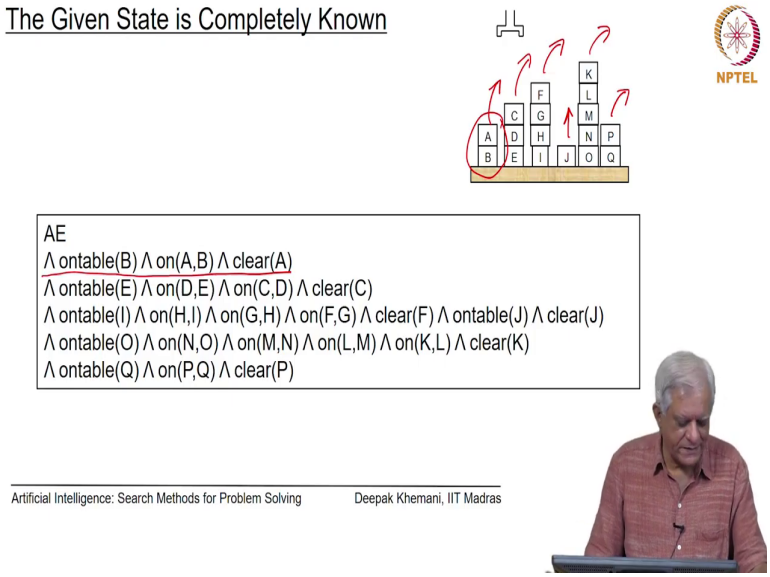
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 07 and 10
A First Course in Artificial Intelligence
Lecture – 60
State Space Planning: Forward and Backward

(Refer Slide Time: 00:16)

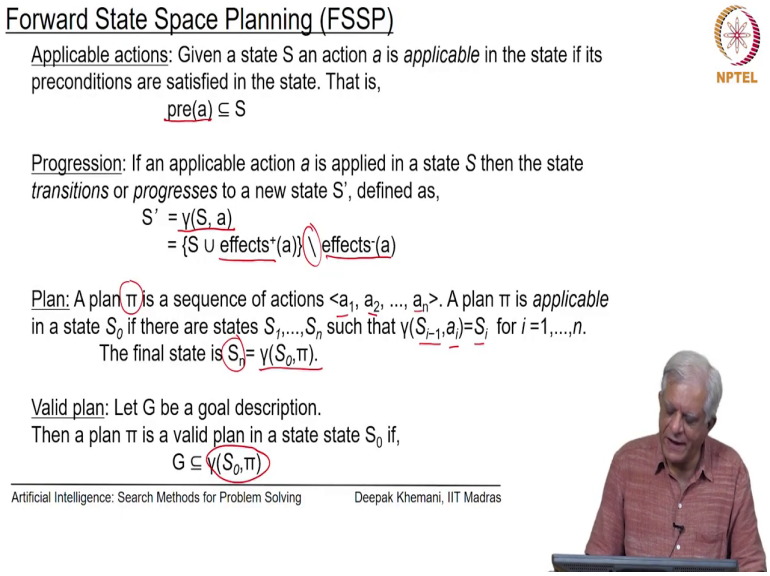


The given set is of course, completely known. So, the given set that is shown here pictorially on the top right is shown in the box as a set of statements in which all the predicates describe the state.

So, on table B says B is on the table then on A, B says that A is on B; I am reading the first line here and clear is shows says that A is clear. So, these 3 statements on table B on A, B,

clear A, are describing this part of the looming. Likewise, other statements describe other part of the state and we have a complete description of the state.

(Refer Slide Time: 00:57)



Now, what does forward state space planning do? That is in some sense analogous to the search that we have been talking about, you are given in your in a start state you want to find which states you can go to and then you want to find the paths to the goal state. We will call this as forward state space planning because you are moving in the forward direction from the start state to some goal state essentially.

The first thing you want to decide is what are the actions that are applicable in the start state essentially. Remember that we said that somebody will give us a move gen function, now we are making this more explicit. We are saying how to construct the move gen function.

And we are saying that the set of actions that are applicable in a given state are those actions such that for each action the preconditions of that action which is described by preconditions of a is a subset of the state S, which means the preconditions whatever the preconditions are for example, on table a and arm empty they must be there in the state. So, the precondition which is a set of predicates is a subset of S which is also a set of predicates. And an action a is applicable if its pre conditions are true in the state.

As we will see in the previous world that we saw there are many actions which are applicable. You can unstack A from B, you can unstack C from D, you can unstack F from G, you can pickup J, you can unstack K from L, you can unstack P from Q. So, you can see that these 6 actions are applicable because there are 6 stacks and you could pickup block from any one of those stacks essentially.

So, the next step is what if what happens if you apply their action. If you apply this action then you are set to progress over that action and you are supposed to transition over that action. And you are into a new state S prime which is defined as saying and remember that we are talk about the state transition system that you are in a state S and you are applying that action a and that gives you a new state and we are calling this new state S prime.

Given this domain description language that we have, the transition function can be defined in terms of the language. It does not have to be something that is given to us from an external user. You are simply saying that if you are in state S and if you are doing an action a then you will be in a stage in which to the state S, you would add the positive effects of a, which is why scripts call it add list and you would remove the negative effects of a from that state.

So, this is a set difference operator which is commonly used. So, we take the set of statements which describe the state S, you add to that the set of statements which are the positive effects of the action and you remove from the state those statements which are negative effects of action. And then you have described what it moves means to say that you have moved from state S to state S prime using that action a or in other words you have progressed over the action a essentially.

Now, a plan, we said at the beginning that we are taking an action centric view of the world and the solution that we are looking at would be called a plan. And very often we use the Greek letter pi to stand for a plan and in this simple worlds where there is this one arm robot manipulating the world a plan is simply a sequence of actions.

Now, remember of course, if they were two arm robots, then you could do actions in parallel. Then you have to do progress alternative presentation, but let us for the moment stick to the simple case, where actions sequence of actions is what is called the plan.

So, you do action first and then you do action a 2 and then a 3 and so on up to action a n. We say that the plan is applicable just like an action is applicable in a state S 0, which is the start state. If there are a sequence of state S 1, S 2, S 3 up to S n such that for each action a i, takes you from the state i minus 1 to the state i.

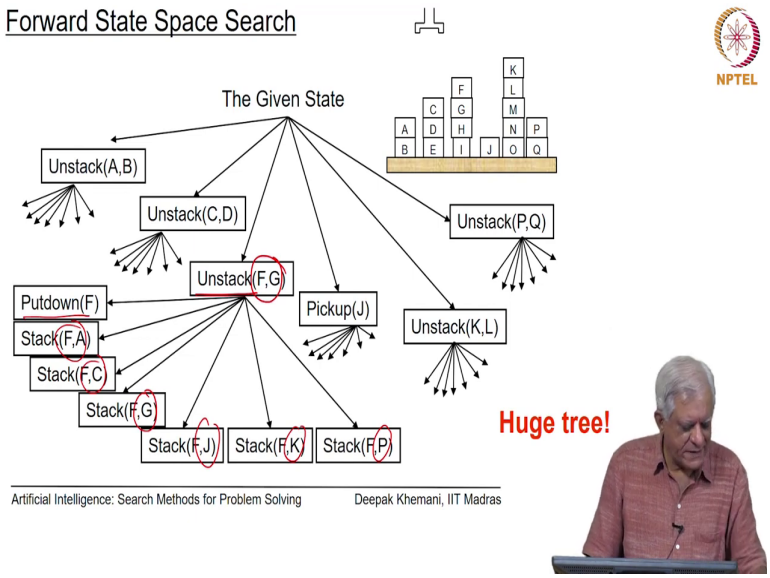
So, essentially it is the state transition from function takes you from S 0 to S 1 then by applying action a 1 then from S 1 to S 2 by applying action a 2 and so on and so forth till the last action a i takes you to state S i and i is varying here for us from 1 to n. So, we have a plan of n actions and we will do n state transitions.

The final state can be succinctly represented by saying that the same state transition function we are kind of overloading it here, takes a plan as a second argument and it results in the final state which is S of n essentially. So, if we have a plan then we will end up in the state S of n. Is that plan a valid plan? So, let us say that we have a goal description given by a set of predicates called G.

We see that the plan pi is valid in a state S 0; that means, that if you were to take be in a state S 0 and you were to apply plan pi and we have already said that the pi is applicable in that state, then you would have found a valid plan if the goal descriptions were true. And what do we mean by goal descriptions were true?.

That the final state S of n as we have said can also be written as this and the goal description must be part of that final state essentially. So, this gives us basically the mechanism for doing the move generation function, traversing the state space, constructing plans as we go along and a goal test function we tells us when to stop essentially.

(Refer Slide Time: 07:21)

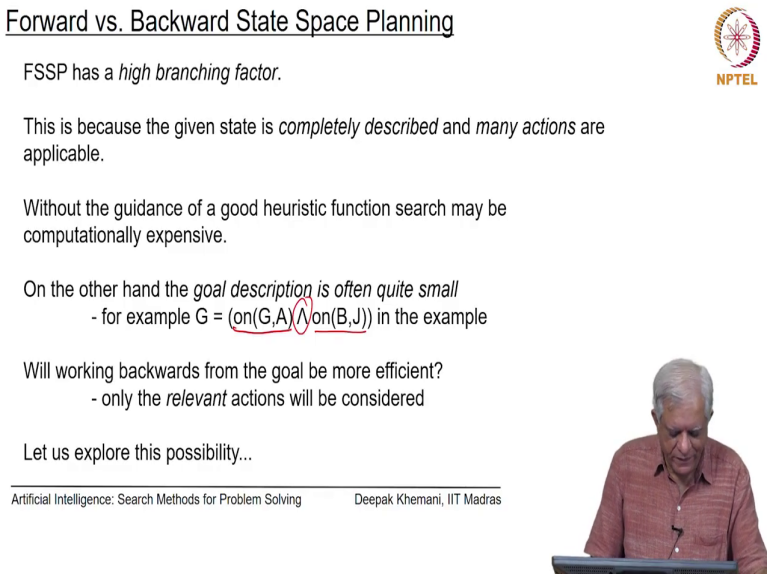


So, this is what forward state space planning would look like. As we had observed in the initial state of the start state which is given to us there are 6 possible actions. You can unstack A from B, you can unstack C from D, you can unstack F from G and so on and so forth. For each of these actions there may be 6 or 7 actions which are possible.

For example, if we have unstack F from G, you can either sorry; let us say this is the action that you have done unstack F from G you can either put it down on the table or you can put it onto A or you can put it onto C or you can put it back onto G or you can put it on the J and K

and so on. So, you can see that the space state space that we have to explore here is a very large state space.

(Refer Slide Time: 08:16)



Now, let us consider this possibility. Does it make sense for us to start searching from the goal description rather than from the start state? So, let us look at this situation here. The forward state space planner has high branching factor and this is because in a given state that is completely described there are many actions which are applicable and each action kind of results in one branch in the space.

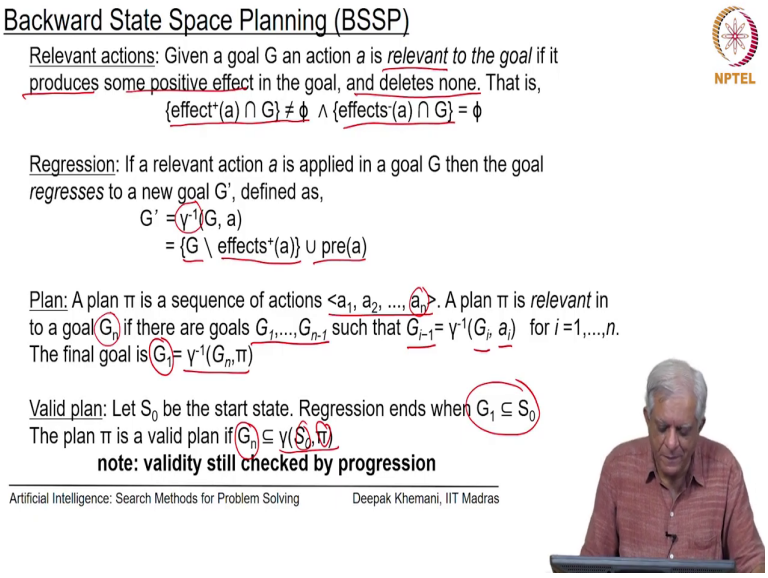
Unless we have a very good heuristic function the search may become very computationally expensive and we have seen that while we are talking about state space search as well. On the other hand as we have repeatedly said, the goal description is often quite small. You simply want to specify what you want to be true in the goal state not the complete state which is a

goal state. You only want to specify some things in the goal state and that description can often be quite small.

For example, in the example that we saw there were only two things we were interested in. We said that G must be on A, which is represented by on G, A and this is a AND sign and B must be on J and that is what we said about the goal. So, if we start searching from the goal state will our planner be more efficient.

That is because the number of things that we may want to do or the number of relevant actions that we want to consider maybe smaller essentially. So, let us explore this possibility quickly and then we will look at how to improve upon that essentially.

(Refer Slide Time: 09:57)



Now, we are looking at backward state space planning. So, let us define first the machinery which will allow us to that. We say that an action is relevants to a goal G, if it produces some positive effect.

It must produce some positive effect which is expressed by saying that the effects of that action intersected with the goal must not be empty and it deletes none. It must not delete anything that is desired in the goal state, which means that effects of the negative effects of the action intersected with the goal must be an empty set. If this is true then we say that the action a is relevant to the goal G.

If an action is relevant and we choose that action then we move from the goal G to a new goal which is often called the sub goal G prime which is defined as the reverse of that state transition function. You are moving in the opposite direction and it can be computed by saying that you take the goal G, remove from that the positive effects of a, because anyway we expect that they would be produced by the action a and that is why we are including that action.

But, add to that the preconditions of a, because we want that action to be applicable in the previous state whatever we regress to. This moment is called regressing over a goal towards sub goal. And similarly, is a sequence of actions a 1 to a n the notion of a plan does not change. Except that in backward state space planning we start constructing the plan from the last action a n and then the previous action and then so on.

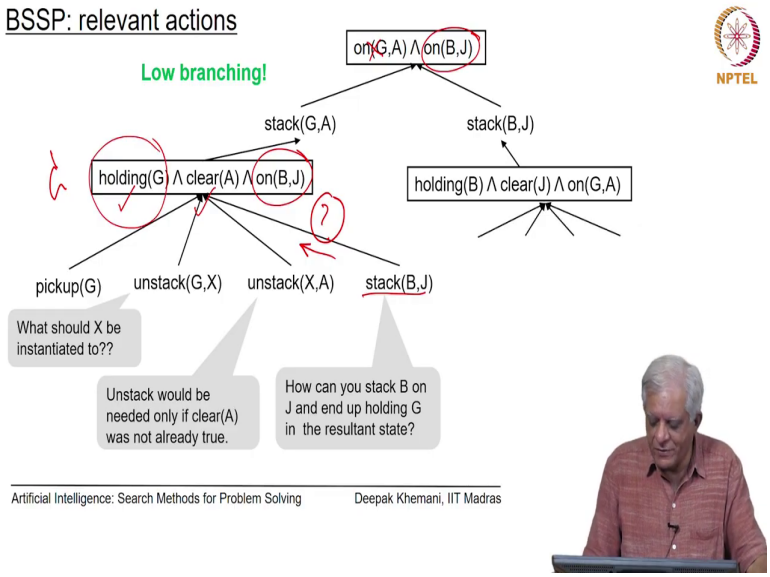
In forward state space planning, we start constructing the planner from the first action essentially. So, there is a difference there. We see that the plan pi is relevant to a goal G n that is a goal that we want to achieve, if there are a sequence of sub goals such that we can regress to them.

So, you start with goal G i, apply action i and you go to goal G i minus 1 and you do this for n, you will end up in the final goal which is G 1 and we can talk about the applicability of the plan to this goal G n. It takes you to the state G 1. When will our algorithm terminate?.

Let S 0 be the start state which is given to us. We will terminate the regression when the goal that we have regressed to is true in the start state that means, the predicates of the goal are present in the start state is 0. However, we have to do an additional check and we will see in a moment why that is the case.

The plan pi is a valid plan only if the goal n that you have is belongs to the state which you get by regressing from the sorry from progressing from the start state using the plan pi and then you reach a goal state. So, validity of a plan is still checked by progression and we will just discuss that quickly before we wind up for the day.

(Refer Slide Time: 13:16)



So, look at this planning problem. We have only two elements in the goal description and there are two relevant actions. You can either stack G onto A, which would achieve the goal on G, A or you can stack B onto J, which would achieve the goal on B, J. There is no other

action that is relevant and the first thing you can observe is that backward state space planning will have low branching which is a advantage over forward state space planning, where all the possible actions that were applicable it had to be considered here.

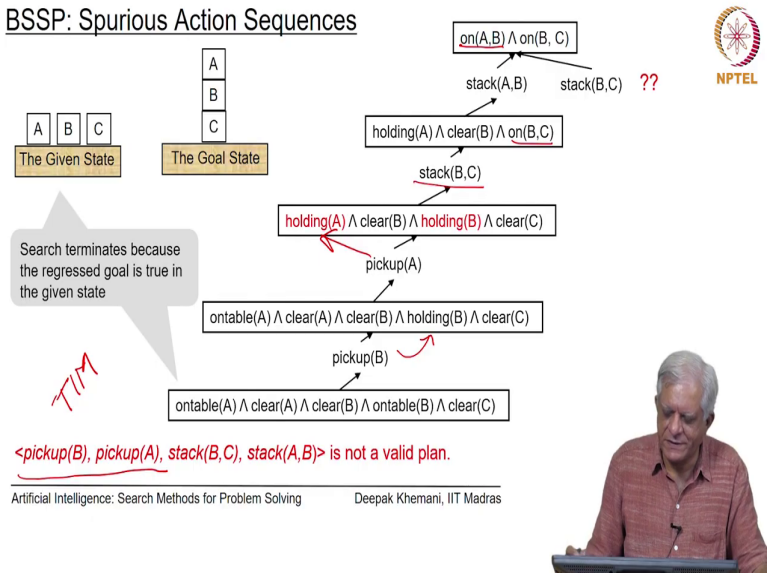
We are only considering relevant actions and since the goal description is small we expect that the relevant actions would be small as well. Then for example, if we choose this action which is stack G, A then we regress to a goal. So, we had this on B, J here which we carry forward to this, but we delete on G, A from this and we add the preconditions of the action stack G, A and the preconditions are that you must be holding G and A must be clear.

If you are holding G and A is clear then you are able to stack G onto A and the resulting action if you want to progress from this to so, the goal set would be the goal set that you have want to achieve. Likewise, in this previous goal which is let us call it G prime or G n minus 1 as the case may be you can do pickup. Pickup is a relevant action because that particular goal description says that you are holding G. So, maybe you want pickup G or maybe you want to unstack G from something else.

But it is not clear what that something else should be and the planning problem will have to address that issue. Or you may want to unstack X from A, why? Because one of the goal conditions is that A is clear and maybe you need to make it clear by unstacking something on top of that. It is not clear whether you need to do this action or not, but the set of relevant actions include all these things essentially.

All you may want to say stack B on to J, but if you are going to stack B onto J then how can you end up in a state, where you are also holding G at the same time? So, you are how can you do stack beyond to J and end up holding G that is not possible. So, this is a problem with backward state space planning in the sense that it can end up with spurious actions. And let us clarify on that little bit with a slightly more complete example.

(Refer Slide Time: 16:07)



And this example is a very simple example. The goal set is similar except that it is a stack of 3 blocks. A is on B and B is on C. The given state is very simple all the 3 blocks are on the table. And what is the plan for transforming the start state or given state to the goal state. Let us see what backward state space planning could do.

You could say ok, I have two goals, let us let me first choose stack A onto B which seems to be a reasonable goal at least as we can see. We have a holistic view of this whole problem. We can imagine that stack A on B would be the last thing that you want to achieve. But, then if you just go by relevant actions you can see that my previous action can be stack B on C why because, I want to achieve on B, C and the definition of relevant actions allow it to do that.

What you cannot see is that in the previous goal there are two things which cannot be true at the same time that you cannot be holding A and you cannot be holding B together essentially.

Nevertheless, the algorithm goes on courageously constructing a plan, it says ok, then I will do pickup a because I want to be doing holding A and then I will do pick a B because I want to be doing holding B. And then it will take me the state in which A is on the table, B is on table, C is on a table, A is clear and B is clear, which is true in the given state, given start state.

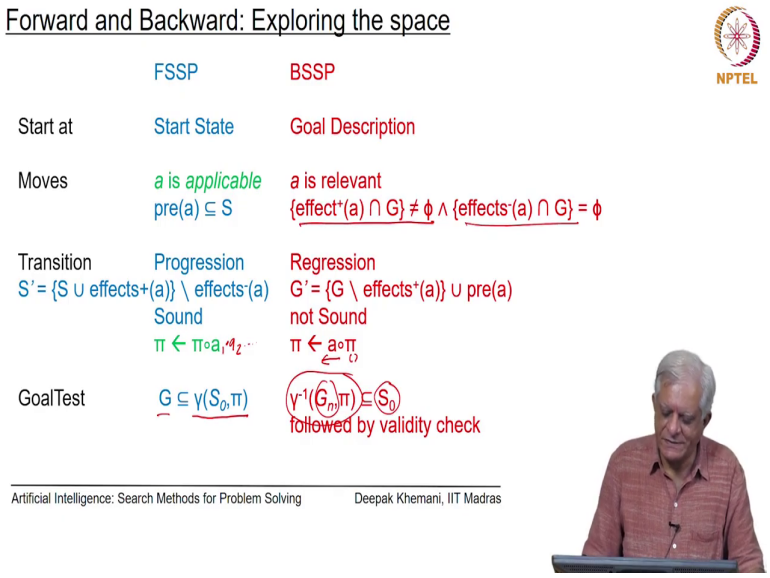
So, our algorithm will terminate but, as you can very well imagine the plan that it constructs which says you first pickup B then you pickup A. Now, clearly that is not possible that action pick up B, pickup A the second action is not applicable in the state in which you have picked up B because you will be holding B. This backwards state space planning cannot solve this problem.

People have attempted to write programs. So, for example, there was a program called TIM with some people in UK had written in which they try to look at more descriptions and read out spurious ones, but it is not such a straightforward problem. And that is why when we talked about backwards state space planning we said that the validity of a plan will be checked only by progressing the plan over the starts from the start space.

So, if you take this plan pickup B, pickup A, then the moment you start to progress over this you can see that the plan cannot be executed essentially because the preconditions for pickup A are no longer true.

So, it is not an applicable action at all. Was if our algorithm it is said that we can have stack B on C is the last action, now you can see that clearly that is not possible because you know A has to be on B somehow and in our simple world we cannot manipulate blocks which are carrying other blocks on top of that.

(Refer Slide Time: 19:05)



So, let us do a quick comparison of these two algorithms; forwards and backward, how do they explore the space. So, we have these two algorithms forward state space planning and backward state space planning. The start the start point is different for the two. Forward algorithms begins at the start states and looks for a plan to reach the goal. Backwards state space planning starts with the goal description and looks for a plan which will be executable in the start state essentially.

If the criteria for considering a move for forward state space planning is that action a should be applicable which is defined by the expression that preconditions of a must be true in the state S or backwards state space planning then action has to be relevant and that is describe that the fact that it must have some positive effect and it must have no negative effect which intersects with the goal essentially.

Then movement we call it progression in the case of forward state space planning and regression in the case of backward state space planning. Progression was defined by the set operation that is we have studied and irrigation also was likewise defined by a set operation that is given here. We saw that progression was sound and by sound we move mean that when you progress from a stat from a state to a new state the new state is indeed a valid state essentially.

Regression is not sound because when you move from a goal to a sub goal G prime we found that the sub goal G prime could not be feasible at all essentially. For example, we saw in the previous example that you cannot be holding two blocks A and B at the same time.

So, that is not a feasible state at all. So, in that sense regression operator is not sound essentially. In progression we start constructing the plan from the first action. So, basically we start with an empty plan pi and then we add the 1st action, then we add the 2nd action and then we keep doing that essentially.

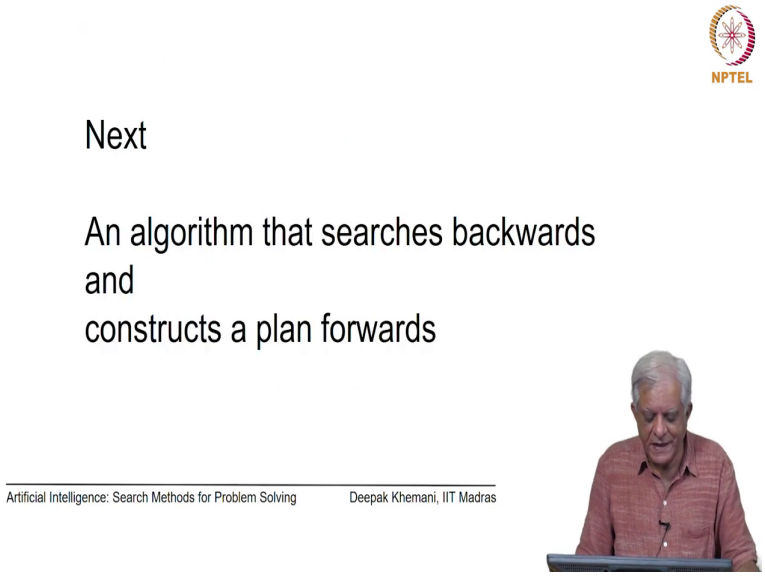
In regression also we start with an empty plan pi here, but we add actions in the reverse direction essentially. The last action first then the second last action and then so on. The goal test function that after you have progress the plan from the start state to applying the entire plan, the goal predicates must be true in the final state. So, there must be a subset of the final state.

In the case of backwards state space planning we said after you have regressed from G n to this new state by applying the plan so many times the regress operator that state must be a subset of S 0, but that was not enough.

We said that we must do a validity check that the plan that has been returned by the algorithm is it a valid plan, is it a feasible plan or not. If not you would have to backtrack and try another plan. Now, clearly in this small problem that we saw where there were 3 blocks on the table A, B and C and you have to stack them on top of other A on B and B on C there is a very simple plan.

You first pickup B stack it onto C then pickup A and stack it onto B essentially. And if we allow our backward state space algorithm to backtrack and try another path it will find that algorithm. But there is a problem here that you may end up generating lot of spurious states and do lot of unnecessary work essentially.

(Refer Slide Time: 22:47)



So, in the next session when we meet we will look at an algorithm that tries to combine the best features of both these algorithms and algorithm which searches backward because that gives you low branching factor, but constructs a plan in the forward direction essentially. It means because it will always be a sound plan. So, we will do that when we meet next plan. So, see you in a next class.