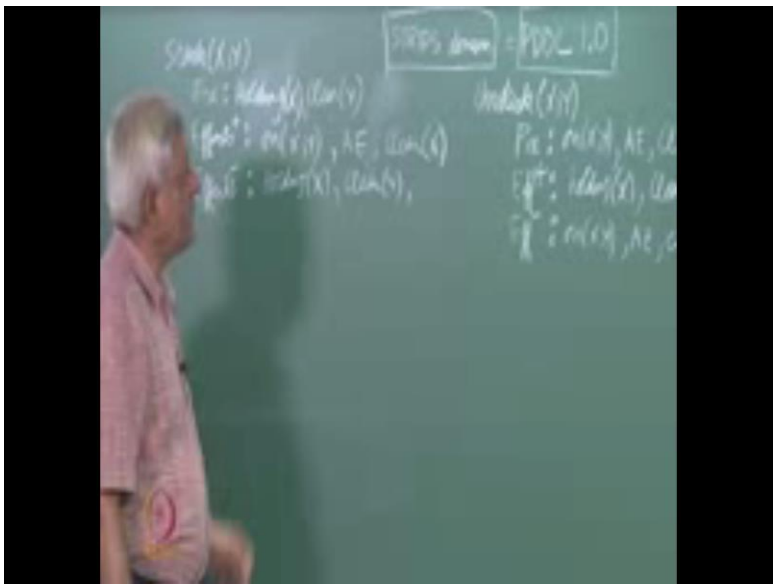
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 34
Planning FSSP BSSP

(Refer Slide Time: 00:14)



So, we are looking at planning and in the last class, we had looked at this language call PDDL very briefly we are talked about this planning domain description language. And we are looking only at the simplest version which we can say version 1.0 which equals to the strips domain. In this simplest version of the language actions are instantaneous the world is deterministic, the world is completely observable, the goals are simple in the sense that only goal state is look that a nothing else. And the agent is only 1 whose changing the world, but we took the first steps towards rejoining. A when we if starts talking about the language like this planning domain description language. We are talking about representation how to represent domains how to represent actions? So typically if you will look at the periodical document you would say they who start of the defining domain.

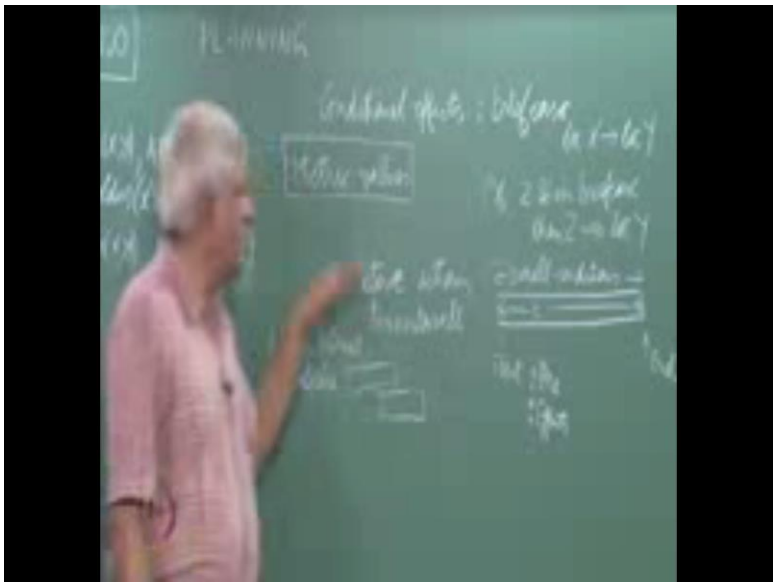
So, the domain that we are looking at is the blocks world domain so you would say domain name is blocks world the predicates or on table holding. And all those predicates and some of the actions we saw. So, let me the couple of the actions stack x on y and the

pre conditions are thus set of predicates which must hold further action to be applicable. And the p conditions first stack was on x y then clear y and arm empty. So a is stands for arm empty will use the short form. So, if x is on y so of x and y have variables here in the operate us, but when we actually talk about domain they will get bound to constance which stand for specific blocks like a and b and c and so on. So, if x is on y and y is sorry somebody should a point of this off. It should be holding x if a holding x and y clear then you can stack x on y. And the effects positive are on x y and arm empty and the effects negative are holding x and clear y. There were one more positive effect which is you can add as clear x. This is not precisely that the syntax with periodical uses, periodical uses less like's intakes so everything is in brackets.

And you say if x then positive effects then negative effects are actually a continuation of positive effects with a not prefix, but will use this simpler notation here essentially. So, basically operators or actions are represented by these 3 kinds of things 1 is set of p conditions holding x clear y for example, and then the effects so what the effects? Effects are what are the changes that this operator makes in the world if that operators in the world executed. So, what we a saying here is that stack x y is possible when you are holding x and y is clear. And if you executes stack x y then x will be on y the arm will be empty. Because you would have draft x some sense and x would be clear and this will no long be true that you a holding x and clear y essentially. Similarly, we have define unstack x y which is a inverse of this so the pre conditions are on x y arm empty and clear x.

So, if you x is on y and arm is empty which means robot arm is not holding anything remember we are talking about 1 arm robot which is moving blocks around. And clear x which means nothing is on top of x essentially then the effects positive are holding. So, you can see here there is a kind of a association will that clear y and the effects negative all these things on x y arm empty and clear x. So, in this version I have add clear x go to effects here and there so if you add it here universe as it in this also yes in that earlier. So, as a say this is a simplest version of period l or period l 1.0 on which actions in sent an yes and the other effects let we talked about. So, just before you move on to the planning of it us let me illustrate a couple of riches language construct essentially.

(Refer Slide Time: 07:00)



So, one thing which a richer language you to apply is called conditional effects. So, conditional effects are effects which are which come in to force only if certain conditions are true essentially. So, as an example of conditional effects let say that you have a domain in which you are talking about moving objects around and so on and so forth and you have trucks and you have this kind of thing. And you want to move let say 1 table from location a to location b. And you define move truck action which says the truck goes from location let us say x to location y. And the conditional affects of that a that if there is any object in that truck then that object also moves from location next to essentially. Or if you look of the something like a briefcase domain, so if a briefcase goes from location x to location y.

So, let us say you have bag that you carry from department to hostel then if there is a book a conditional effect will say if z is in briefcase then z goes to location y. So, and was writing it in sort of semi English, but the basic idea is that the act of moving a briefcase from location x to y has conditional effect within the sense that whatever inside the briefcase we will also get move to the target location essentially. Of course, in this there is no sustain the effects are determines stack an every action very clearly well defined effects essentially, but that said is a language. Another feature which a richer language is have a durative actions which are been explore quit a lot now a days. So, as we said in the last class strips actions of periodical one action have no concept of time in world at all and that is, because actions a in sententious.

So, if you a taking up a object or putting down a object was stacking in object and stacking in object we just assume that it happens in sent essentially. And then once a action happened it is the effects a seen immediately essentially. But in the real world of course, actions are not in sententious. If you go from here to the hostel it will take a certain amount of time from go from here to the hostel so that must be a taken in to account. So, durative actions are represented by durations or intervals so this is a start of the action and this are the end of the action.

So, for example, we consider this action of going from here to the hostel then the actions start at a certain time t and it has a certain duration so there is certain amount of time t that is this action takes. Now, this time t could be constant or it could be a variable for example, move action in the real world go from the here to the hostel it will take you 10 minutes. You go from here to the department it will take you 5 minutes you will go from here to the gate it may take you 15 minutes. So duration could be dynamic in that sense essentially and so this is a start of the action this is the end of the action. So, the action is define over a duration and such search planning domains a called durative such actions a called durative actions. And what you have is that you have start conditions or lets use a term p condition at of you is using and then starts effects.

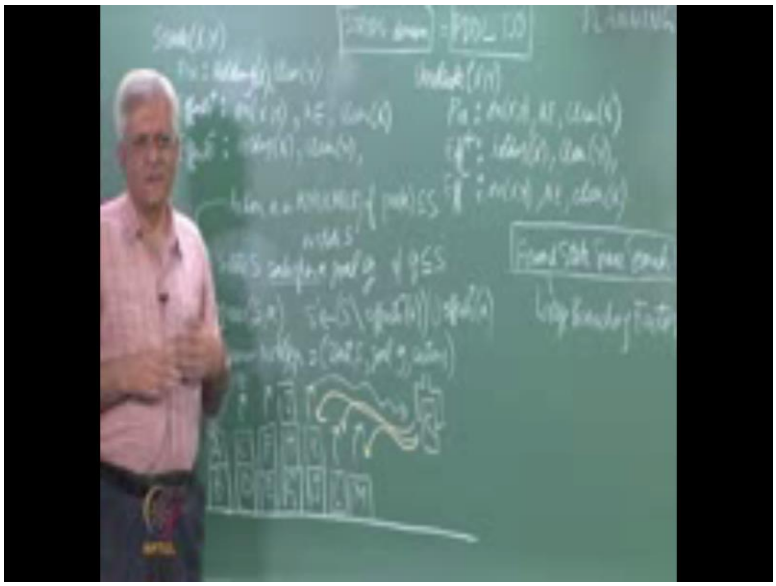
And you have end conditions and you have end effects so what will meaning by this? That let say let you want to go from here to a lab on the department and we have defined an action for that. So, the start 3 condition as you should be here and may be you have a bicycle or something what is, whatever is other 3 condition that we have here, what are the start effects? Start effects are the effect which comes into play as soon as actions begin. So, the moment is start going towards the lab your are no long a here essentially. So, that is a starts negative effect essentially what are the end effects? End 3 end may have 3 conditions so 3 conditional could be the for example, this a lab is opened that is the lab is open at this point of time then the action will execute correctly. And the end effects could be you could be inside the lab or something like that. And then we have overall conditions which have predicate which was being true throughout the action.

So, supposing you are cycling from here to some place then the overall condition could be said that all points your cycle has enough air. It is in so if the condition fails any time then the action wills no longer work. Now, as you can imagine once you start working about durative actions then things become much more complicated. You have to talk

about start effects start conditions end effects end conditions overall effects, overall conditions and so and so for. So listening becomes much more complicated and you can imagine that 1 action may be like this another action may happen like this. And then there are different kinds of relations between these intervals that you have to explore. And there is a well define study about it Aliens interval algebra which basically looks at all the relation between which 2 time intervals can have or 2 interval can have their does not want to be time. So, this is for example, saying that a second interval begins before the first interval. Hence and you can imagine that a there are many different possibilities.

They can happen simultaneously 1 can second one can start immediately after the first one or the second one can be contain totally in the first one or second one can totally contain the first one. So, in there are 13 such intervals and then there is an talks about these thing with a intervals we will not get into to this here. The third that is often introduced is found is like metric values. So, these are numerical values which can take different values and to take an ex to illustrate an example here. It could be that inside of bicycle let us say you using a motor cycle. Then one of the things that you will need one of the condition that you will need is that at all points you must have certain amount petrol essentially. So, this metric values would keep track of amount of the petrol that you have in a your motor cycle. And it could say something like if you have ex amount of petrol at the start then at the end of action you will have x minus some k amount of petrol. So, that is a end effect that is they were abreaction and that introduces a third dimension which is that of continues numerical values essentially. But we will in this course not look at these richer domains and focus more on the basic planning which is done using the strips periodical can of a domain essentially.

(Refer Slide Time: 16:08)



So, we had define something's we at said that an action A is applicable if thus anyone remember the pre conditions of A so will use this as a predicate. This stands of a conditions of A so this basically stands this set you is a subset of S. So we should say applicable in state s. So, with this planning as you will notice we have set of making a gradual transaction into representation. So, for we did things like search, but you do not bother about representation. Now, we a talking about how to represent a domain, how to represent actions and also we have keeping the reasoning part in mind. And the reasoning part is at least to start with will be the same kind of reasoning which is searching for solutions essentially. So, because we have this explicitly representations we can talk about this kind of rule conditions.

So, an action A is applicable in state s if the p conditions of a all the subset of s essentially. Likewise we say a goal so when we say goal g we essentially mean a set of predicates that we want to be true. So, for example, today we look at an example may goal is the following 1 A B and on B B. Let a say hich stands for the fact that A is on B and B is on B in the blocks for domain. So this thing is the goal essentially and this comma is to be in it is and which means at both the things must be true A must be on B and B must be on D essentially. So, goal is basically a partial description of a state that you want to be in of the desire state essentially. So, we say that the goal g such so let say a state S satisfies a goal g remember both of these are sets of predicates. A state is for

example, what is true in the world on A on B on C whatever likewise goal is what you want to be true in the world essentially.

So, use a word goal a satisfies a state satisfies a goal g. If in the representation that we have use g is the subset of the S. So, you can see that this can be use to define the goal test function that have be reach the goal or not. We are working its simple goals here and we want to see whether we have reach the state which is the goal or not. Then this can be used to test for that essentially whether action a is applicable we can define a function call progress let say in a state S using an action A and what this progress does is an a takes a state s and applies they action A. And gives you a new state explain were explain as is on the last class this S minus the negative effects of A union the positive effects of A. So, we have a progress function which allows it a move from one states to next state essentially. So, we have basically the machinery for doing state space search here we can test whether than action is applicable in a state on a not by using this applicability test.

We can progress from the state to the next state so in the in effect we have done the move gen function. By if we just select all the actions which applicable in the state we have the move gen function and then we have the goal test function here which says that you can test for the S on goal is not essentially. So, forward state space search essentially does the kind of the search set we have in talking about. So, let me give you a so a planning problem is define by a start is as before a goal so start s let us call it goal g and a set of actions essentially. So, obviously we have define the domain using a language an having define the domain in the define in we have define the set of actions of the possible domain. And a specific problem is when we have define the starts at s a goal say g and the set of actions that a available to you defines which can be use to go from start to goal; essentially. So, let us that is assume that we have some simple domain.

So, we want that A B and D so let a say a given starts it is the following A is on B lets say this is on the table. And let a say C is on D and then E is on the table Ff is on the table. Some arbitrary starts seat I have given so let us say this is a start and that is a goal say that you want to achieve which is that A should be on B and B should be on D essentially. So, again that we repeat that is a goal is a description and the description may be true in many states as you can imagine. If you just have this a on b on d then the rest of blocks will be available in any way and as long as those 2 predicates on A B and on B B or there in the state. That state is the goal says that is what you have said here that a

state S satisfies the goal g if g is the subset of S. Now, we want to do this planning and we start with forward state space search. Because that is ((refer time 23:41)) familiar with.

So, how it forward state space search? To gets is exactly like space search that we have to been talk to about so far, that given a start state S. You can apply many actions and so inside of writing the actions let me that the action was saying by using this arrows. So, this arrow says that I am unstack A from B that is one action I can apply I can unstack C from D, I can unstack F from E, I can unstack I from H is I can pick up. If you remember there a 4 actions we talked about stack unstack and pick up and put down. The only different between pick up and unstack is it unstack happens from another block and pick up happens from the table. So, just we will a says how to use different action and this one? So, the move gen function in this particular state would generate these actions 1 2 3 4 5 6 7 actions. And your first algorithm will have to take one of them essentially. Let say it picks this action for some reason now, what is the situation? A situation is that this rubato is holding this block K that say it pick this action.

It is the now, holding this block K everything else it is the change so how do we define this new state? Define it by doing this progress action which says that if I am doing this unstack K J action I must apply this I must remove the negative effects of A. The negative effects of unstack that ex so on by in this case x is K on y is J. So, K J is no longer true, on K J is no longer true; arm empty is no longer true, because you are holding K. And clear x which is K is no longer true and then I must add the positive effects of this action. The positive effects of the action he has it has A holding it and wise clear. So, this effect would be present at fact you have holding K and. Because you have lifted it K away from this j will become clear so clear j will be added to the state. So, that is the progress action that you have to implement to get this new state K. Now, you can see that having done this action these actions are still applicable sorry actually they are not. But they would become applicable the moment arm becomes empty this moment is arm is not empty arm is holding K.

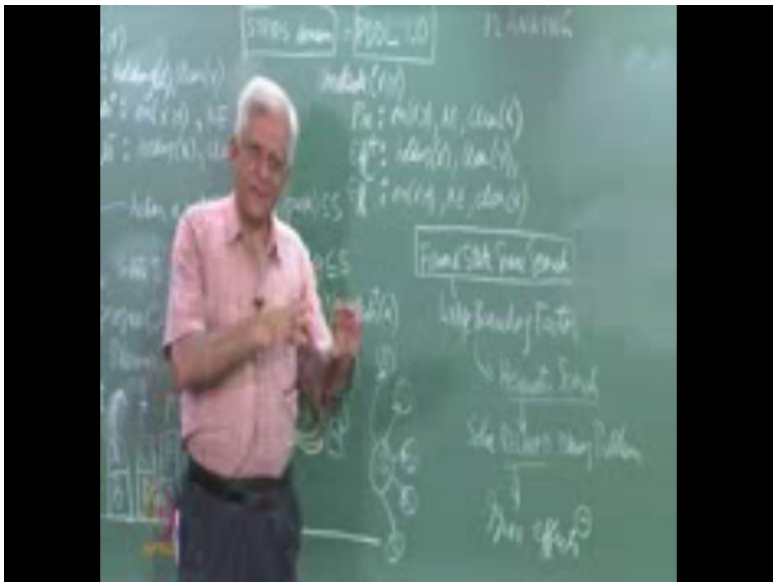
So, you do only action you can do with this one arm robo unfortunately super case some were either put it on the table or put it on the one of the block. From this we can illustrate by saying that from here the actions are either put it on this are either put it either put it on this and so on and so. So, I will have a clause a state space here it 2 difficult to draw,

but you basic you gt the basic idea that you move from state to state. And whichever state you looking at you affair the move gen function which is looking for all the actions which are applicable pick one of them and move forward essentially. So, this representation mechanism allows a rule of overall search you have doing which was we listen even name to it. We are not is that given a name to this we have only set forward stack space search, because we have not talked about what does strategy is now you could do that first search or you could do or you could even do did or some something like that. You could do anything that we were doing here essentially so what is the difficulty with this? What is the drawback of forward stack space search?

If you look at this given start state and that given goal state can you observe what this is doing wrong? And when you make this observation you must keeping mind that you are not solving the only the blocks whole problem. We are writing this as a generic algorithm of a our stack space search in which we can plug in the domain with all it set of action on a operators and all the predicate it is not goal riven essentially. And in fact, you look at the I properly drawn this like this, because the 3 blocks A B and D that you want in the goal to be true or here in the left hand side. And this right hand side is actually will does not care not of course the real world is like that if you are let us say planning for yourself what to do next? Then if you what simply try to describe what is true at this moment in IIT.

You would have 100 in 1000 of statements in something who is there in the office, who is there in the lab, who is there in the canteen what is happening in this class is going on their this class is going on 100 in 1000 of statements. You would have and then you 100 of 1000 actions would be possible essentially when we look at this particular problem we can see that is no point in picking up A or I or K or L or M. Because they are all go on to help us achieve the goal where as something should be done on this part essentially. So, the problem is forward stack space search and this arises from the fact that state description can be large, the number of applications can be large is that there is a large branching factor. The number of actions which are applicable in any given state is thus too many to we consider by a grout force approach and that is true in any real world situations value essentially.

(Refer Slide Time: 30:22)



How do we address this problem? One approach is heuristic search and that is something you will again not explore in this course. Because we do not have enough time and you have most welcome to come for the planning course you entrance it. And when we talk of heuristic search here we are not talking in a domain independent fashion. Earlier we said for this 8 puzzle this is a heuristic you can use of a blocks world. This is a heuristic you can use you have number of blocks which are in the correct place. And that kind of staff, but we are we do not want to define you the 6 domain dependent fashion. We are interested in talking about domain independent fashions. And the idea of domain independent heuristic is to solve simpler problems essentially. And the approach that we use in planning is to solve the, what we called is the relaxed?

So one approach is to solve the relax planning problem and what why mean by that? I mean that if I am in a start stage S and I have all these possible moves from these stages that is call them S 1 S 2 S 3 S 4 S 5. Then I solve a relax planning problem going from this S 1 to goa1l and S 2 to goal and S 3 to goal and S 4 to goal and S 5 to goal. Look at the solution produce by this relaxed or as the solution is call the relax plan. Look at the relax plan and see which is the cheapest relax plan may be in terms of number of moves. So, if S 2 is given in the cheap cheapest relaxed plan then I will say I am going to make this move from S 1 to S 2 exactly like in the heuristic search that we said earlier. We said that we will apply the heuristic function to all these things and choose the 1 with sees to be closes to the goal. Except that here we are literally applying us problem solving

strategy to solve from each of this positions to go to the goal. And the seeing which one seen which is which one seems to be close to the goal.

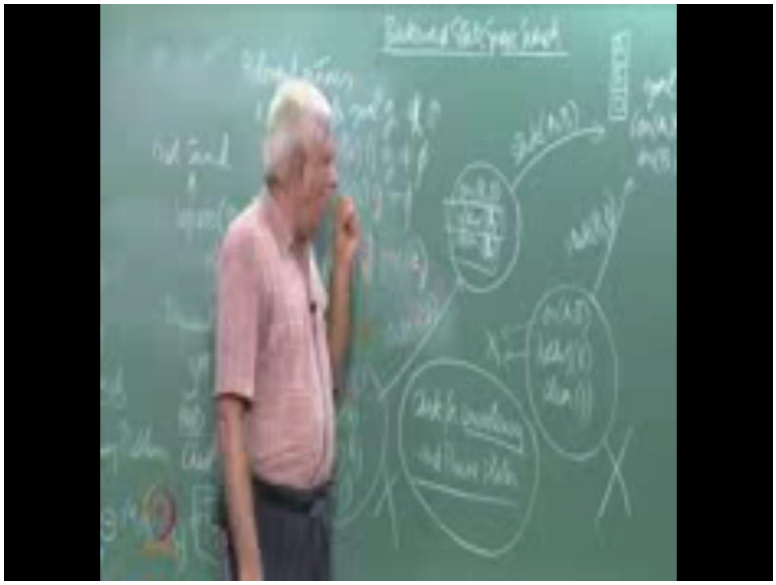
Now, the first question arises to that you would ask is that computing the heuristic functions should not be as expensive as solving the original grout force problem. Because that is the case and what is what is advantages of grout basic function is giving you in the heuristic function that here seen so for the domain dependent heuristic functions. We had said that it is a static function you only look at a state and do a computation of a value. So for example, you do it leaden distance or as we set the number of blocks and so on so they are they were cheap. In this case, we same the relax problem in a such a way that it is a easy problem to solve which means computationally easy to solve and when we say computationally easy time. So if each of the, these states can be solve can be foes as a relax planning problem and it can be solve in palynology time. Then we will just have now, K times that time polynomial time will be able to decide on which 1 to use here.

So, remember we at said that the even in this simple domain planning is hard in fact its p's space complete which means it requires polynomial space and time. And here we us trying to say that we will solve a simpler problem, but try to estimate which of them is closer to the goal. And notice that this is being done independent fashion how to be do this? The relax planning problem is basically says ignore the negative effects of actions and those are called relax actions. So, I just ignore this part essentially and then you can see that if you that visualize this become monotonic situation. You keep adding new predicates to the start set you are not deleting any predicates the complexity of planning comes. Because you are deleting things that you pick up K from J K is no longer on J essentially. But if I am living on K J relax plan then I monotonically increasing the set of predicates. And at some point I will get those to predicates in said my said. And then I will say have solve all the relax planning problem.

So, it terms out you should take my word for it that solving relax planning problems when the effects negative effects are ignore is polynomial time essentially. And heuristic search essentially uses that approach. There are 1 or 2 other ways to compute heuristic, but we will not going that what you will do instead is to focus on doing what we call is a back word search. We are already works some exposure to backward reasoning we said that ((refer time 35:53)) for example, (refer time 35:36)) and goal trees that you work

from goals back words. But they the representation was tailor to backward reasoning that we said that this is how you decompose a given goal into sub goals. And then that is a goals into more sub goals and eventually you have trivially sub goals to solve. Here we are not changing the representation, but we want to back ward reasoning. So, we need a few things know again we have to so this machinery will no longer we have applicable.

(Refer Slide Time: 36:30)



We talk about something called relevant actions. So we say an action A is relevant for a goal g this is an some sense applicable actions. In forward stack space search actions are applicable in a given state in backwards search back ward we use a terms stack space search here again. Actions are applicable to goal descriptions and we say that in action A is relevant for goal g if the following is true let me write. If you the effects plus of A and g is not equal to 5 that an action has a effect which is required in the goal to be true which mean so what you have simply saying is that the inter section of the positive effects of the action and goal must not be empty. It must produce something which is relevant for the goal then we say action A is the relevant. But is another thing you want which is that effects minus of A 1 must be equal to 5. There it not produces any negative effects which you want in the goal to be true. Remember what does the goal is the set of predicate set a want to be cube.

And you are looking for actions which will achieve the goal essentially. So, we say that an action A is relevant to A goal if the positive effects of the a has something for the goal

which means intersection was goal is not empty and the negative effects of the a is empty essentially. So, we now, 2 predicates in a goal and you can see that we 2 actions are relevant essentially. And you tell me what were the 2 actions? So I have moving from goal to sub goal now, I am saying that if I want this to be achieved and if this is my last action then I will achieve it. But only I am use I am only using this definition of relevancy essentially I am not any deep reasoning here and just applying this test to see what actions are relevant. Anyone wants to venture against hazard against.

Student: Stack A, B

Stack A on B and one more is stack B on D. Now, you can see the principle advantage that backward stack space search gives us forward stack space search is in the branching factor. Because backward stack space search is focus on the goal. It is only looking at actions which will achieve the goal. And in this example you can see the branching factor is small we have only 2 actions at we can consider and here they were many actions at we were considered. That is a main advantage of doing backward reasoning ((refer time; 39:52)) reasoning then we have to talk about a regression. So, regress goal g with action A it means how do you refine the sub goal? So he get g plain h g minus effects of A, because action A is producing those effects. We do not then to be clue in the in the previous state or the previous goals description union any one p conditions of a because that actions must be true in must be applicable in the previous goal state essentially.

So this is called regression and if you want to now, over this you can see that I must remove the effects of stack A B which is on A B, but I would be left with on B D. And now, I need the 3 conditions of those stack action which is holding B. And clear D in the similar fashion for this cycle regress to on A B is here holding B clear D. You can go in the stack B on D I am must B holding and D must be clear. So that is why I added it to the so I have moving backward from the goal to sub goals in some sense and this is the process I will search backward. The main advantage that I am looking for is lower branching factor, because I am only looking at actions which needed from a goal to be achieved. But there is a problem here every one there is a problem like what is the problem here? So, look at these second action so if you visualize what you are doing? You have this A on B, B on D and you are saying that may last action would be to stack

B on D. To if you try to imagine that you will see that it practically impossible, because he cannot be holding B when A is on B.

So, if you look at these 2 then they are inconsistent cannot be true at the same time. Moreover if I want to regress from here I could say stack I could take any goal, I could pick an action to achieve this goal which is to stack B on D. I could pick an action to achieve this goal with which will be unstack b from somewhere or pick up b from the table. And I could cube the action to achieve this goal and that action would be unstacking something from D of course, unless D is already true. But I can choose the first one for example, I can say stack b on d and when I regress so this I get those holding B clear D and then the p conditions of this which are holding A clear it should be clear B. So, I will have holding A and clear B and this holding B and clear D come, because of the stack B D action. Again you can see that this is an impossible state which call a this is not possibly a state whereas this is a state it you can agree.

Because B is on D an you a holding A and you about to stack A on to B that is perfectly feasible. In fact it will be part of the plan that you will be a looking at. This is not feasible this says you a holding B and you holding A at the same time and then you have these 1 arm which is trying to do this essentially. So, backward stack space search has this advantage that we branching we get a low branching factor. Because it is goal directed, but it has a disadvantage that what we get are not states essentially may not be states. Now, this progress action is sound and by sound I mean it is close over the other state place. In some sense is that when you apply a progress action to a state what you get is a state?

So, in that is sense its close a other state place the regress action is not sound when you apply a regress action to a goal. You get a goal description which may not necessarily be state essentially which may not be possibly part of a state. So, for example, this is cannot be part of the state how can we having on A B an also holding B essentially that is not allowed in the rules of the game you have defined you have a one arm robo which should be holding B. And then so because is holding b nothing on B on top of B so this not possible. So, how do we get around this problem? So this is not sound so there 2 things you can do basically is when the will stop this backward stack space search, when we what you mean by that what is the test that will you use.

Student: 3 condition of the actions applicable

You terminate this when that g whatever prime that we have is part of the start state. Remember that you have regressing from 1 goal to another sub goal to another sub goal to another sub. If at some points you have set of predicate which are true in a starts set that you do not have to do any more actions. You are already home is not to speak essentially so this is a condition that you will use. So, you should verify there is possible for meet to construct a plan like this using backward stack space search, but this is not a valid plan. So, I can do a check for a valid plan how to a do that? So what is a check for valid plan check for a valid plan takes an as arguments a plan pie. Basically takes a problem description and a plan. And the problem description is a starts states and a goal description and a set of actions.

So, which this input you can like a small routine which will begin with a start states and I will not write the detail essential running out of time. It will begin with a start set and keep progressing with the actions what is the actions pies A 1 A 2 A k. So, the plan is of k actions you begin with a start set apply the force action. And we have seen that progress is a sound you will get a new state apply the second action to that you apply the third action. And till the end you keep applying the actions and then you see whether the state that you reaches the goal set on not. And you will see that if you construct the plan like this test will fail that it will say the, this is not a plan why because. For example, this action will not be applicable at all, because you will ever reach a state where you a holding both these things together.

So, you can do the check essentially another thing we can do a check for consistency and proven, because start state may have 100 of facts. So, for example, here we have facts about F and E and H and G and K there will never be generate it in the back ward face from there we will never talk about K and J. So it will never figure in the goal description see what will happen is that you will say I want to stack A on B and I want to stack B on D. So, first I want to stack B on D and then I want to A on B so stack B on B add to first pick up B. To pick up B I have to remove A so it will remove A pick it on the table then pick up B sorry pick up C and remove it on A table so that D becomes clear. Then it will pick up A and stack on B then it will pick up S and stack into A those will never come.

So, those in the starts it so that is why this subset is necessary. This check for consistency is basically just a moment. This check for consistency basically proving set saying move this state do not go back ward from here. Or remove this state do not go backward from here inset go along this ration. No as you can see in the actual plan this will be the last section stacking A on B that will necessarily with a last section. The previous action must be to make this clue or to make this the previous action must be or maybe it is make clear B true. Once B is made clear and then holding A for that the action will be pick up A as you shall look at this carefully. You have to first put a on the table then you will have to pick up a from the table. So thus third last action would be picking up A so there are parts which are which will yield the plan there are parts which are can be proven.

And then even if you do not have well you have this final check. That after you think he of solve the problem just see whether what you have produce is the valid plan or not. If it is a valid plan then accept the solution otherwise continues searching essentially. And the good thing about this consistency check is at it does not have to be perfect. If it is perfect it will it will proven this state if it is not perfect it own proved. It you may produce a series of actions and you will find it not a plan and then you come back and try again essentially. So, even if you have a in consist in perfect consistency check it will sell work essentially. So, in the next class, we will try as we keep doing all these time. To look at an approach which will combine the good features of backward search and forward search.

The good feature of forward search is that it is sound that whatever acts sequence of actions it produces are valid plans, because the applicability test is the sound. The good feature about backward stack space search is that branching factors is a low, but the bad point is that it produces the poorest states. This is not state at all it is a poorer state and we have to know do unnecessary extra work to remove the states essentially. So, will try and see an a algorithm which will search in a backward fashion like this and construct a plan in a forward fashion like that essentially for this is doing is searching in the backward fashion and constructing a plan also from the last action backwards. This is the last action this is the second last section will soon which as you are seen is a not a sound process. This one is constructing sound plans, but it is doing too much searches you want to combined the 2 and will do that next class.

So, will stop here.