quantifiers by replacing with skolem constants or skolem function
  - ✓ ∀X **∃Y** ((P1(X,Y) v (P2(X,Y))) ≡ ∀X ((P1(X,**F(X)**) v (P2(X,**F(X)**))
- D. Rename variables/ use standard variable to avoid duplicate quantifiers.
- E. Drop all universal quantifiers.
  - ✓ (P1(X,F(X)) v (P2(X,F(X))

Steps for Resolution

- **3. Negate the statement which needs to prove (proof by contradiction)**
  - ✓ In this statement, we will apply negation to the conclusion statements, which will be written as
  - ✓ ¬ likes(John, Peanuts)
- **4. Draw Resolution graph. (unification)**
  - ✓ Now in this step, we will solve the problem by resolution tree using substitution.

Resolution - Example

a) John likes all kind of food.

b) Apple and vegetable are food.

c) Anything anyone eats and not killed is food.

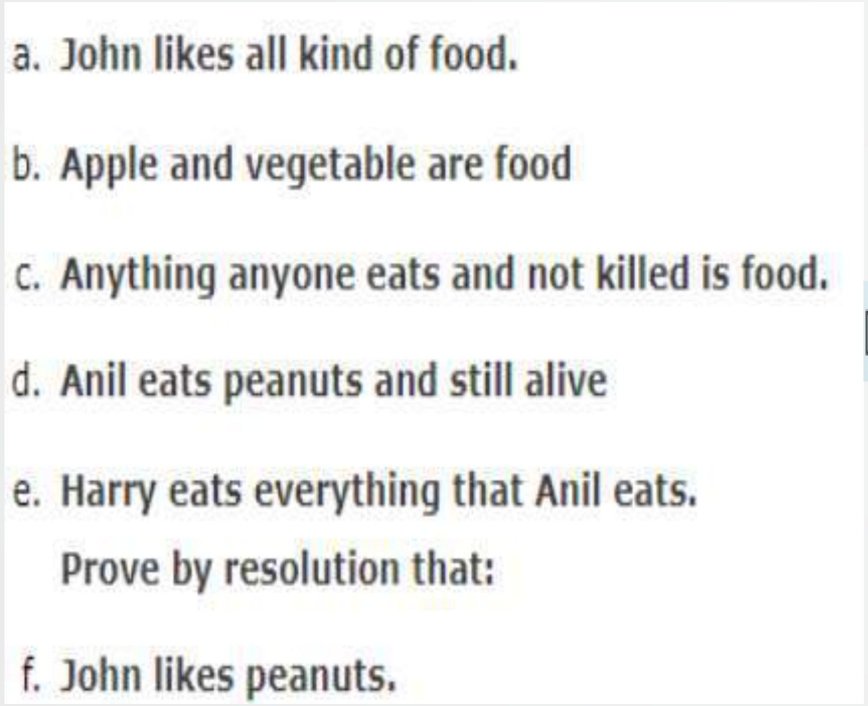
d) Anil eats peanuts and still alive.

e) Harry eats everything that Anil eats.

Prove by resolution that:

f) John likes peanuts.

Step-1 (Conversion of Facts into FOL)



a. John likes all kind of food.

b. Apple and vegetable are food

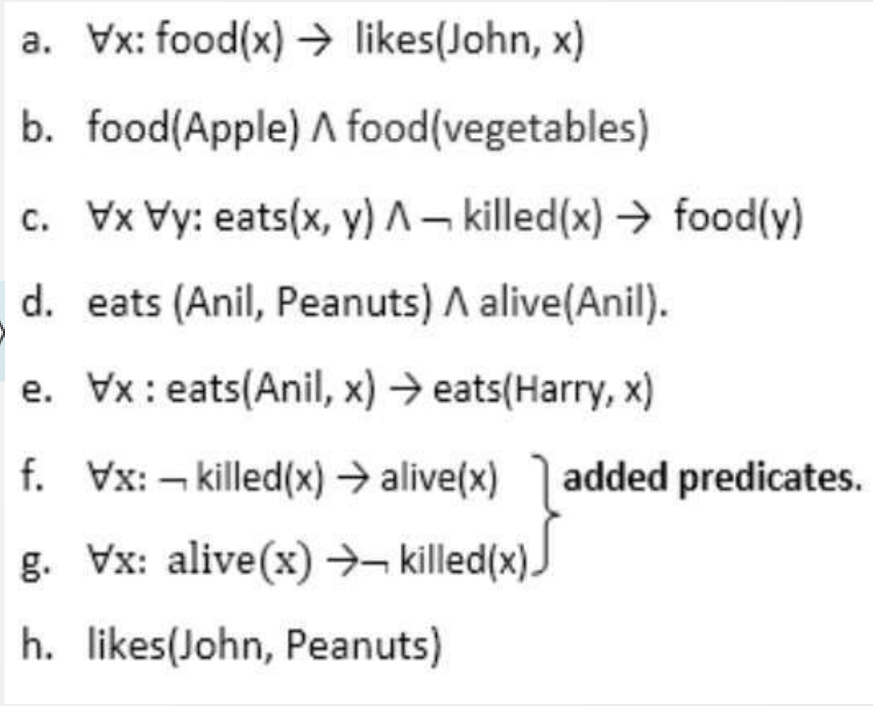
c. Anything anyone eats and not killed is food.

d. Anil eats peanuts and still alive

e. Harry eats everything that Anil eats.

Prove by resolution that:

f. John likes peanuts.



a. $\forall x$: food(x) → likes(John, x)

b. food(Apple) ∧ food(vegetables)

c. $\forall x \forall y$: eats(x, y) ∧ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil).

e. $\forall x$: eats(Anil, x) → eats(Harry, x)

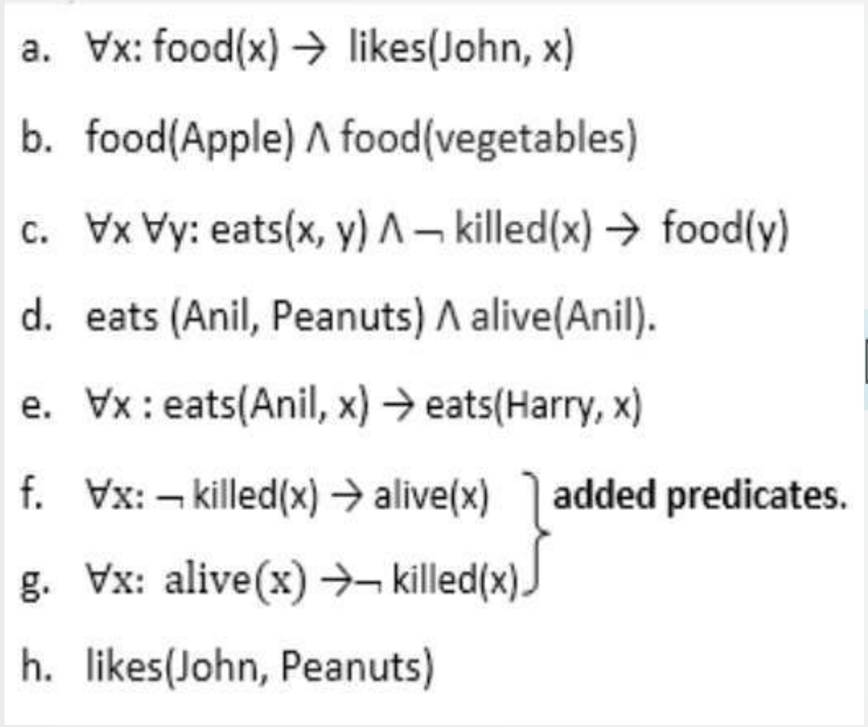
f. $\forall x$: ¬ killed(x) → alive(x)

g. $\forall x$: alive(x) →¬ killed(x)

} **added predicates.**

h. likes(John, Peanuts)

Step-2 (Eliminate all implication (→) and rewrite)



a. $\forall x\text{: food}(x) \rightarrow \text{likes}(\text{John}, x)$

b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$

c. $\forall x\ \forall y\text{: eats}(x, y) \wedge \neg\, \text{killed}(x) \rightarrow \text{food}(y)$

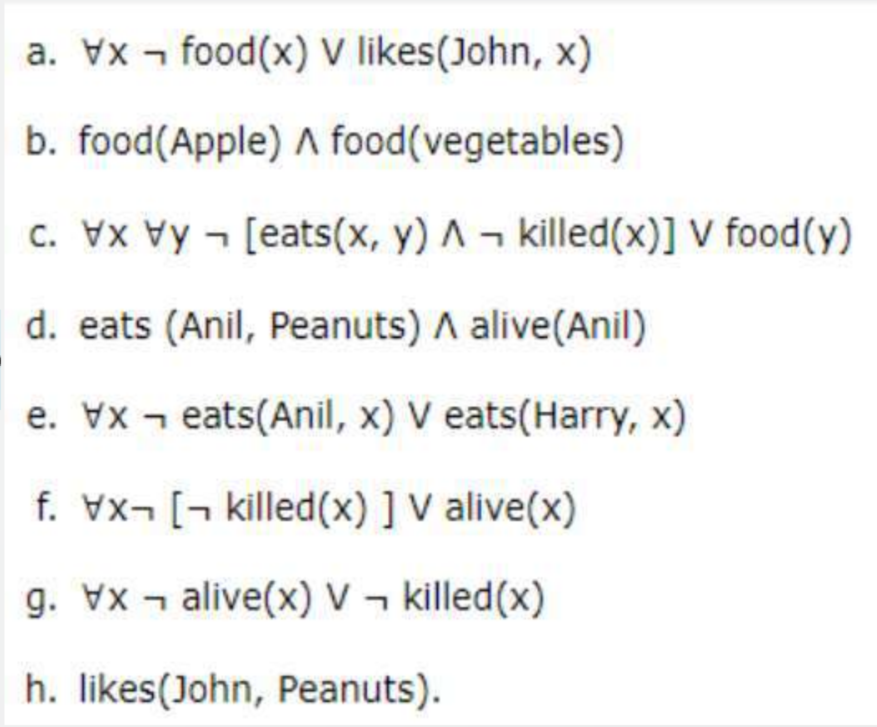
d. $\text{eats}\ (\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$.

e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

f. $\forall x\text{: } \neg\, \text{killed}(x) \rightarrow \text{alive}(x)$ } **added predicates.**

g. $\forall x\text{: alive}(x) \rightarrow \neg\, \text{killed}(x)$

h. $\text{likes}(\text{John}, \text{Peanuts})$



a. $\forall x\ \neg\, \text{food}(x) \vee \text{likes}(\text{John}, x)$

b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$

c. $\forall x\ \forall y\ \neg\, [\text{eats}(x, y) \wedge \neg\, \text{killed}(x)] \vee \text{food}(y)$

d. $\text{eats}\ (\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$

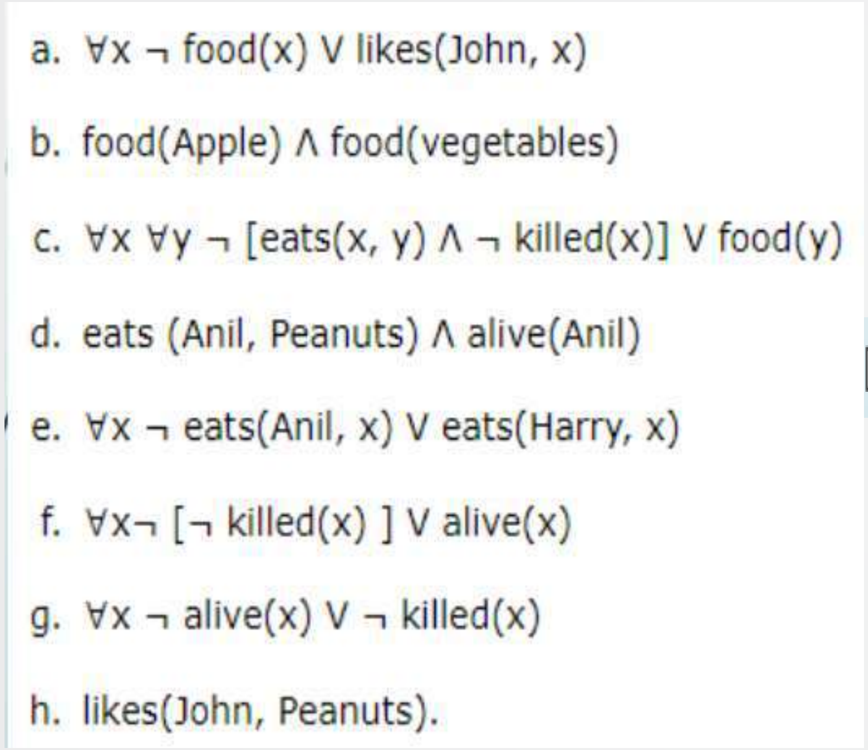
e. $\forall x\ \neg\, \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$

f. $\forall x \neg\, [\neg\, \text{killed}(x)\ ] \vee \text{alive}(x)$

g. $\forall x\ \neg\, \text{alive}(x) \vee \neg\, \text{killed}(x)$

h. $\text{likes}(\text{John}, \text{Peanuts})$.

Step-3 (Move negation (¬)inwards and rewrite)



a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x¬ [¬ killed(x)] V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x killed(x) V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

Step-4 (Rename variables or standardize variables)

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

f. ∀x killed(x) V alive(x)

g. ∀x ¬ alive(x) V ¬ killed(x)

h. likes(John, Peanuts).

a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g killed(g) V alive(g)

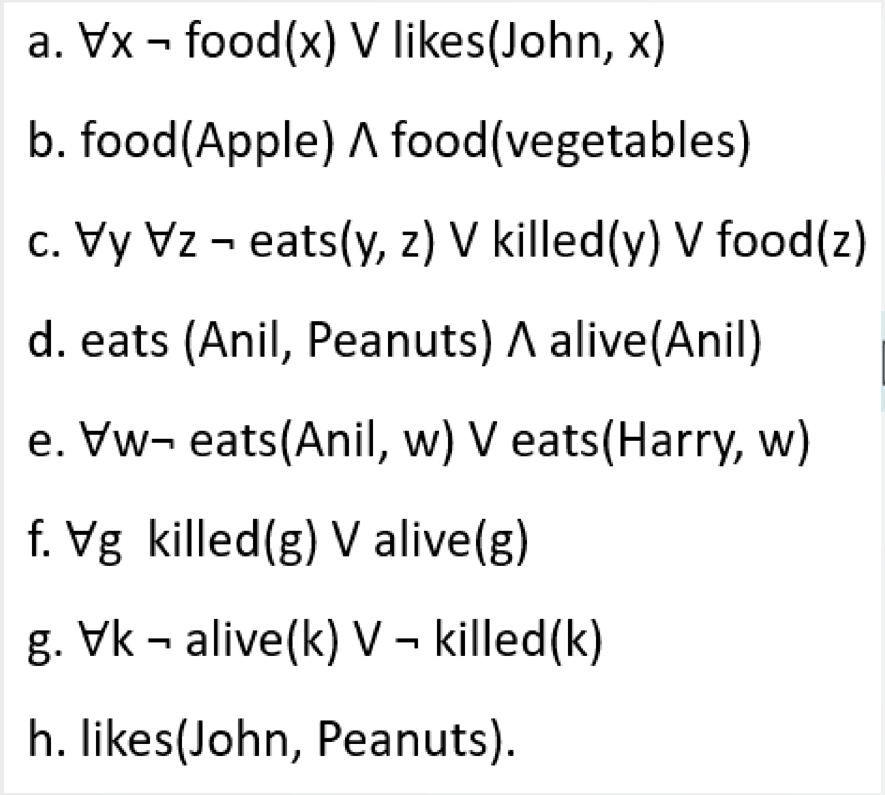
g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).

Step-5 (Eliminate existential instantiation quantifier by elimination.)

In this step, we will eliminate existential quantifier ∃, and this process is known as ***Skolemization***. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

Step-6 (Drop Universal quantifiers)



a. ∀x ¬ food(x) V likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

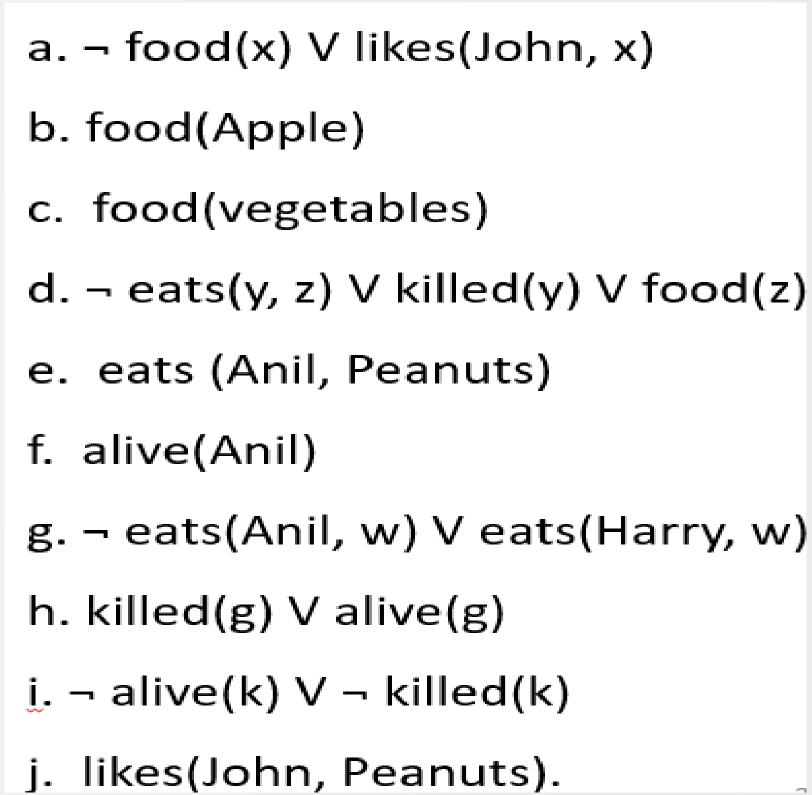
d. eats (Anil, Peanuts) Λ alive(Anil)

e. ∀w¬ eats(Anil, w) V eats(Harry, w)

f. ∀g killed(g) V alive(g)

g. ∀k ¬ alive(k) V ¬ killed(k)

h. likes(John, Peanuts).



a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).

Step-7 (Draw Resolution Graph)

a. ¬ food(x) V likes(John, x)

b. food(Apple)

c. food(vegetables)

d. ¬ eats(y, z) V killed(y) V food(z)

e. eats (Anil, Peanuts)

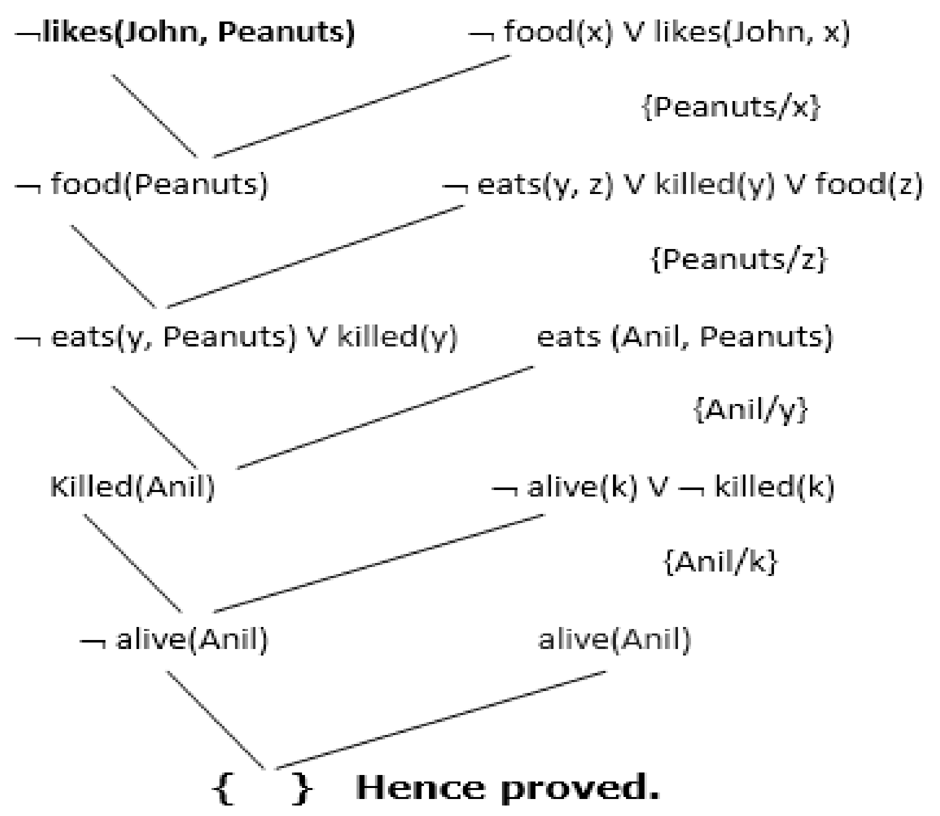
f. alive(Anil)

g. ¬ eats(Anil, w) V eats(Harry, w)

h. killed(g) V alive(g)

i. ¬ alive(k) V ¬ killed(k)

j. likes(John, Peanuts).



Prenex Normal Form (PNF)

Prenex Normal Form - Theorem

THEOREM:

Every first-order formula is equivalent to a formula in a prenex disjunctive normal form (PDNF) and to a formula in a prenex conjunctive normal form (PCNF).

Prenex Normal Form – Algorithm Steps

Any expression can be converted into prenex normal form. To do this, the following steps are needed:

1. Eliminate all occurrences of $\rightarrow$ and $\leftrightarrow$ from the formula in question. (same like CNF and DNF rules).
2. Move all negations inward such that, in the end, negations only appear as part of literals.
3. Standardize the variables apart (when necessary).
4. The prenex normal form can now be obtained by moving all quantifiers to the front of the formula.

Prenex Normal Form

To accomplish Step 1 (eliminate the →,↔), make use of the following logical equivalences:

- **$A \rightarrow B \equiv \neg A \vee B$.**
- **$A \leftrightarrow B \equiv (\neg A \vee B) \wedge (A \vee \neg B)$.**
- **$A \leftrightarrow B \equiv (A \wedge B) \vee (\neg A \wedge \neg B)$.**

Prenex Normal Form

To accomplish Step 2 (move all negations inward, such that negations only appear as parts of literals), use the logical equivalences:

De Morgan’s Laws.

- **$\neg\neg A \equiv A$.**
- **$\neg\exists x\, A(x) \equiv \forall x\, \neg A(x)$.**
- **$\neg\forall x\, A(x) \equiv \exists x\, \neg A(x)$.**

Prenex Normal Form

To accomplish Step 3 (standardize variables apart), make use of the following result.

The theorem allows one to rename the variables in order to make them distinct. Renaming the variables in a formula such that distinct variables have distinct names is called **standardizing the variables apart.**

Prenex Normal Form

To accomplish Step 4 (move all quantifiers in front of the formula) make use of the following logical equivalences:

- **$A \wedge \exists x B(x) = \exists x (A \wedge B(x))$, x is not occurring in A.**
- **$A \wedge \forall x B(x) = \forall x (A \wedge B(x))$, x is not occurring in A.**
- **$A \vee \exists x B(x) = \exists x (A \vee B(x))$, x is not occurring in A.**
- **$A \vee \forall x B(x) = \forall x (A \vee B(x))$, x is not occurring in A.**

(These equivalences essentially show that ***if a formula A has a truth value that does not depend on x, then one is allowed to quantify over x.)***

Prenex Normal Form

More logical equivalences for Step 4

- $\forall x\, A(x) \land \forall x\, B(x) = \forall x\, (A(x) \land B(x))$
- $\exists x\, A(x) \lor \exists x\, B(x) = \exists x\, (A(x) \lor B(x))$
- $\forall x\, \forall y\, A(x, y) = \forall y\, \forall x\, A(x, y)$
- $\exists x\, \exists y\, A(x, y) = \exists y\, \exists x\, A(x, y)$
- $Q1x\, A(x) \land Q2y\, B(y) = Q1x\, Q2y\, (A(x) \land B(y))$
- $Q1x\, A(x) \lor Q2y\, B(y) = Q1x\, Q2y\, (A(x) \lor B(y))$. [Where $Q1, Q2 \in \{\forall, \exists\}$.]

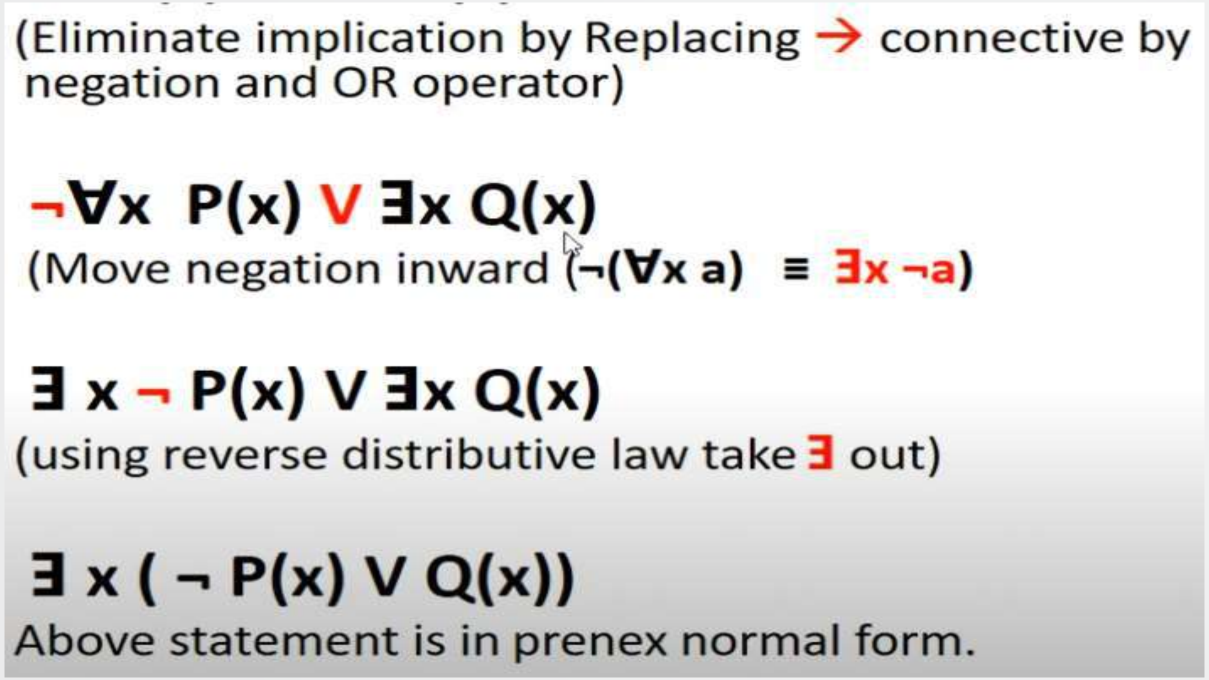
For example, if $Q1 = \forall$ and $Q2 = \exists$, we have

$$\forall x\, A(x) \land \exists y\, B(y) = \forall x\, \exists y\, (A(x) \land B(y))$$

Prenex Normal Form – Example-1

Find the Prenex normal form : $\forall x\, P(x) \rightarrow \exists x\, Q(x)$

Solution:



(Eliminate implication by Replacing $\rightarrow$ connective by negation and OR operator)

$\neg \forall x\ P(x) \lor \exists x\, Q(x)$

(Move negation inward $(\neg(\forall x\ a) \equiv \exists x\ \neg a)$

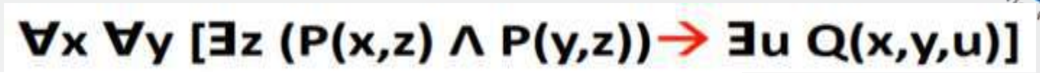
$\exists\, x\ \neg\ P(x) \lor \exists x\, Q(x)$

(using reverse distributive law take $\exists$ out)

$\exists\, x\ (\neg\ P(x) \lor Q(x))$

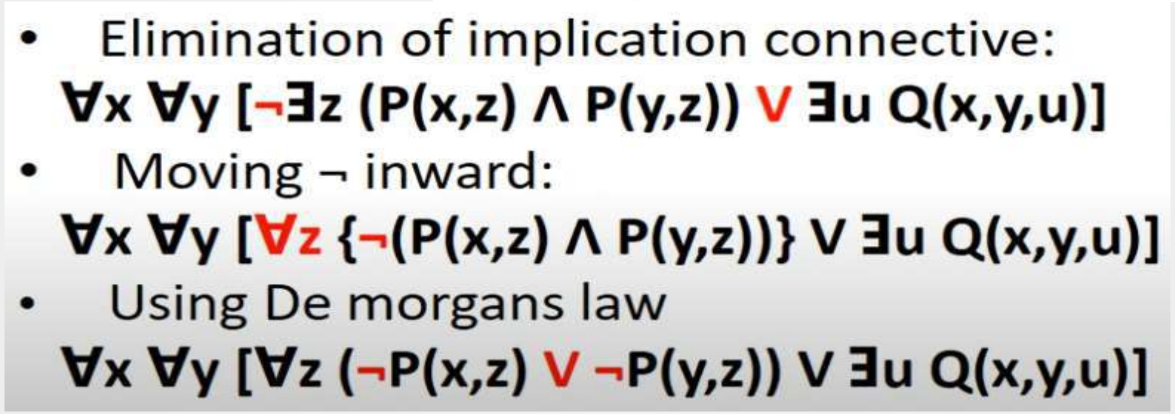
Above statement is in prenex normal form.

Prenex Normal Form – Example-2

Find the Prenex normal form : 

$\forall x \forall y [\exists z (P(x,z) \land P(y,z)) \rightarrow \exists u\, Q(x,y,u)]$

Solution:



- Elimination of implication connective:

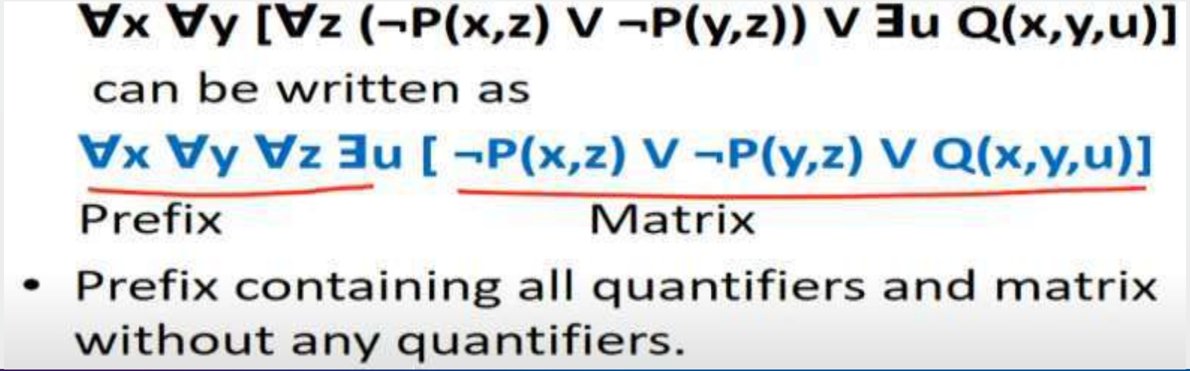
$\forall x \forall y [\neg \exists z (P(x,z) \land P(y,z)) \lor \exists u\, Q(x,y,u)]$

- Moving ¬ inward:

$\forall x \forall y [\forall z \{\neg (P(x,z) \land P(y,z))\} \lor \exists u\, Q(x,y,u)]$

- Using De morgans law

$\forall x \forall y [\forall z (\neg P(x,z) \lor \neg P(y,z)) \lor \exists u\, Q(x,y,u)]$



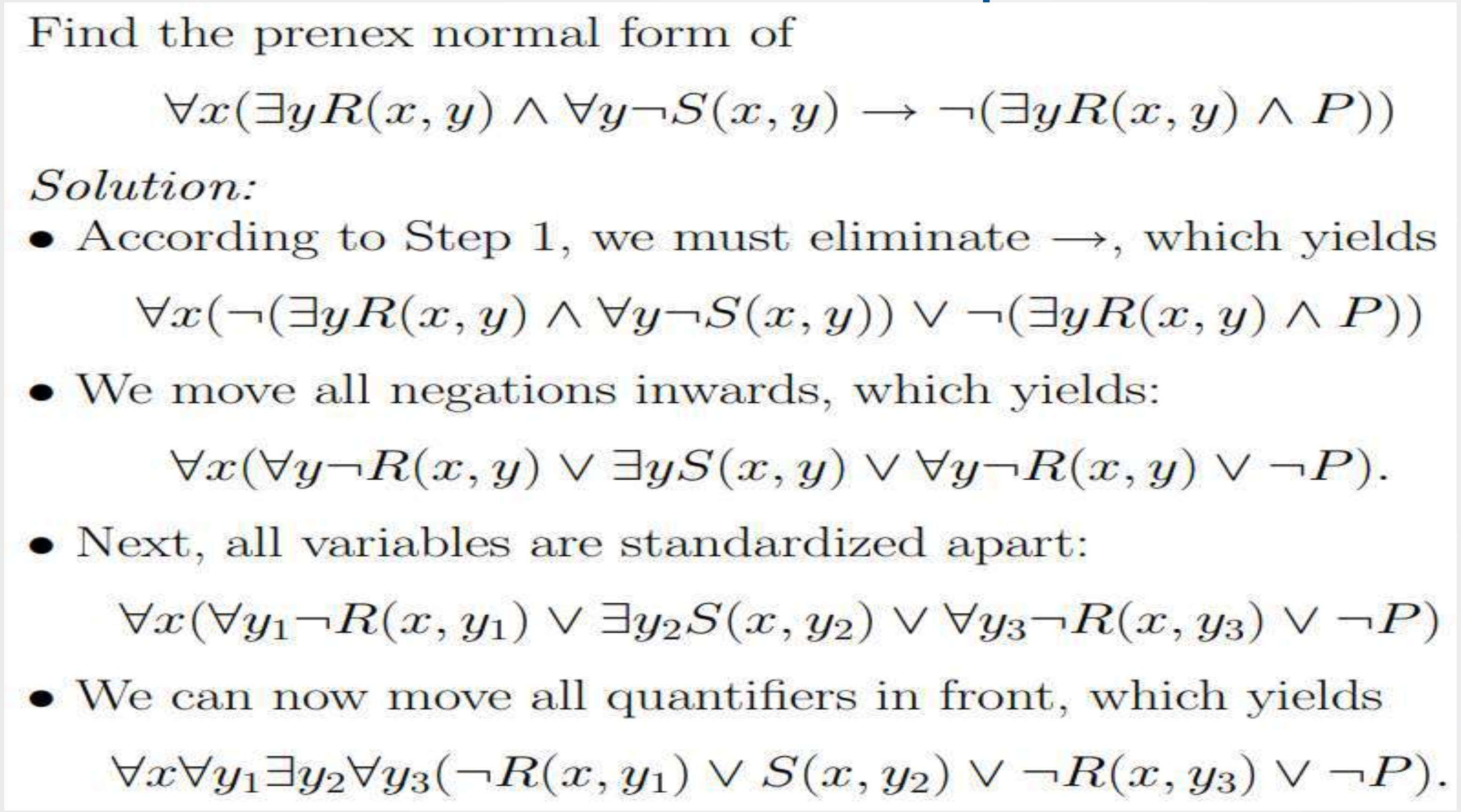
$\forall x \forall y [\forall z (\neg P(x,z) \lor \neg P(y,z)) \lor \exists u\, Q(x,y,u)]$

can be written as

$$\underbrace{\forall x \forall y \forall z \exists u}_{\text{Prefix}} [\, \underbrace{\neg P(x,z) \lor \neg P(y,z) \lor Q(x,y,u)]}_{\text{Matrix}}$$

- Prefix containing all quantifiers and matrix without any quantifiers.

Prenex Normal Form – Example-3



Find the prenex normal form of

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

Solution:

- According to Step 1, we must eliminate $\longrightarrow$, which yields

$$\forall x(\neg(\exists y R(x,y) \land \forall y \neg S(x,y)) \lor \neg(\exists y R(x,y) \land P))$$

- We move all negations inwards, which yields:

$$\forall x(\forall y \neg R(x,y) \lor \exists y S(x,y) \lor \forall y \neg R(x,y) \lor \neg P).$$

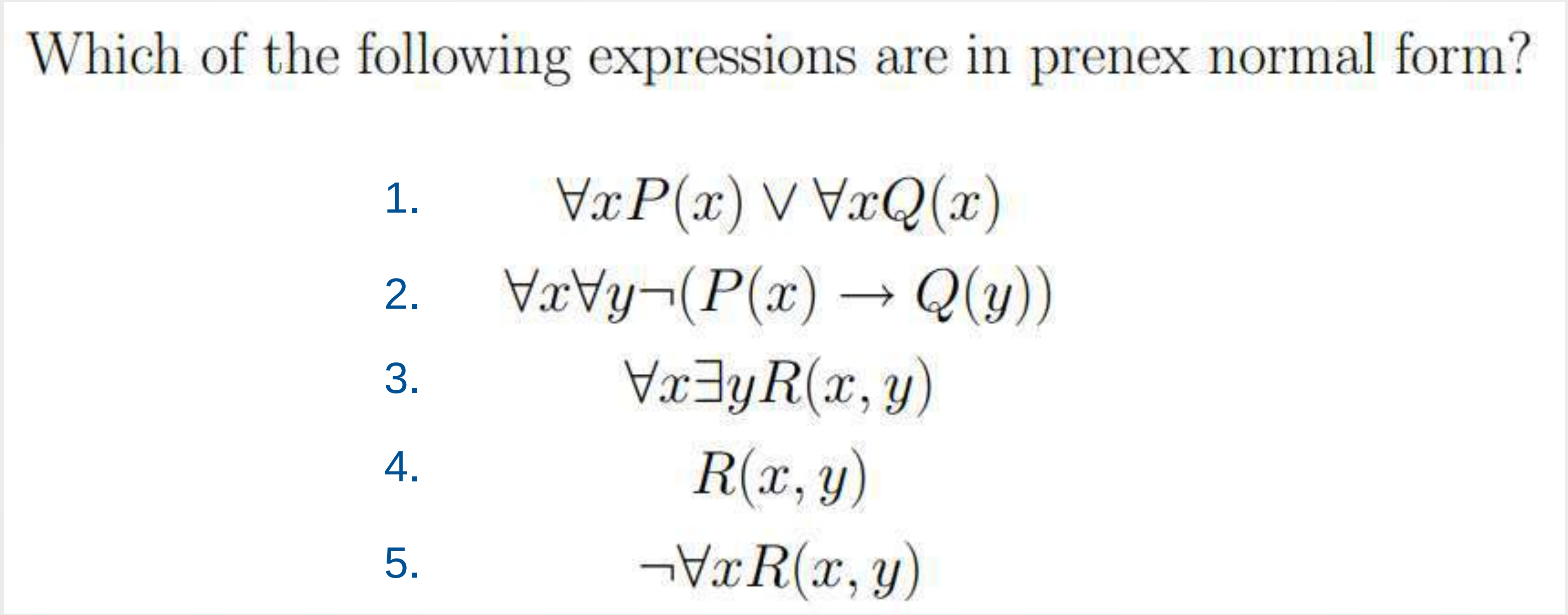
- Next, all variables are standardized apart:

$$\forall x(\forall y_1 \neg R(x,y_1) \lor \exists y_2 S(x,y_2) \lor \forall y_3 \neg R(x,y_3) \lor \neg P)$$

- We can now move all quantifiers in front, which yields

$$\forall x \forall y_1 \exists y_2 \forall y_3 (\neg R(x,y_1) \lor S(x,y_2) \lor \neg R(x,y_3) \lor \neg P).$$

Prenex Normal Form - Try



Which of the following expressions are in prenex normal form?

1. $\forall x P(x) \vee \forall x Q(x)$
2. $\forall x \forall y \neg (P(x) \rightarrow Q(y))$
3. $\forall x \exists y R(x, y)$
4. $R(x, y)$
5. $\neg \forall x R(x, y)$

Skolemization / Skolem Standard Form(SNF)

Skolemization

- In artificial intelligence and logic programming, Skolemization is a process used to ***eliminate existential quantifiers (∃) from logical formulas.***
- It is a technique often applied in first-order logic and predicate logic.
- Existential quantifiers (∃) express the existence of an object that satisfies a certain property.
- For example, the formula **$\exists x\ P(x)$** asserts that ***there exists an object x for which the predicate P is true.***
- Skolemization is used to remove such existential quantifiers by introducing ***Skolem functions*** ***or*** ***Skolem constants*****.**

Skolemization - Rules

There are some **rules** that are followed when we convert first order predict logic into into skolem standard form:-

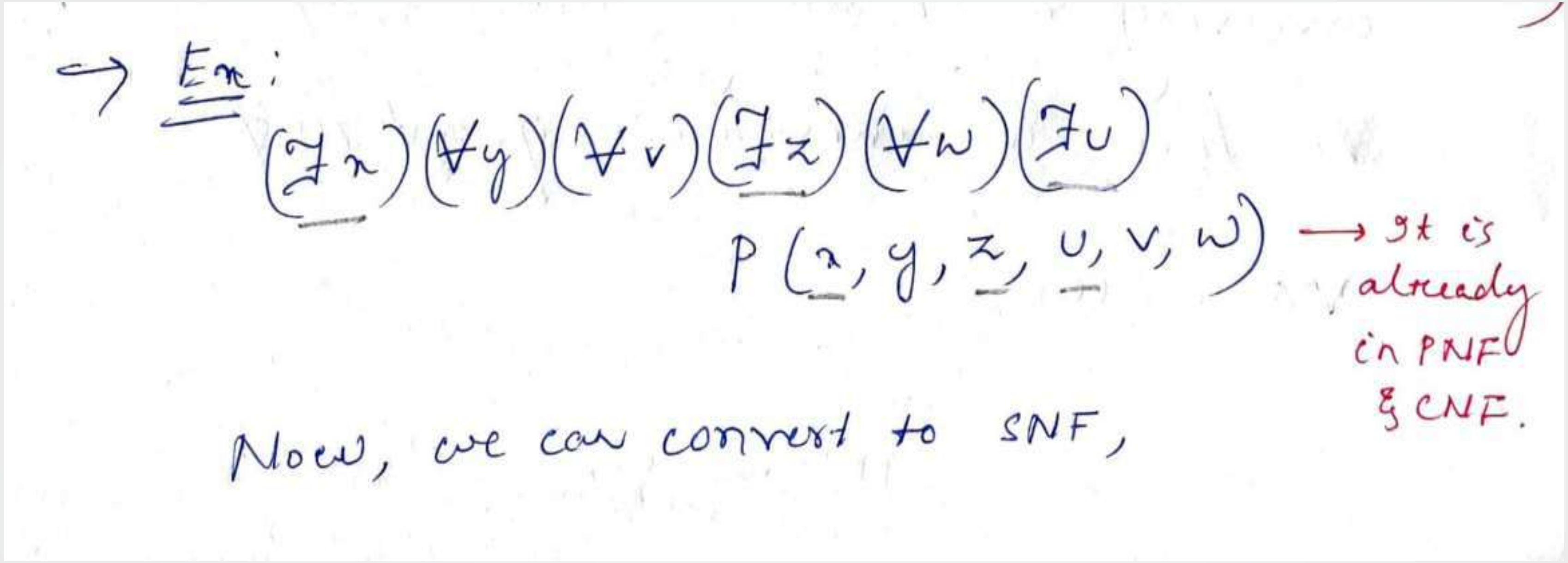
1. **Convert FOPL into PNF(Prenex Normal Form),if it is not in PNF**
2. **Convert PNF into CNF(Conjunctive normal form)**
3. **Apply skolemization**

Skolemization - Rules

Here we follow 2 steps:

- **Check all for some (∃).**
- **Before (∃x) :**
  - **If we have nothing, replace it with skolem constant.**
  - **If we get any for all (∀) symbol before it, replace with skolem function.**

Skolemization - Example



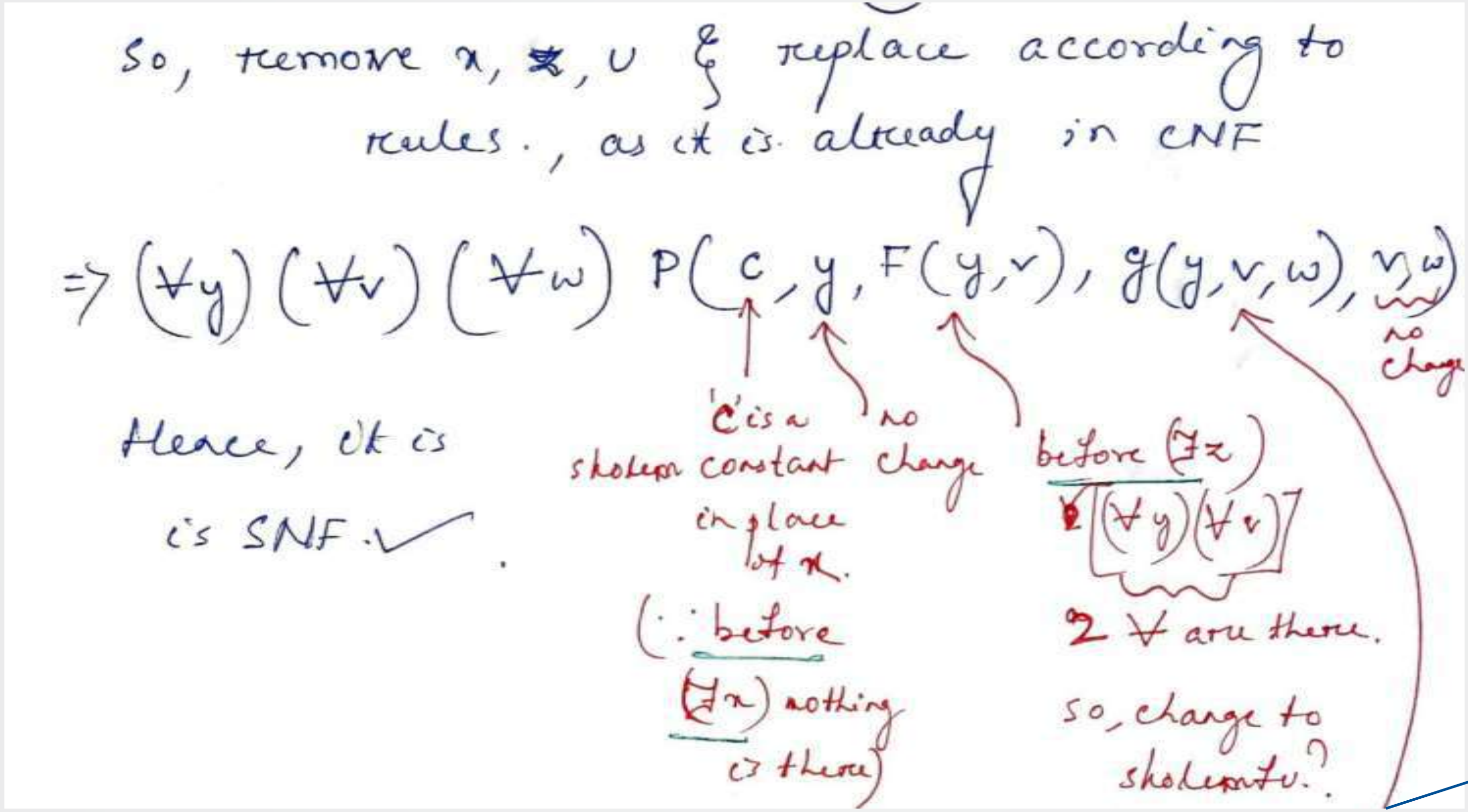
→ Ex:

$$(\underline{\exists x})(\forall y)(\forall v)(\underline{\exists z})(\forall w)(\underline{\exists u})\ P(\underline{x}, y, \underline{z}, \underline{u}, v, w) \longrightarrow$$

It is already in PNF & CNF.

Now, we can convert to SNF,

Skolemization - Example



So, remove x, ~~z~~, u & replace according to rules., as it is already in CNF

$$\Rightarrow (\forall y)(\forall v)(\forall w)\, P(c, y, F(y,v), g(y,v,w), v, w)$$

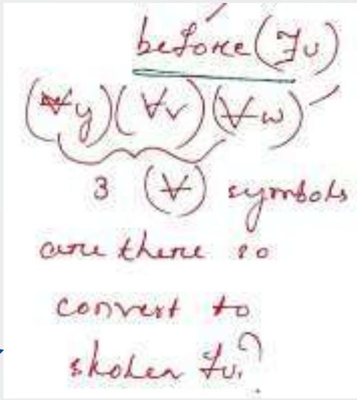
'c' is a skolem constant in place of x. ($\because$ before $(\exists x)$ nothing is there)

no change

before $(\exists z)$ $\underbrace{[(\forall y)(\forall v)]}$ 2 $\forall$ are there. So, change to skolem fn.

no change

Hence, it is is SNF ✓.



before $(\exists u)$ $\underbrace{(\forall y)(\forall v)(\forall w)}$ 3 $(\forall)$ symbols are there so convert to skolem fn.

References

- Stuart Russell and Peter Norvig, “Artificial Intelligence”, 2nd edition, Pearson Education, 2003.
- Saroj Koushik, “Artificial intelligence”.
- NPTEL

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING

Thank You