PLANNING UNIT-4

Role of Planning in AI:

• AI is an Important technology in Future. Whether it is Intelligent robots, Self driving Cars, they will use different aspects of AI.!!

But, Planning is Very Important to make any such AI project.

→ Even, Planning is an Important part of AI which deals with the tasks & Domains of a particular problem.

✱ Everything we Humans do is with a specific goal in mind

Planning is considered as Logical Side of Acting.

Planning is required to reach a particular destination

It is necessary find Best route in Planning, but the tasks to be done at a particular time.

∴ Planning is about Deciding the tasks to be Performed by AI system & System's functioning under Domain- Independent conditions.

Classical Planning in AI:

Classical Planning in AI is a foundational field that traverses the maze of complications across multiple domains.

The Foundation of Everything from Robotics to manufact-uring, Logistics to Space Exploration is Classical Planning, which offers an organized method for accomplishing objectives.

→ It is a key Area in AI to find a sequence of actions that will fulfill a specific goal from an exact beginning point.

PDDL (Planning Domain Defindlation knowledge.)

PDDL is used for representing states in the form of set of actions / variables.

There are multiple versions of PDDL available which are capable of describing the following things necessary for defining a search problem.

* Standard Encoding language for "classical" planning tasks —— PDDL.

Components of a PDDL Planning Task:

- Objects: Things in the world that Interest us.
- predicates: Properties of objects that we are Interested in; Can be true or false.
- Initial state: The state of the world that we start in.
- Goal speification / Goal test: Things that we want to be true.
- Actions / operators: Ways of changing the state of the world.

Planning Tasks speified in PDDL are seperated into two files:

① A Domain file for predicates & Actions
② A problem file for objects, initial state & Goal speification.

Domain file Look like this:

```
( define ( domain < domain name> )
        <PDDL Code for predicates>
        <PDDL Code for first action>
            :
        < PDDL Code for last action>
)
```

<domain name> is a string that [illegible] identifies the Planning Domain Eg: gripper

Eg on web: gripper.pddl

Problem files

problem files look like this.

```
(define (problem <problem name>)
    (:domain <domainname>)
    <PDDL code for objects>
    <PDDL Code for Initial state>
    <PDDL Code for goal specification>
)
```

<problem name> is a string that identifies the planning task. Eg: gripper-four-balls.

<domain name> must match domain name in the Corresponding domain file.

Eg: on web: gripper-four.pddl

Example: Gripper-task with Four Balls.

There is a Robot that can move between two rooms and pick up or drop Balls with either of his Two Arms. Initially, all Balls and the Robot are in first room. we want the Balls to be in the Second Room.

Objects: The 2 rooms, 4 Balls & 2 robot-Arms.

Predicates: If x is a room? is x a ball?
Is ball x inside room y?
Is robot arm x empty? ...

Initial State: All Balls and robot are in 1st room
All robot arms are Empty ...

Goal Test : All Balls must be in 2nd room

Actions : Robot can move b/w 2 rooms, pick up /drop a ball.

Objects :

rooma , roomb

Balls : ball1 , ball2 , ball3 , ball4

Robot arms : left , right

In PDDL :

```
(objects rooma , roomb
         ball1 ball2 ball3 ball4
         left right)
```

Predicates :

Room(x) : true if x is a room

Ball(x) : true if x is Ball

GRIPPER (x) : true if x is gripper (robot arm)

at-robby(x) : true if x is room & robot is in x

at-ball (x,y) : true if x is Ball , y is r

free(x) : true if x is gripper & x doesn't hold a ball

Carry(x,y) : true if x is a gripper , y is a ball , x holds y.

In PDDL :

```
(:predicates (Room ?x) (BALL ?x) (GRIPPER ?x)
             (at-robby ?x) (at-ball ?x ?y)
             (free ?x) ( Carry ?x ?y))
```

Initial state :

ROOM(rooma) and ROOM(roomb) are true

BALL(Ball1) ... BALL(Ball4) are true ;

GRIPPER(left), GRIPPER(right), free(left) and free(right) are true.

at-robby (rooma) , at-ball(ball1, rooma) ....

Everything else is false.

In PDDL:

```
(:init (ROOM rooma) (ROOM roomb)
      (BALL ball1) (BALL ball2) (BALL ball3) (BALL
         ball4)
      (GRIPPER left) (GRIPPER (right)
      (at-robby rooma)
       (at-ball ball1 rooma) (at-ball ball2 rooma)
        (at-ball ball3 rooma) (at-ball ball4 rooma))
```

Goal Test:

at ball(ball1, roomb), ..at-ball(ball4, roomb) must be true.

PDDL:

```
(:goal (and (at-ball ball1 roomb)
            (at-ball ball2 roomb)
            (at-ball ball3 roomb)
            (at-ball ball4 roomb))
```

Movement Action:

Robot can move from x to y
rooma to roomb

in PDDL:

```
(:action move: parameters ?x ?y)
  :precondition (and (Room ?x) (Room ?y)
                  (at-robby ?x))
  :effect       (and (at-robby ?y)
                   (not (at-robby ?x))))
```

Pick up operator/ action

Robot can pick up $\underset{\substack{\downarrow\\ \text{ball}}}{x}$ in $\overset{\substack{\text{room}\\ \uparrow}}{y}$ with $\underset{\substack{\uparrow\\ \text{free}}}{z}$

→ **PDDL** :

```
:parameters (?x ?y ?z)
    (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
         (at-ball ?x ?y) (at-robby ?y) (free ?z))
     (and (carry ?z ?x)
          (not (at-ball ?x ?y)) (not (free ?z)))
```

Drop operator: Robot can drop x in y from z

```
:parameters (?x ?y ?z)
 (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
      (carry ?z ?x) (at-robby ?y))
 (and (at-ball ?x ?y) (free ?z)
      (not (carry ?z ?x))))
```

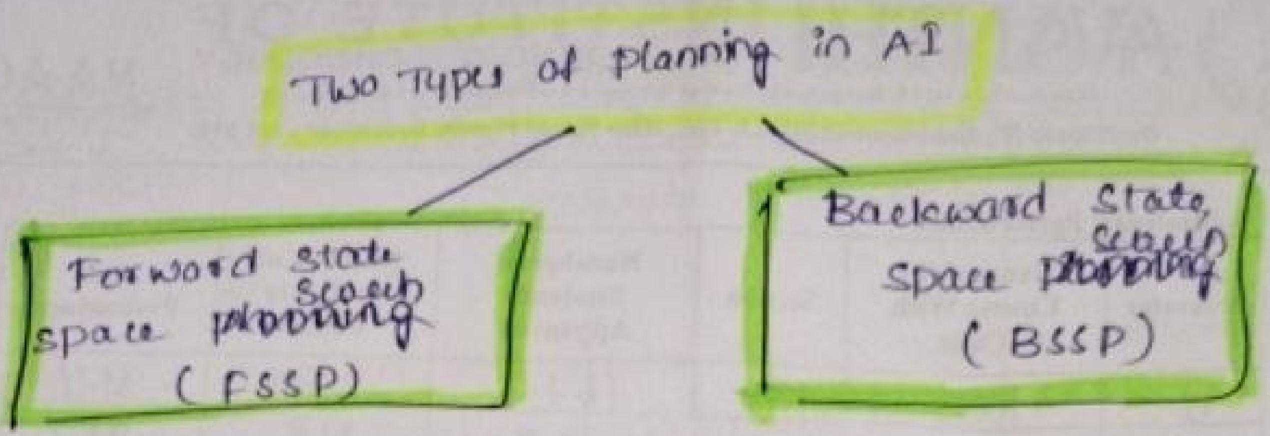
Planning using State space search

State Space Search Algorithm in AI:

State space search is a fundamental technique in AI for solving planning problems.

→ State space search Algorithm explore all possible states & actions to find optimal (or) feasible solution.

→ This approach is crucial in various applications like:
- ✓ Robotics
- ✓ Game playing
- ✓ logistics

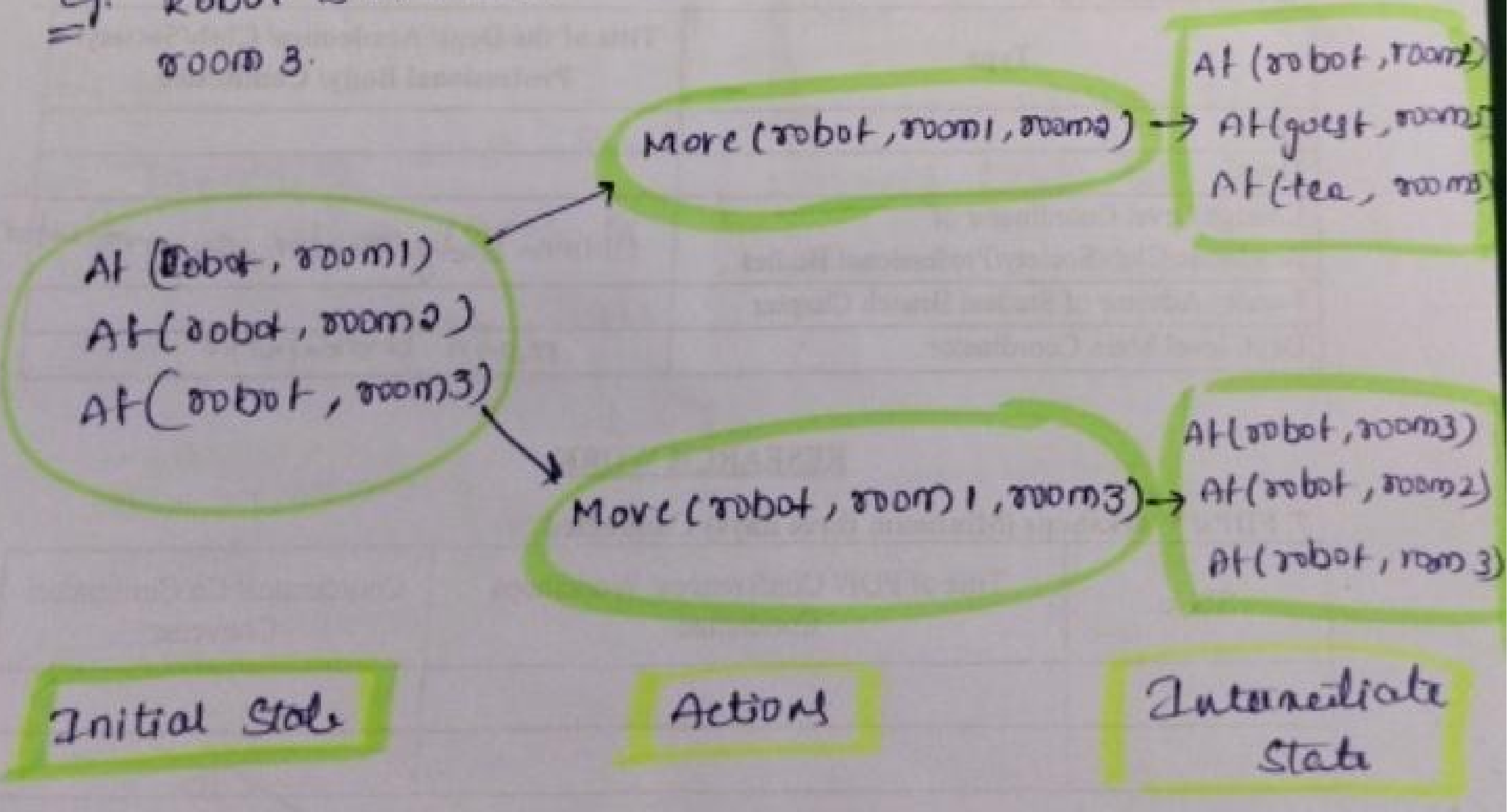


1. Forward State Space Planning (FSSP) / progression planning

Forward state space search also referred to as progression planning.

The planning with Forward state-space search begins with Initial State & moves forward until it finds sequence that reaches the goal state.

Eg: Robot is in room1 and Tea, guest are in room2 and room 3.



ⓐ **Initial - state** : The Initial state of planning problem. It generally assume Each state Corresponds to Collection of positive ground literals & Literals are not involved are false.

ⓑ **Actions** : Sequence of actions which satisfies preconditions that moves to next intermediate state.

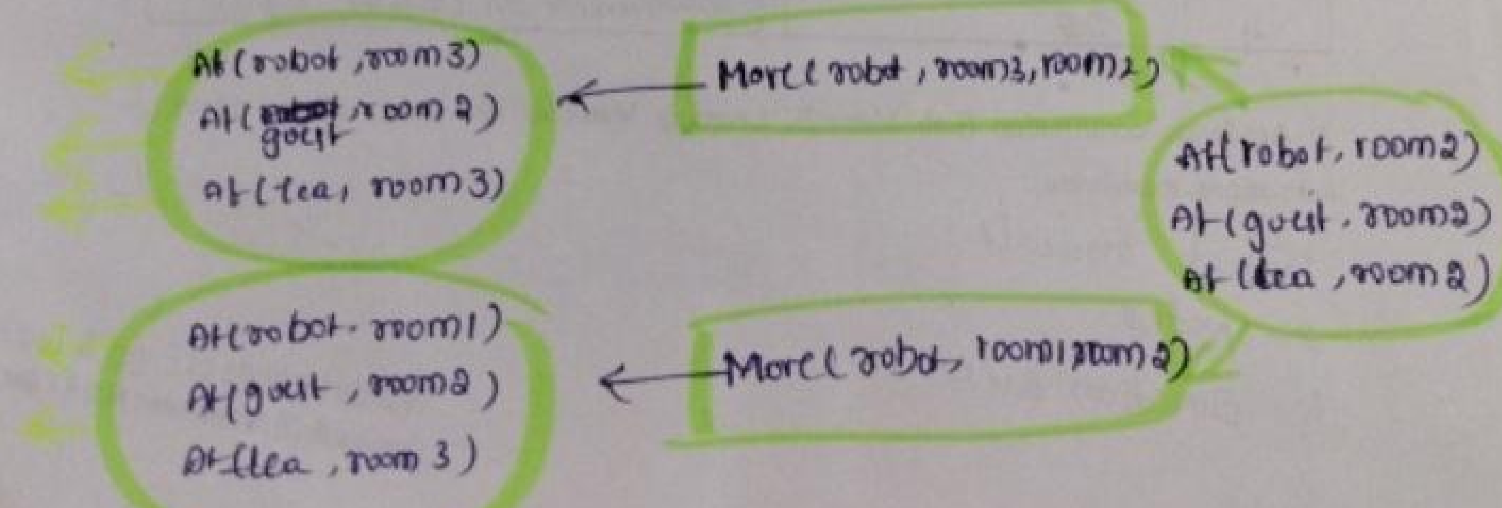
ⓒ **Goal check** : It checks whether state reaches goal of planning problem(or) not.

ⓓ **Step cost** : It refers to cost associated with Each & Every action included in forward planning. Most often, it value is 1.

Backward State space Search:

→ Typically referred as Regression planning / Regressive relevant state space search.

→ Planning with Backward state - space search begins from goal states & moves back till it reaches Initial state.



Steps for Backward State space search

-> Select an action that satisfies all or some of the prepositions in goal state.

-> Reconstruct a new goal.

Heuristics of State space search:

* Methods like forward state-space & Backward State-space search are both inefficient without the use of good Heuristic Function.

-> It is ~~also~~ known that Heuristic function Computes Distance from Initial state to goal node.

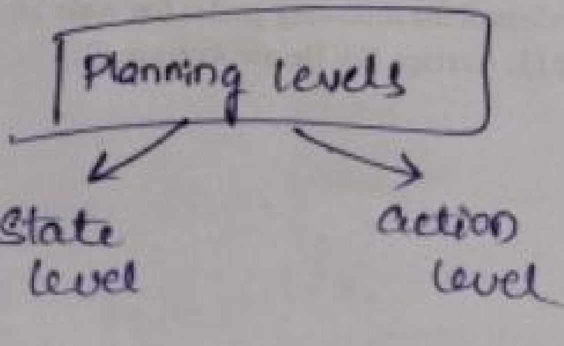
Planning Graph: (only work for propositional problems with no variables)

A special Data structure known as planning graph can be used to give Better Heuristic Estimates.

-> Primarily used in Automated Planning and Artificial Intelligence to find solutions to Planning problems.

Breakdown of its main Components & its functions:

① Levels : A planning graph has two alternating types of levels:



*State level : These levels Consist of nodes representing (11)
logical propositions or facts about the world.

→ Each Successive state level Contains all the propositions of the previous level plus any that Can derived by actions

* Action levels : These levels contain Nodes representing Actions. An action node Connects to a state level if state Contains all preconditions necessary for that action

Eg: Consider a problem of "have pizza and Eat pizza too".

init(Have(pizza))
Goal (Have(pizza) ∧ Eaten(pizza))
Action(Eat (pizza)
PRECOND: Have(pizza)
EFFECT: ¬Have(pizza) ∧ Eaten(pizza)
Action (Bake(pizza)
PRECOND : ¬Have(pizza)
EFFECT : Have(pizza))

Initiate state (S0):

→ Have (pizza) : we have the pizza.

→ ¬ eaten(pizza) : The pizza hasn't been eaten yet

Action level (A0): Represents possible actions that can transistion ~~from~~ state from S0 to next state.

→ Eat(pizza) : Represent the action to eat pizza

Next state (S1): Determined by action taken at action level based on propositional variables.

→ In our Example, we have 4 propositional variables

Have(pizza)	¬Have(pizza)	Eaten(pizza)	¬Eaten(pizza)
No action is peformed, we'll have a pizza	action Eat(pizza) is peformed, we'll no longer have pizza	Eat(pizza) is peformed, pizza will have been Eaten	no action is peformed pizza is uneaten

Mutual Exclusion: (Mutex relation holds between two actions / two literals)

→ if eat(pizza) is peformed, have(pizza) & ¬Eaten(pizza) can't be true.

→ if eat(pizza) is not peformed, ¬have(pizza) & Eaten(pizza) can't be true.

Actions available at A1 from S1

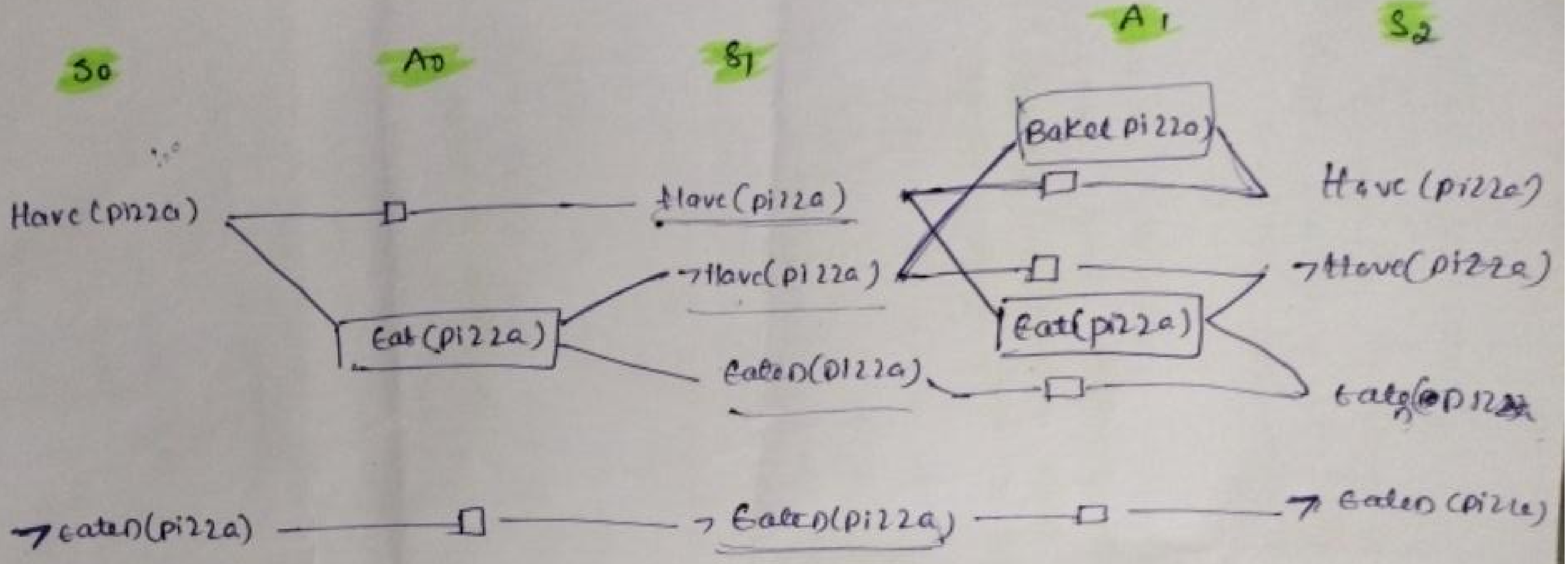
→ Bake(pizza) : precondition of action is ¬have(pizza)
if ¬Have(pizza) is true,

→ Eat(pizza) : precondition of action is have(pizza)

Repeat process until graph levels off:

→ 2 consecutive levels are Identical or

→ contain same amount of literals



∴ A_0, A_1 = Action levels

∴ S_0, S_1 & S_2 = propositional variables at State levels

Planning graph Algorithm:

Step 1: Begin from state S_0

Initialize $i = 0$

Compute each and every action of A_{i+1} applicable in S_i

Find mutexes Between two actions (or) two literals of A_{i+1}

Compute literals reachable at S_{i+1}

Find mutexes in S_{i+1}

If $S_{i+1} \neq S_i$, Increment i by 1 & goto step 3. In our Example. S_1 & S_2 are same, hence stop.

Hierarchical Planning:

Hierachical Planning in AI is a Problem solving and Decision making Technique employed to reduce the Computational Expense associated with Planning.

In AI, Hierachical Planning is a Planning methodology that entails grouping tasks & actions into several hierachies with higher-level jobs being broken down into a series of lower-level tasks.

→ AI systems can effectively handle complicated tasks & surroundings because of hierarchical Planning which enables them to make decisions at many levels of abstraction.

Components of Hierachial planning:

High level goals: Provide initial direction for Planning process & guide the Decomposition of tasks into smaller sub-goals.

sub-goals: are Intermediate objectives that contribute to accomplishment of higher level goals.

→ sub-goals are derived from decomposing higher level goals to smaller.

Tasks: Actions that need to perform to fulfill high level goal.

Task-dependencies & Constraints:

Task dependencies determine order in which task is executed. & preconditions must be satisfic before a-task to be performed is constraint

Hierachical Planning techniques in AI:

① HTN (Hierachical-task network)
② HRL (Hierachical Reinforcement Learning)
③ HTNs (Hierachical task networks)
④ HSSS (Hierachical state space search)

① **HTN planning**: Decomposing High level tasks into simpler sub-tasks using hierachical structures called task Network.

learn from outcomes & Decide what action to take next

② **HRL**: Extension of Reinforcement Learning.
In HRL, tasks are organised into hierachy of sub-goals & agent learns policies for achieving these sub-goals at different levels of abstraction.

③ **HTNs**: Consist of set of tasks organised into a hierachy, where High level tasks are decomposed into sequence of lower-level tasks.

④ **HSSS**: * Most Efficient.
Instead of direct Exploring Individual states. Hierachical state space search organises states into Hierachical structures (Higher levels are decomposed into sub-goals & sub-goals are decomposed until reach primitive actions).

Classical Planning as Boolean Satisfiability :

PLANSAT (OR) SATPLAN.

SAT : propositional satisfiabity problem, given a Boolean formula in CNF (Conjuctive normal form), find an Interpretation (assignment of truth values to literals propositions).

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

possible Interpretation are :

A : T , B : T , C : T

A : T B : F C : T

A : F B : T C : T

A : F B : T C : F

→ Complexity of Classical planning ;

Illustrated by Considering Two Decision problems. They are PlanSAT and Bounded PlanSAT.

→ PlanSAT : Indicates the availabity of a Plan which can solve a specific problem.

→ Bounded PlanSAT : Indicates the availability of a solution of length less than (or) Equal to k for a optimal plan.

But when Addition of function symbols.

PlanSAT does Impact whereas Bounded PlanSAT doesn't Impact. Such problems are solved with use of Turing machine having polynominal amount of space