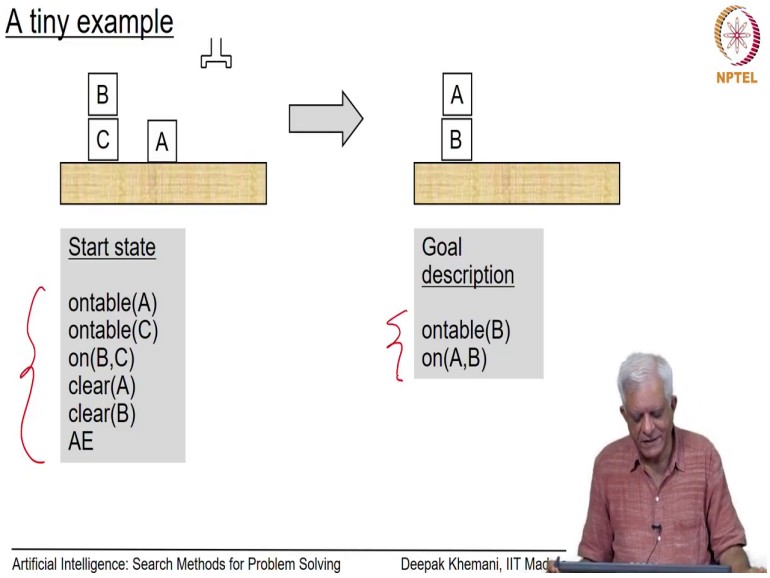
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 07 and 10
A First Course in Artificial Intelligence
Lecture – 64
PSP: A Tiny Example

(Refer Slide Time: 00:14)

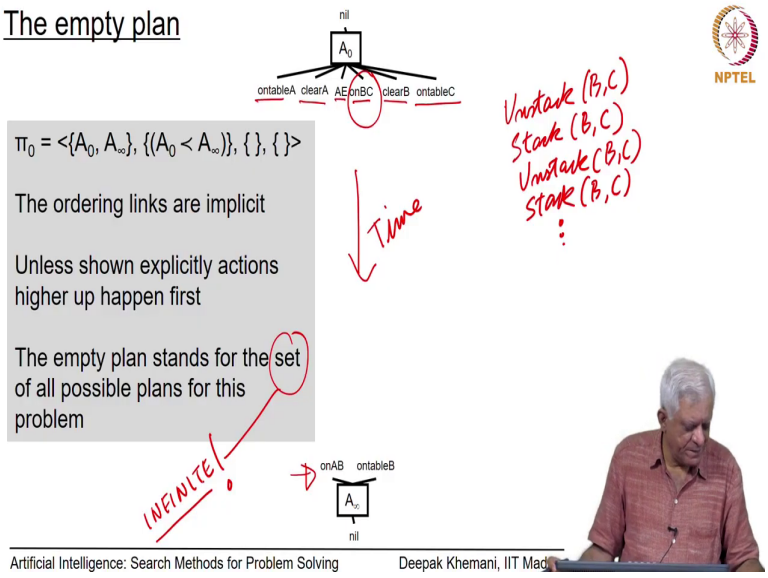


So, welcome back, we just finished a study of the plan space planning algorithm. And, just to recap we start by creating an empty plan of two actions A 0 and A infinity. A 0 produces the predicates of the start action and if you look at this particular example the predicates are these.

So, these must be the effects of the initial action A 0 when we start doing plan space planning. These actions are basically describing a diagram on the top and which says that A is on the table, C is on the table, B is on C, there is nothing on top of A and there is nothing on top of B and the arm is empty.

The last section A infinity will consume the goal predicates, the goal predicates are two in this case. One says that B is on the table and other says that A is on B; we do not care about anything else. We do not care where C is and whether the arm is holding C or whether the C is somewhere else; maybe it could be on A or maybe it could be on the table. It does not matter to us.

(Refer Slide Time: 01:43)



So, this is a starting plan that we begin with. So, as you can see the goal the star predicates are the effects of the start action A 0. And, the goal predicates have been consumed by or will be

consumed by the last action A infinity. The plan as we said before is A 0, A infinity and ordering link which we are not showing here; just so that with the diagram does not get cluttered up essentially.

We will assume in the next few slides that unless shown explicitly the actions which are higher up in the slide happen before actions which are lower down in the slide. It basically means that time is flowing like this on top to bottom; any action which is higher happens first and the any action which is lower happens later.

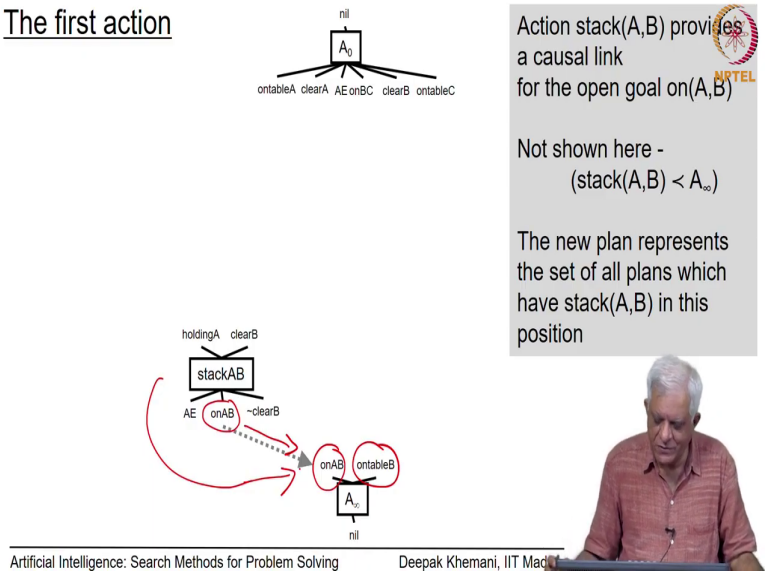
Now, this empty plan A 0, A infinity stands for the set of all possible plans that you can construct in the (Refer Time: 02:53). Now, it so turns out that this set is actually infinite. Why is this infinite? Because you can simply have a sequence of actions, which go into a kind of an infinite loop or if not in any infinite loop in a small loop. So, for example, you as you can see you have onBC in this star state.

So, you can say unstack B from C, then you can change your mind and say stack B onto C, then you can again say unstack B from C and again you can change your mind and say stack it on to C. So, we can have a sequence of plans that unstack B from C, then stack it back; nothing is stopping us from doing that. Then again unstack B from C and again stack it back and so on.

We can do this any number of times and any plan which incorporates these four actions would still be a valid plan. Of course, it would not be an optimal plan ah, but it would still be a valid plan. So, in that sense the space explode by plan space planning even for this tiny problem, that we are looking at is infinite.

Whereas, the space explode by state space planning forward, state space planning backward, state space planning, goal stack planning would have been finite essentially. So, that is one little bit of a difference between the two approaches.

(Refer Slide Time: 04:44)

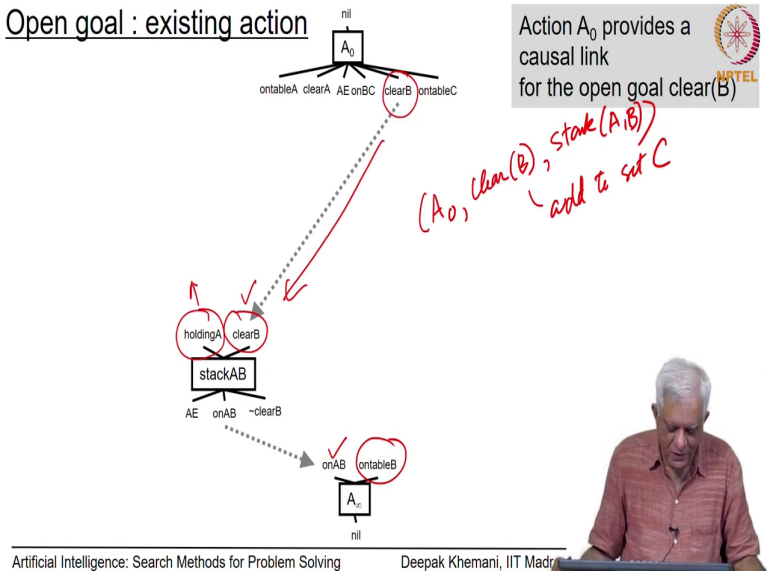


So, we add the first action, remember that basically the planning algorithm is driven by solve removing flaws. And, to start with we had two flaws in this thing there were two open goals and one of the ways of catering to an open goal was to add an action. So, clearly as we have seen by now that to cater to this open goal on A B, the action that is relevant is stack A on B.

So, we add this action to the plan essentially. So, what does that mean? That we add this action stack A, B into the first set which is the set of actions, we establish a causal link from AB to on onAB to onAB and we add that as a explicative element in the set of causal links. And finally, what is not shown here is that we act add a ordering link.

We should add a link saying that this action happens before this action. But, we will not show it because the slide would become more cluttered; we show it towards the very end.

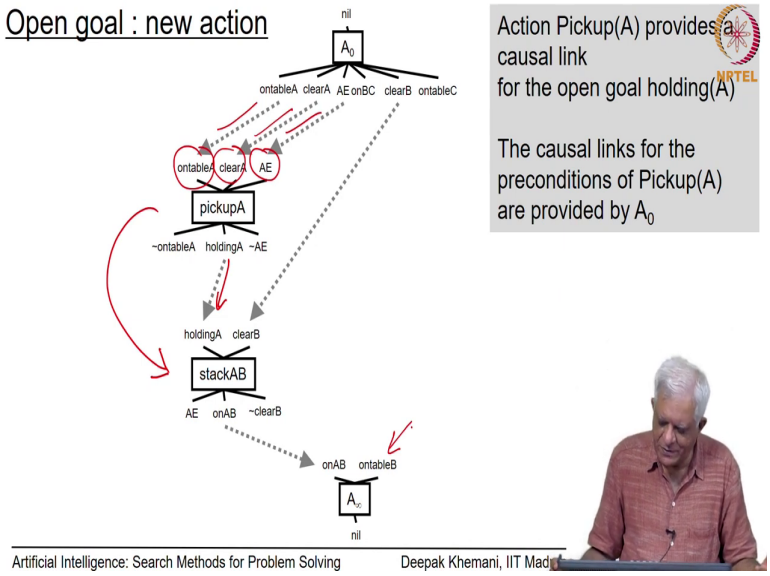
(Refer Slide Time: 06:08)



Now, another way of solving an open goal is to find an existing action. Now, you can see here that clearB is fit is true in the in the start state which means it is produced by the first action and that is what you needed. So, you can just go ahead and establish a causal link between this.

The causal link is represented by this arrow, it would be added to the plan by saying that this triple A 0 produces clear B and stack A, B consumes clear B. So, we add this to the set which we called as C essentially. So, we add it and as we have been seeing we will also add an ordering link between these two actions, the other.

(Refer Slide Time: 07:13)



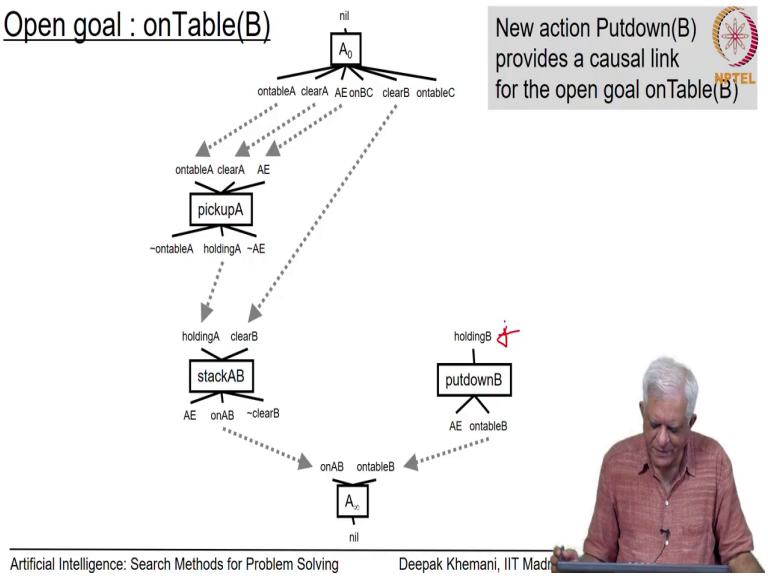
So, what we saw here was one open goal. So, at this moment you can see that this open goal has been taken care of, this open goal has been taken care of, this one still remains and this one still remains. So, let us assume that our algorithm is going to address this open goal holding A and because to execute the action stack A on B, we need holding A to be true.

So, we need a causal link for holding A and that will be provided to us by Pickup A. We could have chosen like in goal stack planning either Pickup A or we could have chosen unstack A from something. But, let us say that somehow we have managed to choose this action Pickup A, inserted into the plan.

That means, added to the set of actions, establish this causal link here, establish the ordering link which we have not shown and so on. And, it steps that I will skip for each of these open goals, we can establish links from the start action because they are true in the start state.

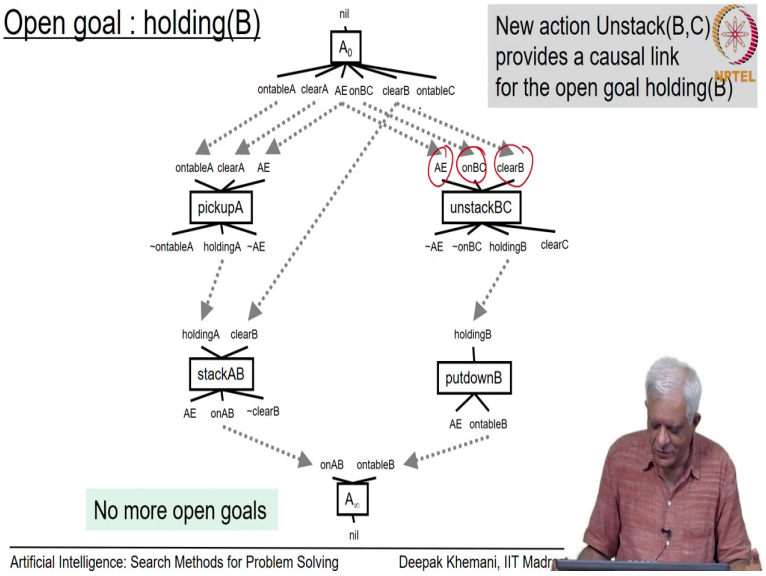
So, we have now addressed many open goals, but we are left with one which is to be addressed still. And, you can see that that says that B should be ontable and we know that the action which will do that is to putdownB.

(Refer Slide Time: 08:49)



So, we add that action to the plan, add the corresponding causal link, add the corresponding ordering link. And, then we find that there is still one open goal left in the plan which is holdingB; holdingB like holdingA that we saw earlier could have been achieved either by unstacking B from something or by a picking up B from the table.

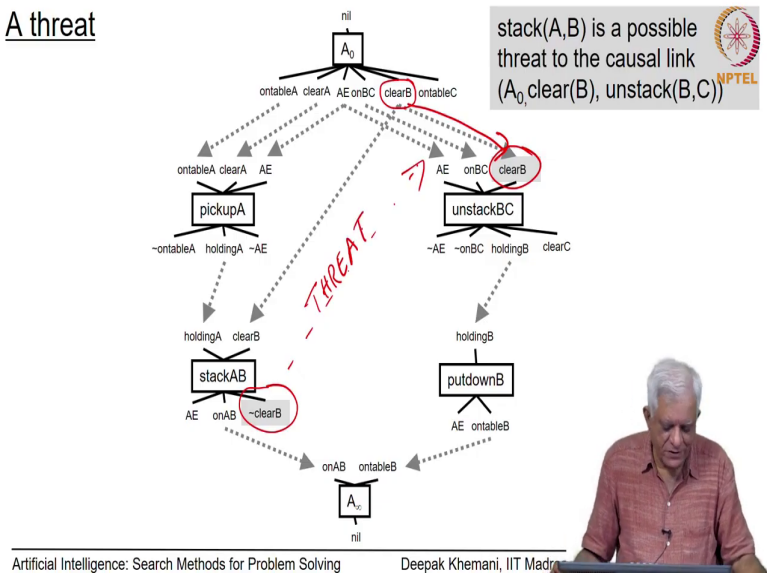
(Refer Slide Time: 09:23)



And, let us say somehow we managed to pick the right action in this case; maybe we can look at the effects of the start state or maintain that somewhere to guide search in a way; so, so that we only select meaningful actions ah. But, even if we cannot do that, we can always backtrack and try something else.

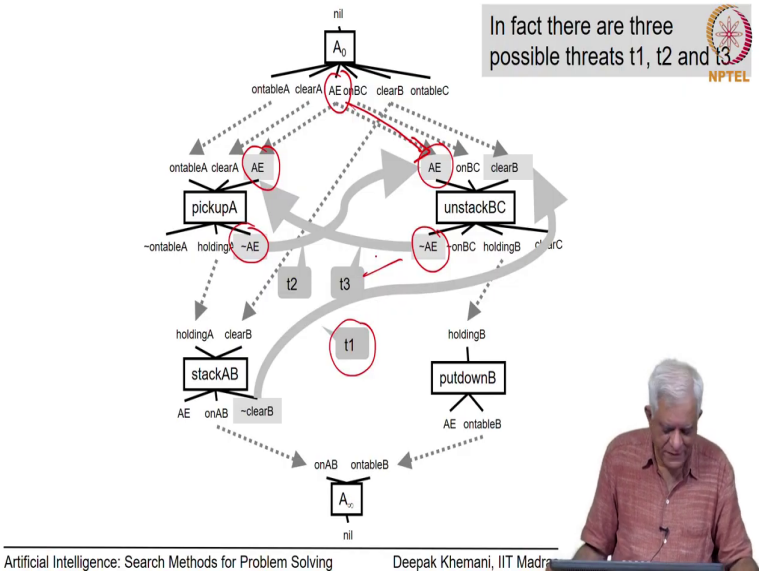
So, the action that we have added now is unstackB from C and the preconditions for this action which are arm empty, onBC, clearB are all true in the start state. And, at this point we have solved all open goals; there are no more open goals in the plan.

(Refer Slide Time: 10:06)



What about threats? So, let us look at what happen what is happening with threats. You can see that this action stack A, B is a threat because there is a causal link between clearB and clearB here and this is doing not clearB. So, clearly this action is a threat to that causal link.

(Refer Slide Time: 10:44)



So, at least there is one threat, but it so turns out that in fact, there are three threats; all of them are to do with arm empty. It is not surprising that this is a case because, the problem that we have defined the domain we have working in the STRIPS domain. It works with a one armed robot and this one arm robo can only handle one thing at a time.

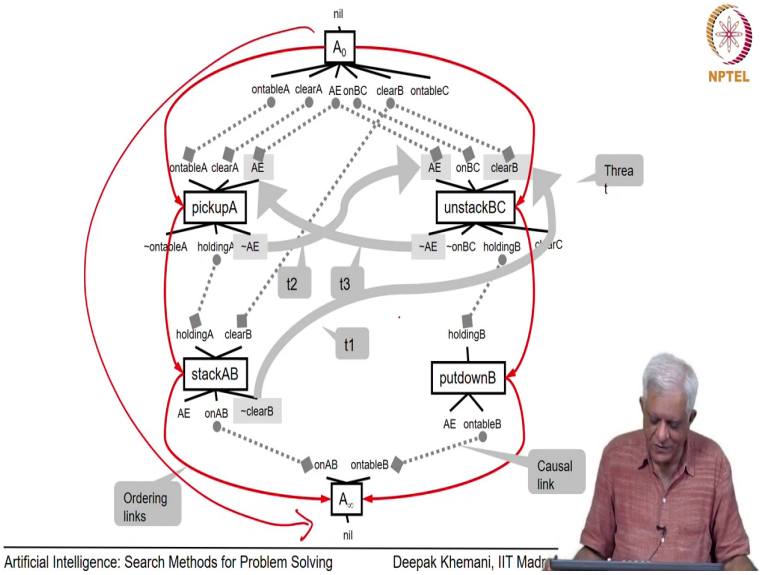
So, the moment it picks up something it arm is no longer empty and if some other action wanted arm to be empty, then clearly there would be a threat somewhere. And, we can see that there are three threats here in this case. The first one we have already identified as we let us call it t1. But, there is another threat which says that pickupA also produces not arm empty and unstackBC wanted to consume arm empty.

So, pickupA is a threat for this link which is this link right. And, likewise unstackBC is producing not arm empty and that arm empty would have been consumed by pickupA. So, we

have this third threat which is t3 essentially. So, we have three threats and we can remove them one by one. Now, in this example there are no variables ah.

We are not said unstack B from something; we could have said that, but it would not have affected our plan. We just took the liberty of choosing the action unstack B from C because; we could see that onBC was true in the start state.

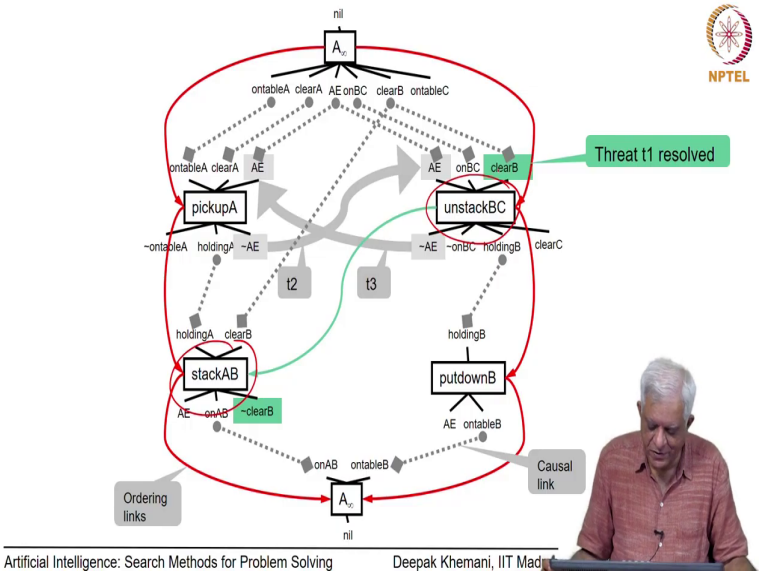
(Refer Slide Time: 12:29)



So, we can remove this states one by one. So, here I have introduced the ordering links which we have found so far. When we added each of these four actions, we added some ordering links for those actions. And of course, there is a original ordering link which says that this must happen before this thing which I have not again drawn here.

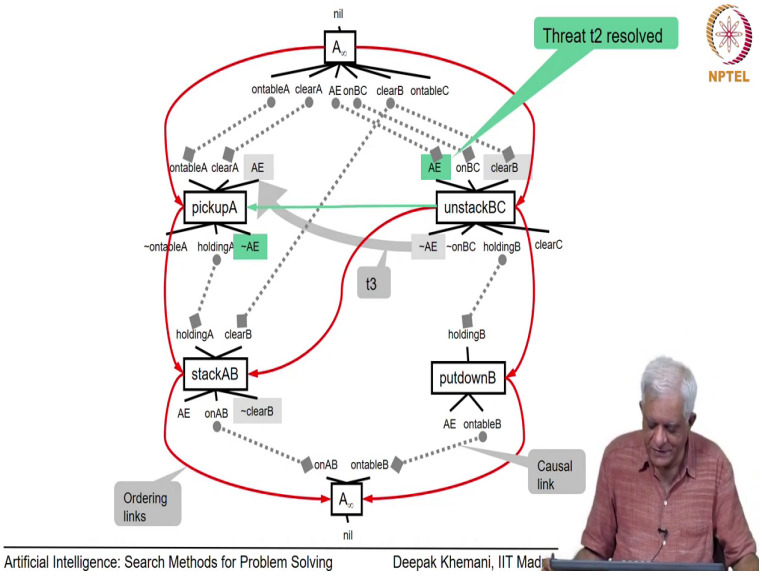
So, these are the red coloured links are the ordering links, the dotted links are the causal links. And the this thick shaded, this thick grey allows these are the threats to the causal links.

(Refer Slide Time: 13:09)



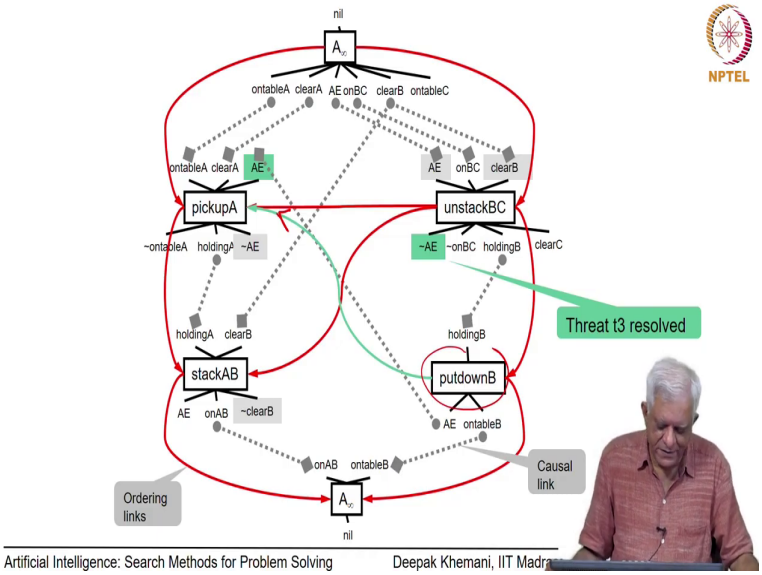
So, there are three threats as we saw. The threat number 1 can be resolved by saying that unstackBC which is this action must happen before stack B on C. So, once it is unstackBC is done; that means, it is consumed arm empty and it is no longer bothered with about the fact, that stackAB is going to sorry its consumed clearB. And, its no longer bother about the fact that stackAB is going to destroy or clobber clearB.

(Refer Slide Time: 13:43)



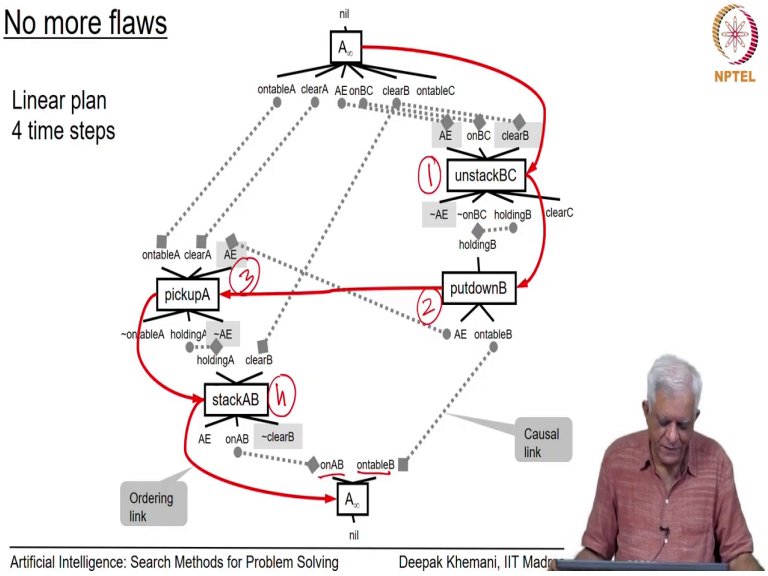
Likewise we can add ordering link to say that unstackBC must happen before pickupA. So, that this arm empty condition is not conflicting as it was in threat 2.

(Refer Slide Time: 13:55)



And, in a similar fashion by saying the putdownB must happen before, this is a link that we are adding pickupA and then the third threat will also vanish. So, by at this point we will see that there are no more threats in the plan.

(Refer Slide Time: 14:20)



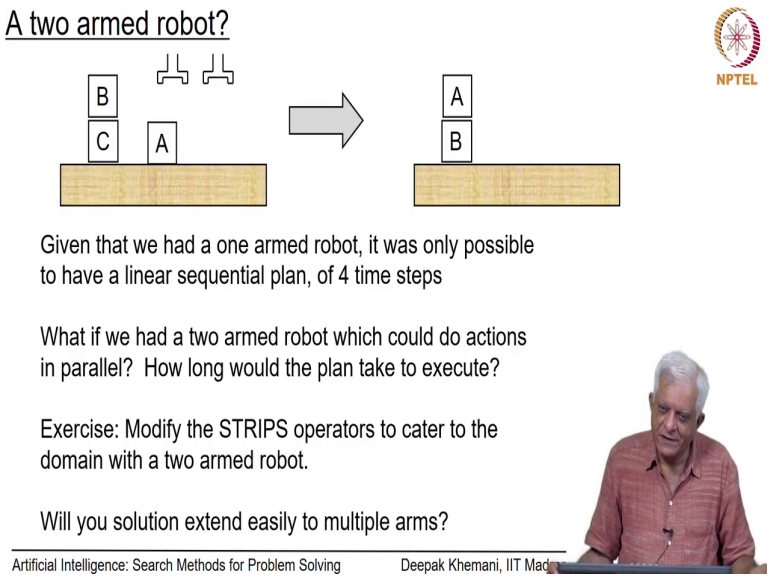
And, we can kind of draw a neater version of this to say that this is what finally matters. There are other ordering constraints that we spoke about. But, eventually what this produces is a plan which happens to have a linear structure which says that you in the 1st action that you do is unstack B from C.

The 2nd action in do is putdownB, the 3rd action you do is pickupA and the 4th action you do is stack A on to B. So, at the end of which you have stack A on B and you have B on the table essentially and, this is a plan.

Now, it is not surprising that it is a linear order because, as I have said that we are working in a domain in which there is a one armed robot. And, this robot will do this four actions in a

sequence one by one and produce the final goal, a produce the plan which will achieve our goal.

(Refer Slide Time: 15:20)



What if we had a two armed robot? So, for the same problem that we are trying to solve, let us say that the domain is allowing us to or the domain has a robot which has got two arms. In the given domain as it was in the STRIPS domain, it was only possible to have a linear sequential plan of 4 time steps. What if we had a two armed robot which could do actions in parallel?

So, now as you can see in the diagram there is a two arms and how will that change the process? So, as you can imagine for example, the two arms could; so, one arm could unstack B from C; while other arm is picking up A. So, these two actions could be done parallel. So,

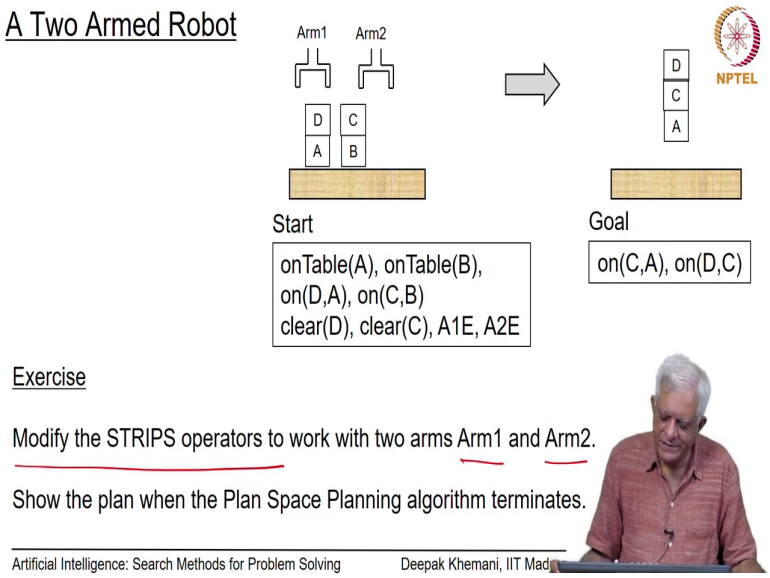
maybe you can get a shorter plan in terms of the number of time steps. We called that as makespan in terms of planning.

And, it is possible that these things may happen in parallel essentially. How long would the plan take to execute? So, that is a small exercise I would ask you to think about and we will take it up in the next session. The real problem here is to for you to modify the STRIPS operators to cater to the domain in which there is a two armed robo.

Now, in the original STRIPS operators that we have we had actions like pickup for example, A. Now what is the meaning of pickupA in this world where is the arm the robot has two arms? We should be able to specify which arm is picking up A and that needs to be now part of the modified domain description.

And, I will leave that as a small exercise for you. I will give you one suggestion now, but hopefully you will do look for better suggestions and come back before the next lecture. In particular what I would like to ask is; does your solution extend to multiple arms?

(Refer Slide Time: 17:41)



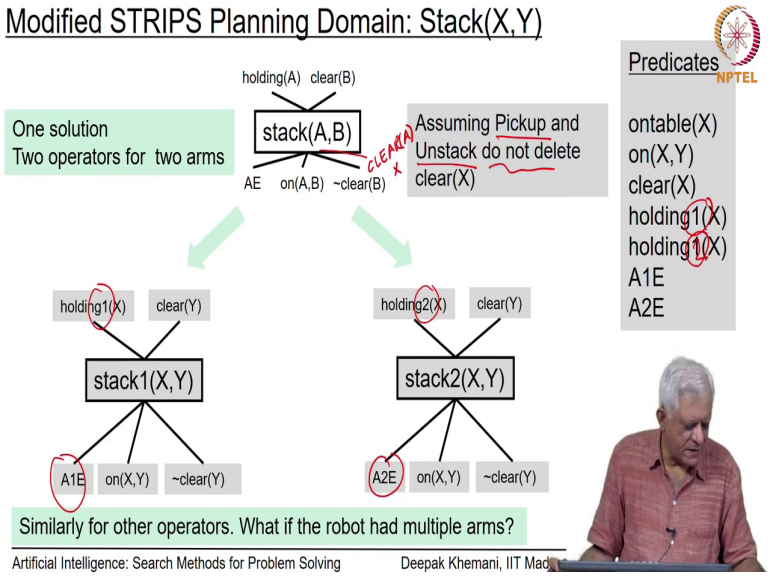
So, let us take the two arm thing first and here is another problem for which I will give you a solution a little bit later. This is a problem in which in the starting state there are two stacks D is on A and C is on B. And, you have two arms Arm1 and Arm2. And, the goal state is that you have to stack D onto C and C onto A essentially.

So, as I said modify the STRIPS operators to work with Arms1 and Arm2 and also as an exercise show the plan that partial space a plan space planning will produce at the end of it. So, the first thing you should be able to observe is that it is not necessary that the plan that is called as a solution plan has a sequence of actions.

The very fact that we have introduced two arms into our domain allows for the possibility of parallel actions. And, you should be able to observe how that would be possible in plan space

planning, how would the planner give us a plan which could be executed in parallel wherever it can essentially.

(Refer Slide Time: 19:01)



Let me give you one brute force or simple method and this method says now here you have this stack operation. There is something that I have not kind of emphasized during this talks on STRIPS planning is when you for example, pickup an object from the table or unstack it from another object that object X let us say.

So, at the moment when you pickup or unstack, one of the precondition says that object must be clear. So, clear X must be true. So, should we delete clear X or should we let it be? The reason for letting it be would be that anyway after you picked it up and after you have all after you have unstacked it, the only thing you can do is put it down or stack it on to something else and then anyway it will become clear.

So, why bother deleting clear X in pickup and unstack and then adding it again in put down all stack. So, at least you know for the single arm robot that did not make too much sense and so, let us begin with that. You may want to consider the possibility of our arm robo being very versatile, two arm robo being versatile and being able to for example, stack one block on another while holding the other block.

So, then we have to be a bit careful about this clear predicate. In the simpler case we assume that, that we had not deleted clear X. So, when we stack a onto B for example, in this case we do not have to add another effect which says clear A. So, because you have assumed that they are not deleted we are not adding this here, but that would be an option essentially.

But, then you would have to modify a pickup and unstack appropriately. So, working with this, this version where we are not deleting that clear thing; the simplest thing to do is to simply have two separate actions just with different names for the two arms with. And, without going into too much creativity let us just assume that the action is called stack1 and the other action is called stack2 essentially.

So, they are just different names, I could have given them any other names; after all there is what is in a name you know, the Shakespeare said a rose by any other name would smell as sweet. So, from the planning perspective or from the company's perspective and action by any other name would still do the same thing. Now, but the simplest brute force approach is to just introduce two new actions.

One is called stack1 and the other is called stack2 and the precondition we need to modify the predicates. So, that we have now holding1 and it should be holding2. So, holding1 is a precondition for 1 and holding2 is a precondition for the other and likewise the arm which becomes empty is clearly specified. So, we are just treating this as a separate action, they are just two different actions.

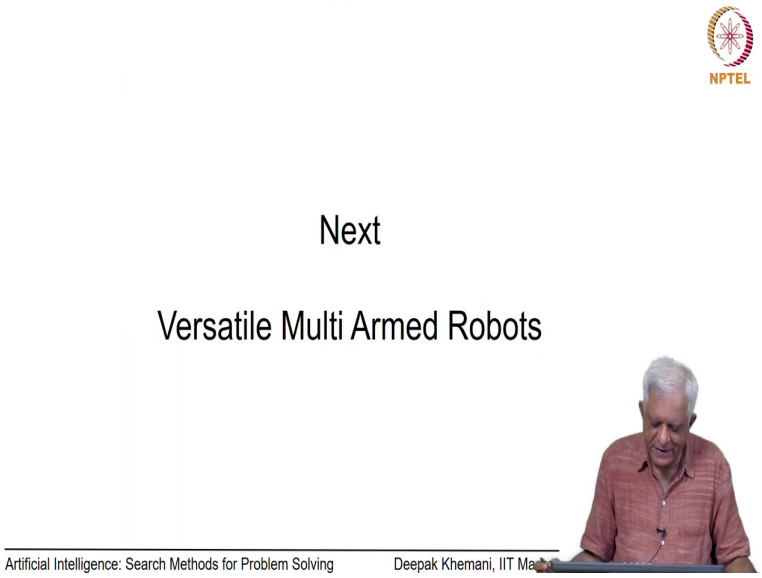
In some sense we have separating the world of Arm1 and Arm2, but what would happen if we had multiple arms essentially? What if I give you a robot with two arms and three arms or four arms and things like that?

Maybe you should look for an way of modifying the STRIPS actions; so, that they can be extended easily to any number of arms essentially. The other thing that I would like you to think a little bit about is that if you wanted to allow the robot to become versatile which means that it can.

For example with one arm it can pickup block A, on another arm it can pickup block B and then while holding it, it can put A onto B without putting it down on the table essentially. So, if you wanted to do that sort of a thing, where you could do things in parallel or let me take you back to the example that we started with. Supposing, in the first time step Arm1 pick on stacks B from C and Arm2 unstacks or picks up A from the table.

Then is it possible for the two arms to in one time step achieve this goal of saying that B is been put down, but at the same time A is been stacked onto B; how would you think of modifying the planning operators to do that? So, I will leave you with this small exercise and we will ignore this simple solution that I have given.

(Refer Slide Time: 24:21)



And, in the next session we will see how to model versatile multi armed robot essentially. If there are many robots, many arms in a robot or maybe more than one robot with let us say two arms each.

And, if they are versatile in the sense that they can manipulate objects which are not on the table; then how would you model them? One place where this kind of a approach would be meaningful is if you are talking of an a multi agency scenarios.

So, if it is not just one robot that you have many robots and they have to you know coordinate and collaborate and then do things together. Then as an example for example, if there is a table that you want to move from one corner of the room to another corner.

Then the two robots have to coordinate their actions in such a manner that both of them lift the table from either end, from the two ends simultaneously and carry it and then put it down at the same time. So, it is just an exercise in trying to see how you can extend simple STRIPS domains to more complex domains. We will do a little bit in the next session.

So, see you then right.