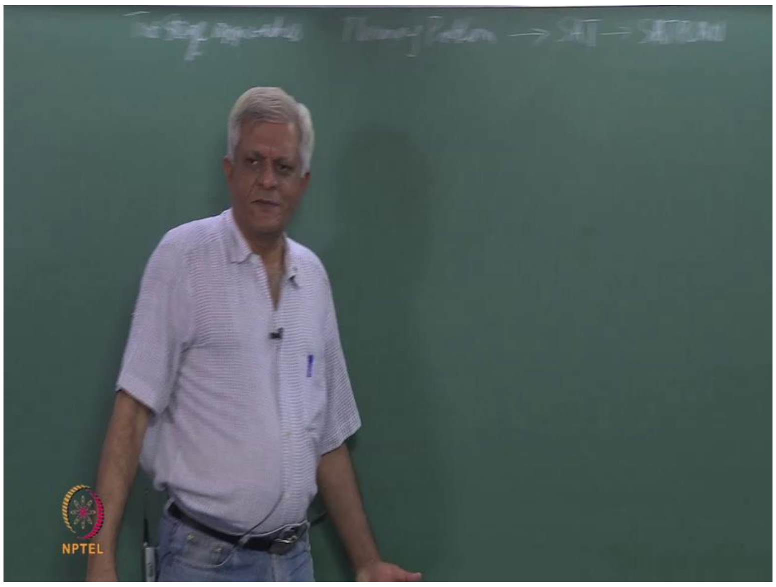
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 38
Graph Plan

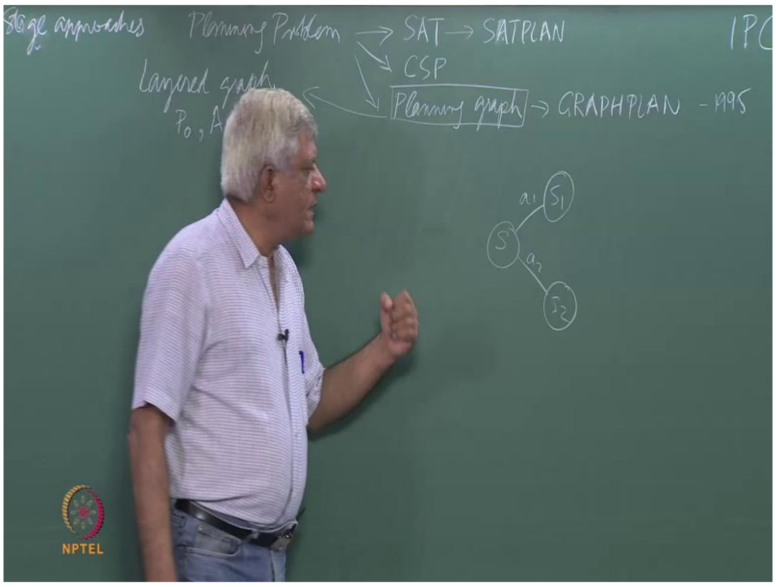
So, let us for the last time this in this course look at planning and so as I said in the mid nineties, there were a series of approaches to planning which produce much longer plan, which means, they were much faster algorithms and could the solve larger problems.

(Refer Slide Time: 00:37)



All these approaches were 2 stage approaches which means you have at we have a planning problem, you converted into something else and solve that problem essentially. So, one thing that was very successful was that if you converted into a S A T problem and we already familiar with S A T and then you could use some solver. In fact, there was an algorithm called S A T plan which was very successful in the mid nineties and early part of this century. So, how do we know it was a good algorithm?

(Refer Slide Time: 01:29)


Planning Problem → SAT → SATPLAN
CSP
Planning graph → GRAPHPLAN - 1995
Layered graph
IPC
NPTEL

They used to be, there is every 2 years a competition call I P C international planning competition, in which people submit their programs. So, just like you had this travelling salesman problem competition, in this you submit your planner and they try it out to set up problem and then the planner which was best is does the to be the winner.

So, S A T plan was one of those planners which did very well for the period of time, and what they did was to convert a planning problem into a satisfiability problem and then solved it using it so on algorithm because, but you could solver first, that is ((Refer Slide Time: 02:13)). Another approach was to convert a planning problem into a C S P problem, which as a name suggests converts is into a C S P and, by C S P mean constant satisfaction problem we look at a, at them briefly in the next 2 lectures and then solve the C S P.

So, here posing it has a different problem and then using some solver, to solve this is recurring team in many problem solving situations. What graph plan does is to convert it into structure called a planning graph? This is algorithm we want to look at today and it is called graph plan, it was given in 1995 by two researchers called Blum who is now as C M U I think in first. So, failed recent by in contrast to the other algorithm that you have seen all those algorithms were quite old.

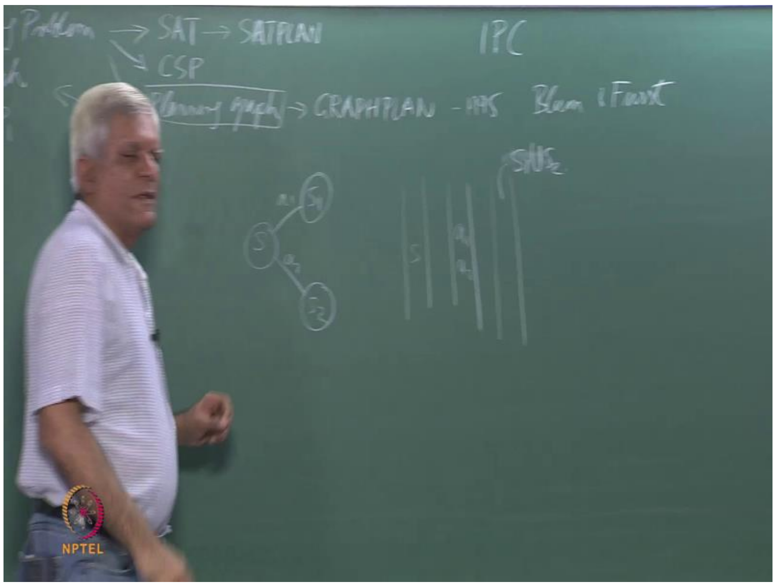
So, as a name, as I, as I said what these types is it construct a structure called a planning graph and then searches in the structure for a plan. All these 3 algorithms have one thing in common is that the conversion that they do in the case of either S A T or C S P or planning is incremental in nature. So, they construct a structure of certain size and then see if there is a plan there little bit like D F I D you might say. Keep exploring longer and longer plans to see when you can find the plan and because of that all these algorithms, end of finding the optimum solutions as well because they incrementally search for longer plans.

So, you can imagine that whatever the mechanism for convert, take it into a S A T problem, it is a relation between action and states and new states and new action so on and you express this as a satisfiability problem. Things like re condition of this must be satisfied by the action, by something you know that kind of some and they always find shortest answers. So, let us start where looking at the panning graph structure, the panning graph is a layered graph with alternate layers of prepositions and actions essentially.

So, there is a layer P 0 which is, which contains all the star prepositions, then there are, there is a layer A 1, which is a layer of all possible actions, which can be done at the first time instance. But, followed by layer P 1, which consists of all the prepositions, which could possibly which you at the, at the end of the first layer essentially. Now, the difference between the other algorithms heuristic search, we just briefly mentioned in the passing that we know. We could have powers express search or backward express search with the heuristic function which was computed by solving a relax planning problem a to decide which successor is best.

That search heuristic search or set space search searches in this set of states plan space search that we saw in the last class searches in the space of plans. Here, we searched in a structure called a planning graph which is some sense are union of all possible states that can happen essentially. So, if I have, for example a start state and I can do action A 1 to go to sate S 1 and I can do action A 2, do to the state S 2, that is what heuristic search would do, powers express search would do, it is a should, I go to S 1 or should I go to S 2, in graph plan what you do is.

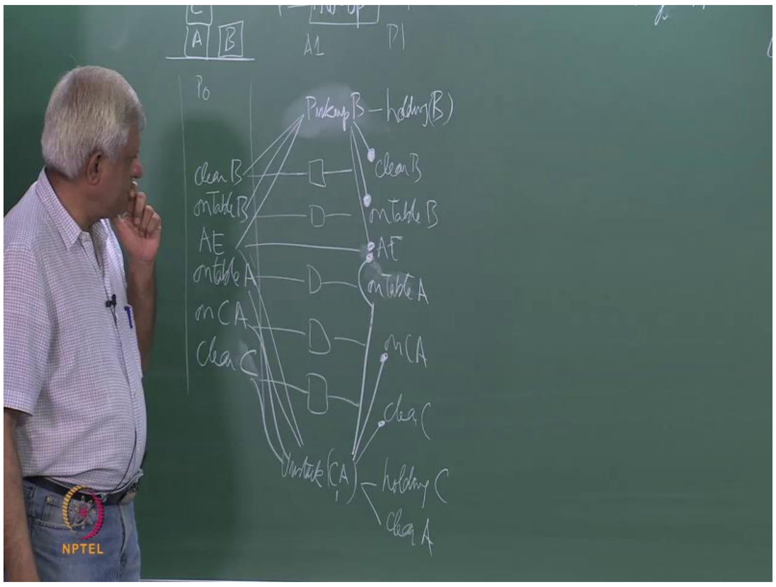
(Refer Slide Time: 06:57)



You have this layer start layer, then you have first action layer which includes both this actions A 1 and A 2 and, then you have layer which is basically the union or will be S, right S 1 union S 2. We put together all the possible predicates into one set which is in this next layer or the, so we have this proportion layers and action layer essentially. So, one thing with all these algorithms do is to first convert this problem into a proportional problem which means, then if I will operator like a pick up a put down. So, on you look at what the starting state is what the objects in the domain are, so that let say I have 5 blocks A, B, C, D, E, then it will produce all possible actions.

So, with this 5 blocks pick up a stack A 1 to B, stack A 1 to C, all possible actions that produce and work A goes actions. So, I want to give you a flavor of this, so let us search work with this, with this same example that we are looking at the Sussman's anomaly.

(Refer Slide Time: 08:20)



Which means we are starting with the state A C is on A and B is on the table and they are on the table, so in the initial layer, in layer P 0, we would have all these things, so let say arm empty then on table B, clear B on table A on C, A clear C these act these. So, this is P 0 while layer P 0, now there are 4 kinds of edges in the planning of the, first is preconditions edges which go from an action to some precondition in the previous layer. Then effects positive edges, it takes minus edges and one more set of edges, which is called Mutex Ulatian.

Now, already familiar with the precondition edges, the positive effects and the negative effects we look at work Mutex is in the movement. So, in the, in the initial layer, we have this proportion, so it is a proportion layer 0, P 0 then we have action layer 1 in which we want to insert all actions which are possible in this propositions.

We want to put a slight additional constant that the preconditions of those actions must be possibly true at the same time if they are possibly true, at the same time we will consider such an action. Let us, why this notion of Mutex relations will come? So, as he told with, now this is an often use term some mutual exclusion essentially, so we will have this notion of. So, this Mutex relation is going to be a, these edges are going to be inside east layers essentially, so this layer will have it is own Mutex relations.

This layer will have Mutex relation, they are from one action to another, in an action layer or from one proportion to another in a, in a proportion layer. If they might exists between two entities, either two actions of the propositions it means at, though A cannot be true at the same time. So, it if there is a Mutex between A 1 and A 2, it means they both cannot be done at the same time. So, as you can see in this example, we have 2 actions unstacks A from A or pick up B and, we can see that they cannot be done in the same time because they both required the arm to be empty and only one can use the arm essentially.

So, you would have a Mutex relation between them, likewise they are Mutex relation between proportions which say that those things cannot be true at the same time one thing which happens constantly as we construct planning. So, we construct the planning graph from left to right, we keep inserting actions, then proportion and actions then proportion and actions then proportion. Till what time do we do that? Till the time when the goal propositions appear in a propositions layer and they are not mutually exclusive.

So, what are the goal propositions that we are interested in? In our case, we are interested in on A B and on B C, if these 2 occur in a proportion layer. In fact, the first time they occur in the proportion layer and they are not Mutex, which means they are not mutually exclusive as per the planning graph. Then there is a possibility that we have solved the problem so that at that point, the first stage of the algorithm which is a powered space and the second stage begin. So, that says, that this is that actually a plan which will achieve a both those put goal conditions for me or not essentially. So, it is a 2 states problem, you construct a planning a search for solution, if you cannot extend the planning graph by more search for a solution and you keep doing that.

Now, one action which is a uniform action which we always insert is called a no op and a no op has any predicate P as an input and the same predicate P as an output. So, as a name suggest, it says that no operation we are doing on this predicate. So, we will depict no ops by these lines like this initially, but we will not draw them after that because they really take up to a space. So, as a effect of this, in P 1 we will have the same actions clear B on table, B arm empty on C A clear C, this we will do all the type in every layer, we would always insert no op actions.

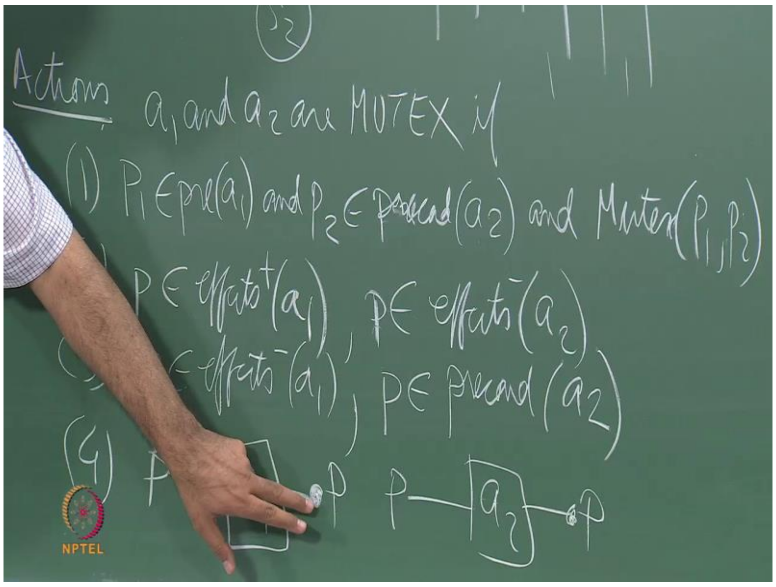
You can see the effect of this no op actions is going to be that the planning graph is going to grow monotonically. The sets of propositions in the proposition layer is going to grow monotonically because whatever is present in previous layer will always, we carried forward to the next layer plus of course new things may be added essentially. So, what are the new things which can be added in our example? We have the action unstack no, so it be write here pick up B, so that I can add to this. Why can I, why can I add to this? Because it is preconditions that B is clear, that B is on the table and arm empty is true maybe I should have written this a little bit that side we will manage then.

So, this is a first kind of links we have been edges, we have maintained the precondition edges then the second kind of edges are positive effects. So, the effect of pick of B is holding B, so I have a positive edge from this to holding B and I have negative edges to it is negative effects which is on table B. So, I have negative edge going from here, so we will defect to net, let say we can defect net negative edge leg this, and for argument sake we will urgent that this also becomes falls at the end of A and arm empty.

So, that is a third kind of edge, so first kind is precondition edges which go from a propositions to an action layer and they basically capture the preconditions that A is necessary, thus action to be applicable. The second kind is the positive effects which are what would be true in the next layer of proposition, if there action word to be executed and in that sense. You can see that no op, basically say where if the no op action word to be executed then on table layer would continue to be true a. So, this we have been here on table A was true here and if I do a no op then on table A will be true in the next layer essentially.

That is one more action and one more action we can add is unstack C from A and you can see that the preconditions for that is that arm must be empty, that A must be on the table A and C must be clear. So, all these 4 conditions of preconditions for that the effect of that is holding C and clear A and the negative effect is that we have assume that we will mark this like this. But, that C is no longer clear C is no longer on A and arm is no longer empty, so when do we have Mutex solutions? That is A. How do we define that two entries in a layer are Mutex essentially? So, let us talk of actions first.

(Refer Slide Time: 18:47)



Actions: a_1 and a_2 are MUTEX if

(1) $P_1 \in pre(a_1)$ and $P_2 \in precond(a_2)$ and $Mutex(P_1, P_2)$

$P \in effects^+(a_1)$, $P \in effects^-(a_2)$

$effects^-(a_1)$, $P \in precond(a_2)$

(4) P P $P - \boxed{a_2} - P$

There are 4 conditions under which action are Mutex, so we say a 1 and a 2 are Mutex if one of these conditions holds. The first is that P 1 belongs to precondition of a 1 and p 2 belongs to precondition of a 2 and Mutex P 1, P 2, Mutex P 1, P 2, I mean there is an edge between P 1 and P 2 which says that they are Mutex. So, if a 1 requires P 1 as precondition and a 2 requires P 2 as a precondition, and the previous layer, thus to mark as Mutex then those actions convert be done together. Essentially, 2 effects P belongs to effects plus of a 1 and P belongs to effects minus of a 2, the second condition for two actions to be Mutex is that there is some predicate P which is a positive effect of.

So, that is in the layer, remember that this effect links go from the action layer to the next layer and we are trying to decide whether any 2 actions in the action layer or Mutex or not. So, if action a 1 produces P and if action a 2 deletes P, so that means it is a negative effects of a 2 then a 1, a 2 are Mutex. They can never happen the same time because the semantics of these 2 actions happening in parallel is not defined essentially. So, graph line in general, will allow us to develop parallel plan which means if we are the 2 arms robot, for example. Then the 2 arms robot could have simultaneously picked up C from and picked up B from the table and then something with them essentially.

So, parallel actions are in general allowed, of course we have put both is actions in our action layer, but we know that, now familiar with this one arm robot. How it can operate? That only one of those actions can happen which means that these actions must be mark as Mutex somehow essentially. So, we will see this is not a condition for that, but one condition is that the one action is producing something P and the other action is deleting P. So, we do not know it both were to happen together whether P would be true or P would be false and, if you go to linearise them then the final effect of those sequential actions would change depending on the order essentially.

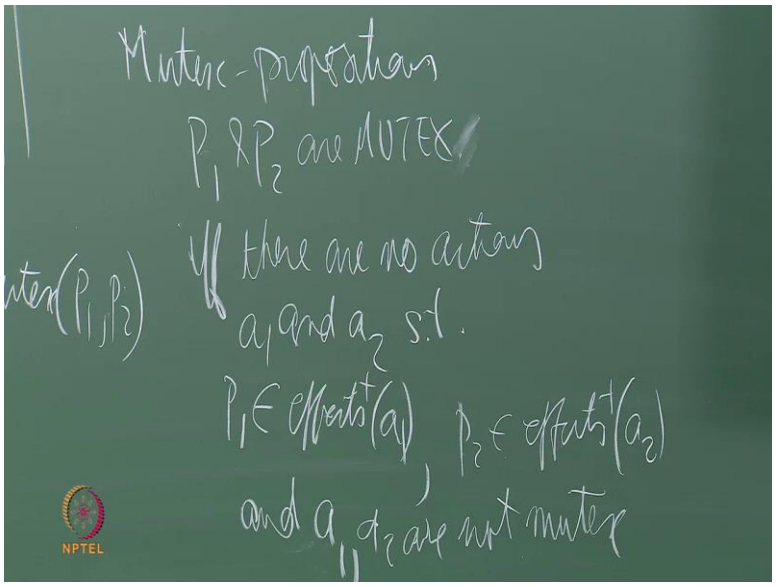
So, we cannot do that, one condition the third condition is the P belongs to effects minus of a 1 and P belongs to precondition a 2. So, if a, if a predicate P is required by action a 2 and predicate P is being deleted by action a 1, then we say that a 1 and a 2 cannot be parallel. Again, the reason for this is, that if you got to linearise them then the 2 orders will not produce the same effects essentially, for example if you going to do a 1 first then it would delete P and then it convert to a 2 after that you could do a 2 first and a 1 afterwards.

But, then you can see that the effect of these 2 out actions is different, we want our parallel pans to web search that they can always be linearise to give us a solution plan. So, that is one more condition and the fourth condition is that, so it as draw it like this is a 1 produces P and deletes P and if a 2 also does that, it consumes P and deletes P if both of them are consuming something. So, this remember this dot stands for deleting P, it is a negative edge, it saying that P is a condition for a 1 and not P is a effect of a 1, this is also saying that P is condition of a 2 and not P is an effect for.

So, if both the actions, so it have something, you have consuming I mean there only 1 cake and there are 2 people, so only one of them eat essentially. So, both cannot be put into the running after saying a is eating the cake and b is eating the cake. In our example you can see that there is a negative edge from to arm empty from pick up b that this is going to delete arm empty and this is also going to delete arm empty both are consuming arm empty. So, this fourth case applies to this and we mark Mutex edge between this and this we can also say it that this no op is Mutex would pick up b because it no op is producing clear b and this pick up b is deleting clear b.

So, the a, second clouds which say that one of them is producing it and other one is deleting it and these 2 actions cannot happen in parallels. So, we have to, so as you can imagine there would be many Mutex suggest between a actions a, that we will have to. So, one Mutex edge that we have is between this and this, another Mutex edge is between these two, and there are others, of course that you can find, so that takes care of the action layer.

(Refer Slide Time: 25:29)



Mutex-propositions

P_1 & P_2 are MUTEX

utex(P_1, P_2) iff there are no actions

a_1 and a_2 s.t.

$P_1 \in effects^+(a_1)$, $P_2 \in effects^+(a_2)$

and a_1, a_2 are not mutex

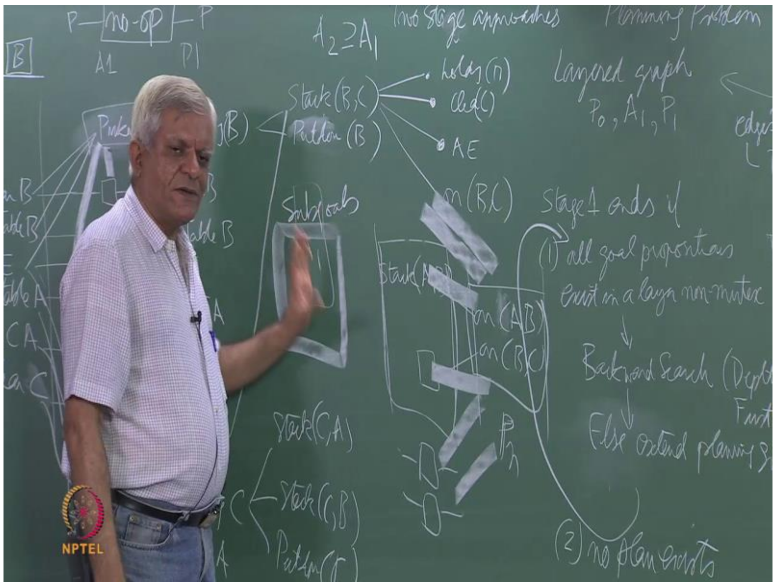
Then Mutex relation between proposition P and P 2 are Mutex, so P 1 and P 2 are proportions in the same layer. Obviously, Mutex relations are always within the layer if there is, I will just write a wrong explanation that we see what to write, what we want to say is that P 1 and P 2 are Mutex. If all ways that P 1 can be produce and all way is Mutex with all ways that P 2 can be produce, and by this mean any action with produces P 2, if Mutex with every other action with produces P 2 essentially.

If there is a no actions a 1 and a 2 in the presiding layer such that P 1 belongs to effects a P 2 belongs to of a with 2 and, in other words if we can find 2 actions a 1 and a 2 which are not Mutex and a 1 is producing P 1 and a 2 producing P 2. Then we can say P 1 P 2 are not Mutex essentially, but if you cannot find, so in there are no actions, a 1 and a 2 is satisfy this condition then we say that they are Mutex essentially.

So, let us solve, try to construct this example from here, so the first thing you will observe is that in this layer a 2. So, a 2 is a superset of a 1 everything which is there in a 1 will always be there in a 2, why because we are just no op operations which a carrying forward everything which was here, which made this, all these action possible. We are being carried forward here and, therefore there will be possible, so apart from this pick up b unstack c from a and so on. So, for example it is always possible for me to say that my first action is to do nothing, for some strange reason that I will do a no op.

If I do a no op then I can always do a pick up b after that essentially or I can always do a unstack c a, after that which means these 2 actions must come in my layer, so everything which is there in a 1 plus something new. What are the new things we can have? So, we can see, so I am not going to write all these again here ideally you should copy it here, but now it is a bit of work. So, I will just write this as a 1 plus the following that you have holding b, you can do something with b.

(Refer Slide Time: 29:18)



You can stack B on to C, why because you know all the conditions are there if you are picked up B, then you are holding B and then C is clear because no op, we will make it clear in that, is that is about what you need.

So, we can stack B on to C, or you can stack B on to, no you can put down B these are two extra actions that I will add to my, likewise for this one holding see I will add stack C on to A, stack C on to B putdown C. So, I have the same A 1 set of actions and remember all these no ops are always there for every predicate in my act proposition layer any proposition that proposition layer there is a no op action which takes it power to the next layer.

So, that is always part of the graph plus anything new that I can add essentially the only condition, I need for adding an action is that it is preconditions must be available in a non Mutex fashion essentially. So, again to repeat you want stack, you want to see what do I need, I need holding B, of course I need clear C and that is what it is. So, of course it will have its effects, it will say arm empty not clear C and not holding B and so on essentially. So, in this process, I construct the planning a layer by layer, at every layer I have to find what are the Mutex and put them in.

As you can observe the planning op is going monotonically essentially it is start with these propositions then it expands and because whenever a proportion gets into the planning of a no op will take it to the next layer, only new ones will be added. So, for example A on a stack B on C, one thing that gets on B C plus of course all these P, so it is growing monotonically, it is also. Can we show in? It is going as a polynomial of the original problem size and general, the size of the planning of grows increases polynomial essentially.

So, that is a first space of graph land you construct the panning graph. So, till what stage? The stage 1, they are 2 possibilities that either you have reach the stage in which a plan may exists. So, let say we have 2 arm robot for arguments, say we can pick up C in and pick and pick up and pick up the unstack C A and pick up B in as action 1, then at the second layer we can put down C and also no.

So, let say in the first layer, you unstack C A and in the second layer you put down C and pick up A at the same time. In the third layer, you pick up B and the fourth layer, you stack B on to C and fifth layer you stack A on to B. So, you can get a 5 step solution if you had A 2 arm robot.

We know that for A 1 arm robot, as you on a last class there is a 6 steps solution, so the stage when ends if one all goal predicates of propositions exists in a layer non Mutex if you can find all the goal proposition. What were the goal proportions to interest in on B on A B and on B C? If they happen be in A, in a proposition layer and they are non Mutex, then we stop this forward planning graph construction exercise and we go to a backward search.

So, typically you could do something like depths first here remember that the problem is still in hard, but you could use other method, so for example people have try to understand satisfaction methods and so on. What are we looking for? We are looking for, so at some stage, so let me write that, those things here on B A and, I can write them here on B A and, sorry on A B and on B C, I am looking for these propositions. So, if they occur in some layer and they are not some let us say P n and they are not Mutex then I start a backward search face and say. Can I find all the sub goals of this, in the previous layer in the non Mutex fact fashion?

Now, hobbies these have positive links coming from actions and those positive links could be in the final solution that we have going to construct. What are these positive links going to be? What are the two actions will produce this stack? But, are they non Mutex? I am looking for non Mutex actually, so I need to do this backward search looking for non Mutex actions with non Mutex preconditions. So, as you can guess this is a no op action essentially, so in a previous layer if I have stack, and I am I am saying that because I know where the solution is a last action must be stack A on B.

But, essentially what they, I will go to them in neat will lead to do is to search, so from this goal set find an action set which assumes goal sets which is non Mutex. So, stack A on B and stack B on C will be Mutex because both are them, so stack A on B makes B, A makes B not clear and this needs or something like that. So, something will come and they will be Mutex essentially, I need a goal set here and then an action set here. Then a goal set come, A here sub goals and then actions and then sub goals and then actions till I come to the initial layer and nowhere do a encounter a Mutex solution.

If that is a case, then found up line, imagine as you can imagine this will be found only in the 6 layer, for this particular problem because A we know by now that it need 6 actions. But, in the sixth layer, we will find this stack A B Mutex with nothing and its condition will be Mutex, will be not Mutex, A envelop them will be holding A and clear B and on B C already the here and, so essentially will be able to find a path back. So, this backward space is a search space which will try to solve every goal set with sub goal set with another sub goal set right up to the start set in a non Mutex fashion.

So, do this, of course it, let it do some it try this sub goal set another sub goal set could be just 2 actions, for example you can see that that it may try. But, of course it will not find a solutions because for 2 no op action to be happen in parallel this must need 7 steps, we know we need this 6 steps anywhere. But, it could try this it could try this combinations or it could try this combinations or it could try com third combination we will do some kind of the search and typically the original graph and algorithm which in the albumin first backward space was depth first search.

So, in the forward space, you go all goal proportions like this in the layer non Mutex, you do backward search, if it fails then you extend the graph by one more layer and do the same thing again. Now, notice that when you extend the graph by one more layer, you will still get those things non Mutex or you mean, you need to convinced about this that once a goal set is non Mutex, it will remain non Mutex in the next layer also. But, it is possible that some sub goal may not the Mutex some sub goal may be Mutex which is why we need to extend the planning a further and further essentially.

So, there are 2 cases, one is that a pan exists in which case stage 1 ends when all goal proportions are found then the backward space begins else. So, I will just, else meaning we are not found the plan extend panning graph and you keep doing this extend planning graph search for a plan, extend the planning graph, search for a plan. So, you can imagine that because we are doing that, if you find the plan it will be shortest possible plan essentially, the other situation is no plan exists or this is the little bit trickles see. So, I do not think we will have the time to go into details to stretch out the conditions when the algorithm should say that no plan exists, but the general idea is as follows.

So, what is happening in this in this planning graph extension stage that more proportions a being added, so more and more things will come in to play as non Mutex. So, initially for example you might say pick up B is possible, unstack C from a possible and then because we all, we already have clear C somewhere. You can say after B you can stack C on stack B on to C and because we have unstack C from we can pick up A in the next you can and so on.

So, it might be that you might see the proportions appear in the proportion layer, but they may not be Mutex, at some point they will be Mutex and then you will search for in a backward space. You will find that at some layer some preceding layer they become Mutex and then you cannot complete backward space search all way all the way to the start set. So, you find that their Mutex for you, if you have tried all possible combinations and at some layer and they are Mutex, then we extend the planning graph to let say it was that layer T, and then extend it to layer T plus one and then we come back and search.

So, what Blamen first showed was this condition, if the sub goal set that you are trying to solve the main constant when you extend from planning. What is the sub goal set? Sub goal set is the set of sub goals where each sub goal is the set that you want, so for example for on A B on B C, they will have the sub goals as we want to do stack on B holding A clear B and on B C. These three actions would be in the sub goal set, this action will have different actions at the sub goal set, some other actions will have different actions in the sub goal set.

So, we have a set sub goals set of sub goal sets, if the set of sub goal sets does not change as you go from layer T layer, T plus 1 as you extend the planning graph from layer T to layer T plus 1 then we can terminate best in it. There is no plan or to get some intuitive feeling behind this, what happens is that as we construct the planning graph first the set of propositions stabilize that no more propositions added to the set. We added everything possible essentially and then the set of Mutex relations stabilize, it means at no more Mutex relations are added.

But, even then you have to be a bit careful that you can extend it by one more layer and you might still find and what Blamen, first this is straightly complete condition which says that if extend the planning graph from layer T to T plus 1. The sub goal set at some intermediate layer n remains constant, then n you can be sure that you will never because if you are not be. If B able to add a new sub goals, at when you move from T to T plus 1 you can never add anymore because a set rule set up propositions as stabilized set up set up Mutex is stabilized.

So, there is nothing more you can do after that it takes a bit of a time to figure this, so it is summarize a graph plan algorithm works in 2 stages. In the power stage construct a planning graph which contains is 4 kinds of edges A, the precondition edges, the positive effects negative effects and Mutex relations. Once it finds all the goal properties, it exists in a non Mutex form in a layer it goes to the backward space tries to search for a non Mutex substructure in the planning graph where they are no Mutex relations which means that those actions are possible even if they are in parallel.

If it cannot find, it goes back and extends the planning graph by one more layer, again goes it will the backward space and keep doing that. So, still at some point either it finds a solution or this criteria which we described somewhat briefly is met where it says that no plan can be found essentially. So, the planning graph gives as a optimal plan in terms of the most the shortest times span or the shortest number of the steps in which parallel solutions can be found if there is a parallels solution.

So, of course keep in mind that domain that we are talking about has no parallel solutions because it has single arm robot. But, many domains may have parallel solutions and what graph plan does give us is the shortest time in which a parallel solution can be found and we can of course always linearise that by putting all the actions in any layer can be linearise in any. What it gives us is the set of actions at layer 1, set of actions in layer 2, set of actions in layer 3 which is the parallel shortest possible planning. So, it is an optimal algorithm and it gives to the shortest plan and it can solve must larger problems than the earlier I will go to essentially.

So, I think I will stop here with planning we do not have time to go into the other algorithms. So, what you will do in the next couple of lecture is get a flavor of constant satisfaction problems which is actually a very big area in computer science in a that there are people who work, try to pose everything as a C S P and there is a huge amount of activity writing solve us for constraint satisfaction. So, what we will do is, we will just get a flavor of how to pose a problem as a constraint satisfaction problem. What is the constant satisfaction problem? What are the basic ideas of? What are the algorithms at we use for solving it?

The interesting thing about studying C S P constraint satisfaction problems is that it gives us a opportunity to combine search with reasoning essentially know that you can do some amount of search. But, you can contain the search while doing some amount of reasoning and constant satisfaction problems give us a national way of doing that essentially. We will try to get a flavor of that in the next few lecturers essentially, and then we will move on to representation proper with login essentially. We will stop here.