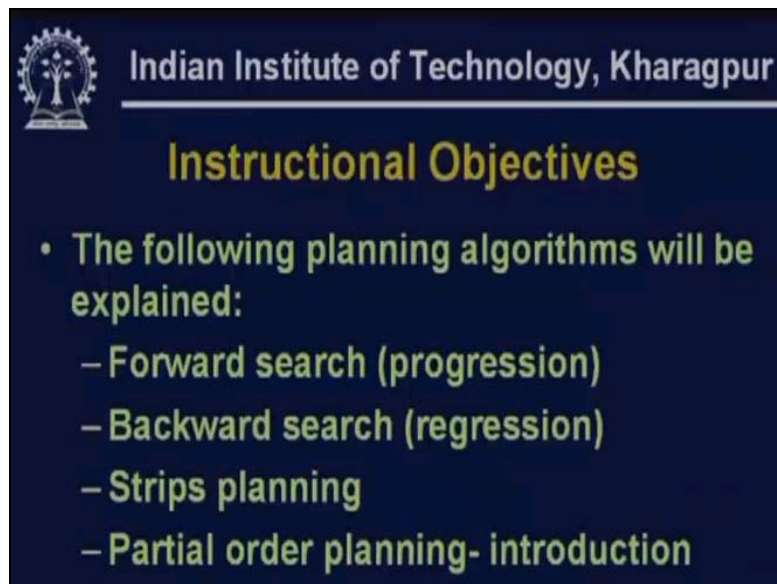



**Artificial Intelligence**  
**Prof. Sudeshna Sarkar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture # 22**  
**Planning - 2**

In the last class we had a first introduction to planning systems. Today's class would be the second lecture on planning.

(Refer Slide Time: 00:35)



 Indian Institute of Technology, Kharagpur

---

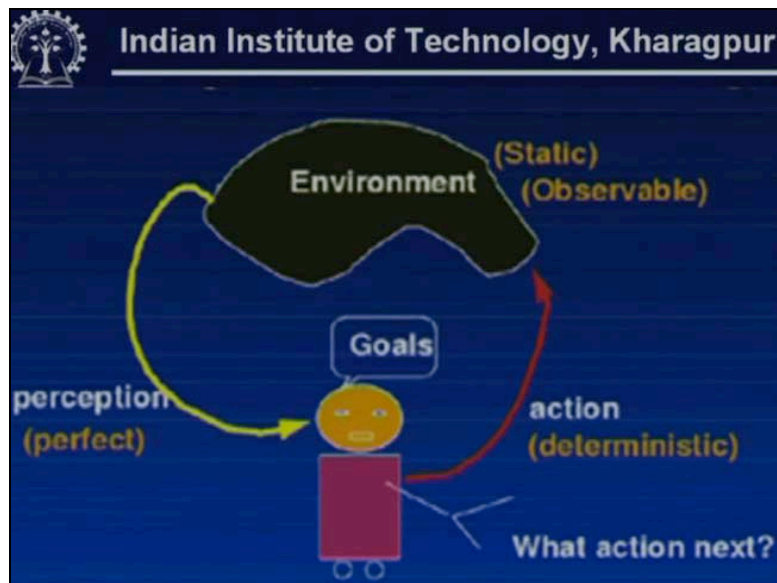
**Instructional Objectives**

- The following planning algorithms will be explained:
  - Forward search (progression)
  - Backward search (regression)
  - Strips planning
  - Partial order planning- introduction

The instructional objectives of today's lectures are the following:

We will look into a little more detail of forward search as used for planning problems, forward search or progression. Then we will look at backward search or regression. We will also look at the planning algorithms used in the STRIPS system. And finally we will start with partial order planning the framework and we will explain the basic partial order planning algorithm. So let us get back to a quick introduction of the planning problem.

(Refer Slide Time: 01:18)



As we have noticed earlier we have an agent which works in an environment. The agent has goals, the agent takes actions and the actions affect the environment, the agent receives percepts through its sensors from the environment. The objective of the environment is to plan the next action or the next sequence of actions so that it can achieve its goals. So this is the basic idea of a planning system.

(Refer Slide Time: 02:22)

The slide features the IIT Kharagpur logo and name at the top. The title 'Questions' is centered in yellow. Below it, a list of questions is presented in green text:

1. Explain how planning systems differ from classical search problems.
  - Decomposable sub-goals of a goal
  - States decomposable (conjunction of variables)
  - An action typically changes only a few of the variables.

Before we get into today's lecture I would like to briefly discuss the questions we posed in the last lecture. The first question was, explain how planning systems differ from classical search problems. Planning is also one type of search. What sets apart from a general search problem is that in the case of planning we can decompose the goal we do not look at the goal as a whole. So it is not just that we look to see whether a state satisfies the goal fully or not. Usually as we have noted, a goal is described as a conjunction of different conditions different propositions or conjunction of different

literals so these can be looked upon sub-goals. So instead of looking at the goal as a whole it may be more useful if we look at the goal as a conjunction of sub-goals so that we can reason separately about solving each sub-code and then combine the results. This will make the planning problem easier to solve.

Secondly, states are also not indivisible entities, states are decomposable. So states as we have seen are described by a conjunction of those propositions that hold or that are true in the state. So state is described as a set of propositions. And actions are also explained in terms of what propositions must hold at the previous state in order to the action to be executed that is the preconditions of the action and effect of the action. So we concentrate on how the action changes the propositions in the state. So this is what sets a planning problem apart or which distinguishes the planning problem from the more general search problems. And the algorithms for planning that we will consider is look at these characteristics of the planning problem in order to devise the right algorithms.

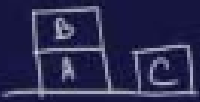
(Refer Slide Time: 04:29)

Indian Institute of Technology, Kharagpur

### Questions

2. Formulate the blocks world planning problem.

- Initial state : Conjunction of propositions OnTable(A), On(B, A), Clear(B)
- Actions: Pickup(X), Stack(X, B)
- Goal state: On(B, C)



Formulate the blocks world planning problem. For the blocks world planning problem let us just look at a sample of a blocks world planning problem. Suppose we have the following situation: We have two blocks A and B, A then B is on top of A and we have a third block C. So this is the initial state of the planning problem.

How do we express this initial state?

On table A, on B, A, clear B, on table C, clear C. So the initial description of the initial state of the planning problem can be given as conjunction of on table A and on B, A and clear B and clear C and on table C. This is how we can describe a particular initial state in terms of the propositions on table on and clear. Then we have to specify a description of the goal state.

Suppose in the goal state we need to only give a partial specification then for example I can say that B should be on top of C. So this as we see is a partial specification of a goal state and there could be several states which satisfy this condition. For example,

this is a state which satisfies the goal condition B is on top of C, this is another state which satisfies the goal state, there are several states which can satisfy the goal state. Then we can represent actions. For example suppose in the blocks world we can have actions like pick up, pick up x and we specify an action in the STRIPS domain by giving the preconditions of the actions and the effects of the action. For each action you need to specify the precondition.

What would be the preconditions of the pick up x actions?

For the pick up x action the preconditions could be hand empty and clear x. If x is clear and hand is empty then the pick up x action can be executed. Now, if pick up x action is executed what would be the effect?

The effect would be not, so if x is on top of something suppose x is on top of y if on x, y was true in the previous state on x, y will not be true in the next state so that would be deleted and hand empty will no longer be true after the pick up x action so not hand empty. Similarly, if we look at the actions stack (x, B) what would be the precondition the precondition as to be that clear B so B has to be clear and then hand should not be empty and the hand should be holding x. So, if holding x and clear B then the actions stack (x, B) can be performed. After the action stack (x, B) is performed B will no longer be clear x will be on (x, B) and x will be clear.

(Refer Slide Time: 08:44)

The image shows handwritten notes on a slide defining two actions in STRIPS notation:

- Pickup(x)  
 Pre:  $Clear(x), Handempty(), On(x,y)$   
 Effects:  $\neg Handempty(), \neg On(x,y)$   
 $Holding(x)$
- Stack(x,y)  
 Pre:  $Holding(x), \neg Handempty(), Clear(y)$   
 Effects:  $\neg Holding(x), \neg Clear(y)$   
 $Handempty, On(x,y)$

For the action pickup x the precondition has to be clear x hand empty and let us say on (x, y), what will be the effects?

Effects will be not hand empty not on x, y and the effect will also be holding x. For the action stack (x, y) what would be the preconditions?

The precondition has to be that holding x, not hand empty, clear y and what will be the effects?

The effects will be not holding x, not clear y, hand empty on (x, y).

(Refer Slide Time: 11:00)



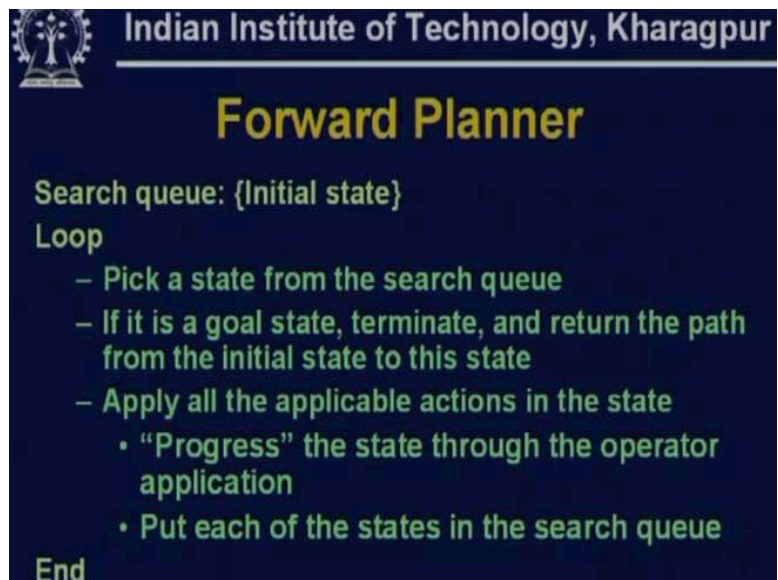
Indian Institute of Technology, Kharagpur

## Search in Planning

- Given an initial state and a goal state
- How do we generate a sequence of actions?
- Forward state space planning

As we have noted that planning problems can be cast as search problem and in planning we are given an initial state and a goal state and we have to generate a sequence of actions. And we have also discussed that there are two standard ways in which we can generate this sequence of actions. One is we can do forward state space planning in which we go from the initial state towards the goal state in order to find the plan or we can do backward planning. So let us review the algorithm for forward planner.

(Refer Slide Time: 11:29)



Indian Institute of Technology, Kharagpur

## Forward Planner

Search queue: {Initial state}

Loop

- Pick a state from the search queue
- If it is a goal state, terminate, and return the path from the initial state to this state
- Apply all the applicable actions in the state
  - “Progress” the state through the operator application
  - Put each of the states in the search queue

End

In a forward planner initially the agent is in the initial state and we maintain a search queue and we put the initial state in the search queue. Then we have a loop. In the loop the first step is to pick a state from the search queue and examine the state. If it is a goal state then terminate and return the path from the initial state to this state. Otherwise if it is not a goal state we apply all the applicable actions in the state to

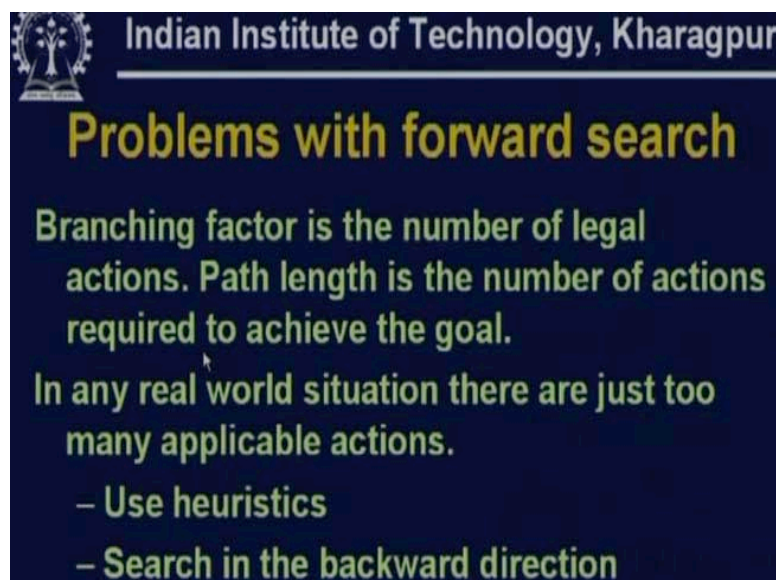
generate all successors of the state. This is called progressing the state through operator application. And each of the successor states so generated are put in the search queue. So basically we can do a breadth first search in the state space. We can go from one state to the states which are distance one ahead and distance two ahead and so on until we get a state which satisfies the goal description. This is the general breadth first search framework and this is called forward planning algorithm.

This forward planning algorithm is also called progression and we can write this algorithm in more detail as follows:

We are given an initial state  $I$  and we need to go to a goal state which satisfies  $G$  so the progression from the current state to the goals given the set of actions available and the path so far. So, if the current state satisfies the goals then we return the path otherwise we choose actions. Choose from the actions those actions such that preconditions of  $A$  is satisfied in the current state. So actions is the set of all actions, we choose a which are part of actions and those  $A$  for which the precondition is satisfied in state. If there is no such  $A$  then we cannot progress that state further. But if such an  $A$  exists or if more than one  $A$  exists then for each  $A$  we get the next state by applying the action  $A$  to state  $S$  to the current state then the goals remain the same the actions remain the same and path is augmented by adding the current action. So this is a recursive formulation of the forward planning algorithm. And first call we call progress initial state  $G$  actions and then empty path.

Now, the main problem with forward search is that in certain domains the number of possible actions that you can execute can be very high. And many of these actions may have no bearing to achieve the current goal. So in such systems forward search is not an effective mechanism. Basically in forward search the branching factor is the number of legal actions.

(Refer Slide Time: 15:02)



Indian Institute of Technology, Kharagpur

## Problems with forward search

Branching factor is the number of legal actions. Path length is the number of actions required to achieve the goal.

In any real world situation there are just too many applicable actions.

- Use heuristics
- Search in the backward direction

The path length is the number of actions that are required to achieve the goal. In any real world situation there are just too many applicable actions, there are too many things that you can do. So if you are considering all possible actions then the search

space can become really huge. Therefore if you want to use forward planning you better use some good heuristics to decide which actions you should use. And the other alternative is do not search forward but search in the backward direction. Let us take a very simple example of forward search.

(Refer Slide Time: 15:44)

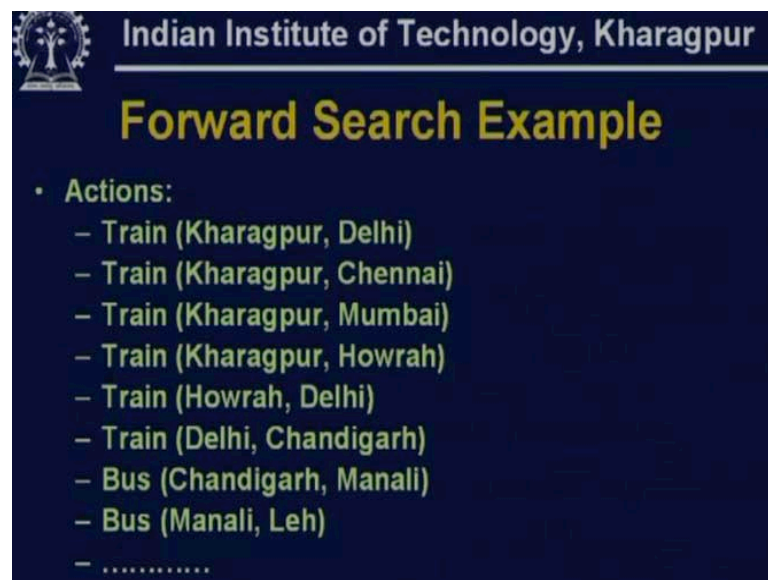


Indian Institute of Technology, Kharagpur

## Forward Search Example

- Initial state: At (Kharagpur)
- Goal state: At (Leh)
- Actions:

(Refer Slide Time: 15:55)



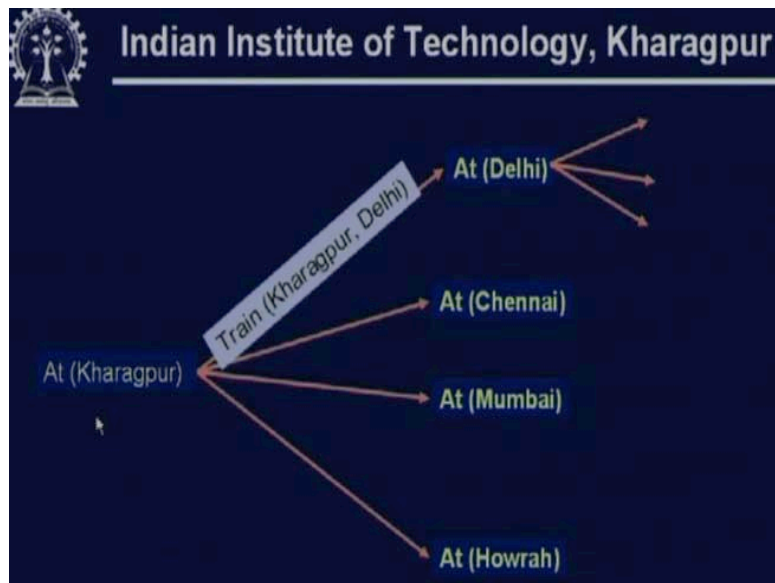
Indian Institute of Technology, Kharagpur

## Forward Search Example

- Actions:
  - Train (Kharagpur, Delhi)
  - Train (Kharagpur, Chennai)
  - Train (Kharagpur, Mumbai)
  - Train (Kharagpur, Howrah)
  - Train (Howrah, Delhi)
  - Train (Delhi, Chandigarh)
  - Bus (Chandigarh, Manali)
  - Bus (Manali, Leh)
  - .....

Suppose initial state the agent is at Kharagpur and the goal state is the agent has to be at Leh and the actions available are; the agent can take a train from Kharagpur to Delhi, take a train from Kharagpur to Chennai, from Kharagpur to Mumbai, Kharagpur to Howrah so there are a huge number of possible actions that the agent can do. Agent can take a train to different places, can take a flight to different places, can walk to different places so the total number of choices is just too huge. Some of these choices are applicable in the current state. Even that can be very large.

(Refer Slide Time: 16:18)



The agent can take a train from Kharagpur to Delhi, Kharagpur to Chennai, Kharagpur to Mumbai, Kharagpur to Howrah and so on. Then for each such state consider what are the possible actions that it can take?


If the agent is not directed by heuristics towards a particular motive then the state space which is explored can be just too huge as you can see here. Now one possibility when the branching is very high is to use backward search.

What does backward search mean?

Backward search means instead of doing a search forward from the initial state the agent does a search backward from the goal state. So the agent finds out what he has to achieve and then finds what actions can achieve the goal. If there are only a few actions with which he can achieve the goal the branching factor can be small in the backward direction compared to the forward direction. But backward search we have to take care of certain issues. You cannot do backward search from the goal just like we do forward search from the start state because for one the goal does not uniquely specify a state but it is only a partial description of the state. **In the goal state we just said these are the conditions that must be true in the goal state.** We do not have the full goal state. But we are given a set of sub-goals that we have to achieve. And then we can consider each sub-goal separately and we consider actions that usually achieve one or more of these sub-goals.



(Refer Slide Time: 18:18)



Indian Institute of Technology, Kharagpur

## Backward Search

An action  $A$  is applicable in state  $S$  in the backward direction if:

- The effect of  $A$  is consistent with  $S$
- There is at least one effect of  $A$  that is part of  $S$

The state resulting from applying  $A$  in the reverse direction (the result of regressing  $S$  through  $A$ ) ?

In backward search we say that an action  $A$  is applicable in state  $S$  in the backward direction if the effect of  $A$  is consistent with  $S$ . There is at least one effect of  $A$  that is part of  $S$  and the state resulting from applying  $A$  in the reverse direction is called the result of regressing  $S$  through  $A$ .

Firstly let us see the actions we will consider from our goal. According to the second point we must consider an action  $A$  whose effect makes at least one condition in the goal valid. So you achieve at least one condition of the goal so you must select such an action. But you must be careful that the effects of the action is not consistent with what you have at the current goal state. So the current goal state says clear  $C$  and your effect is not clear  $C$  then that action cannot be executed. So an action can be executed only if its effects are consistent with the current goal state and it achieves at least one of the propositions in the current goal state. So basically we are doing goal directed actions which achieves some goal. If the number of actions that can achieve a sub-goal is not very high in such cases backward search can turn out to be much more effective than forward search.

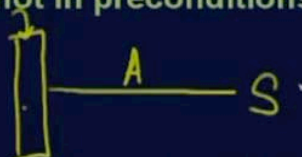
(Refer Slide Time: 19:29)

Indian Institute of Technology, Kharagpur

## Backward Search

The state resulting from applying A in the reverse direction (the result of regressing S through A):

- Precondition of A +
- the variable value assignments of every state variable not in preconditions of A, but in S.



The second question is: How do we find the state resulting from applying A in the reverse direction?

So, that state is obtained by this. First of all we collect the preconditions of A. And secondly we add to it the variable value assignment of every state variable which are not in the precondition of A but in the current state. So, this is my current state and we are considering the effect of the action A.

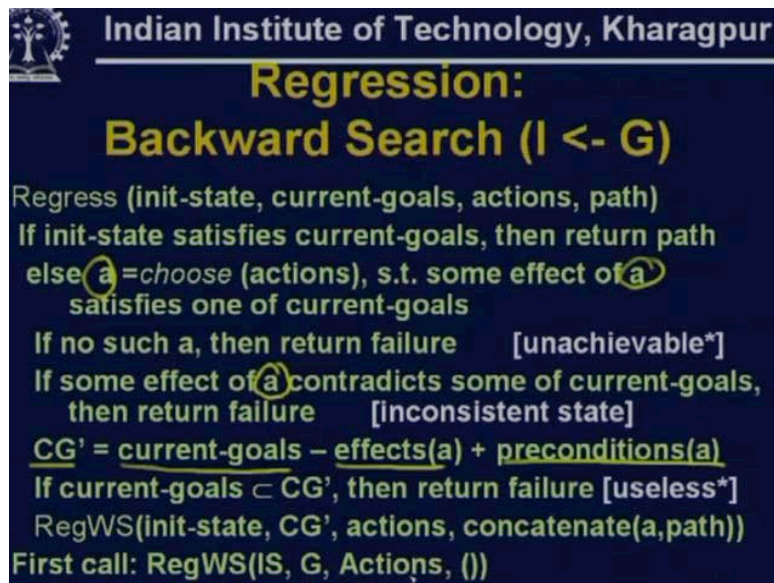
Now, in this state what would be the propositions?

Number 1 all the preconditions of A, number 2 all those propositions which are true in S. But they are not in the preconditions of A we should also add here so that would be the partial description of the state which we get by regressing action A in state S.

Finally in backward search when do we terminate?

In forward search we terminate when the agent gets to the goal. In backward search we terminate when the current backward states partial assignment is consistent with the variable assignment in the initial state. So initial state we have a full description of a set of propositions. If the current backward state is consistent with the initial state description then one can value it. Now let us look at the algorithm for backward search which is called regression.

(Refer Slide Time: 22:02)



Indian Institute of Technology, Kharagpur

## Regression: Backward Search ( $I \leftarrow G$ )

```
Regress (init-state, current-goals, actions, path)
If init-state satisfies current-goals, then return path
else (a) = choose (actions), s.t. some effect of (a)
satisfies one of current-goals
If no such a, then return failure [unachievable*]
If some effect of (a) contradicts some of current-goals,
then return failure [inconsistent state]
CG' = current-goals - effects(a) + preconditions(a)
If current-goals  $\subset$  CG', then return failure [useless*]
RegWS(init-state, CG', actions, concatenate(a,path))
First call: RegWS(IS, G, Actions, ())
```

In regression we go from G the goal state to I the initial state. This function Regress takes the following parameters:

Init state is the initial state current goals are the goals and current goals are the current state of the goals then we have a set of actions then we have the current path. Now in this function if init state satisfies the current goals then we have got to a state from G to I and we have finished so we determine the path. Otherwise we choose those actions from the set of actions such that there is at least one effect of this action which satisfies one of the current goals.

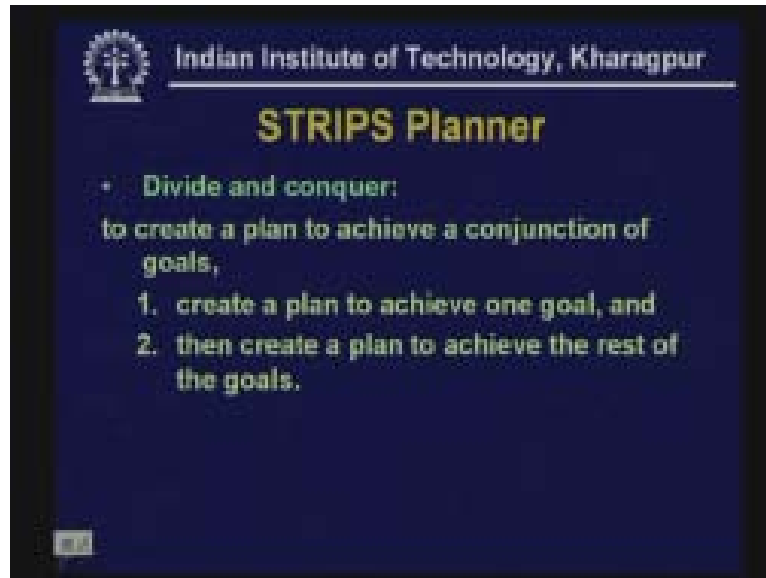
We choose those actions such that some effect of action satisfies the current goals and the effects are not inconsistent with the current goals so we choose all such actions. If there is no such action then we return failure. We say that this goal is unachievable. If some effect of the action A contradicts some of the current goals then also that action leads to an inconsistent state so it cannot be applied. Therefore you only consider those actions which achieve some sub-goal and whose effect does not contradict the current state.

Now we have to find out what is the state that we get by regressing the action in the state? So let  $CG'$  be the set of current goals minus effects A plus preconditions A. So the goal achieves some of the sub-goals. These are the preconditions and these are the effects. So we get the previous state by taking current goals minus what as been achieved as a result of the action A plus we add the preconditions of A. Now we get  $CG'$ .

If the current goal is a subset of the  $CG'$  then we return failure because we do not achieve anything. If we get back  $CG'$  something equal or more then the current goal then it is useless we do not consider such states and we break search at those points. Otherwise we again regress. We call regress with initial state, the current goals become  $CG'$ , actions remain the same and with path we concatenate A. And first time we call regress with the initial state, the goal, the set of actions and the empty path. So this is the regression algorithm or the backward search algorithm. Notice that we

searched from the goal so you have to handle choosing the correct actions, finding out whether the actions are consistent for the current state as you have to look at that carefully and you have to know how to get back the regress state from the current state.

(Refer Slide Time: 25:44)



Now that we have talked about progression and regression let us just go back a little bit and discuss the planning algorithm which is used in the STRIP system. Basically the STRIP system is a divide and conquer search system. So the objective is to create a plan, to achieve a conjunction of sub-goals and this is done in this way. We create a plan to achieve the goals in isolation that is each goal separately and then we combine this plan so we create a combined plan to achieve all the goals. What is done is usually you will serialize the goal. First we achieve one goal then we have the problem of achieving the rest of the goals until all the goals have been achieved. So in STRIPS we divide the goals and then achieve one goal at a time.

Now how does the STRIPS planner achieve a list of goals?

It first chooses one of the goals to achieve. If that goal is not already achieved then the STRIPS planner chooses an action that makes the goal true. So it chooses one goal to achieve, finds an action which makes the goal true backward search then it achieves the precondition of the action. So it finds the new goals that the system must achieve and then it carries out the action and achieves the rest of the goals. This is the algorithm of the STRIPS planner. Now let us see the characteristics of the algorithm and the difficulties we may face.

(Refer Slide Time: 27:38)



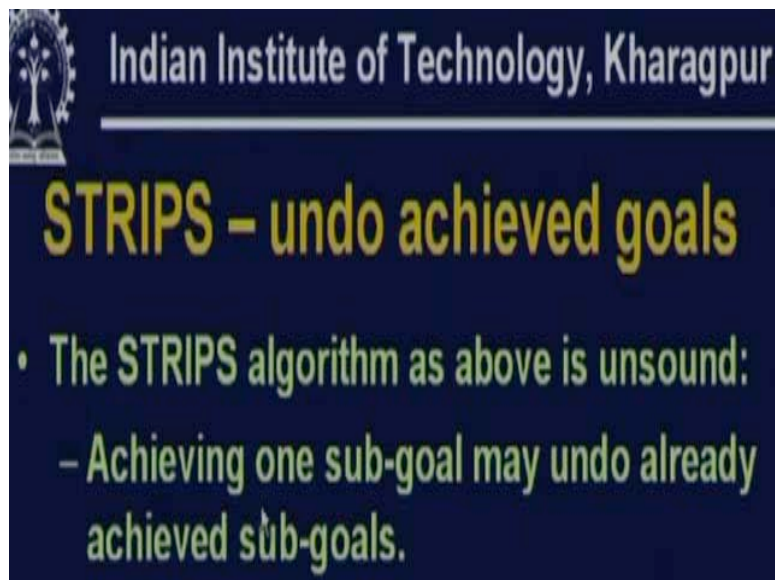
Indian Institute of Technology, Kharagpur

## STRIPS Planner

- To achieve a list of goals:
  - Choose one of them to achieve
  - If it is not already achieved
    - Choose an action that makes the goal true
    - Achieve the preconditions of the action
    - Carry out the action
  - Achieve the rest of the goals

Now the problem is that this algorithm is unsound because suppose the goal  $G$  is specified as a conjunction of  $g_1$  and  $g_2$  and  $g_3$  suppose the agent finds a plan to achieve  $g_1$  so this is the initial state agent takes some actions and as a result of the actions  $g_1$  holds.

(Refer Slide Time: 28:22)



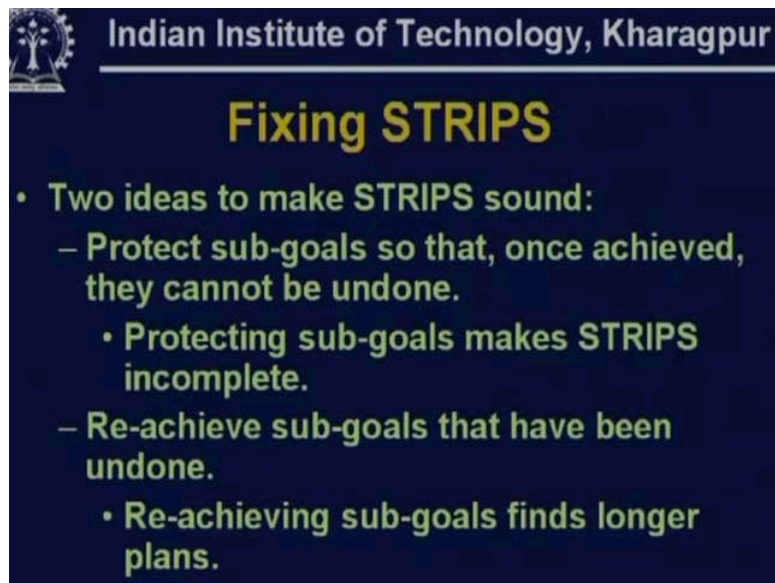
Indian Institute of Technology, Kharagpur

## STRIPS - undo achieved goals

- The STRIPS algorithm as above is unsound:
  - Achieving one sub-goal may undo already achieved sub-goals.

Now the agent tries to satisfy the rest of the goals  $g_2$  and  $g_3$ . But while the agent finds a plan to satisfy  $g_2$  it may happen that in this state  $g_1$  does not hold. So, achieving  $g_2$  may undo the effect of achieving  $g_1$ . So we have to be careful that while achieving the goals in sequence the previous goals are not undone. So STRIPS does not take care of it properly and that is why the STRIPS planning algorithm is unsound. Achieving one sub-goal may undo already achieved sub-goals. In the example we did, achieving sub-goal  $g_2$  may undo the effect of achieving sub-goal  $g_1$  because the objective of the agent is to go to one single state where both  $g_1$  and  $g_2$  and  $g_3$  hold.

(Refer Slide Time: 29:44)



The slide features the IIT Kharagpur logo in the top left corner. The title 'Fixing STRIPS' is centered in a large, bold, yellow font. Below the title, there are two main bullet points in green text, each with a sub-bullet point. The first main bullet point is 'Two ideas to make STRIPS sound:', followed by a sub-bullet 'Protect sub-goals so that, once achieved, they cannot be undone.' and another sub-bullet 'Protecting sub-goals makes STRIPS incomplete.' The second main bullet point is 'Re-achieve sub-goals that have been undone.', followed by a sub-bullet 'Re-achieving sub-goals finds longer plans.'

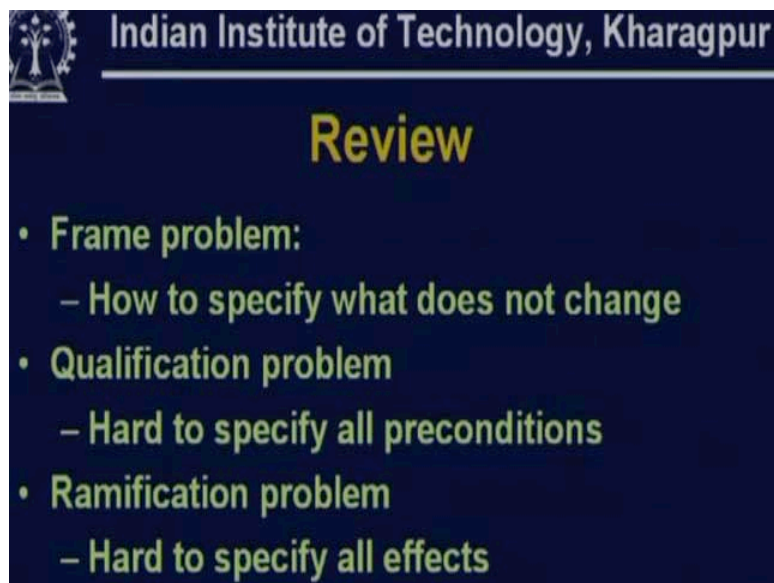
Now how do we fix the STRIPS algorithm?

There are two ideas that have been tried to make STRIPS sound. The first idea is to protect sub-goals. That is, once a sub-goal  $g_1$  has been achieved do not take any action that undo the effect of  $g_1$ . Therefore the first strategy to make STRIP sound is to protect sub-goals which have been achieved. So once  $g_1$  has been achieved do not allow any action that undoes  $g_1$ . Unfortunately sometimes it will not be possible. So, if you have to go to a state in  $g_1$  it may be the case that you cannot get to a state where  $g_2$  will be satisfied unless you undo the effects of  $g_1$ . So such situations can arise that  $g_2$  cannot be achieved without undoing  $g_1$ .

For example, suppose your objective is to make a cake and buy balloons. Suppose the objective is to make a cake and be at the park. Suppose you first try to achieve the goal of being at the park and then you want to achieve the goal of making cake. Now in order to make cake you have to be at home. If you are at home you cannot be at the park. So you cannot achieve the goal make cake without undoing the effect of being at the park. So this condition makes the STRIPS algorithm incomplete.

The second strategy to deal with STRIPS is to re-achieve sub-goals that have been undone. Therefore if you first achieve  $g_1$  and then try to achieve  $g_2$  and  $g_3$  and while trying to achieve  $g_2$  if  $g_1$  is undone then your objective would be after  $g_2$  is completed re-achieve  $g_1$ . Now, if you re-achieve sub-goals you might end up in finding longer plans. Before we move on to the next topic mainly partial order planning we will like to do a brief review of some of the characteristics of planning problems we discussed in the earlier class.

(Refer Slide Time: 31:11)



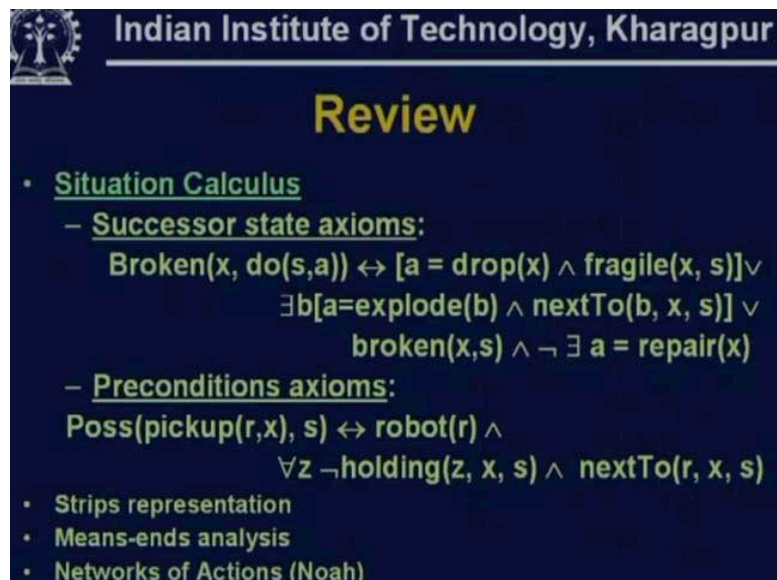
Indian Institute of Technology, Kharagpur

## Review

- **Frame problem:**
  - How to specify what does not change
- **Qualification problem**
  - Hard to specify all preconditions
- **Ramification problem**
  - Hard to specify all effects

The frame problem: In the frame problem we wanted to specify what does not change. So, when the agent carries out some action for example **the preconditions and effects of the action change** but the rest of the world does not change. Then we talked about the qualification problem. In qualification problem we said it is hard to specify all possible preconditions so we try to qualify them. And thirdly we looked at the ramification problem where we said that it is hard to specify all possible effects. Then we looked at situation calculus.

(Refer Slide Time: 33:28)



Indian Institute of Technology, Kharagpur

## Review

- **Situation Calculus**
  - **Successor state axioms:**
$$\text{Broken}(x, \text{do}(s, a)) \leftrightarrow [a = \text{drop}(x) \wedge \text{fragile}(x, s)] \vee \exists b [a = \text{explode}(b) \wedge \text{nextTo}(b, x, s)] \vee \text{broken}(x, s) \wedge \neg \exists a = \text{repair}(x)$$
  - **Preconditions axioms:**
$$\text{Poss}(\text{pickup}(r, x), s) \leftrightarrow \text{robot}(r) \wedge \forall z \neg \text{holding}(z, x, s) \wedge \text{nextTo}(r, x, s)$$
- Strips representation
- Means-ends analysis
- Networks of Actions (Noah)

In situation calculus we looked at successor state axioms. We said that frame problem can be handled by using situation calculus such as successor state axioms. For every action let us say drop action we say that the effect of drop action is broken x. But we also want to say that if the drop action has not be performed x will not be broken. So we say broken x in the state resulting from doing action A from state S if the action is

dropped and if x is fragile or there exist b such that b as exploded and x is next to b and x is broken and we cannot repair x. So we will try to specify all actions which results in broken x and we say no other condition x will be broken. This is a way of incorporating the successor the frame condition. Then in the classical planning framework that we are following we made certain assumptions.

(Refer Slide Time: 34:55)

Indian Institute of Technology, Kharagpur

## “Classical” Planning Assumptions

- Deterministic Effects
- Omniscience
- Sole agent of change
- Goals of attainment

We assumed that all actions have deterministic effect. We assumed that the agent is omniscient and it is a sole agent of change there are no other agents in the world and the agent has to achieve certain goals. Now let us look at a very famous problem the blocks world planning problem and what are the difficulties some algorithms face to solve this problem and basically the problems that STRIPS face in solving this problem. This problem is called the Sussman blocks world problem or called the Sussman Anomaly.

(Refer Slide Time: 35:36)

Indian Institute of Technology, Kharagpur

## Example Problem Instance: “Sussman Anomaly”

Initial State:                      Goal:

C		A
A	B	B
		C

Initial State: (and (on-table A) (on C A) (on-table B)  
(clear B) (clear C))

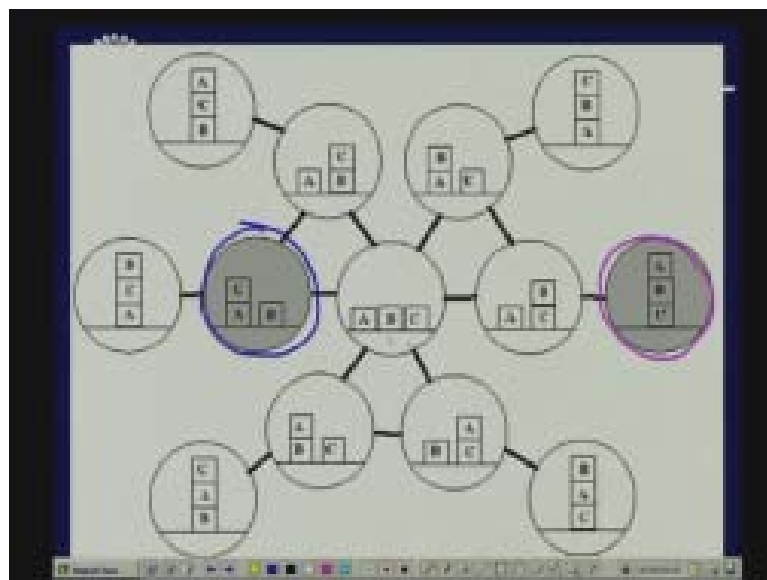


The initial state is given by the following diagram where A is on the table C is on A, C is clear, B is on the table, B is clear. So this is the initial state and the goal state we want A to be on top of B and B to be on top of C. So this is the initial state and this is the goal state. The goal is on A B and on B C. Suppose you take the second of these goals, that is, you want to first achieve on B C.

Now how can you achieve B C?

You can pick up B and stack B on top of C. So, if you put B on top of C you can achieve on B C. Now after you have achieved on B C if you are using the STRIP formalism you would like to achieve the other goal on A B. After you have achieved on B C you get a situation like this B C A. Now if you want to achieve on A B you cannot do it without undoing the effect of the previous goal. Therefore this is a very famous problem for which the STRIPS algorithm would run into difficulty. Now let us look at the blocks world search space.

(Refer Slide Time: 36:32)



This is the initial state of the Sussman Anomaly problem. This is the state where C is on top of A and B is there on the table and A is there on the table. And this is the goal state that we wish to achieve. So if we are using forward search then what would happen?

The agent would get here, here and then from here he can go here, here, he can go to all these places etc so in forward search one can achieve the goal condition. But if you want to serialize the sub-goals in a particular way it might be difficult to achieve this goal. What we can do is, once we have this state space and because it is a very small state space we can do forward search or backward search. So let us look at an example of the progression algorithm applied to this problem. Now, from the initial state the agent has recourse to two possible actions.

The agent can move C from the top of A to B and go to this state or the agent can move C to the table, the agent has a third action of moving B to the top of C so these are the three possibilities that the agent can get to from the current state. Now, suppose the agent chooses this action then the agent will go to this state. Now at this

state also the agent has 6 possible actions from this state. He can move B to top of C, he can move B to top of A, he can move C to top of B and so on so there are 6 possible actions the agent can perform. Suppose the agent takes the action moving B to the top of C again from this state the agent has two possible actions this and this and if the agent takes this particular action and puts A on top of B then the agent reaches the goal state.

So, from the initial state by taking the actions move C from A to table then move B from table to C and then move A from table to B the agent can reach a state in which the goal is satisfied and this is a case of success for the agent. And these choices however are non-deterministic choices of the agent. And in the actual practice the agent will have to do a lot of search. So this is a particular problem where the state space is not very large. Therefore this order problem can be executed by the agent but for large problems search can turn out to be very expensive.

Now let us just briefly look at the regression algorithm we discussed earlier and see what happens when on the same state space we use regression. We start from the goal and then we apply action from the backward direction. Again there is a particular sequence of action move A from table to B then move B from table to C and then move C from table to A. So these actions bring us back to the goal initial state. But again these are non deterministic choices.

(Refer Slide Time: 40:54)

Indian Institute of Technology, Kharagpur

## Regression Example

I: (on-table A) (on C A) (on-table B) (clear B) (clear C)  
 G: (on A B) (on B C)

R(I, G, BlocksWorldActions, ())

R(I, ((clear A) (on-table A) (clear B) (on B C)), BWA, (move-A-from-table-to-B))

R(I, ((clear A) (on-table A) (clear B) (clear C), (on-table B)), BWA, (move-B-from-table-to-C, move-A-from-table-to-B))

R(I, ((on-table A) (clear B) (clear C) (on-table B) (on C A)), BWA, (move-C-from-A-to-table, move-B-from-table-to-C, move-A-from-table-to-B))

current-goals  $\subseteq$  I => Success!

Therefore if the agent takes these choices the agent can get the initial state but finding these will involve a lot of search. The challenge in finding a good planning algorithm is to make the search space smaller. The reason why STRIPS tries to serialize the goal and achieve one goal at a time is in order to make it easier to achieve the goal and cut down on the search space. So, before we proceed further let us look at the comparison between progression and regression. Both the algorithms of progression and regression are sound that is the resulting plan is valid, they are complete and if a valid plan exists they find one. But in these algorithms we need to do a non-deterministic choice and therefore you have to use search.

(Refer Slide Time: 41:44)

Indian Institute of Technology, Kharagpur

## Progression vs. Regression

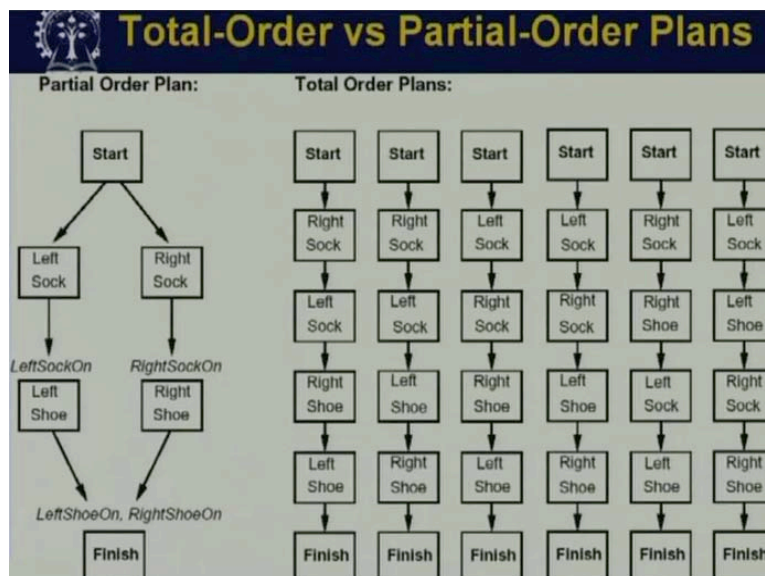
- Both algorithms are:
  - Sound: the result plan is valid
  - Complete: if valid plan exists, they find one
- Non-deterministic choice => search!
  - Brute force: DFS, BFS, Iterative Deepening, ...
  - Heuristic: A\*, IDA\*, ...
- Complexity:  $O(b^n)$  worst-case  
b = branching factor, n = |"choose"|
- Regression: often smaller b, focused by goals
- Progression: full state to compute heuristics

If you are using search you can do different types of search such as depth first search or breadth first search or iterative deepening search, A star, IDA star etc. But as you know the complexity of search is exponential in the worst case. So in the worst case the time taken for search is  $O(b^n)$  where b is the branching factor. In regression usually the branching factor b is small. And in progression we need a larger branching factor. So in progression it is imperative to compute a good heuristics in order to do progression search.

Now we will introduce you briefly to a different type of algorithm for finding plans.

Instead of looking at state space search that we have been looking at so far we have been looking at search in the space of plans or in the space of partial plans. So before we get into that let me explain what a partial plan means.

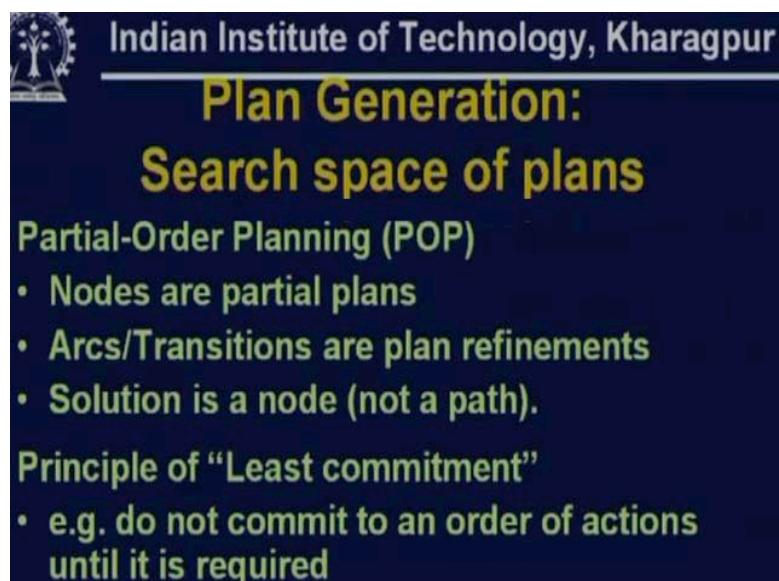
(Refer Slide Time: 43:08)



Suppose this example is from Russell and Norwich's book Artificial Intelligence of Modern Approach. The objective is to put on your socks and shoes every morning before you go out you put on your socks and shoes. Now you can put on your socks and shoes in many different orders but you must put your right sock on before you put your right shoe on. And you must put your left socks on before you put your left shoe on. Now here is a plan to achieve right shoe on and right socks on and left shoe on and left sock on. There are many ways in which you can achieve this. So there are six different ways of achieving this plan. These are called six different linearization of the plan. You could either first put right sock then left sock then right shoe then left shoe or right sock right shoe left sock left shoe or there are six different choices as you can see in this diagram. Or we can represent these choices by the partial order plan.

This partial order plan says you must put your left shoe after your left sock and right shoe after your right sock. But then the order between them may vary. So this is the start state this is the finish state and these four actions have to be performed in any order so long as they satisfy this constraint that left shoe is after left socks right shoe is after right sock. This is an example of a partial order plan. This is the least commitment plan or partial order plan which only expresses the constraints on the ordering of the different actions. Thus, in partial order planning the idea is that we have a search space of partial plans when we search in that space in order to find a partial plan to achieve the current goal.

(Refer Slide Time: 46:00)



Indian Institute of Technology, Kharagpur

## Plan Generation: Search space of plans

**Partial-Order Planning (POP)**

- Nodes are partial plans
- Arcs/Transitions are plan refinements
- Solution is a node (not a path).

**Principle of "Least commitment"**

- e.g. do not commit to an order of actions until it is required

Now, when we talk about partial plans we can immediately handle the problems that STRIP faced with serial solving of sub-goals. We can elegantly solve that problem. Therefore in partial order planning which we call P O P the nodes are partial plans and the arcs or transitions are refinements to the plans and the solution is a partial plan and is not a path. In partial plan we follow a principle of least commitment. That is, we do not commit to a particular order of the action unless it is required.

Now let us see how we represent the partial plan in the P O P framework. A plan is represented by a 3-tuple A, O and L where A is the set of actions in the plan, O is the

set of temporal orderings between the actions,  $A$  less than  $B$  means  $A$  must be executed before  $B$  can be executed and  $L$  is a set of casual links linking actions via a literal. We will see some examples.

(Refer Slide Time: 47:18)

Indian Institute of Technology, Kharagpur

## Partial Plan Representation

- Plan = (A, O, L), where
  - A: set of actions in the plan
  - O: temporal orderings between actions ( $a < b$ )
  - L: causal links linking actions via a literal

$A_p \xrightarrow{Q} A_c$

- Causal Link:  
Action  $A_c$  (consumer) has precondition  $Q$  that is established in the plan by  $A_p$  (producer).

(clear b)

move-a-from-b-to-table  $\longrightarrow$  move-c-from-d-to-b

This is a notation; this is an example of a casual link  $A_p$  to  $A_c$  to  $Q$  which means that the action  $A_c$  as a precondition  $Q$  and this precondition is achieved by the action  $A_p$ . So the action  $A_p$  achieves a condition  $Q$  which is required for the action  $A_c$  to be executed. These are called **causal links**. Therefore we have a set of nodes which correspond to partial plans, we have a set of temporal links and a set of **causal links**. Temporal link simply says that if we have two actions  $A_1$  and  $A_3$  and we say that this is a temporal link that means  $A_1$  must occur before  $A_3$ . Causal links  $A_p$  to  $A_c$  means  $A_p$  achieves some condition  $Q$  which is required as a precondition for  $A_c$ .

For example, from the blocks world domain let us take the following example: The action moves  $c$  from  $d$  to  $b$ . Suppose this is  $d$ , this is  $c$  we want to move  $c$  from  $d$  to  $b$  and it requires that  $b$  is clear, moving  $c$  from  $d$  to  $b$  requires that  $b$  is clear. And  $b$  is clear is achieved by the action move  $a$  from  $b$  to table. So this action achieves clear  $b$  and clear  $b$  is required for this action. So we will have a causal link from the action move  $a$  from  $b$  to table to move  $c$  from  $d$  to  $b$ .

Now, suppose we have a **causal link** from  $A_p$  to  $A_c$  we cannot insert an action another action or third action between these two the actions if that action threatens this causal link. Now which other actions can threaten this causal link? A step  $A_t$ . So, if we have a casual link from  $A_p$  to  $A_c$  on  $Q$  so  $A_p$  achieves  $Q$  and  $A_c$  requires  $Q$ . Suppose this is a temporal link and if I put in an action  $A_t$  between  $A_p$  and  $A_c$ ,  $A_t$  is a temporal link such that  $A_t$  has NOT $Q$  as its effect. So if  $A_t$  has NOT $Q$  as its effect what will happen?

(Refer Slide Time: 48:44)

Indian Institute of Technology, Kharagpur

### Threats to causal links

Step  $A_t$  threatens link  $(A_p, Q, A_c)$  if:

1.  $A_t$  has (not  $Q$ ) as an effect, and
2.  $A_t$  could come between  $A_p$  and  $A_c$ , i.e.  $Q \cup (A_p < A_t < A_c)$  is consistent

What's an example of an action that threatens the link example from the last slide?

$A_p \xrightarrow{Q} A_c$   
 $A_t$  (interfering)

$A_p$  achieves  $Q$ ,  $A_t$  achieves  $\neg Q$ . So, if  $A_t$  achieves  $\neg Q$   $A_c$  cannot be executed immediately after  $A_t$  because this  $Q$  which  $A_c$  requires is gone. So a step  $A_t$  can threaten the causal link from  $A_p$  to  $A_c$  on  $Q$  if  $A_t$  as  $\neg Q$  as an effect and in that case  $A_t$  cannot come between  $A_p$  and  $A_c$ . Let us look back at the example from the last slide. Move a from b to table achieves clear b and then we can execute move c from d to b.

Now can you tell me an action that threatens this causal link?

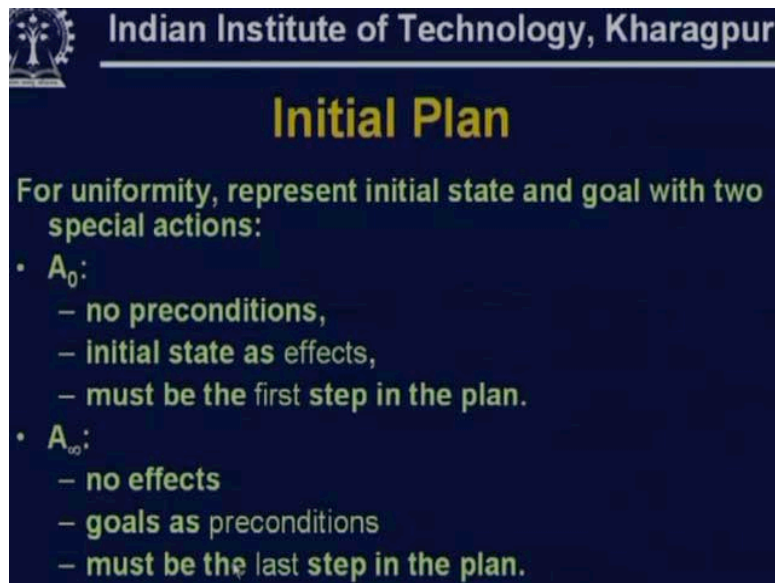
Suppose between these two actions (... 51:19) in another action which is move x to b, if you have move x to b now what happens?

b is no longer clear as a result of this action so not clear b. So if b is not clear this action cannot be executed. So, if we have two causal links and this action achieves the move a from b to table action achieves clear b which is required for move c from d to b action no other action should come in the middle that undoes this action. As we said we will search in the plan space now what would be our initial plan?

We will introduce two special initial plans. So initially we have two special actions.

One we call  $A_0$  and one we call  $A_{\text{infinity}}$ . We will put these two special actions.

(Refer Slide Time: 52:33)



Indian Institute of Technology, Kharagpur

## Initial Plan

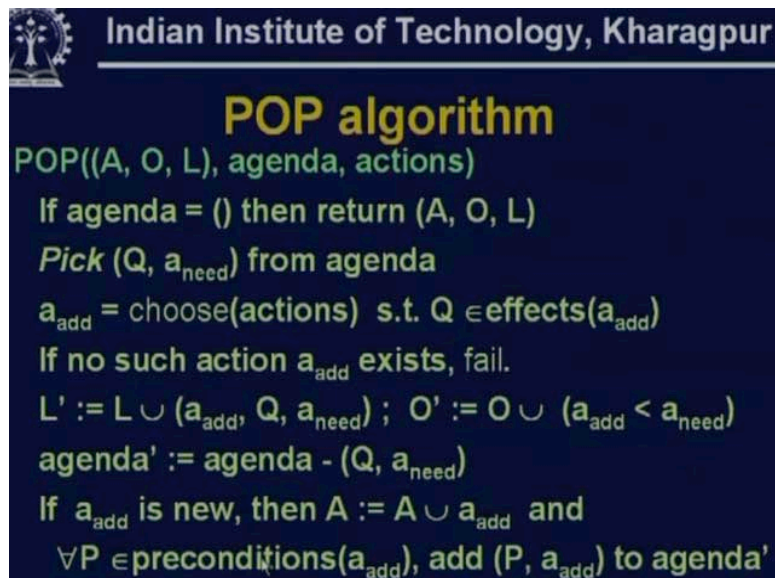
For uniformity, represent initial state and goal with two special actions:

- $A_0$ :
  - no preconditions,
  - initial state as effects,
  - must be the first step in the plan.
- $A_\infty$ :
  - no effects
  - goals as preconditions
  - must be the last step in the plan.

What is  $A_0$ ?

$A_0$  is a special action which has no precondition and the effects of  $A_0$  is the description of the initial state. So it has null precondition and its effect is the initial state. We also introduced action  $A_{\infty}$  whose preconditions are the goals and it has no effect. So  $A_0$  must be the first step in the plan and  $A_{\infty}$  must be the last step in the plan. So this is what we constrain any partial plan to have  $A_0$  as the initial action and  $A_{\infty}$  as the final action.

(Refer Slide Time: 53:53)



Indian Institute of Technology, Kharagpur

## POP algorithm

$POP((A, O, L), agenda, actions)$

If  $agenda = ()$  then return  $(A, O, L)$

Pick  $(Q, a_{need})$  from agenda

$a_{add} = \text{choose}(\text{actions})$  s.t.  $Q \in \text{effects}(a_{add})$

If no such action  $a_{add}$  exists, fail.

$L' := L \cup (a_{add}, Q, a_{need})$ ;  $O' := O \cup (a_{add} < a_{need})$

$agenda' := agenda - (Q, a_{need})$

If  $a_{add}$  is new, then  $A := A \cup a_{add}$  and

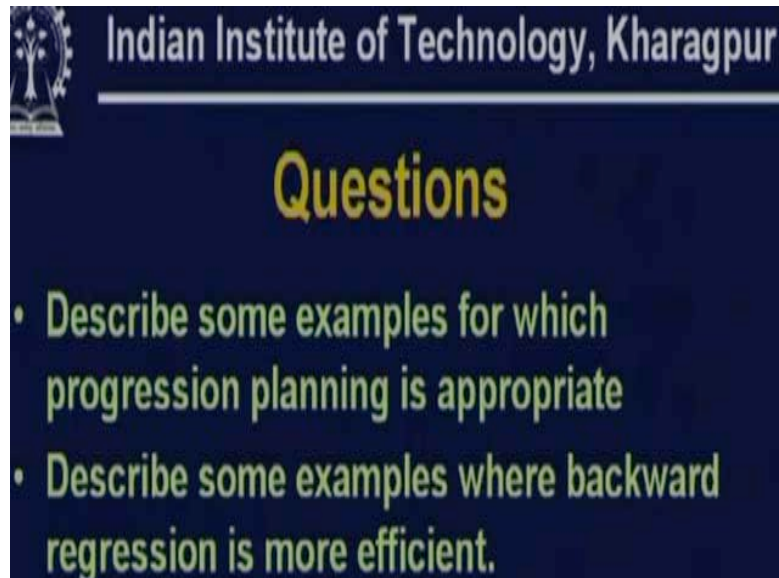
$\forall P \in \text{preconditions}(a_{add}), \text{add}(P, a_{add})$  to  $agenda'$

In the next class we will cover the partial order planning algorithm in greater detail.

The basic idea in the partial order planning algorithm is that we start with an empty plan which has only the initial action and the last action and no other commitment then we try to insert other actions so that goals are achieved. And where do we pick the goals from?

We try to find out a sub-goal which has not yet been achieved and we try to find an action which satisfies some of the sub-goals and we continue until we get a correct plan. So we will discuss in detail the partial order planning algorithm in the next class and then show how it works. Come up with some examples of planning problems for which forward planning is an appropriate choice for which progression can be used. Then think of some example planning problems for which backward search or regression is the right choice.

(Refer Slide Time: 55:25)



Indian Institute of Technology, Kharagpur

## Questions

- Describe some examples for which progression planning is appropriate
- Describe some examples where backward regression is more efficient.