Experiment Number: 2

Aim:

Assignment on State space formulation and PEAS representation for Wumpus world Problem.

Theory:

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

Following is a sample diagram for representing the Wumpus world. It is showing some rooms with Pits, one room with Wumpus and one agent at (1, 1) square location of the world.

```
    +---------+---------+---------+---------+
4   | stench  |         | breeze  |  pit    |
    |         |         |         |         |
    |         |         |         |         |
    +---------+---------+---------+---------+
3   | wumpus  | stench  | pit     | breeze  |
    |         | breeze  |         |         |
    |         | gold    |         |         |
    +---------+---------+---------+---------+
2   | stench  |         | breeze  |         |
    |         |         |         |         |
    |         |         |         |         |
    +---------+---------+---------+---------+
1   | start   | breeze  | pit     | breeze  |
    |         |         |         |         |
    |   ==>   |         |         |         |
    +---------+---------+---------+---------+
         1         2         3         4
```

Fig.2.1 Initial state

Procedure:

PEAS description of Wumpus world:

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

Performance measure:

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

Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].

- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

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

Sensors:

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

The Wumpus world Properties:

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

Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.

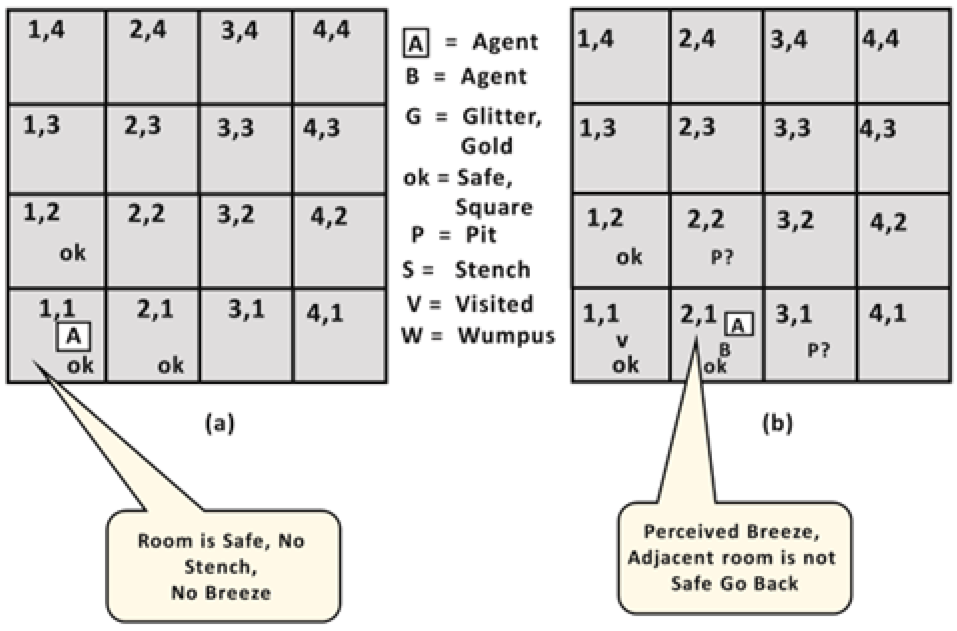


Fig 2.2 1st step

Agent's second Step:

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].

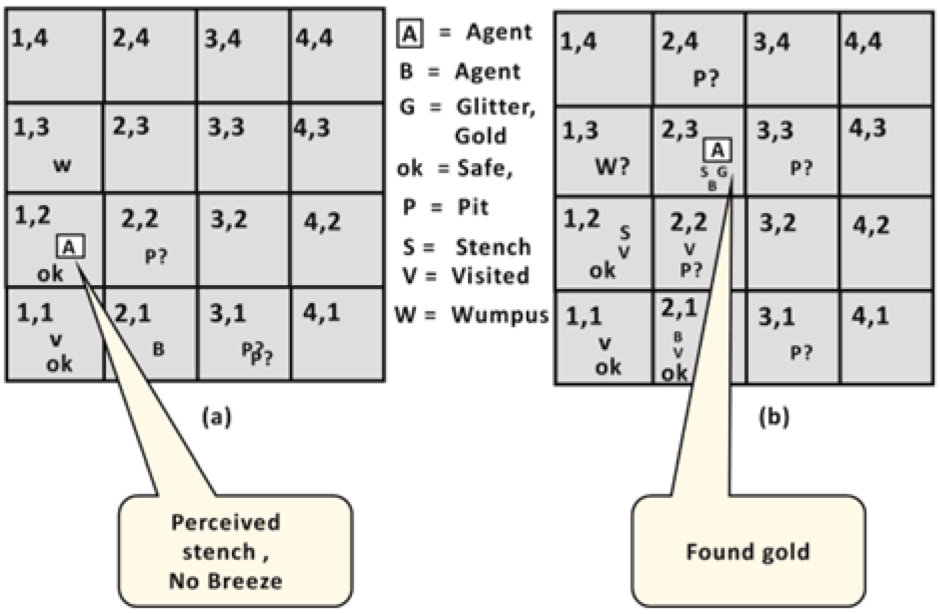


Fig 2.3 2nd step

Agent's fourth step:

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

Conclusion: Thus, we have studied PEAS representation for Wumpus world Problem.

Experiment Number: 3

Aim:

WAP for BFS algorithm using uninformed search method.

Theory:

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. Breadth-first search can be implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out (FIFO) queue, assuring that the nodes that are visited first will be expanded first. It uses two queues for its implementation: open, close Queue**.** Children are added from backend of queue.

Performance Comparison:

- Completeness: yes, it gives shallowest goal
- Optimality: yes, provided path cost is non- decreasing
- Time complexity: O(b^{d+1})
- Space complexity: O(b^{d+1})

Algorithm:

1. Create single member queue comprising of root node.
2. If 1st Member of Queue is GOAL then goto Step 5.

3. If first member of queue is not GOAL then remove it and add to CLOSE or Visited Queue. Consider its Children/ successor, if any add them from BACK/REAR [FIFO]
4. If queue is not empty then goto Step 2, If queue is empty then goto Step 6
5. Print “SUCCESS” and stop.
6. Print “FALIURE” and stop

*** Solve One example based on BFS**

Conclusion: Thus, the program of Breadth first search has been executed successfully.

.

Experiment Number: 4

Aim:

WAP for DFS algorithm using uninformed search method.

Theory:

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking. Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

Performance Analysis: Depth-first search visits every vertex once and checks every edge in the graph once. Therefore, DFS complexity is O(V + E)O(V+E). This assumes that the graph is represented as an adjacency list.

Algorithm:

1. If the initial state is a goal state, quit and return success.
2. Otherwise, loop until success or failure is signalled.

   A] Generate a state, say E, and let it be the successor of the initial state.If there is no successor, signal failure.

   B] Call Depth-First Search with E as the initial state.

   C] If success is returned, signal success. Otherwise continue in this

   Loop.

*** Solve One example based on DFS**

Conclusion: Thus, the program of Depth first search has been executed successfully.

Experiment Number: 5

Aim:

WAP for A* algorithm using informed search method.

Theory:

An informed search strategy-one that uses problem-specific knowledge-can find solutions more efficiently. Akey component of these algorithms is a heuristic function h(n)
h(n)= estimated cost of the cheapest path from node *n* to a goal node.
Admissible /heuristic never over estimated i.e. h(n)≤ Actual cost. For example, Distance between two nodes(cities)=> straight line distance and for 8-puzzel problem- Admissible heuristic can be number of misplaced tiles h(n)= 8.

A* Search technique

It is informed search technique. It uses additional information beyond problem formulation and tree. Search is based on Evaluation function f(n).Evaluation function is based on both heuristic function h(n) and g(n).
f(n)=g(n) + h(n)
It uses two queues for its implementation: open, close Queue. Open queue is a priority queue which is arranged in ascending order of f(n).

Algorithm:

1. Create a single member queue comprising of Root node
2. If FIRST member of queue is goal then goto step 5
3. If first member of queue is not goal then remove it from queue and add to close queue.

4. Consider its children if any, and add them to queue in ascending order of evaluation function f(n).
5. If queue is not empty then goto step 2.
6. If queue is empty then goto step 6
7. Print ‘success’ and stop
8. Print ‘failure’ and stop.

Performance Comparison:

- Completeness: yes
- Optimality: yes

Limitation:

- It generate same node again and again
- Large Memory is required

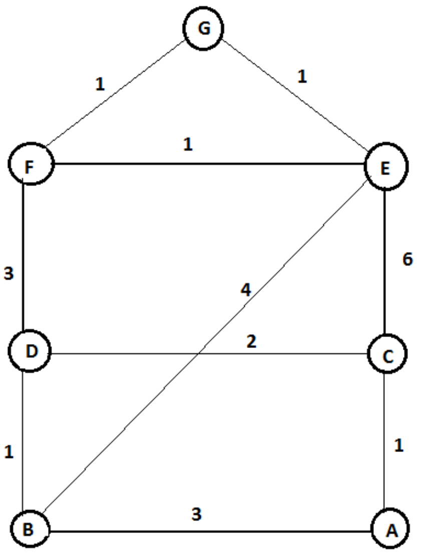


Fig.5.1. Example of A*

OPEN/FRINGE	**CLOSE**
[A]	[ ]
[C,B]	[A]
[D,B,E,A]	[A,C]

[F,E,B,C,A]	[A,C,D]
[G,E,B,C,A,D]	[A,C,D,F]
SUCCESS	

Node A:

f(B)=g(B) + h(B)=3+5=8

f(C) =g(C) + h(C)=1 + 6=7

Node C:

f(A) = g(A) + h(A)=2+7=10

f(D) = g(D) + h(D)=3+4=7

f(E) = g(E) + h(E)=7+1=8

Node D:

f(F) = g(F) + h(F)=6+1=7

f(C) = g(C) + h(C)= 5+6=11

f(B) = g(B) + h(B)=4+5=9

Node F:

f(E) = g(E) + h(E)=7+1=8

f(D) = g(D) + h(D)= 9+4=13

f(G) = g(G) + h(G)=7+0=7

Final path: A → C → D→F → G

Total cost= 7

Conclusion: Thus, the program of A*algorithm has been executed successfully

Experiment Number: 6

Aim:

WAP for Tic-Tac-Toe using game playing algorithm.

Theory:

The min-max algorithm computes the min-max decision from the current state. It uses a simple recursive computation of the min-max values of each successor state, directly implementing the defining equations. The recursion process all the way down to the leaves of the tree, and then the min-max values are backed up through the tree as the recursion unwinds.

The min-max algorithm performs a complete depth-first exploration of the game tree. If the maximum depth of the tree is m, and there are b legal moves at each point, then the time complexity of the min-max algorithm is O(bm). The space complexity is O(bm) for an algorithm that generates all successors at once, or O(m) for an algorithm that generates successors one at a time. For real games, of course, the time cost is totally impractical, but this algorithm serves as the basis for the mathematical analysis of games and for more practical algorithms.

Algorithm:

```
function MINIMAX-DECISION(state)returns an  action
inputs: state, current state in game
v ← MAX-VALUE(state)
return the action in SUCCESSORS(state) v

function MAX-VALUE(state)returns a utility value
if TERMINAL-TEST(state) then return  UTILITY(state)
```

```
v ← -infinity
for a, s in SUCCESSORS(state)do
v ← MAX(V, MIN-VALUE(S))
return v

function MIN-VALUE
if TERMINAL-TEST(state) then return  UTILITY(state)
V←  infinity
for a , s in SUCCESSORS(state)do
v ← MIN(V, MAX-VALUE(S))
return v
```

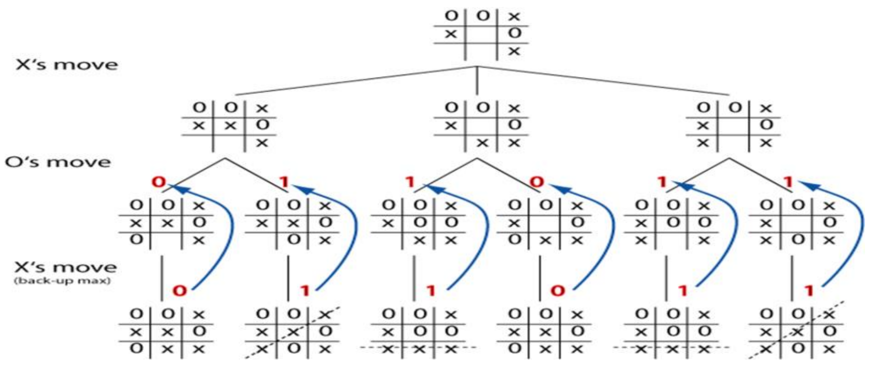


Fig.6.1. Tic-Tac-Toe

Conclusion: Thus, the program for Tic-Tac-Toe game has been executed successfully

Experiment Number: 7

Aim:

WAP for Water Jug Problem in Prolog.

Theory:

This is the jug problem using simple depth-first search of a graph. The modified water-jug problem is as follows: Jug A holds 4 liters, and jug B holds 3 liters. There is a pump, which can be used to fill either Jug. How can you get exactly 2 liters of water into the 4-liter jug?

Assumptions:

- We can fill a jug from the pump
- We can pour water out of the jug onto the ground
- We can pour water from one jug to another
- There are no other measuring devices available

To solve the water jug problem, apart from problem statement we also need a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to state is made as described in corresponding right side and the resulting state is checked to see if it corresponds to a goal state. As long as it does not the cycle continues.

Algorithm:

Rules for water jug problem

Table 7.1. Rules

Rule	Current state	New state	Rules
1	(x, y) if x<4	(4,y)	Fill the 4-gallon jug.
2	(x, y) if y<3	(x,3)	Fill the 3-gallon jug.
3	(x, y) if x>0	(x-d, y)	Pour some water out of 4-gallon jug.
4	(x, y) if y>0	(x, y-d)	Pour some water out of 3-gallon jug.
5	(x, y) if x>0	(0, y)	Empty the 4-gallon jug on ground.
6	(x, y) if y>0	(x, 0)	Empty the 3-gallon jug on ground.
7	(x, y) if (x+y)>=4 &(y>0)	(4,y-(4-x))	Pour water from 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full.
8	if (x+y)>=3 &(x>0)	(x-(3-y),3)	Pour water from 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9	(x, y) if (x+y)<=4 &(y>0)	(x+y,0)	Pour all the water from 3-gallon jug into the 4-gallon jug.
10	(x, y)if	(0,x+y)	Pour all the water from 4-gallon

	(x+y)<=3 &(x>0)		jug into the 3-gallon jug.
11	(0,2)	(2,0)	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug.
12	(2,y)	(0,y)	Empty 2 gallons in the 4-gallon jug on the ground.

Conclusion: Thus, the program of water jug problem has been executed successfully.

Experiment Number: 8

Aim:

WAP for Block world Problem using planning agent.

Theory:

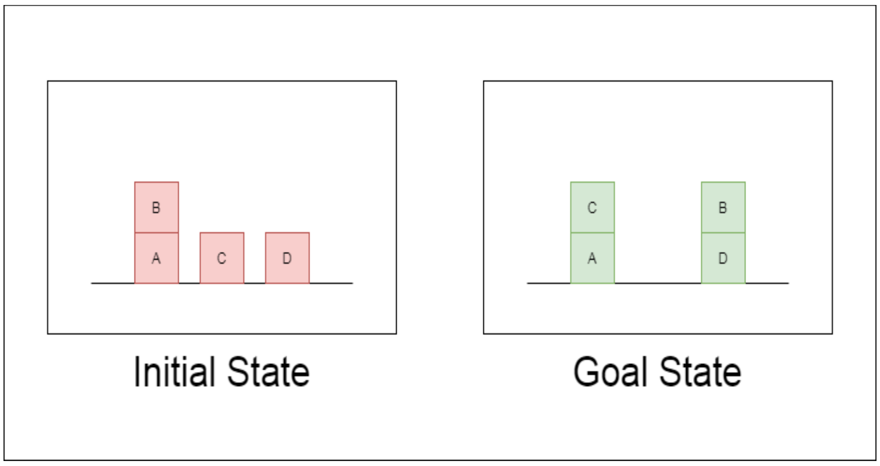


Fig. 8.1.Block word Problem

What is Blocks World Problem?

This is how the problem goes — There is a table on which some blocks are placed. Some blocks may or may not be stacked on other blocks. We have a robot arm to pick up or put down the blocks. The robot arm can move only one block at a time, and no other block should be stacked on top of the block which is to be moved by the robot arm.

Our aim is to change the configuration of the blocks from the Initial State to the Goal State, both of which have been specified in the diagram above.

What is Goal Stack Planning?

Goal Stack Planning is one of the earliest methods in artificial intelligence in which we work **backwards from the goal state to the initial state.**

We start at the goal state and we try fulfilling the preconditions required to achieve the initial state. These preconditions in turn have their own set of preconditions, which are required to be satisfied first. We keep solving these “goals” and “sub-goals” until we finally arrive at the Initial State. **We make use of a stack to hold these goals that need to be fulfilled as well the actions that we need to perform for the same**.

Apart from the “Initial State” and the “Goal State”, we maintain a **“World State”** configuration as well. Goal Stack uses this world state to work its way from Goal State to Initial State. World State on the other hand starts off as the Initial State and ends up being transformed into the Goal state.
At the end of this algorithm we are left with an empty stack and a set of actions which helps us navigate from the Initial State to the World State.

Representing the configurations as a list of “predicates”
Predicates can be thought of as a statement which helps us convey the information about a configuration in Blocks World.
Given below are the list of predicates as well as their intended meaning

1. ON(A,B) : Block A is on B
2. ONTABLE(A) : A is on table
3. CLEAR(A) : Nothing is on top of A
4. HOLDING(A) : Arm is holding A.
5. ARMEMPTY : Arm is holding nothing

Using these predicates, we represent the Initial State and the Goal State in our example like this:
Initial State — ON(B,A) ∧ ONTABLE(A) ∧ ONTABLE(C) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(C) ∧ CLEAR(D) ∧ ARMEMPTY

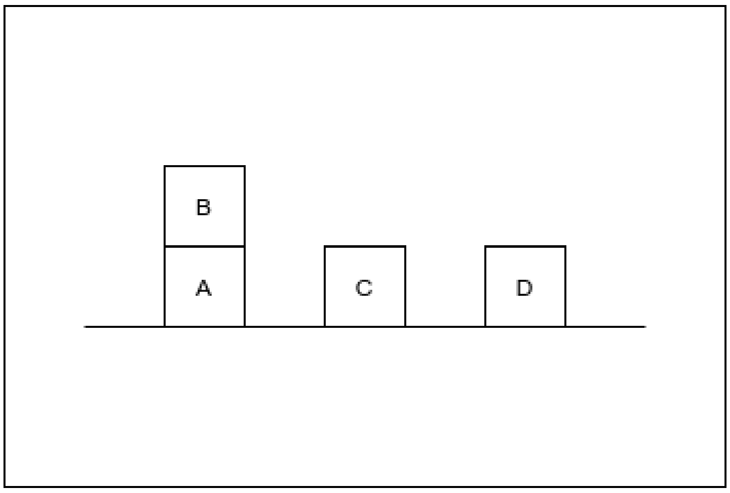


Fig 8.2 Initial State

Goal State — ON(C,A) ∧ ON(B,D) ∧ ONTABLE(A) ∧ ONTABLE(D) ∧ CLEAR(B) ∧ CLEAR(C) ∧ ARMEMPTY

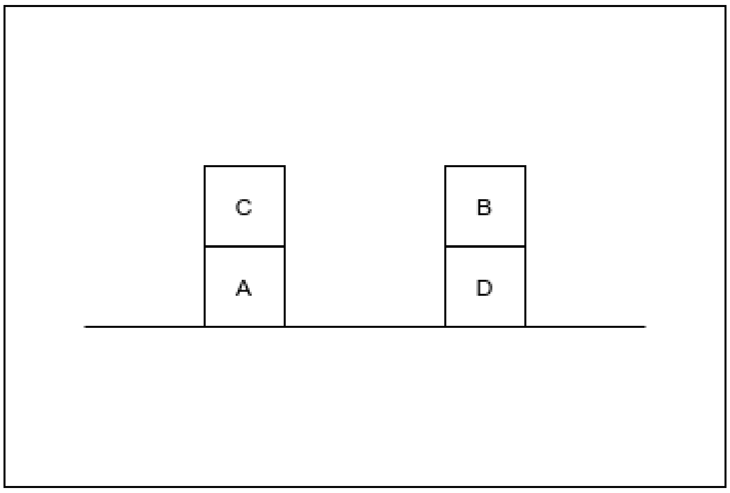


Fig 8.3 Goal State

Thus a configuration can be thought of as a list of predicates describing the current scenario.

"Operations" performed by the robot arm

The Robot Arm can perform 4 operations:

1. STACK(X,Y) : Stacking Block X on Block Y
2. UNSTACK(X,Y) : Picking up Block X which is on top of Block Y

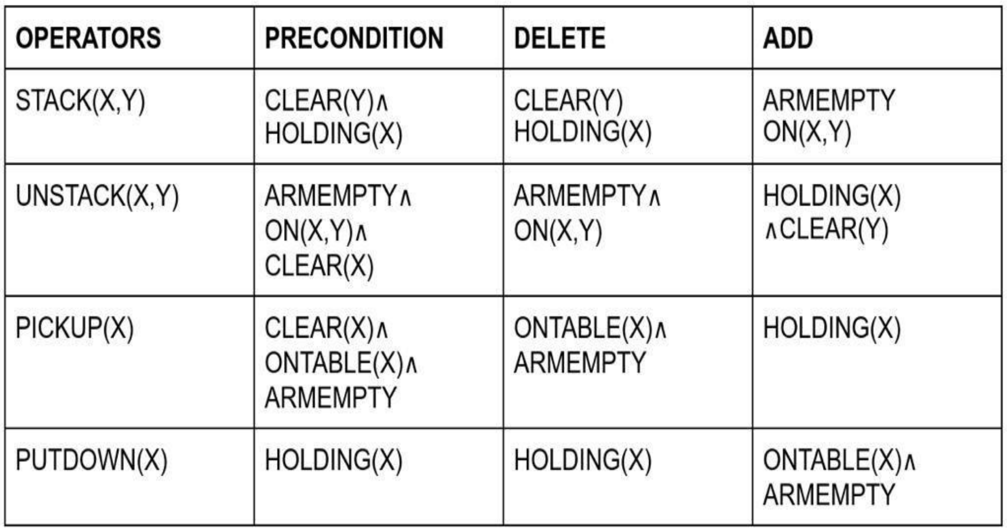
3. PICKUP(X) : Picking up Block X which is on top of the table
4. PUTDOWN(X) : Put Block X on the table

All the four operations have certain preconditions which need to be satisfied to perform the same. These preconditions are represented in the form of predicates.

The effect of these operations is represented using two lists **ADD** and **DELETE**. DELETE List contains the predicates which will cease to be true once the operation is performed. ADD List on the other hand contains the predicates which will become true once the operation is performed.

The Precondition, Add and Delete List for each operation is rather intuitive and have been listed below.

Table 8.1 Operations



OPERATORS	PRECONDITION	DELETE	ADD
STACK(X,Y)	CLEAR(Y)∧ HOLDING(X)	CLEAR(Y) HOLDING(X)	ARMEMPTY ON(X,Y)
UNSTACK(X,Y)	ARMEMPTY∧ ON(X,Y)∧ CLEAR(X)	ARMEMPTY∧ ON(X,Y)	HOLDING(X) ∧CLEAR(Y)
PICKUP(X)	CLEAR(X)∧ ONTABLE(X)∧ ARMEMPTY	ONTABLE(X)∧ ARMEMPTY	HOLDING(X)
PUTDOWN(X)	HOLDING(X)	HOLDING(X)	ONTABLE(X)∧ ARMEMPTY

Operations performed by the Robot Arm

For example, to perform the **STACK(X,Y)** operation i.e. to Stack Block X on top of Block Y, No other block should be on top of Y **(CLEAR(Y))** and the Robot Arm should be holding the Block X (**HOLDING(X)**).

Once the operation is performed, these predicates will cease to be true, thus they are included in **DELETE List** as well. (Note : It is not necessary for the Precondition and DELETE List to be the exact same).

On the other hand, once the operation is performed, The robot arm will be free (**ARMEMPTY**) and the block X will be on top of Y (**ON(X,Y)**).

Conclusion: Thus, the program for Block world problem has been executed successfully.

Experiment Number: 9

Aim:

Study of Bayes Belief Network

Theory:

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**. Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts: Directed Acyclic Graph and Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an Influence diagram.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:

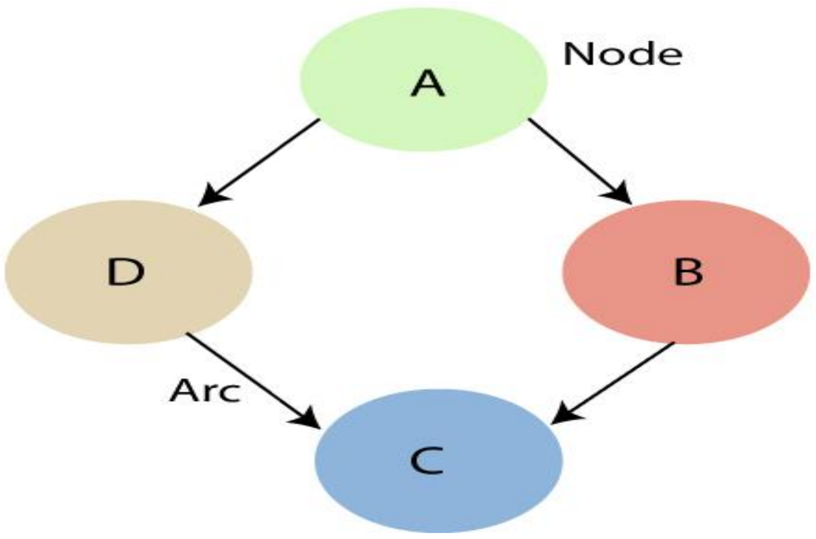


Fig 9.1 Graph

- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
    - **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
    - **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
    - **Node C is independent of node A.**

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution **$P(X_i | Parent(X_i))$**, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$P[x_1, x_2, x_3, \ldots, x_n]$, it can be written as the following way in terms of the joint probability distribution.

- **$P[x_1 | x_2, x_3, \ldots, x_n] P[x_2, x_3, \ldots, x_n]$**
- **$P[x_1 | x_2, x_3, \ldots, x_n] P[x_2 | x_3, \ldots, x_n] \ldots P[x_{n-1} | x_n] P[x_n]$.**

In general for each variable Xi, we can write the equation as:

$P(X_i | X_{i-1}, \ldots, X_1) = P(X_i | Parents(X_i))$

Conclusion: Thus, we have studied Belief networks.