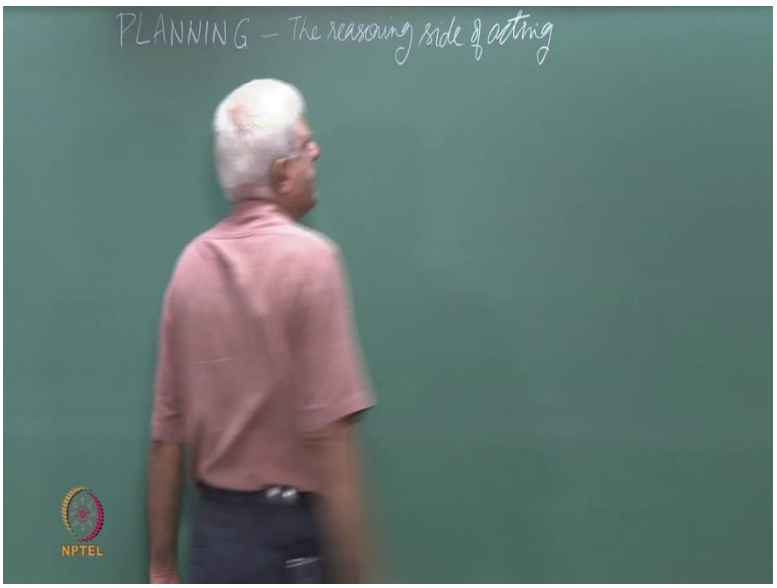
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 33
Planning

(Refer Slide Time: 00:15)



So, today we move on to this topic on planning. So, in the next 6 or 7 lectures, I will cover little bit of planning and little bit constraint satisfaction, which is I think enough for a introductory course on A I. But next semester, Dr Narayan swami and I offering a course on planning and constraint satisfaction, which will go into much more detail in the topics. So, what we are doing here is just a basic, but the more recent advancements will probably do in that course essentially.

So, there is a nice book on planning, which we are not following here by Malik Galab and Dina Noven, call automatic planning. They start the book by this definition of planning and they say it is the reasoning side of acting. So, planning is concerned with actions. Essentially, we are not deal with them explicitly on and off. For example, we looked at heuristic for the blocks world. We talked about moving blocks around, but we did not really explicitly module domains and we do not explicitly module actions. We will do that in the planning path that we are doing now.

Now, just imagine an agent. So, this is world and the agent inside the world, but will draw it here. It perceives the world and deliberates and then acts. So, this is a model of an agent in an environment that we have. The agent perceives the environment or the world, does some deliberation or some reasoning as Galab and all call it. Then, based on that reasoning the agent does acting in the world. Of course, we as human beings do it all the time. We are planning consciously or unconsciously. Whatever, we do these are not random actions that we do. Anyway almost everything we do has a certain goal in mind. We think about how to achieve that goal and our actions are oriented towards that achieving the goal essentially.

So, goals of course, can be long term or short term and things like that, but all the time in the world we are planning. Now of course, in the world of computing, planning is becoming more and more important as systems are becoming more and more autonomous. So, for example, you must have heard about autonomous cars now a days. So, there is stand forth car and if there is a car rally for autonomous cars. So, you have to let loose the car in some sense and it should be able to go to a destination.

Now, if you think about what is involved here? It is not just route finding, which is what we have discussed to some extent, but to drive a car there are many many actions that you have to do. You have to look out for other cars, you may have to slow, you may have to break, you may have to accelerate. All these are actions and somebody has to decide or some agent or some program has to decide, which actions to be done at what time essentially. That decision making or this reasoning side of acting is what planning is all about essentially.

So, we imagine that we have a planning system and that interacts with some people call an executive, which interacts with the world essentially. So, I using the term executive, it was used by NASA in one of their early systems, the remote agent architecture. So, you should look up on the web about this remote agent. It was experiment done by NASA experiment called deep space.

Sometime in the mid 19^{th}, they had this aircraft flowing spacecraft, flowing somewhere in space going to earth some asteroid or some planet or something. In between for 2 or 3 days, they had removed ground station control from that space craft, which means they had let it function autonomously, deciding for itself what it would do in the next 2 or 3 days. During that period they had also introduced deliberately a fault in the space craft and which was used

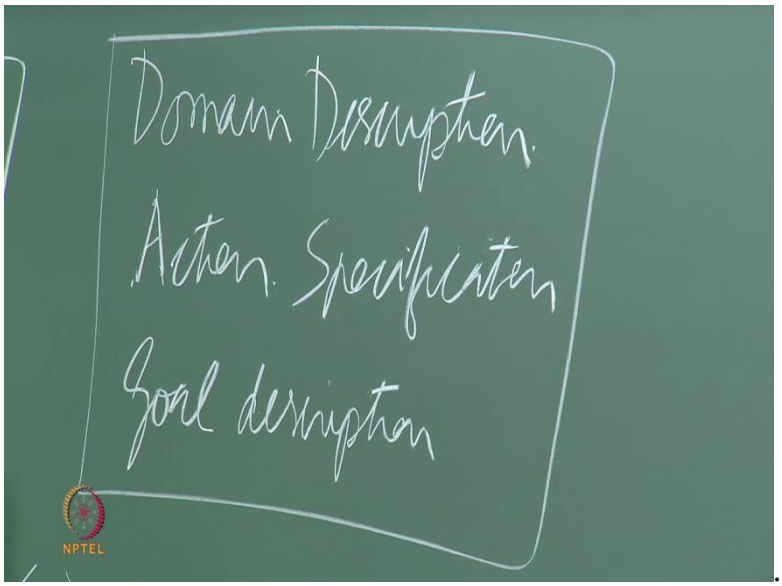
to testing their automatic fault diagnosis system, which of course, we are not doing here, but apart from fault diagnosis planning was a experiment that they did.

Then, you must have heard about mars lowers and the more recent lowers that have been sent to mars. Given, the fact that it takes of the order of 20 minutes, I do not know the exact figure for a signal to travel from earth to mars. You cannot control vehicle on mars as you would be controlling a vehicle, let us say in your lab. If you had a remote or something like that because the signal takes that much more time in such situations, it is inevitable that you have autonomous planning built into the system.

So, both in the remote agent architecture and in the mars lowers, what the scientist do? is that they give high level commands go and explore that area or go and collect samples of this in from this area. Then the rest of the activity is planned by the system on board. So, planning of course, these are some of extreme examples space example, but planning happens all the time essentially. You know scheduling holiday planning, you go to some flight reservation systems, they will tell you that if you want to go from here to basils, what are the flights you should take? So, all this comes under planning some way or the other essentially.

So, let us look at this model of what is happening in the world. Let us see to what degree of complexity, we can capture this essentially. So, in planning systems we are going to do the following. We are going to talk of domain description, action let me use the word specification or action description, describing what actions are available to you and of course goal states.

(Refer Slide Time: 07:44)



We will look at this in little bit while, the early planning systems were kind of domain specific. People would say I want to build a system for planning in this domain or planning in that domain and so on. But following the theme that we have been in this course, which says that a problem solver must be removed from the domain. You want to find general purpose planning systems or else this community calls it domain independent planning essentially, which means that we will devise a language to express all this.

So, this is like domain functions that we talked about in search. The planner would be independent of this. You would write the planner assuming that, you would read a domain description and a problem description or goal description and the planner would automatically compile and run essentially. So, one of the earliest planners that was built was called strips.

(Refer Slide Time: 09:17)

We had possibly talked about it at least in the context of this Robert called Shakey. So, if you remember Shakey was one of the earliest Roberts, that was built in the computing community, which could roam around a corridors of stand Ford University. May be looking for forward points anything like that know where you could share itself. It had a on board camera and it was on 2 wheels. So, if you just look up Shakey, you will see some photographs and I had shown the photographs during my introduction as well. It was fairly autonomous cam of course, it could not do much, it could just roam around the place and the planner that they used was called strips.

Some people say, stand ford research institute planning system essentially and that was the earliest kind of language description, the simplest kind of language description that you can use for planning. We will sought of look at this language to start with this essentially, what would do all this people had started working with devising languages. So, these are called planning domain, description language or in short PDDL and there are versions of PDDL. You start with PDDL 1.0 and we will look at only PDDL 1.0 here in this course and which is basically, what strips does essentially?

So, what happens when you go to higher levels of language? Basically, the richness of the language increases, you are able to describe the domain as well as actions of more complex nature. So, let us see what are those kind simplifications that we are making in strips. So, we had describing strips domains now. So, it is a term that has become common to use for the

simplest kind of planning domains. We call them strips domains because that is what this strips planner use essentially and I will describe them in a short way.

So, these are the simplifications that we simplifying assumptions that we use, one is that the space is finite. So, you can still work with this idea of space search that we started this whole course with where you are in some given state. You want to be in some desired state and the actions that you have to chose only now we will. So, that time we said there is a move generation function, which takes a state and gives you successors state. Now, we are saying that we have access to actions and we will reason with those actions essentially.

So, the simplest assumption is that the space is finite essentially. The second assumption that we make is that, it is fully observable which means just as we discussed when we were talking about games. We said we have complete information games and incomplete information games fully observable planning system is one where, the agent can sense the entire world, which means this perception part of the agent is perfect. The agent can see the entire world and there is no missing information essentially.

Each of the assumptions that we are talking about we can relax and we can richer in some sense planning problems of planning domain essentially. The third is, it is static. By static we mean that agent is the only one making changes in the world essentially. So, there is no other influence in the world. All this is not allowed, there is no there no cloud, there is no sun moving around, there is no rain falling, nothing. Agent is the only one, which is making changes in the world and such a domain is called static.

Obviously, when we talk about multi agent systems advisable systems like game playing that we saw, then this domain this assumption we are violating and the other problem becomes more complex essentially. Then the actions are deterministic. All these are simplifying assumptions that we are making and what do we mean by this? That when an agent does an action, let us say pick up this watch essentially or know pickup this remote. Whatever action the agent does, it happens in the real world and as planned essentially.

So, which means in a deterministic world, if I want to say throw this ball into the basket on a basket ball court, then you put ball into the basket essentially. Of course, the real world is not deterministic when you play basketball, you would realize that it needs a considerable amount of practice to get better at it, but we will make a simplistic assumption. That the

world is deterministic, which means that whatever actions agent does, they will happen in the real world.

Just imagine that what would happen, if the world was not deterministic, then you create a plan that you will get out of this building, you will board your bicycle and go off to the hostel and have some tea. Then when you go down, you find that somebody has stolen your bicycle or may be bicycle is punctured or something like that, then you can no longer do the planning. So, the effect of deterministic action is, once you planned your plan, plan can be executed without any false essentially because the world is fully observable and static and deterministic. If you make a plan, the plan will execute in the world.

So, you want to worry about monitoring the plan and things like that essentially. So, now a days of course, there is a big community looking at stochastic actions or probabilistic actions. So, you must have heard about probabilistic planning. So, they follow a entirely different approach to planning, what we are doing is something like search. They do something which is quite different. So, we have this mark of decision processes and so on.

Doctor Ravidran talks more about such things in his course and I think he is also offering a course on probabilistic reasoning next semester. Then simple goals, I mean that we have the goal test function is on applied on the final state. You say that this is what you want to achieve, you want to be in the hostel and you want to be eating dosa and having some coffee and if that is a situation, then the goal is satisfied. So, that is what we will call it as simple goals. Goals tested only on the final state as opposed to this, there is a community.

So, if you look at higher levels of prejudice for example, they would talk about what we call as trajectory constraints, which means you have conditions on the path that you have are finding for the goal essentially as well. So, the trajectory is the solution and you have conditions on the solutions essentially. So, for example, if you are planning a long trip, you might say that that at all points I must be within 5 kilometer, some reason let us say your friend is not too well and or susceptible something. You say that my route must be such that all points I must be within 5 kilometers of a hostel.

So, that is a trajectory constraint you are may be saying that, I want to go from here to Nagpur, that is a final goal, but I have constraints on the path as well. We will not look at those, we will only simple goals or you might have constraints like a every time you go into a

room and out of the room, you must switch on, switch off the lights or something like that. These are constraints on the path essentially also soft constraints.

Our goal description is rigid that only if those conditions are met. we will say that a goal has been achieved essentially. You might have soft constraints or soft goals, which might say I would like to go to the market and if possible I want to go and see a movie. You know by all these stuffs and do some, you may have a plan, but you may accept the plan, which does not achieve all the goals completely, but achieve as many as possible. Then you would have to save a penalty associated with that.

So, in that if I do not know take this goal of buying a tooth paste, my plan is not as good as a plan, if I had also both the toothpaste. Then it becomes a optimization problem because you have to now find a plan, which satisfies certain optimality criteria. As refers to this satisfaction criteria which says, if these things are true, then I have achieved my goal whereas, here you are willing to relax it and that is why it is called soft constraint that you are willing to allow for some things not to be achieved. You would still accept the plan essentially; obviously, with each such condition being relaxed the problem becomes harder than harder.

Then a plan is equal to sequence of actions. We will assume that our plan is a sequence of actions. So, we do not want more complicated plans, which have networks of actions some things like that. And one more constraint which is very important, it is being relaxed quite often now a days which is the notion of time. So, will assume that we do not have a notion of time to start with. We will just say action a happens, then action b happens, then action c happens. There is only a notion of sequence that actions happen in a sequence whereas, in the real world, some actions may take more, some actions may take less amount of time.

So, you may say I will take a bath and they have a cup of tea and then cycle to the gate. Each of these actions may take a different amount of time. If you want to take that into account, then you want to have what we call is. So, what we have is instantaneous actions, but what we can have is just write versus durative actions. So, by durative actions we mean actions, which have durations and then of course, you can even start talking about doing things in parallel. Once we were talking about time and durations.

So, you can say that while my sambhar is being cooked on the left side gas, I will make something else in the right hand gas. So, that things will happen in parallel, this will take so

much time. Then you might worry about what is the total amount of time that my plan will need to execute, which is call make span in planning terms, but durative actions brings these kind of situation.

So, one interesting problem which was raised by a P. SE. student from one of these universities, I think Washington university is the following. He says that and this was a problem, which could not be solved by the techniques that were there till 2007 and it is a task of repairing a fuse essentially. So, it is a very simple problem. Let us say this is a duration whatever time t you need to repair the fuse. So, the situation is that lights have gone out in the apartment and you are repairing the fuse and it takes you that much time.

So, the action is durative, it means it takes that much amount of time. You have one match stick essentially and the match stick can give you light for so much amount of time. So, this is along the x axis though y x is meaningless. Basically, this is a interval over time so, this takes so much time, this takes so much time and the task is to repair the fuse, but you need light when you inserting the fuse in to the socket. So, essentially this match stick must be leaked in such a way that it overlaps with the end part of this fuse repair action.

Now, the trouble with this is from first yearly we are not into details here. The thing is that when you are dealing with durative actions, you cannot talk about time going in know seconds or something like that, that would become simply unmanageable essentially. So, you do not do that, what you do is I do anything at this time point, when I starting this action. May be I can start another action or no something like that. Essentially can I do something when this action is running. So, time jumps from beginning of action to end of action essentially and then you plan to do something.

Now, the whole problem with this is that you cannot reason about it here, you cannot reason about it here. You have to set of reason about it in such a way that, this comes into here. So, this gave rise gives gave rose to some many interesting problems in how to build systems, which will do. Now of course, we have planners which can do this and they would place an action somewhere in between so that this saddles this essentially. So, durative actions of course, increase the complexity manifold essentially.

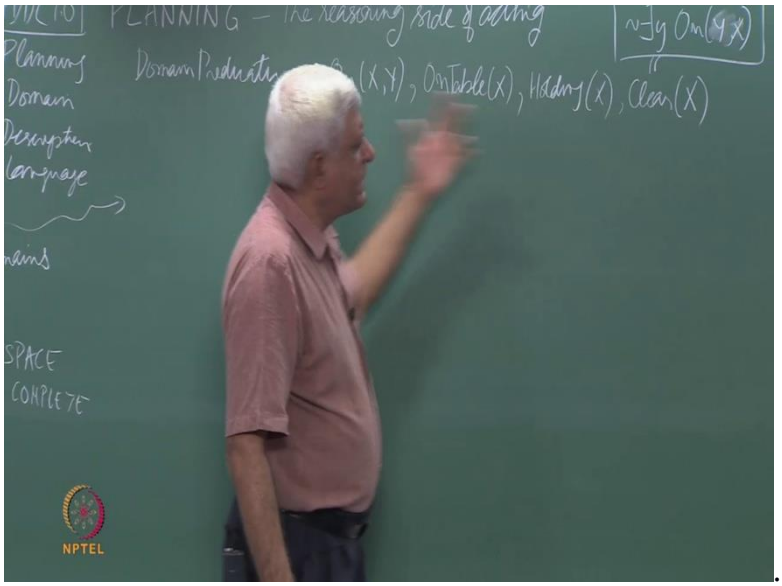
So, one way that some people have tried to model durative actions by seeing to split it into two actions saying, this is a start action and this is a end action, but durative actions will have know conditions like that. Something must be proved throughout or something must be. So,

for example if you are taking a bath then you might have condition that the light must be on throughout that, you are afraid of the dark or something like that.

Now, if you traveling from place a to place b, then at the end of the action you should be at place b. At the beginning of the action you must be at place a and in between you must be at neither place. So, this kind of constrains on actions make complexity of planning more and more difficult essentially. So, I should say here some sense cross. So, we are not using durative actions in strips domain. These are the simplifying assumptions that you make in strips domain. Actions are instantaneous, the goal is fully observable actions are deterministic, the world is static and we have simple goals.

Even in these simple domains it was shown to be p space concrete. So, planning is those one of the hard problems that you are trying to solve and it will always need some exponential amount of time essentially. Every time you relax this constraints, the problem becomes more complex in terms. So, let us look at how strips describes actions. So, we will use this blocks world. You know we already talked about blocks world, moving blocks around. So, we will we will see how blocks world is described in strips essentially and PDDL is basically is kind of a standardization of what we are seeing here essentially. So, first we have domain predicates.

(Refer Slide Time: 27:26)

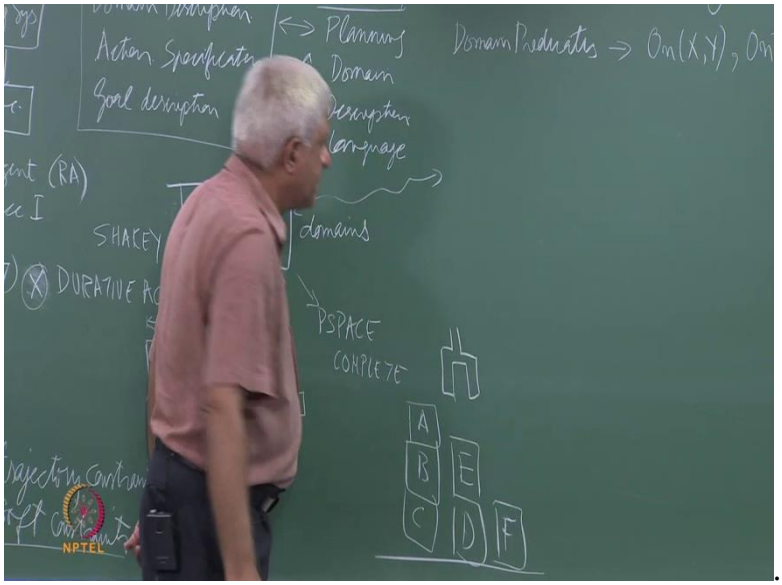


So for example, we might say on x, y where, x and y are variables to signify that block x is on block y, then we can have I use a once used in strips on table x, then holding x. We assume

that there is a one arm Robert, which is moving the blocks around, so that one arm Robert can hold one block and that is described by this predicate saying holding block x. So, this x y would be filled up with variable with constraints in real world, in a real problem essentially. Clear x says that there is nothing on top of x essentially, which is a kind of a short form or saying that another case there exists y.

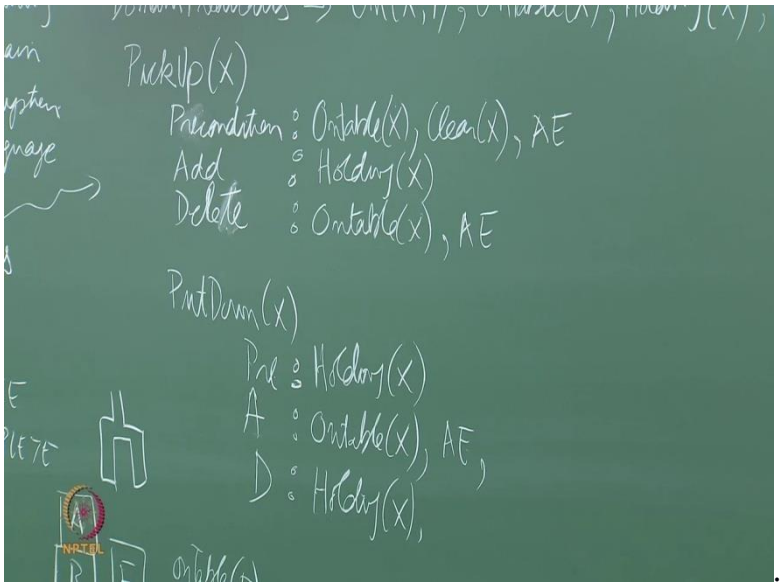
So, we will look at first or logical little bit later in the course case, you are not familiar with the language. So, whenever you define a domain, you have first define what are the predicates which will define the situation essentially. So, instead of saying that we have some state representation, now we are saying that you must be define a predicates which describe the state and the state will be a collection of such sentences.

(Refer Slide Time: 29:35)



So, for example, if I have a situation like this A is on b on c something like this. Then I will describe it by saying that on a b, on b c, on a d, on table c, on table d, on table f, clear a, clear e, clear f and this is my one arm Robot. I need a predicate for that which we will call arm empty. So, either you are holding something or arm is empty essentially. So, we can describe this state using these predicates essentially plane values of the variables. We have the operators or the actions. So, in this simple world we will assume that there are 4 actions, one is pickup.

(Refer Slide Time: 30:33)



So, in PDDL you first describe what are the predicates that you will use to describe the world, then you describe actions. Now, actions are described by 2 things, one is what is necessary to the action to be applicable? What should be clue for the action to be applicable and secondly, what will become clue after the action is applied essentially?

So, in some sense we have the left hand side and the right hand side like in the rule that we talked about earlier, but in this strips kind of a thing, we would say that they are three list, one is a precondition list, one is the add list and one is the delete list. This is the original way in which strips describe the actions. So, precondition list is those predicates which must be true for the action to be applicable. So, if you want pickup list to be applicable, it should be true that it should be on the table and it should be clear, which means nothing must be on top of it. I will just use for this AE and arm must be empty.

So, if these 3 predicates happen to be true in my domain, which means of course, I have to just see the precondition list is a subset of my state description. The state description is the set of predicates which is describing this thing essentially. So for example, this will have on table f and it will have clear f and arm empty. So, because these things are in the state description, this action becomes applicable. I can pickup f for example, that is the way that strips original description is given.

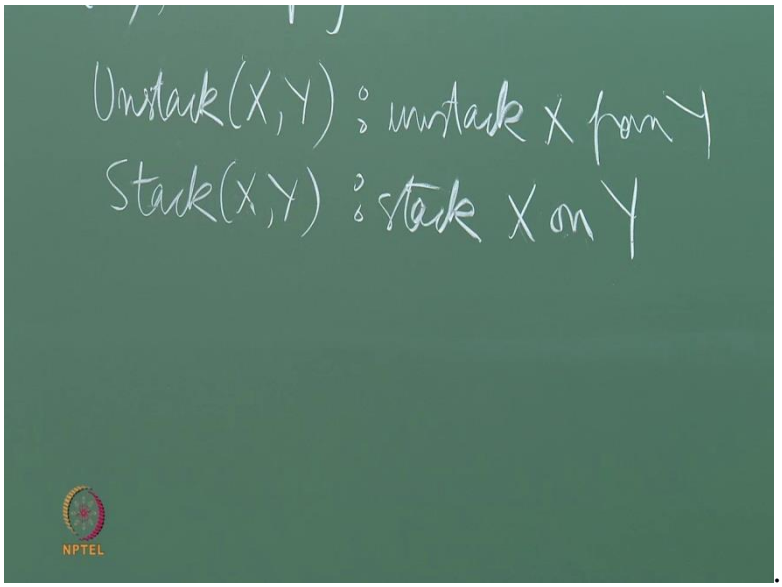
So, if you look at any text book, we use that we will distinguish between the 2 actions in which we pickup something from the table and we pick up something from top of another

block. So, that will have a different precondition just to illustrate what is happening? It has to be owned something either on the table or on another block. So, this pickup is only for picking up things from the table.

So, the add list as they used to call it is a list of effects, which will become true if this action were to be executed. In this case the only thing we are put is that the Robert will be holding this block x and the delete list contains those things, which will become true, but no longer true. That includes things like on table x and arm empty x. So, it also include clear x, but as I will discuss in the moment. You can either like it or you can ignore it, it does not matter. So, corresponding to this, I have an action called put down. So, precondition list says holding x whose add list contains on table x arm empty.

Now, if I want to remove this clear x from here, then I should add it here. If I do not remove it then I should not add it here. You just think about this that either I must add it in both these places or I can live it out of both these places. What is important is that if you are picking up there must be nothing on top of it. So, I need this clear x at that point. It is obvious that once you pick it up and put it down on something else, there will be nothing top of that at that instant. So, clear x will remain true after that essentially.

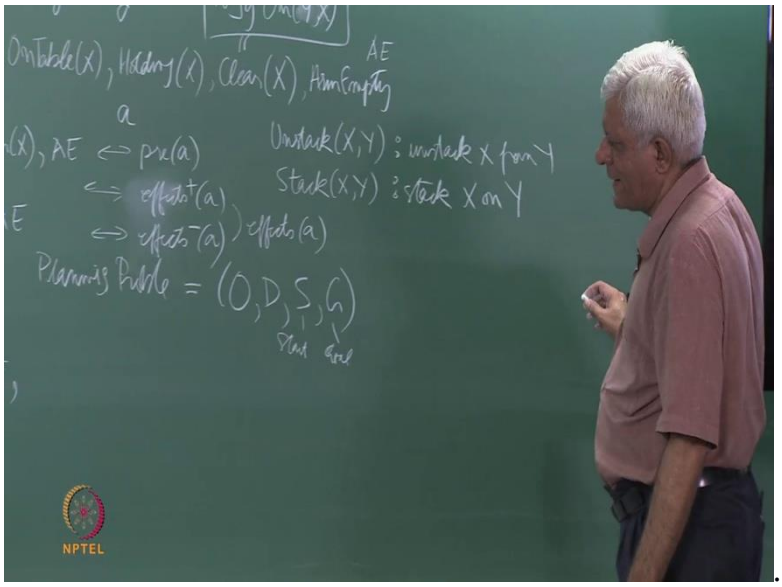
(Refer Slide Time: 35:51)



Corresponding to these there are two more actions which we call as unstuck because I will just name the actions or operators. I will expect you to them to describe it in more detail with this precondition list, add list and delete list. So, the only difference between unstack and

pickup is that, the pickup is from the table and unstack is from some other object. So, you can unstack a from b or you can unstack e from d. The conditions are similar a must be clear, a must be on b and arm must be empty to unstack it and likewise you can stack it. In the more modern PDDL language, I would call this as positive effects.

(Refer Slide Time: 37:06)



So, if my action is a then this list is the precondition list of a, this is the effects positive of a and this is effects negative of a. So, in the modern PDDL language we do not use this name precondition list, add list and delete list. We will say preconditions and effects and effects can be positive or negative. Positive means they become true after the action is applied, negative means that they become false after the action is applied and the two of them together call effects.

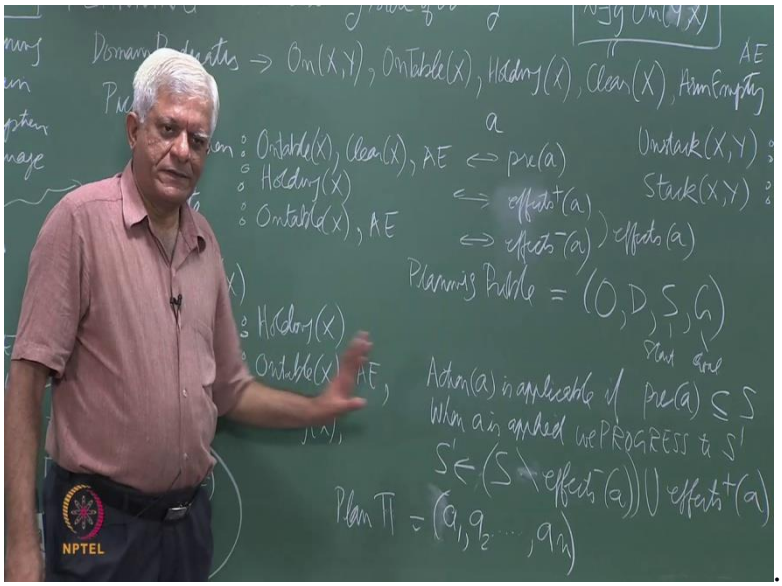
So, we describe a set of action using the precondition list and the effects list of the actions and we can basically now have a language to define any domain in which you want to do planning. What we will need to do is, define a set of predicates which describe the domain which means describe the states of the domain and then define the actions which can be done essentially. So, what is the planning problem? A planning problem is defined by a set of operators or actions a domain description, starts state and a goal description essentially. So, this is the start and this is the goal. So, for example, this could be my start state, I have not written it in this language, but you can write it quite easily. My goal description could simply be something like a, I could say on, this could be my goal description.

(Refer Slide Time: 39:22)

So, observe of course, that I do not need to completely specify the goal as a state. The start state I need to completely specify, you say that it is fully observable. Everything that is true must be given, but a goal does not have to be completely specified. Your goal may simply be that as long as those 2 things are true, then I am happy with my plan essentially. I do not care where the other blocks are as long as e is on c and c is on f, then I am happy. So, e is on c, c is on f, this f could be on something else or it be on the table and I do not care about.

Now of course, this is like what we do in the real world. We do not necessarily describe the goal completely. Let us say this is what I want to be true, I do not care about anything else essentially. So, the goal description is a set of predicates, the start description is also a set of such predicates incenses of predicates. When do we say that action a is applicable?

(Refer Slide Time: 40:54)



If precondition of a is a subset of start, it is just like saying that if my preconditions are true, then well s does not have to be start it can be any state. So, an action a is applicable in a state s, if preconditions are part of the state essentially, which means that they are true essentially. We say that when the action is applied we get a new state. So, we progress new state s prime where, s prime is given by the given state that we are in. I will use this as a minus sign or I can use a standard minus sign whatever the effects minus of a union effects plus a.

So, once you define a state and once you define an action, we know that if an action a is applicable in a state s, then I can apply action a to state s to go to a new state s prime, which is given. I remove everything which other negative effects of the action from the state and add all the positive effects of the action, so I get a new state s prime.

A plan pie is a sequence of actions. So, we have assumed here that our plan is a simple thing like sequence of action. So, plan we will use a term pie is a sequence of actions a 1, a 2 an essentially. So, the first thing you want to do is to write small routine, which will validate a plan that this plan is doing my task. What is the task? The task is that I have been given a start state, I have been given a goal state and somebody tells me that, this is a plan somebody meaning some program that I have attend. The program should be able to validate that, this is a plan how do I do that before we do that, how do I do goal test? How do I say that a given state is a goal state?

So, goal test is simple g should be a subset of state s, g is a set of predicates describing the goal. So, if g is a subset of s, then that s is a goal state. s is a state which describes the whole state. We just some conditions have set on those on the state. So, it is like saying that I should be stick on the canteen and having a dosa, I do not care what everybody else is doing, if that is true then i am in a goal state essentially and that we can test by simply doing a subset essentially.

So, now having a goal test function how do I and somebody gives me a plan pie. How do I know that the plan is a plan for solving a problem. I must able to write a small program to validate that what will that program do it will progress over the state. So, it will apply first action a 1 to given state, then in the resulting action it will apply a 2, which means it check whether a 2 is applicable and apply a 2 and then keep doing that till we apply all the actions. In the resulting state that we have progressed to remember this, we have this progress operator. How can we move from state to state that last state must satisfy that condition, that the goal must be a subset of the state essentially.

So, this all we have done today is to describe the planning domains were to speak. The planning domain is described using a language called a planning domain description language. I have urged to look at some website, which will tell you a what which will give you a actual syntax of PDDL. Essentially it constitutes of a set of predicates which are used to describe the domain or define a domain and a set of operators or actions, which are the actions that you can do in the domain.

The real actions are the instances of these operators, if you want to distinguish between them. Then you can describe a planning problem by saying this is my start state. So, I can say on ab is true, on b c is true, on a d is true, on table f is true, on table d is true, on table c is true, clear a, clear e, clear f and arm empty. If I state all these things, I have said I have described my state. I say my goal state is that c must be on f and e must be on c that is my goal state. So, I can now have goal state.

Now, all I need is an algorithm to find a plan and then of course, I also need to be able to validate a plan. We will see that some algorithms will need this validation check because it can produce plans sequence of actions, which are not necessarily plans. So, we need this check essentially. So, in a next couple of classes we will look at a few algorithms to find this plan. What is the simplest approach you can think? You are talking of a specific problem

which is a blocks world domain. Now, this is just using as a example to illustrate our planning algorithms. Our planning algorithms have to be domain independent, we do not know what are the predicates or what are the actions, but we should be able to still write an algorithm to use that.

So, this solution of course, is a very nice solution which sought of defines the fact that I kept saying it is peace space complete, but in fact, in the blocks world domain, there is a very simple algorithm. It says first put everything on the table, which will take you if the n blocks at most some order n by 2 or something like that or order n. Let us see just put everything on the table and then assemble the thing that you want to assemble.

So, you want c to be on f pickup c put it on f, you want e to be unseen, pickup e and that can be done in linear time essentially, but that is a very specific solution to a very specific problem. We are interested is in domain independent planning, which means we do not know the domain, we do not know the predicates, we do not know the actions. We have a language to describe the domain, the predicates and the actions, and the start state, and the goal state and then the algorithms should be able to operate.

So, the simplest approach as you can get is the power state space search that we have been doing all this along. Go to the start state apply the move gen function. How can you write a move gen function? You have to simply test for all the actions, which are applicable and they will basically constitute a move gen function. They will give you what are the actions you can do, keep doing that and then we also have goal test function. So, the simplest is forward state space search, but we will look at other approaches and we will do that in the next class onwards.

We will stop here.