Shivam. Tiwari.
5117060. B.E. Comps.

Assignment.

Q1 Taking a suitable agent as illustration explain PEAS.

→ Agent :- An automated taxi driver.

- Performance Measure :-
  Safe, fast, legal, comfortable trip, maximize profits.
- Environment :-
  Road, pedestrians, traffic, customers.
- Actuators :
  Staring acceleration, break, signal, horn, display.
- Sensors :-
  Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine, sensors, keyboard.

Q2 Explain an illustration, explain the various types of task environment.

→ 1) Fully Observable V/s Partially Observable.

- If an agent's sensors gives it the access to the complete state of the environment at each point of time, then it is fully observable
- In some environment, if there is noise or agent with inaccurate sensors or may be some states of environment are missing then such environment is partially observable.

Partially Observable Example	Fully Observable Example.
- Pocker game environment - Interactive Science Tutor - Military Planning	- Puzzle game environment. - Interactive Image Analysis. - Tic - Tac - Toe.

2) Deterministic V/s Stochastic :-

- Deterministic : In image analysis whatever is current percept of image, agent can take a next action or can process remaining part of image based on current knowledge.
- Strategic :- Agent playing tic-tac-toe game is in strategic environment as from the current state agent decides the next state action except for the action of other agents.
  Example :- Video Analysis, Trading Agent.
- Stocastic :- Boat driving agent is in stocastic environment as the the next driving is not based on the current state.
- Car driving.
- Robot firing in Crowd.

3) Episodic V/s Sequential:

→ In episodic environment agents experience is divided into atomic episodes such that each episode consists of, the agent perceiving process and performing single action.
- This choice of action depends only on the episode itself, previous episode doesnot affect current actions.

In sequential environment on the other hand, the current decisions could affect all future decisions.

Example:

→ Episodic.

a. Agent finding deffective parts.
b. Blood testing for patient.
c. Card Games.

Sequential Environment. (i) Chess with a clock
(ii) Refinery controller.

4) Static V/s Dynamic.

If the environment can change while the agent is deliberating then the environment is dynamic for the agent, otherwise it is static.

Examples:

Static	Dynamic
a. Crossword Puzzle	a. Agent driving a boat
b. 8 queen Puzzle.	b. Car driving.
c. Semidynamic	c. Tutor.

5) Discrete V/s Continuous:

→ In discrete environment the environment has fixed finite discrete states over the time & each state has associated percepts and actions.

- When, where as continuous environment is not stable at any given point of time & it changes randomly true by making agent to learn continuously.

Example

Discrete	Continuous.
a. A game of tic-tac-toe	a. A boat driving environment.
b. 8 queens puzzle	b. Part Picking robot.
c. Crossword Puzzle.	c. Flight controller.

vi) Single Agents v/s Multiagent.

- In single agent environment we have well defined single agent who takes decision and acts.
- In multiagent environment there can be various agents. or various groups who are working together to take decisions & act.

Example:-

a) Multiagent Independent environment.
   - Many agents inc game of maze.

b) Multiagent cooperative environment.
   - fantasy football.

c) Multiagent competitive environment.
   - War Traging agents.

d) Multiagent and antagonistic environment
   - Wargames.

Q3 Explain a complete utility based agent with suitable block diagram, write psucdocode for it.

→

- Utility Based Agent:

In complex environment only goals are not enough for agent designs. Addihonal to this we can have utility functions.

- Property.

1. Utility functions map a state on to a real number, which describes the associated degree of best performance.
2. Goals gives us only two outcomes achieved or not achieved But utility based agents provide a way in which the likelihood of success can be measured against importance of the goals.
3. Rational agents which is utility based can maximize expected value of utility function. ie. more perfection can be achieved.
4. Goals give only two discrete states.
   a) Happy b) Unhappy.

Example:- Military Planning robot.

which provides certain plan of action to be taken. Its environment is too complex; and expected performance is also high.

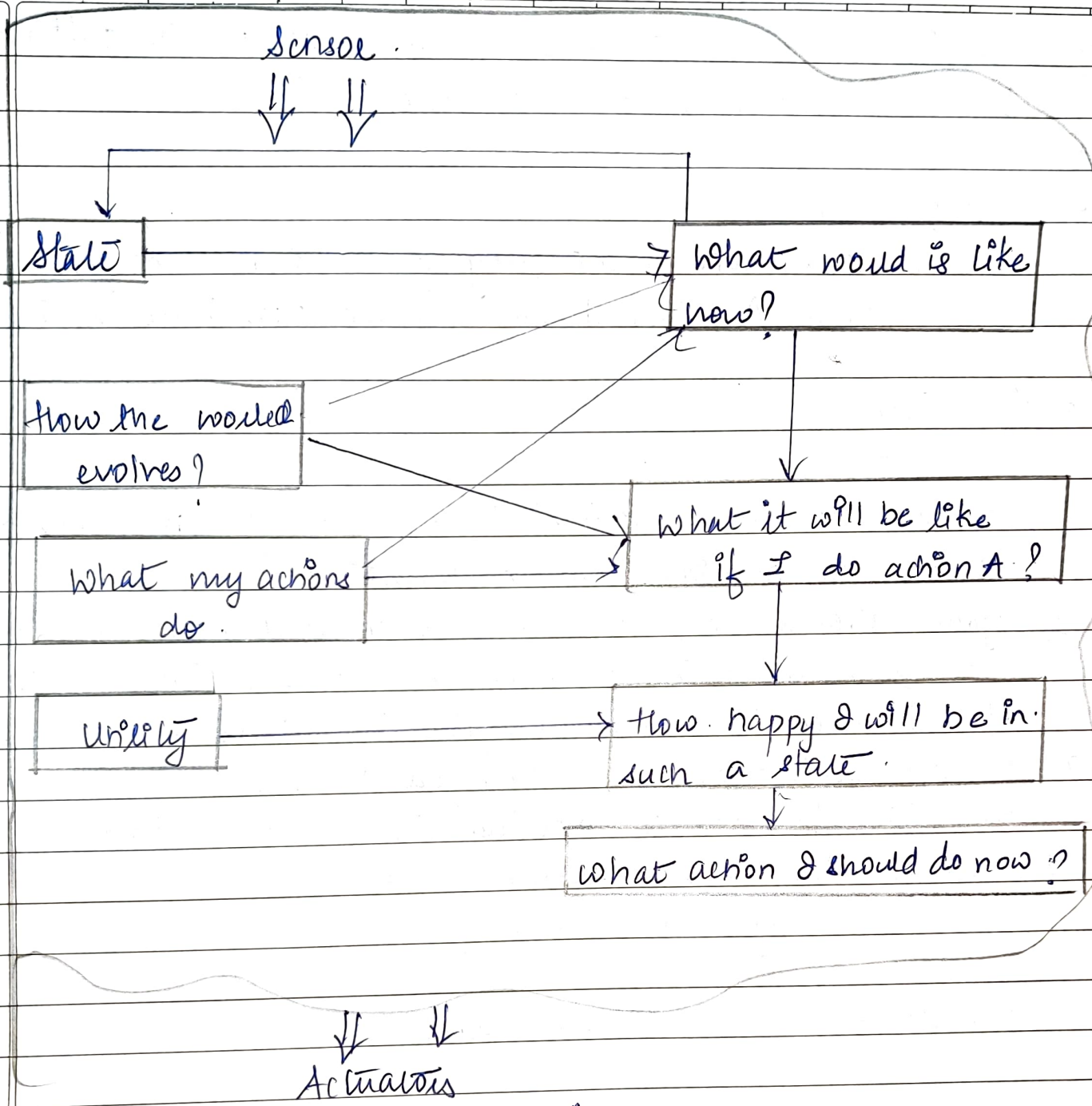


Fig:- complete Utility Based agents.

Q4 Explain the components of a problem with a suitable example.

→ Problem formulation: is the process of deciding, what actions and states to consider, given a goal.

A problem can be defined formally by 4 components.

1) Initial state that the agent starts in:-

eg:-

Consider a agent program Indian Traveller developed for travelling from Pune to Chennai travelling through different states.

The initial state for this agent can be described as In (Pune)

2) A description of the possible actions available to the agent.

The most common formulation uses a successor function.

Given a particular state x, successor function(x) returns a set of (action, successor) ordered pairs, where each action is one of the legal actions in state x & each successor is a state that can be reached from x by applying the action.

for example:

From the state In (Pune) the successor function for Indian Traveller problem would return.

{ < Go (Mumbai), IN (Mumbai) >
< Go (Ahmednagar), IN (Ahmednagar) >
< Go (Solapur), IN (Solapur) >
< go (Satara), In (Satara) >
}

Together, the initial state & the successor function implicitly define the state space of the problem.

The state space forms a graph in which the nodes are the states & the arcs between nodes are actions.

A path is the state space is a sequence of states connected by a sequence of actions.

3) The Goal Test.

It determines whether a given state is goal (final) state. In some problems we can explicitly specify the set of goals. If a particular state is reached we can check it with the set of goals & if a match is found success can be announced.

Example:-

In chess, the goal is to reach the state called "checkmate" where the opponents king is under attacks & cannot escape.

4) A path cost function.

That assigns a numeric cost (value) to each path.

The problem solving-agent is expected to choose a cost function that reflects its own performance measure.

The above 4 elements define a problem & can be put together in simple data structure which can be given as input to a problem solving algorithm.

A solution to the problem is a path from the initial state to a goal state.

Q5 How can you measure the performance of an Algorithm.
// Explain Illustratively.

→ When we are solving a problem we have three possible outcomes.

1) We reach at failure state.
2) Solution state.
3) Algorithm might get stuck in an infinite loop.

Problem Solving Algorithms performance can be evaluated on the basis of 4 factors.

1) Completeness:
Does the algorithm, surely finds a solution, if really a solution exist.

2) Ophimality:
Sometimes it happens that there are multiple solutions to a single problem. But the algorithm is expected to produce best solution among all feasible solution which is called an ophimal solution.

3) Time Complexity:
How much time the algorithm takes to find a solution.

4) Space Complexity.
How much memory is required to perform the search algorithm.

Major factors affecting time & space complexity.

(i) size of state space graph.
The size of state space graph is affected by.

(1) Branching factor:
- maximum no of successor for any node. Lesser the branching factor faster the search.

(2) Depth of goal node.
- Depth of the shallowest goal node, where one can reach fast,
- smaller value, goal node can be achieved easily.

(3) Maximum length of path:
- It is maximum length of any path in state space.
- If length value is more, complexity is more.
- measured in terms of no of nodes generated.
- Path cost is time bound cost which is incurred for reaching or going to a particular node.

Total cost = search cost + Path cost + Memory usage.

Q6 Given the Depth First search & state its Advantages & disadvantages. Explain how some of the problems of DFS can be overcomed by using Depth limited search.

→ Depth first search: (DFS)

- DFS always expands the deepest node in the current unexplored node set (fringe) of the search tree.
- DFS is implemented using a stack & recursive approach

o Performance Evaluation.

1. Completeness :- Guarantees soln, if exists.
2. Optimality :- Since it explores the deepest node first, it may ignore some shallow goal state. Optimal only if all goal states have same path.
3. Time & Space complexity:

DFS requires memory to store the path from node to a specific node along with its unexpanded siblings. When some node is fully explored its complete branch, along with the descendents will be removed

eg: with branching factor 'b', depth 'd' it stores 'bd+1' nodes.

```
Algorithm: DFS (G) // G → Undirected or directed graph.
    // G=(V,E) with 'n' vertices an array initially set to zero.
    // This algorithm visits all vertices reachable from V in G.
    // G and visited [] are global.
    {
      visited[v]=1
      for each vertex w adjacent from v do:
        {
          if (visited[w]=0 then DFS(w);
        }
    }.
```

Time complexity: $O(v^d)$ Space Complexity $O(v^{d+1})$

Drawbacks of DFS:

- If we limit the depth unlike the normal procedure it will become effient.

Advantages:

- Guarantees solution
- If the goal node is the left most deepest node it can be reached quickly

- **Implementation of DLS:**

- It is same as DFS but with a limited depth 'l'
- DLS will terminate with two kinds of failure.
  (a) standard failure → No solution
  (b) Cutoff value → no solution within depth limit.

- **Performance evaluation:-**

(1) **Completeness:** DLS suffers from incompleteness because if we choose level 'l' (levels to be searched) < d (actual levels), when shallowest goal state is at level > 'l'.

(2) **Optimality:** Non optimal if $l > d$.

(3) **Time & space complexity:**
Time → $O(V^l)$
Space Complexity $O(V^l)$

Steps:-

1. Determine start point & maximum search depth 'd'
2. Check if current node is goal node
   if not: Do nothing
   if yes → return
3. Check if current node is within 'd'.
   if not → Do nothing
   if yes → (i) Expand the vertex & save successor in stack
   (ii) Call DLS. (Step 2)

Algorithm :-

```
DLS (node, goal depth)
{
   if (depth >=0)
    {
      if (node == goal)
      for each child expand (node)
      DLS (child, goal, depth -1)
    }
}
```

Q7 Explain heuristics & heuristic function. prove that A* is complete & optimal along all search algorithms.

→

Informed search strategy - one that uses problem-specific knowledge beyond the definition of the problem itself - can find - solutions more effectively & efficiently than a uninformed strategy.

Heuristic function :-

- There are many different algorithms which apply best first search.
- The main diff b/w all algorithm is that they have diff eval fn's
- The heart of such algo is a heuristic fn defined as h(n) = estimate of the cheapest path from current to goal node. { shortest & cheapest paths }
- It is a key component of Best first search.
- It guides search in efficient ways.
- one can give additional knowledge about problem to the heuristic fn.
- It relies on the state of the node
- h(n) = 0, if n → goal state

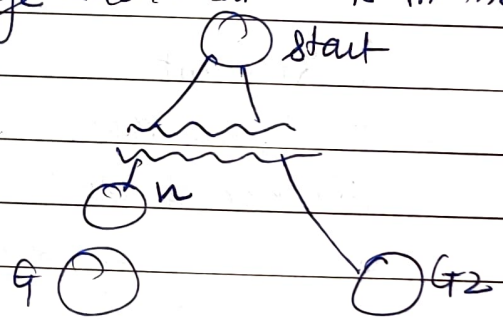
A* Search.

- Avoid expanding paths that are already expensive.
- Evaluation function $f(n) = g(n) + h(n)$.
  $g(n)$ = cost so far to reach 'n'.
  $h(n)$ = estimated cost for 'n' to goal.
  $f(n)$ = estimated total cost of path 'n' to goal.

→ Admissible heuristic → $h(n) \leq h^*(n)$. It never overestimates the path to reach goal node i.e. it is optimistic (true cost)

→ Optimality of A*

- Suppose some suboptimal goal G_2 has been generated & is in the fringe.
- Let n be an unexpected node in the fringe such that 'n' is in the shortest path to an optimal goal 'G'



→ $f(G_2) > f(G)$ (from above)
$h(n) \leq h^*(n)$ since h is admissible
$g(n) + h(n) \leq g(n) + h^*(n)$
$f(n) \leq f(G)$
Hence $f(G_2) > f(n)$ & A* will never select G_2 for expansion

- A* expands nodes in order of increasing f values.
- Gradually adds "f-contours" of nodes.
  contour i has all nodes with $f = f_i$
  where $f_i < f_{i+1}$.

Q8 Describe the WUMPUS world according to properties of task environment.

→ Wumpus word → cave consisting of rooms connected by passageways
- hurking somewhere in cave → terrible wumpus, a beast that eats anyone who enters the room.
- Wumpus can be shot by agent but agent has 1 arrow.
- some room contains bottomless pits. that will trap anyone wandering
- The only way is to find a heap of gold. { wumpus world ↔ conq game

- PEAS standard.
- +1000 for coming out of cave with gold.
- -1000 for falling in pit or being eaten by the wumpus.
- -1 for each action taken & -10 for using arrow.
- The game ends either when agent dies or agent comes out

- Environment:
- Agrid of rooms; Agent starts with square labelled [1,1] facing r
- location of gold & wumpus are chosen randomly with uniform distribution of sequence other than the start square.
- Except the start square, each square can be a pit with probability 2.0.

- Actuators :-
- Agent can more forward, Turn left (90°), Turn right (90°)
- Agent dies if it enters square having pit or wumpus.
- If Agent moves forward & bumps into wall if doesn't moves.
- Action grab → Pick gold if same square as agent.
- Action shoot → shoot in straight line where agent is facing.
- Only 1 Arrow.
- Action climb → To climb out of cave.

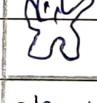
Sensors:

5 sensors as follows.

- If the square containg wumpus & square directly adjacent to it agent will get stench.
- In the square directly adjacent to a pit; the agent will percieve breeze.
- In the square where there is gold, it will percieve glitter.
- If Agent walks into wall it recieves a bump.
- When the wumpus is killed it emits a useful scream that can be percieved any where in cave.
- These percepts will be given to the agent program from the list of 5 symbols.

eg:- if there is only stench & breeze the agent will get

[stench, breeze, none, none, none]

4	stench		Breeze	Pit
3		Breeze Gold stench	Pit	Breeze
2	stench		Breeze	
1	start	Breeze	Pit	Breeze

Q9 Represent the following sentences into first order logic

(a) Not all students take history & Biology

→ $\exists x : student(x) \wedge \neg(Take(History) \wedge Take(Biology))$

(b) Only one student failed in both History & Biology

→ ~~exists x exists y Fails (x, History) ∧~~

~~$\exists x \exists y \forall y : [[student(x) \wedge fails(x, History) \wedge$~~

$\exists x \forall y : [[student(x) \wedge fails(x, History) \wedge fails(x, Biology)) \wedge student(y) \wedge fails(y, History) \wedge fails(y, biology)) \leftrightarrow x = y]$

(c) Well I like sandy & I don't like sandy.

→ ~~like(I, sanday) ∧ ¬like(I, sandy)~~

$\exists x$: like(x, sandy) ∧ ¬ like(x, Sandy).

(d) James father is married to king John's mother

→ ~~married (father (James), mother (king John's)~~

$\exists x \exists y$: father(x, James) ∧ mother(y, kingJohn) ∧ married(x, y).

(e) There is someone who is loved everyone

→ $\exists x \forall y$: loved(x, y).

(f) There are is none who does not like Ice-cream

→ $\neg \exists x$: ¬ likes(x, icecream).

(g) Spot has atleast two sisters

→ $\exists x \exists y$: sister(spot, x) ∧ sister(spot, y) ∧ $x \neq y$.

(h) Mujeeb eats everything Hussein eats

→ $\forall x$: food(x) ∧ eats(Hussein, x) ⟶ eats(Mujeeb, x).

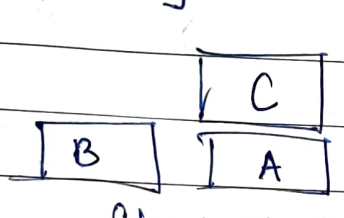
OR $\forall x$: eats(Hussein, x) ⟶ eats(Mujeeb, x)

(i) Anything (y) anyone (x) eats & is not killed by is food

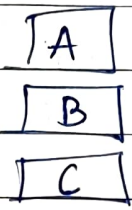
→

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

Q10 Explain non-interleaved planner. figure below shows a block-word problem known as the sussman anomaly. Write a definition of the problem in STRIPS notation & solve it by interleaved planner.



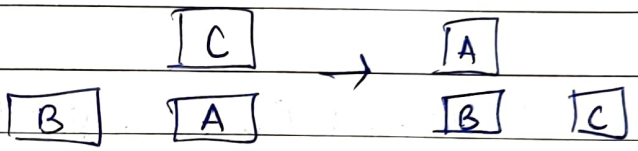
Start state



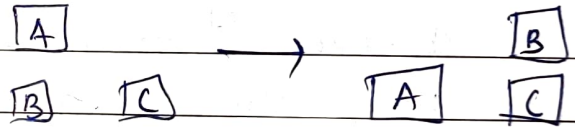
Goal state.

→

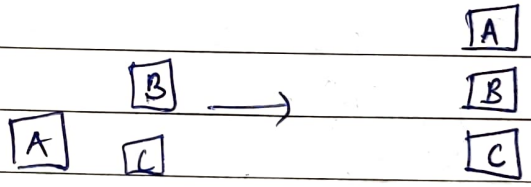
1- Achieve on (B, A)
Unstack (C, A)
putdown (C)
Pickup (A), stack (A, B)



2. Achieve on (B, C)
Unstack (A, B),
putdown (A), pickup (B)
Stack (B, C)

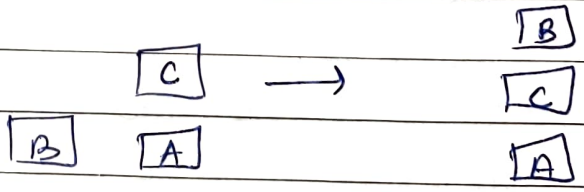


3. Reachieve on (A, B)
pickup (A), stack (A, B)

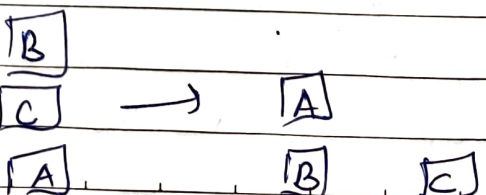


Solution 2:-

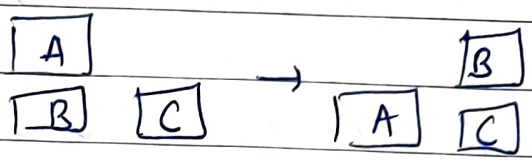
(1) Achieve on (B, C)
pick (B), stack (B, C)



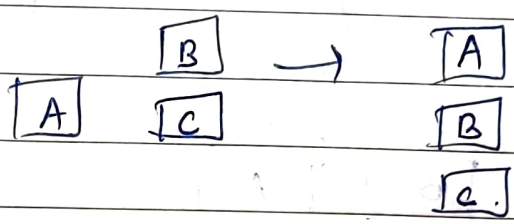
(2) Achieve on (A, B)
unstack (C, A), putdown (C)
pickup (A), stack (A, B)



3. Re achieve on (B, C)
Unstack (A, B) putdown (A)



4. Re-achieve on (A, B)
pick up (A), stack (A, B)



→ A strip is not able to find out an ophimal plan.

→ Unstack (C, A), putdown (C), pickup (B), stack (B, C), pickup (A), stack (A, B).

- The two sub-goals on (A, B) & on (B, C) are not independent that's why it can't find ophimal plan. STRIPS difficulty derives from deleted condition interactions.
- The action chosen to achieve a sub-goal has the side-effect of invalidating a previously achieved goal.
- Thus planning algorithms must be able to reason about multiple plans for achieving sub-goals at the same time.

Q11 The monkey and banana problem is faced by a monkey in a laboratory with some bananas hanging out of reach from the ceiling, a box is available that will enable the monkey to reach the bananas if he climbs on it.
Initially Monkey is at A, Banana at B & Box at C. Height of Monkey and the height of the box is low, but if the monkey climbs on it, they will have height high, same as the bananas actions available to the monkey include go from one place to another.

- Push an object from one place to another to climb onto & climb down from an object and grasp or ungrasp an object. Grasping results in monkey holding the object, if the monkey and objects are on the same height.
(a) Write down the initial state description
(b) write down STRIPS style definition of the six actions.
(c) Write down STRIP-STYLE plan for a possible success.

→

Initially, the monkey is at location 'A', the banana is at location 'B' and the box is at location 'C'. The monkey & box have height "low", but when monkey climbs on to the box, it will have height "high". The same as the bananas.
The action available to the monkey include.
"GO", from one place to another
"PUSH", an object from one place to another.
"CLIMB", onto an object
"GRASP", an object.

- GRASPING results in holding the object provided monkey & object are on the same height & same place.

(i) Initial Description.
At (monkey, A), At (banana, B), At (Box, C), position (monkey, low), position (Banana, high), position (Box, low)

(ii) Definition
a) Go (x, y)
Precondition : At (monkey, x)
Effects → At (monkey, x), At (monkey, y)

b.) Push (object x, y, height)
Precondition : At (monkey, x), At (object, x)
Posihon (Monkey, height), posihon (object, height)
Effects : → At(monkey, x) ~~,~~ At (object, x).
At (monkey, y), At (object, y).

c) Climb up (object, y):
Precondihon: At (monkey, x). At (object, x)
Posihon : (monkey, low), posihon (object, low).
Effects : > posihon (monkey, low), Posihon (monkey, ~~object~~ high),
on (monkey, object)

d) Climb down (object)
Precondihon : Posihon (monkey, high)
on (monkey, object)
Effects : > posihon (monkey, high), → On(monkey, object)
Posihon: (monkey, low).

e) Grasp (object x, height)
Precondihon : At (monkey, x), At (object, x).
Posihon (monkey, height), Posihon (object, height).
Effect : Hold (object)

f) Ungrasp (object x, height):
Precondihon : Hold(object), At(monkey, x)
At(object, x), Posihon (monkey, height)
Posihon (object, height).
Effect : release (object)

(iii). GO(A, C)
2. PUSH (BOX, C, B, LOW)
3. CLIMB UP (BOX, B)
4. GRASP (BANANA, B, HIGH)
5. CLIMB down (BOX)
6. PUSH (BOX. B, C, LOW)