Unit II

Problem Decomposition and Planning.

- Syllabus
- Problem Decompostion.
  - Goal Trees.
  - Rule Based Systems.
  - Rule Based Expert Systems.
- Planning.
  - STRIPS.
  - Forward State Space Planning
  - Backward State space Planning
  - Goal Stack Planning
  - Plan Space Planning
  - Unified Framework for Planning.
- Constraint Satisfaction.
  - N-Queen
  - Constraint Propagatⁿ
  - Scene Labeling
  - High Order & Directional Consistencies.
  - Backtracking & Lookahead Strategies.

- Problem Decomposition:-
  - The problem solving approach we have seen uptil now is to search for sequence of moves starting from given state / or a candidate solⁿ till desired state or solⁿ is reached.
  - This kind of strategy is generally forward looking trial & error.

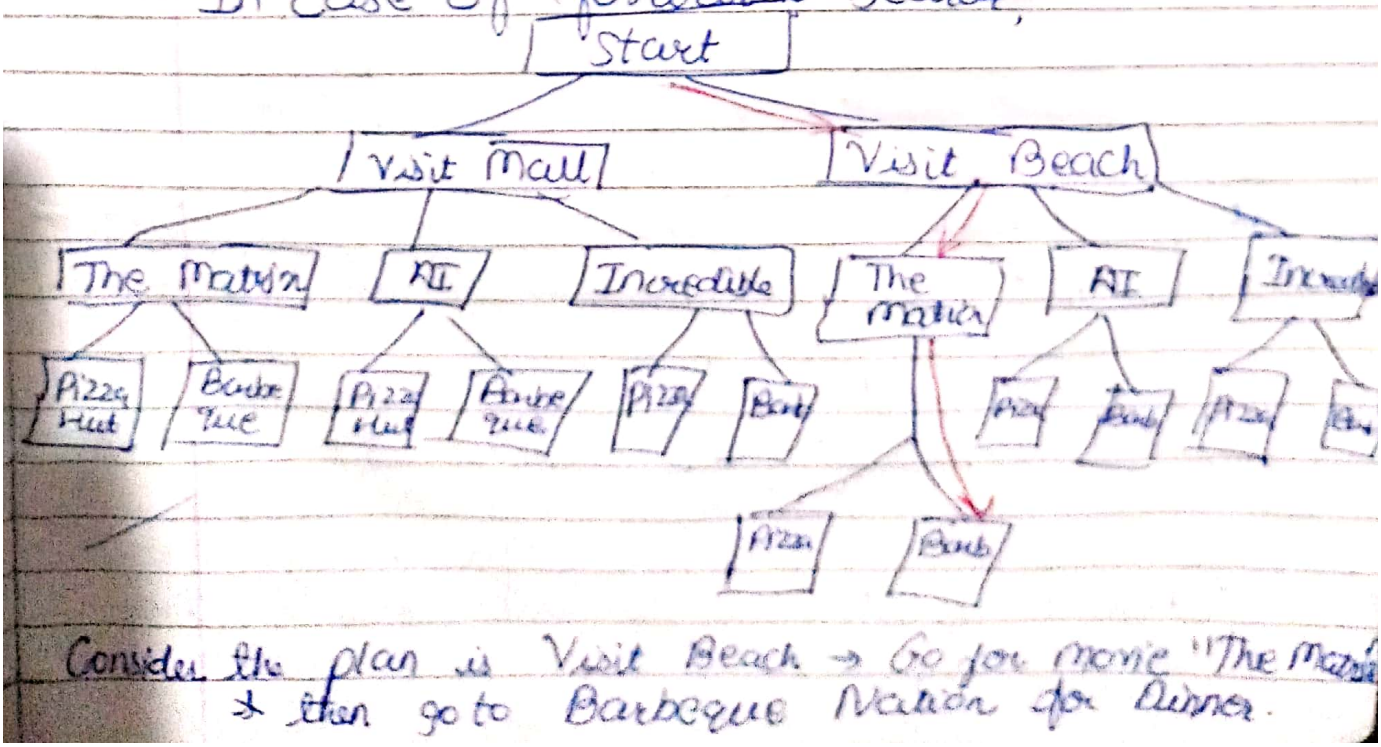
- In many situations, one can reason in backward fashion i.e from the goal to determine what needs to be done [Backward Reasoning].

- Eg:- Finding a city on a map can use both approaches.

- But there are some problems where backward reasoning can help to break up the problem into smaller parts that can be tackled independtly. This leads to search in smaller spaces.

- Eg:- Consider the task of designing a treat for your friend.
- Let us say this constitutes of fall 3 plans
  Evening plan → Visit Mall / Visit Beach
  Movie → The Matrix | AI | Incredible
  Dinner → Pizza Hut / Barbeque Nation

- In case of forward search,



Consider the plan is Visit Beach → Go for movie "The Matrix" & then go to Barbeque Nation for Dinner.

- One can inspect the tree & conclude that it is fruitless to search in the left sub-tree which has Visit Mall activity.
- DFS will waste time in exploring left-half subtree below.
- When it fails in one subtree below, it backtracks to the last choice made & tries the next subtree.
- That is, it does Chronological backtracking.
- The above search was done using bottom-up approach in which different combinat^n of moves are brought together & then tests for the goal being achieved.

- An alternative approach is top-down process where problems can be decomposed into sub-problems. This decomposition process continues till problem is easy to solve.
- The above tree using top-down approach will be:

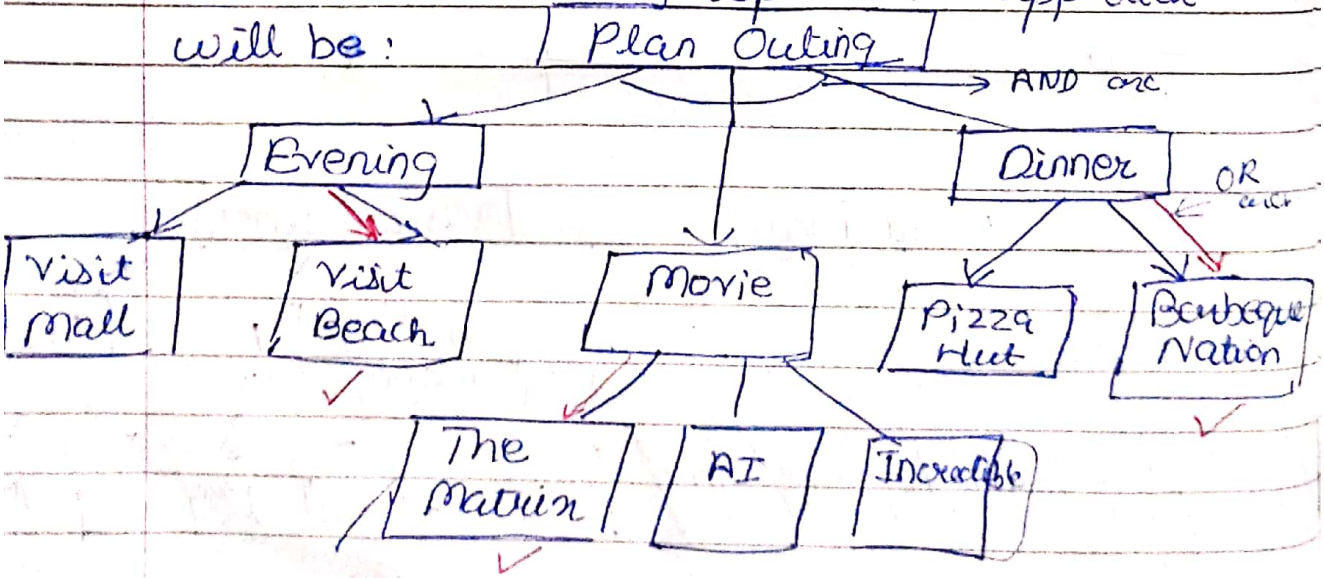


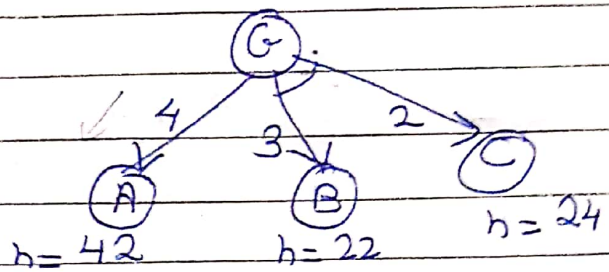
Fig:- AND-OR Tree

- Notice solution is a sub-tree rather than a path.

1) Goal Trees:-

- Heuristic fun can be used to reach goal trees.
- Heuristic fun^n estimates the cost of solving each node.

Eg:-



$\therefore$ G-A = (42+4) = 46.

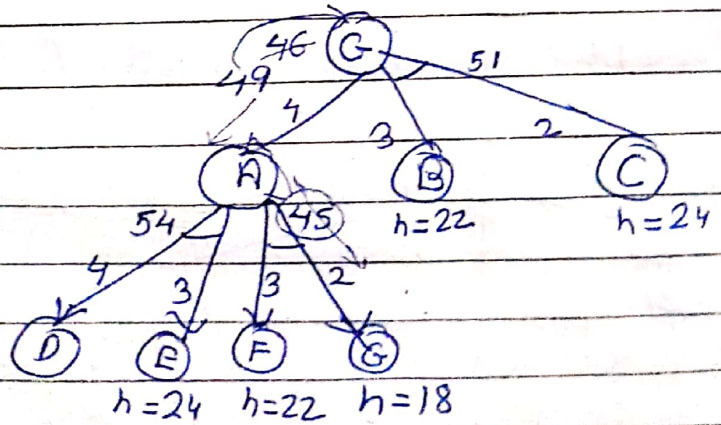
G-B-C = ((22+3)+(24+2))

= (25+26)

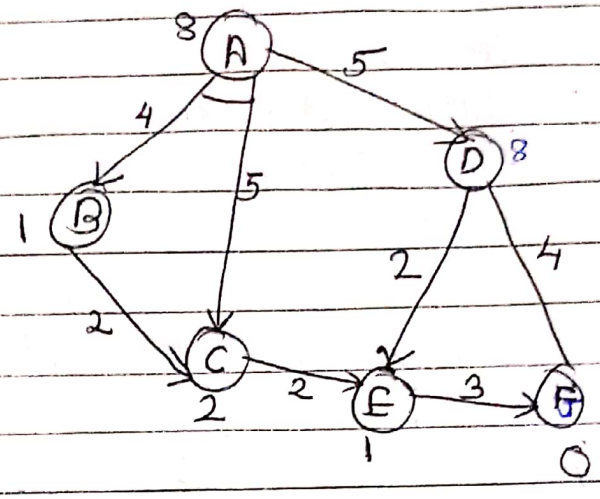
G-BC = 51

since 46 < 51 & hence G = 46.

- Node A will be expand further,



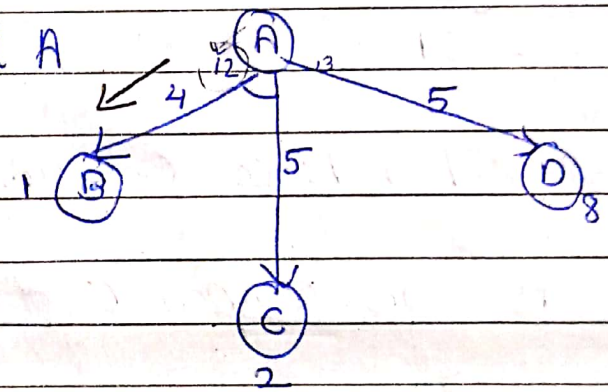
* AO* Example:-



Soln :-

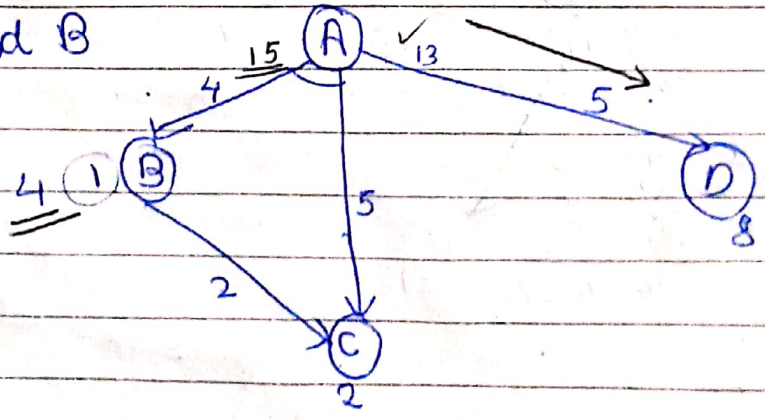
Step1 :-

Step2:- Expand A

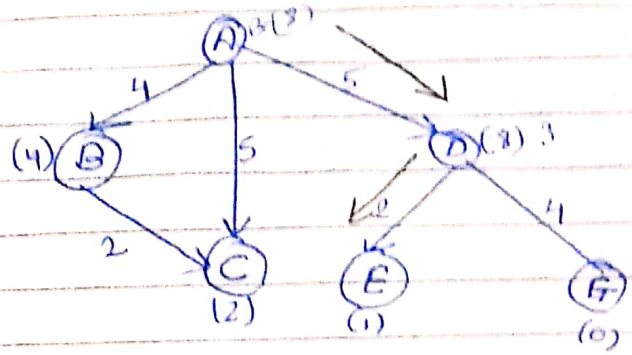


Since 12 < 13. We choose path A-BC.

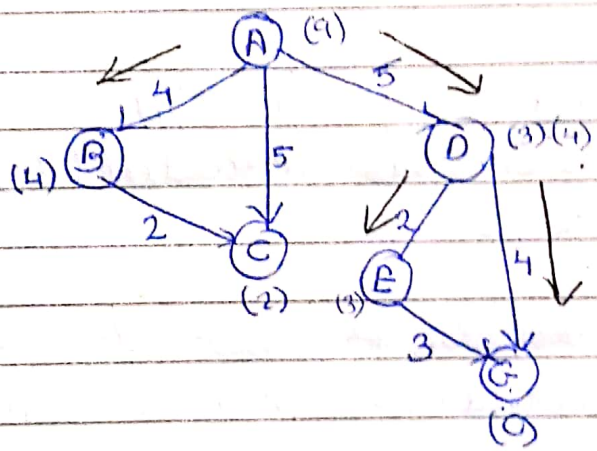
Step 3:- Expand B



Step 4 :- Expand D



Step 5:- Expand nod E



Node G can be expanded further and all its ancestors uptil root node are solved. Hence the algo. stops with A=9.

* Rule Based Systems:-

- Rule based s/m or product° s/ms have been used in general to decompose a problem & address it in parts.

- A rule is a statement of the form Left Hand side → Right Hand Side.

Eg:- Dinner → Pizza Hut / Barbeque Nation

says that the goal of having dinner can be reduced either to goal of eating at Pizza Hut or Barbeque Nation

- Product's can also be used in forward direction, in a data-driven manner.
- The production then looks like,
  Pattern → Action.

- **Rule Based Expert S/m :-**
- Also known as production systems or expert systems.
- A rule based Expert S/m is one whose knowledge base contains the domain knowledge coded in the form of rules.
- Rules are used for representing knowledge.
- Rules are expressed as a set of if-then statements.
- Rules tells what to do or what to conclude in different situations

- Components of Expert System:-
  - An expert system is typically composed of following parts:-

1. Knowledge Base:-
   - The knowledge base is a collectn of rules or other informatn derived from the human experts

- Agenda :- When the rules are satisfied by the program, they are added to a queue called as the agenda.

2. Inference Engine:-
   - It is the main processing element of expert s/m.
   - It chooses rules from the agenda to fire.

3. User Interface:-
   - A U.I is the method by which the expert s/m interacts with the user.

4. Working Memory:-
   - It contains the data that is received from the user during expert s/m session.
   - Values in working memory are used to evaluate rules in the knowledge base.

- Knowledge base / expert s/m architecture

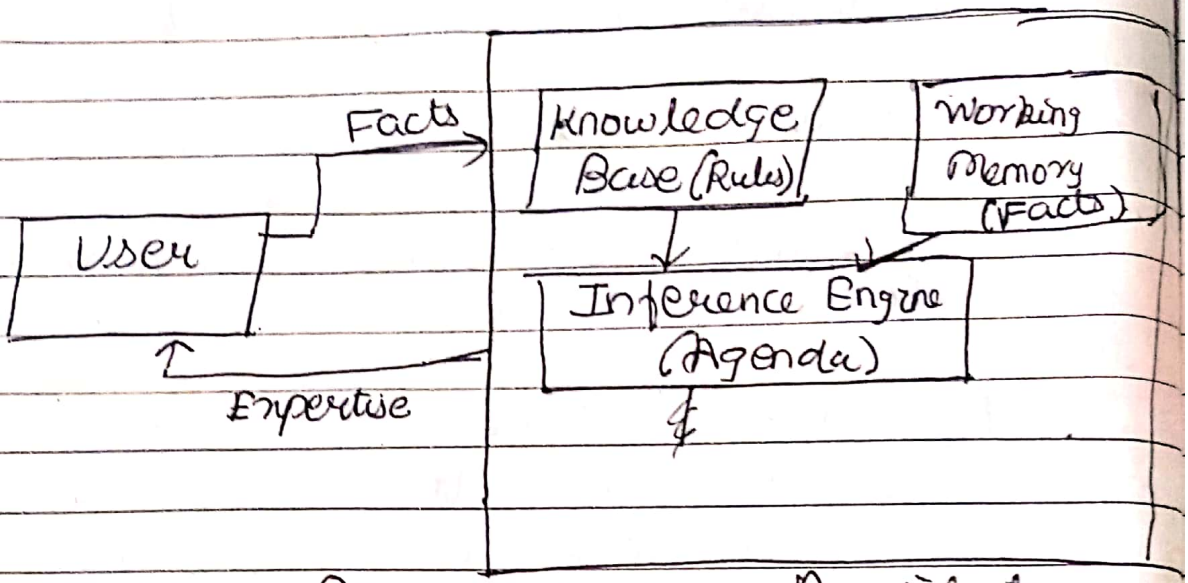


Fig:- Expert System Architecture.

- Types:-

1. Data Driven Rule Based Expert System:-

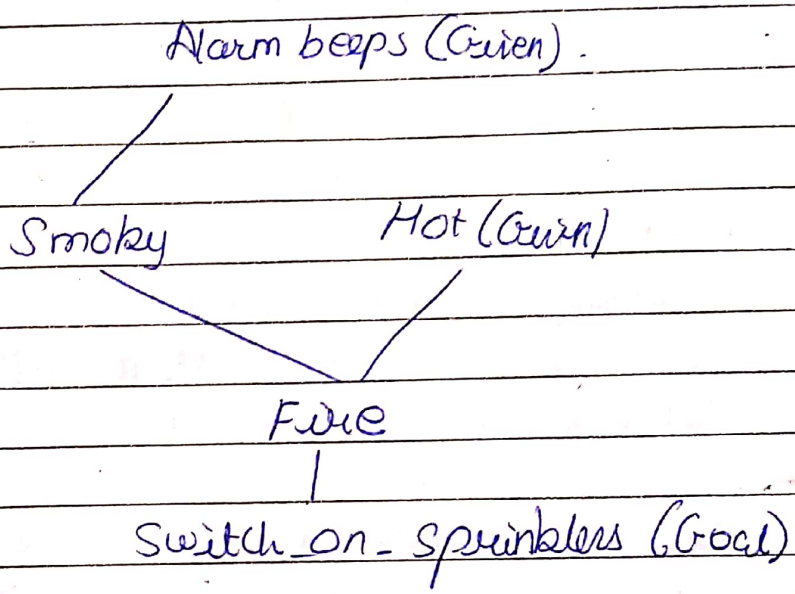
- Also called forward reasoning / forward chaining.
- The process starts with the start states i.e with given facts
- At each iteratn, next level is generated with selection of rule, whose left side matches with the root.
- The process is from left to right evaluatn of a rule.
- The process is continued till goal is achieved.

Eg:-

Rule R1: If hot and smoky Then Fire.
Rule R2: If alarm_beeps Then Smoky
Rule R3: If Fire Then Switch_on_sprinkler
Fact F1: alarm_beeps [Given]
Fact F2: hot [Given].

Goal is :- Should sprinklers be switched on?

Soln :-



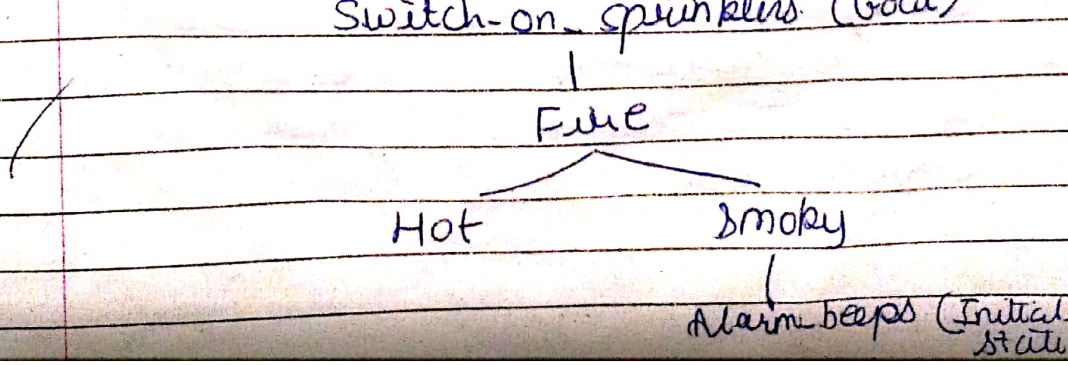
Fact F4: Smoky [from F1 by R2]

Fact F2: Fire [F2, F4 by R1].

Fact F6: Switch_on_sprinkers [F2 by R3].

2] Goal - Driven Rule - Based Expert S/m.

Soln :- For same example :-



- Also called as backward reasoning/ chaining.
- The rules start with the goal.
- The rules whose right side matches with the root are considered.
- The process of selectn is from right to left.
- Process is repeated until initial state is reached.

- Planning:-
- Planning is the process of deciding sequence of actions that will lead to achieve goal optimally.
- Planning determines how & when to do the necessary steps given the set of goals.
- Planning needs proper representatn of what has to be achieved.
- Planning involves reasoning.

- Planning Problem:-
- It is actually question that how to go the next state from the current state.

- Planning problem is defined as
1. Domain model
2. Initial State
3. Final State

• Planning Representat^{n} :-
- Representat^{n} of planning problem involves mapping of states, actions & goals.

1. State Representat^{n} :-
- State is represented as conjunction of positive literals using first order Logic (FOL).

- Literals should be the ground terms.

- Conditions that are not mentioned are assumed to be false. (Close world assumpt^{n})

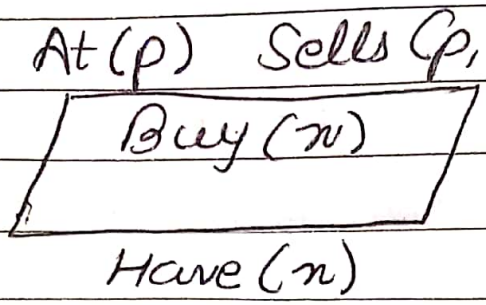
Eg:- At (Ram, Lanka) — Correct.
At (Son_of (Dashrath), Ayodhya) — Incorrect.
Funct^{n}

2] Goal Representat^{n} :-
- Goals are represented as conjunct^{n} of positive ground literals.

3] Representing Actions :-
- Action is represented using 2 components :-
1. Preconditions :-
That should hold before the action can be executed.

2) Effects / Post Conditions:-
They are the o/p that will come after executing an action.

- Example:-



Action : Buy(x)
Precondition : At(p), Sells(p, x)
Effect : Have(x).

* Planning Languages:-
- Lang. should be expressive.
- Every planning lang. make use of the representation schema so that algorithms can be operated on it.
- Few languages used for planning are:-
1. STRIPS.
2. ADL.
3. PDDL.

1. Standford Research Institute Problem Solver (STRIPS):-
- STRIPS was developed in 1970's at Standford for first intelligent robot.
- It makes use of first order predicate.
- It allows function free literals.
- Only positive literals can be present in states i.e. Poor ∧ Unknown.

- Closed world assumption is followed. It means unmentioned literals are considered false.
- Goals are conjunctions.
  Eg:- Rich ∧ Famous. Conjunctions of goals should be there.
- Effects are conjunction.
- No support for equality.
- No support for different Data Types.

Eg:- Consider car driving case-
Action - drive (c, from, to)
Pre condition - at (c, from) ∧ car(c)
Post condition - Add list: at (c, to)
Delete list: ∧ at (c, from)

Add list - List of things that becomes TRUE after performing the action.

Delete list - List of things that becomes FALSE after performing the action.

- Advantages:-
1. Easy to understand.
2. Use of propositional logic makes implementation and building of KB easy. Hence inference and reasoning becomes simple.

- Disadvantages:-
1. The language is restricted.
2. Cannot handle large no. of propositions.

2] Action Descriptor Language (ADL):-
- ADL is used to overcome the limitation of STRIPS ie ADL is more expressive than STRIPS.

Properties of ADL are:-
1. It allows negative literals.
2. It make use of quantified variables along with Conjunction and disjunction.
3. Conditions in post conditions are allowed.
4. Variables with different data types are allowed and also equality property is available.

Example:-
Action:- drive (c, from, to)
Pre condition:- at (c, from) ∧ car (c) ∧ (from ≠ to)
Post condition:- ∧ at (c, from) ∧ at (c, to).

- Not equality property is applied in this example which can't be applied for STRIPS.
- Hence, ADL is more expressive than STRIPS.

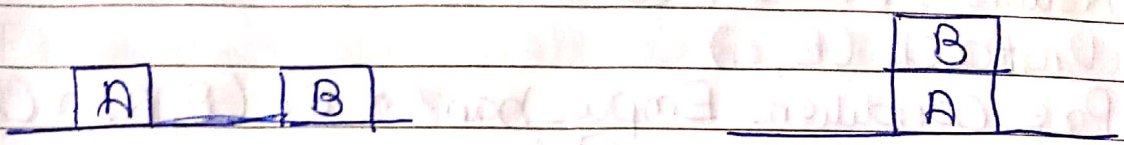
3] Planning Domain Description Language (PDDL)
- It is a standard planning language.
- It is a superset of STRIPS and ADL.
- It allows features like:-
1. Objects can have type specifications.
2. It can have negative preconditions.
3. The add and delete lists can be conditional.
4. It allows numeric values.

- PDDL is used in case of classical planning tasks.
- The planning tasks in PDDL are seperated into domain file and problem file.
- A domain file identifies planning dom
Domain file consists of:-
<PDDL code for predicates>
<PDDL code for actions>

- A problem file consists of-
(define (problem <problem_name>)
(:domain <Domain_name>)
<PDDL code for objects>
<PDDL code for intial state>
<PDDL code for goal specification>
..)

* Blocks World:-
- It is known example to demonstrate planning using STRIPS.
- Block world consists of:
1. A table
2. Identical blocks with unique letters.
3. Blocks can be put one on another to form a stack.
4. This stack is built with robot arm.

Eg:- The arm can perform operations of lifting a single block at a time and placing it.



Initial State Goal State

- Predicates used here are:-

On (A, table) — Block A is on table.
On (B, table)
On (B, A) — B is on A.
Clear (A) — Block A has nothing on it.
Holding (B): Robot's hand is holding block(B)
Empty_Hand : Robot's hand is not holding anything.

- State Representation:-

1. On(A, table) ∧ On (B, table) ∧ Clear (A) ∧ Clear(B) ∧ Empty Hand ——— Intial State.

2. On (A, table) ∧ Holding (B) ∧ Clear (A)

3. On (A, table) ∧ On (B, A) ∧ Clear (B) ∧ EmptyHand.

- Actions:-

1. Unstack (B, A) - To lift/pop block B from A.
2. Stack (B, A) - To push/add block B from A.
3. Lift (B) - Lift block B from table.
4. Place (B) - To put block B on table.

• Actions Performed:-

1. Unstack (B, A).

Pre Condition: Empty_hand ∧ On (B, A) ∧ Clear(B) ∧ On (A, table)

Post Condition: Delete list: Empty hand ∧ On (B, A) ∧ Clear(B)

Add list: Holding (B) ∧ Clear (A) ∧ On (A, table).

2. Stack (B, A):

Pre Cond :- Holding (B) ∧ Clear (A) ∧ On (A, table)

Post Cond :- Delete list: Holding (B) ∧ Clear (A)

Add list: On (B, A) ∧ Clear (B) ∧ Empty hand ∧ On (A, table).

3. Left (B)

Pre Condtⁿ :- On (B, table) ∧ Clear (B) ∧ Empty hand ∧ On (A, table) ∧ Clear (A)

Post condtⁿ :-

Delete list :- On (B, table) ∧ Clear (B) ∧ Empty Hand

Add list :- Holding (B) ∧ On (A, table) ∧ Clear (A).

4. Place (B) :-

Pre Cond :- Holding (B) ∧ On (A, table) ∧ Clear (A)

Post Cond :-

Delete list :- Holding (B)

Add list :- On (B, table) ∧ Clear (B) ∧ Empty hand ∧ On (A, table) ∧ Clear (A)

* Planning Approaches:-
- Basically there are 2 approaches for planning:-

1) State Space ~~Search~~ Planning:-
- Also called as state space search.
- It works on state & operators.
- Search takes place is both direction i.e forward & backward.

a) Forward State Space Planning / Progression Planning:-
- Planning can be from intial state to goal state.

b) Backward State Space Planning / Regression Planning:-
- Planning starts from goal state to initial state.

2] Plan Space Planning:-
- Search is carried out through space of planning.
- We begin with some plan that may be incorrect / incomplete.
- Ordering of actions is done to get correct plan.
- Steps may be added or deleted.

Eg:- Partial Order Planning.

1.] Forward State Space Planning [FSSP]:-
- The search algos start with the given state as start state, generate set of sucessor states, generate set of successor states, & search through them generating more successors till they find a state that satisfies the goal conditions.

- Effects (a) denote all the effects of action (a).

- Let effects +(a) denote set of positive effects of action a. ⇒ Add list.

- Delete list is given by −(a).

- Hence new state s',

$$s' \leftarrow s - \text{effects}^-(a) \cup \text{effects}^+(a)$$

i.e. we delete facts that are not true after the action, & add the facts that become true, resulting a new state s'.

- Let us define a funct^n progress (A, S) which returns sucess state when action A is applied to the state S.

- Drawback :-
- Huge search space will be generated
- Search algo will have no sense direction

Eg:-

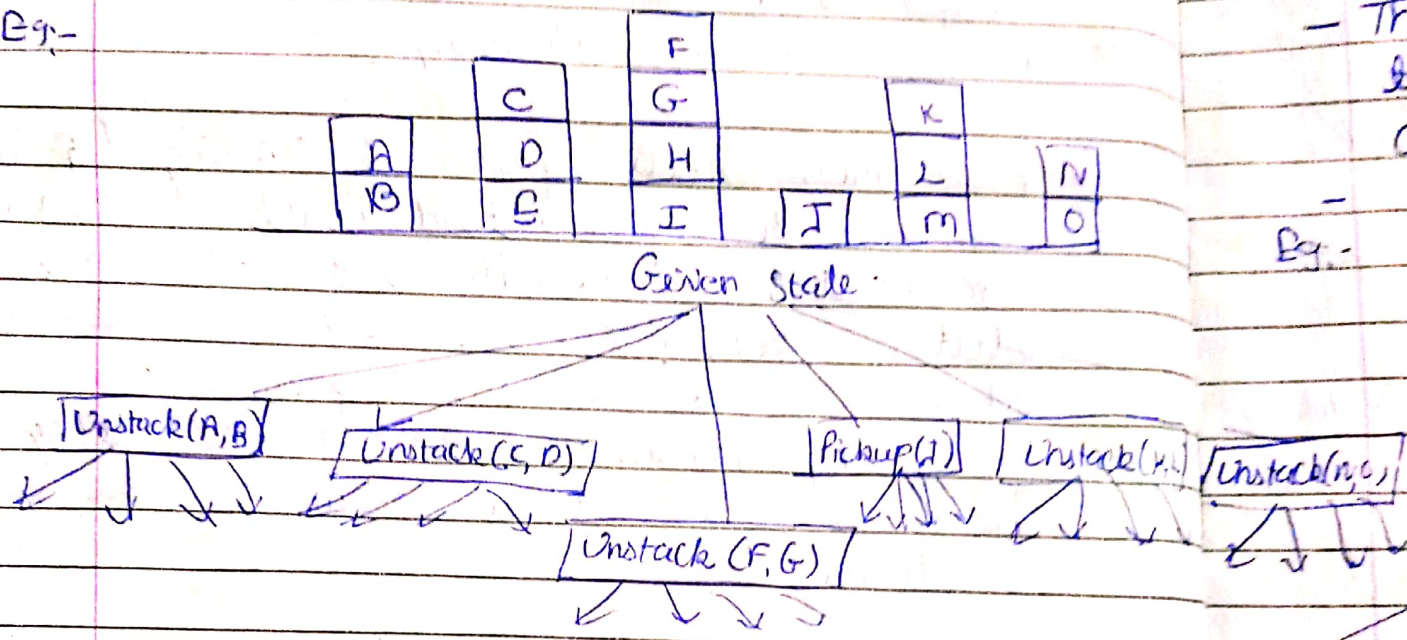
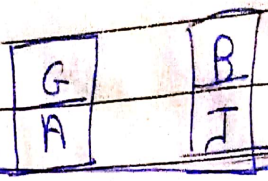


Fig:- Search space generated by FSSP is huge

2) Backward State Space Planning:-
- Here, plan search is started from goal state to initial state.
- Main advantage of backward search is that relevant actions are considered

- Here regression funct^n is used
- It allows us to move back from a set of goal clauses to a set of subgoal clauses.

Eg:- Goal State is

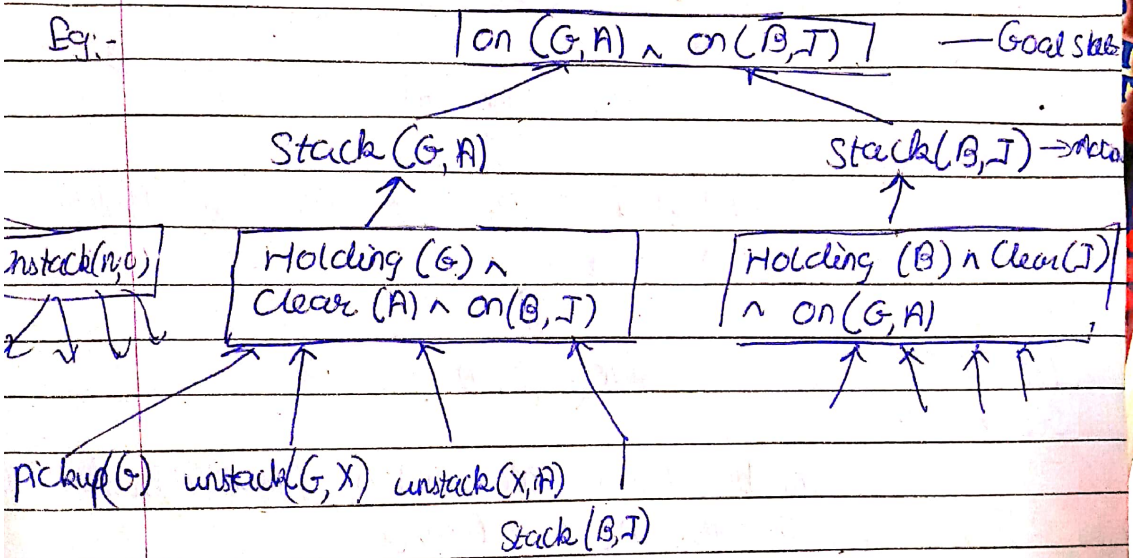


- Given a goal g and a relevant action a, one can regress to the goal g' as follows:-
  $g' \leftarrow \{g - effects(a)\} \cup preconditions(a)$.

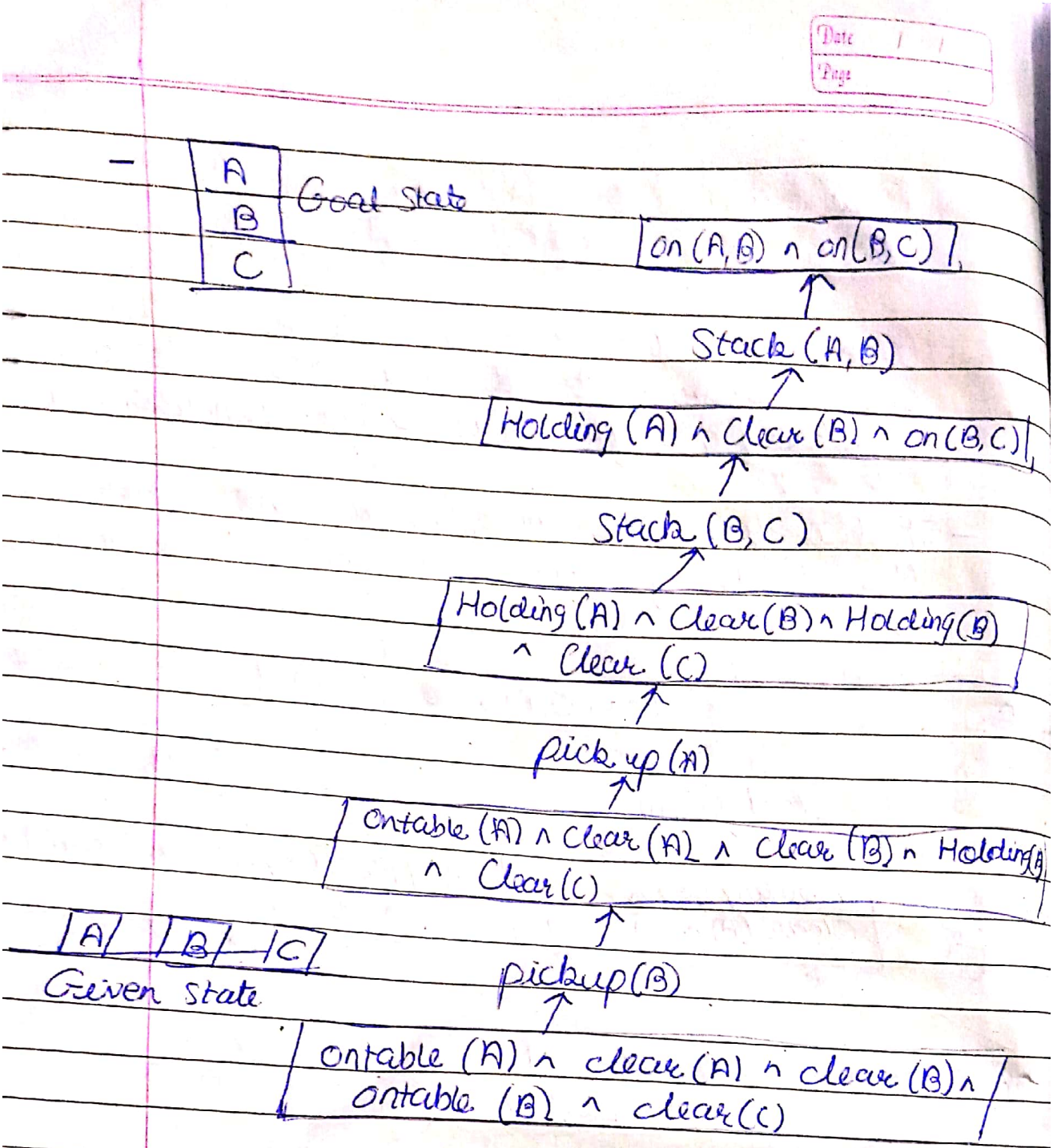
- That is from the set of goal facts, remove the effects of action and add precondtn of actions.

-

Eg:-



-



* Goal Stack Planning:-

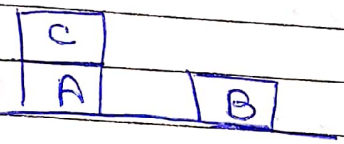
- Main problem with BSSP is goal regression is not sound & algo may be handling goal descriptions that are not consistent with any state.

- Adv. of BSSP over FSSP is that it focuses on goal so search space generated is smaller.

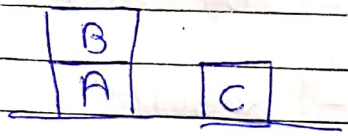
- FSSP is on other hand is sound but has large branching factor.

- Soln is to combine the focused search of BSSP with soundness of FSSP.
- It considers actions by reasoning in backward manner, but commits itself to actions only in forward manner.

Eg:- Initial State



Goal State



- In goal stack planning, a stack of goals is maintained.
- Accordingly, the corr. actions are carried out to get the soln.

Goal state:- On(A, table) ∧ On(B,A) ∧ On (C, table) ∧ Clear (C) ∧ Empty-Hand ∧ Clear(B).

- For simplicity, we neglect clear & empty_hand conditions.

- The refined goal will be
On(A, table) ∧ On(C, table) ∧ On(B, A).

- The stack for goal will be.
On (C, table)
On (B, A)
On (B, A) ∧ On (C, table) ∧ On (A, table).

- Here last operat" leads to final goal, where A is already placed on table.

Soln On (C, table), we need,
- Place (C) and Unstack (C, A).

- So, we replace the On (C, table) & add this to it.
  Unstack (C, A)
  Place (C)
  On (B, A)
  On (B, A) ∧ On (C, table) ∧ On (A, table).
- For unstack (C, A), pre-conditions are to be added.
  On (A, table)
  Clear (C)
  On (C, A)
  Empty_hand
  Empty_hand ∧ On (C, A) ∧ Clear (C) ∧ On (A, table)
  Unstack (C, A)
  Place (C)
  On (B, A)
- On (B, A) ∧ On (C, table) ∧ On (A, table)

- The first state, of A block on the table is true, hence we remove it.
- Further, clear C is also valid & hence it is also removed.
- On (C, A) & Empty_hand are also valid so remove them.

- Finally, stack has :
  Unstack (C, A)
  Place (C)
  On (B, A)
  On (B, A) ∧ On (C, table) ∧ On (A, table).

- The sequence is continued.
- Once the 1st goal is achieved, the next goal is looked upon.
- Similarly, the actions & condtns are added.

- This approach is good for smaller problems but complicated for the larger ones.

* Constraint Satisfaction Problems:-

1) N-Queen:-

- Consider the problem of placing N queens on an N×N chessboard, so that no queen attacks another.

- N-queen problem can be represented as a CSP problem.
- A constraint satisfaction problem consists of 3 components..

1. A set of variables.
2. A set of values for each variable.
3. A set of constraints between various collectn of variables.

. Constraint..

* In N-Queens:

- No queen can attack any other queen.
- Even any two queens Q_i and Q_j they cannot attack each other.
  ∴ Q_i cannot attack Q_j $(i \neq j)$

- Queens can attack each other.

1. If they are placed in same column.
2. If two queens are placed in same row.
3. Along a diagonal.

Problem:-

$X = \{Q_1, Q_2, Q_3, Q_4\}$

$D = \{D_1, D_2, D_3, D_4\}$ where $D_1 = D_2 = D_3 = D_4 = \{a, b, c, d\}$.

$C = \{C_{jk} / 1 \le j \le k \le 4\}$ where $S_{jk} = \{Q_j, Q_k\}$

Eg:-

	a	b	c	d	e	f
1	Q_1	x	x	x	x	x
2	x	x				
3	x		x			
4	x			x		
5	x				x	
6	x					x

6 x 6 queen problem.

	a	b	c	d	e	f
1	Q_1					
2			Q_2			
3					Q_3	
4		Q_4				
5	x	x	x		x	x
6	x	x	x	x	x	x

Fig:- Intermediate sol^n - After placing Q_4, there is no place for queen 6.

Sol^n:-

	1	2	3	4	5	6
1				Q_1		
2	Q_2					
3					Q_3	
4		Q_4				
5						Q_5
6			Q_6			

4 Queen problem:- Soln:-

	1	2	3	4
1		Q_1		
2				Q_2
3	Q_3			
4			Q_4	

8 queen problem soln:-

	1	2	3	4	5	6	7	8
1			Q_1					
2	Q_2							
3							Q_3	
4					Q_4			
5								Q_5
6		Q_6						
7				Q_7				
8						Q_8		

* Constraint Satisfaction Problems:-
- Constraints are the natural medium for people to express problems in many fields.
- Many real problems in AI can be modeled as Constraint Satisfaction problems and are solved through search.
- Eg. of contsraint:
The sum of three angles in a triangle is 180°.

- Constraint is a logical relation among variables.
- Constraint satisfaction is a process of finding a solution to a set of constraints.

• Types of Constraints -
1. Unary Constraint which involves a single variable. Eg:- South ~~A~~frica ≠ green. (~~green~~ ~~is variable~~)

2. Binary Constraint involves pair of variables. Eg:- South Africa ≠ Washington.

3. Higher Order Constraints:-
Involves 3 or more variables.
Eg- Professors A, B, C cannot be on committe together.
- Can always be represented by multiple binary constraints.

4. Preference (Soft) Constraints:-
Eg:- Red is better than green.

* Constraint Satisfaction Problems Representation:
- A CSP consists of :-
1. Variables -
A finite set $X = \{x_1, x_2, \ldots x_n\}$ where $x_1, x_2, x_3, \ldots x_n$ are the variables.

2. Domain -
A finite set D_i of possible values that each variables x_i can take.

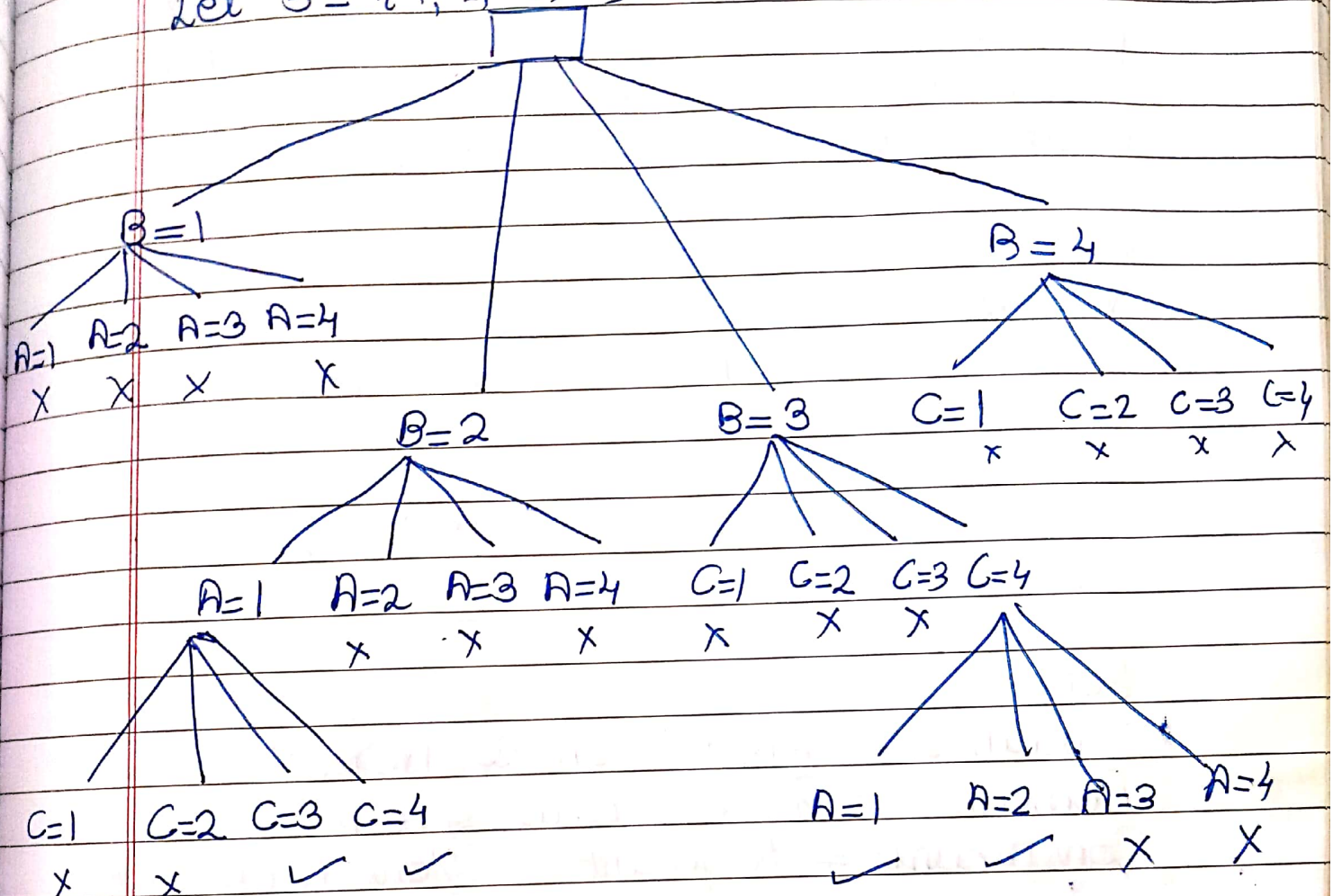
3. Constraints:-
A set of ~~solutio~~ values that variables can simultaneously satisfy the constraint.
Eg:- $(D_1 != D_2)$.

• Example:-
- Suppose you a CSP with variables A, B, C each with domain (1, 2, 3, 4). Suppose the constraints are A < B and B < C. Draw a possible search tree.

Soln:-
Variables ⇒ {A, B, C}.
Domain ⇒ {1, 2, 3, 4}
Constraint ⇒ 1.) A < B
2.) B < C

So the search tree will be,

Let B = {1, 2, 3, 4}.



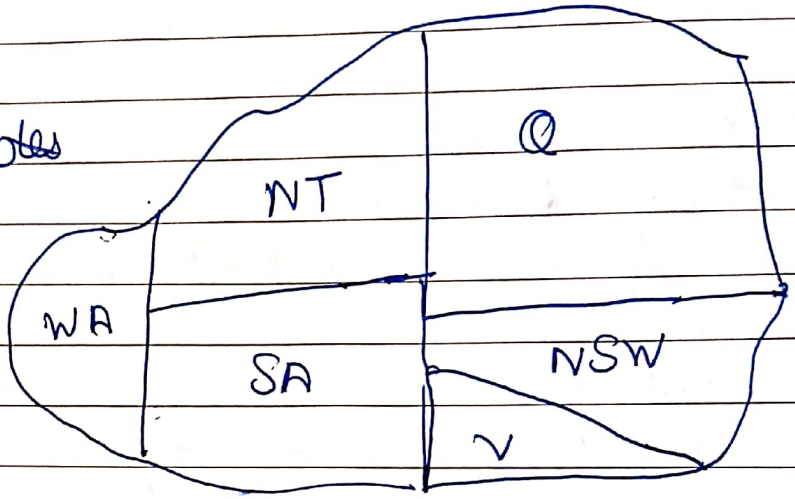
* CSP in AI:-

- Some of them are:-

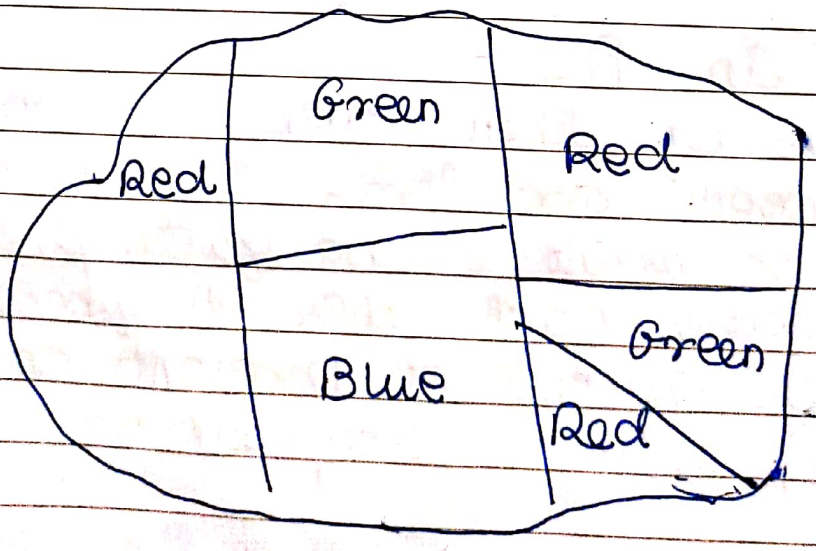
1. 8 queens problem:-
The constraint is no queen should threaten each other. A queen threatens other queen if present on same row, column, diagonal.

2. Map Coloring Problem:-
- Given a map and no of colors, the problem is to assign colors to those area in map such that no adjacent nodes have same color.

Soln:-
Variables



Soln:-
Variables = WA, NT, SA, Q, NSW, V.
Domain = {red, blue, green}.
Constraints = Adjacent regions must not have same colors.



* Scene Labelling:-

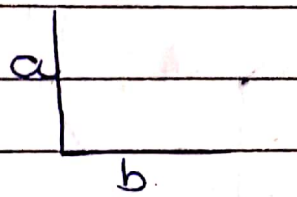
- It is a constraint propagation problem.
- In this problem the lines of an image has to be assigned a label describing them.

- Following are the meanings of the labels:-

1. + ⇒ Convex line.
2. − ⇒ Concave Line.
3. → ↗ ⇒ Right / Up Arrow Line
4. ← ↙ ⇒ Left / Down Arrow Line.

- Types of Vertices:-

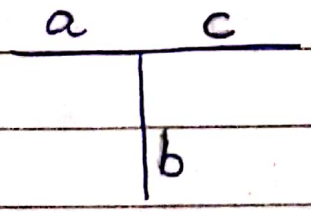
1] L joint:-



(a, b)

(→, →)
(←, ←)
(→, +)
(←, −)
(+, ←)
(−, →)

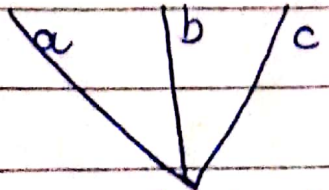
2] T joint:-



(a, b, c)

(←, ←, ←)
(←, →, ←)
(←, +, ←)
(←, −, ←)
(→, +, →)
(→, −, →)

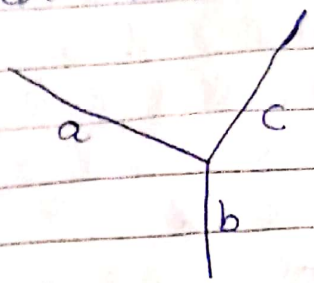
3] W joint or Arrow:-



(a, b, c).

(a, b, c)
(+, −, +)
(−, +, −)
(←, +, ←)

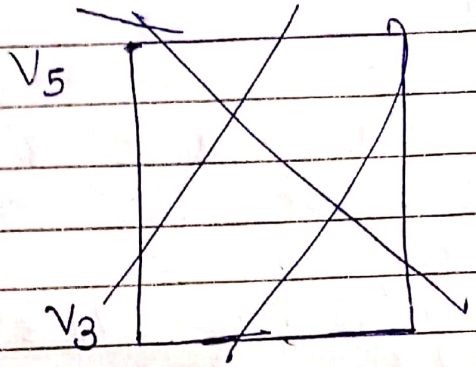
4] Y Joint or Fork:-

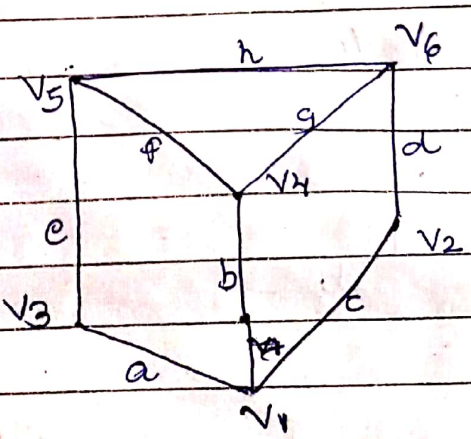


(a, b, c)

a	b	c
(+	+	+)
(-	-	-)
(-	←	←)
(←	-	←)
(←	←	-)

Eg:-





V_1 (W vertex) = (a, b, c).
V_2 (L vertex) = (c, d)
V_3 (L vertex) = (e, a)
V_4 (Y vertex) = (b, f, g)
V_5 (W vertex) = (h, f, e)
V_6 (W vertex) = (d, g, h).

* Backtracking & Lookahead Strategies:-

- If we apply DFS ~~BFS~~ to generic CSP problems then we can notice ~~that~~ the branching factor at the top level is 'nd' & any of 'd' values can be assigned to any of 'n' variables.
- At the next level, branching factor is '(n-1)'d & so on for n levels.

• Basic Steps in Backtracking Search:-

1. DFS selects values for single variable at a time.
2. Apply constraint to the variable when variable is selected.
3. Backtracking is performed if a variable has no legal values left to assign.
4. Backtracking search is basic uninformed algo. for CSP's.

• Limitations:-

- Backtracking is not effective for very large problems.
- General performance is not so good

• Look ahead Strategies:-

- Backtracking suffers from thrashing that is there is partial solut^n's that cannot be extended to a full sol^n & may be reprocessed several times (always leading to a dead end in the search space).
- To improve performance, two search strategy schemes are used:-

- Lookahead & lookback strategies:-

- Lookahead strategy is invoked when next variable or next value is selected.

Eg:- Which variable should be instantiated next? OR
Which value should be choosen for next variable.

- Lookback strategy is invoked when the backtracking step is performed after reaching a dead end.