Artificial Intelligence
Prof. P. Dasgupta
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture No - 20
SAT plan

In the last lecture, we had studied the algorithm graph plan. Today, we will see that with slight modifications to the style in which graph plan works, we can convert the planning problem into a Boolean satisfiability problem and solve it by using SAT solvers.

(Refer Slide Time: 01:14)

SATPlan

Course: CS40002

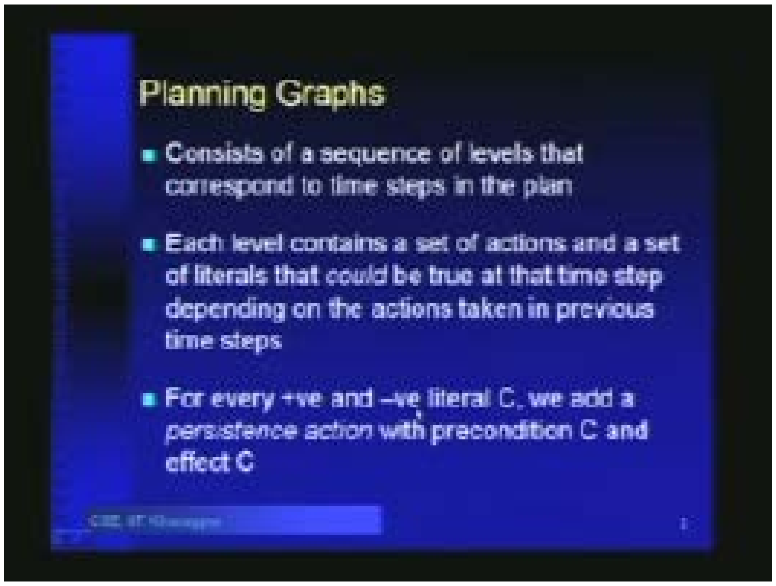
Instructor: Dr. Pallab Dasgupta

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

The reason why SAT plan was born was that there were a lot of new SAT solvers which could handle very large satisfiability problem instances and they could solve them very quickly. So, to harness the power of this kind of SAT solvers, we can actually translate problems into SAT problems and solve them with SAT solvers, which was previously not practical because of the inherent complexity of sat. So, the inherent complexity of SAT is still the same; it is still NP complete, but what we people have found is that many problems, many instances can be solved very efficiently.

And in practice, the instances which are very hard to solve do not appear very commonly. Let us quickly trace back to what we had seen in the last class; we had- just a minute- we had seen planning graphs, which was simply unfolding the state space tree of the planning problem level by level.

(Refer Slide Time: 02:48)



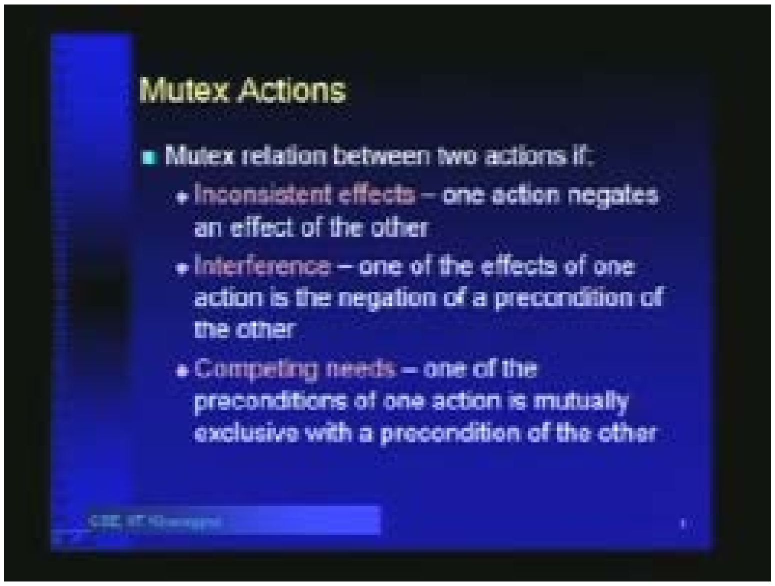
Planning Graphs

- Consists of a sequence of levels that correspond to time steps in the plan
- Each level contains a set of actions and a set of literals that *could* be true at that time step depending on the actions taken in previous time steps
- For every +ve and –ve literal C, we add a *persistence action* with precondition C and effect C

And we had seen that because the set of mutexes monotonically decrease with every iteration and also the set of literals monotonically increase through the iteration. So, this planning graph is finite and tapers off after some time. The first thing that we are going to try in SAT plan is to do this unfolding, but by producing SAT clauses, so that we have a SAT instance at every time step. So, what we are going to do is, we are going to start with the initial state which will be a set of clauses; then, we will create the set of clauses by applying 1 set of the axioms. And then, the new set of classes will be checked against the goal for satisfiability. I will explain with an example.

It is the same planning graph that we will get unfolded, but as SAT instances. But in addition to the basic set of classes that are generated, we will also have to model the mutex actions, because all the set of classes will not be in conjunction, so, there will be disjunction between classes. All this different kinds of mutex relations that we studied, namely inconsistent effects, interference and competing needs, all these will be modeled as SAT instances. Now, 1 thing that we must understand is that manually writing down an enormous number of classes is not possible.

(Refer Slide Time: 04:22)

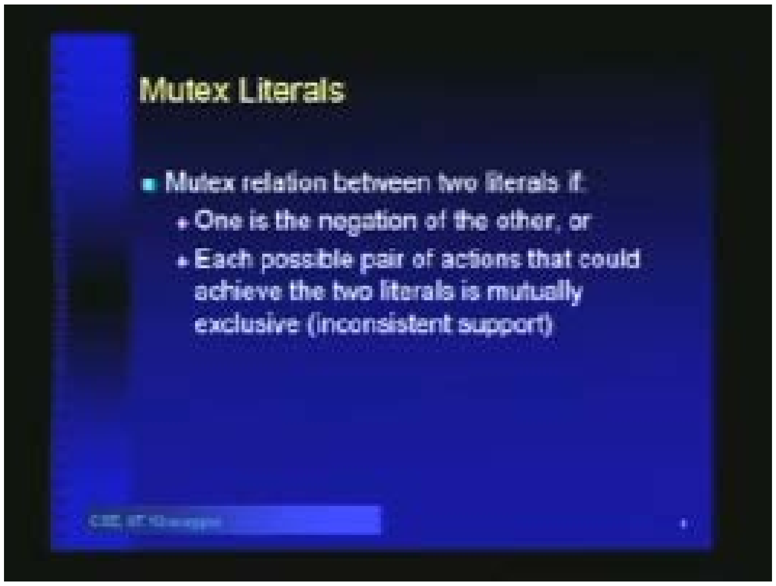


Mutex Actions

- Mutex relation between two actions if:
  - Inconsistent effects – one action negates an effect of the other
  - Interference – one of the effects of one action is the negation of a precondition of the other
  - Competing needs – one of the preconditions of one action is mutually exclusive with a precondition of the other

The front end of the planner will have to be something which allows us to simply express many classes- maybe in form of first order kind of predicates with for all x, etc., and then internally, the planner will expand out that first order clause in terms of propositional clauses. So, suppose we have for all domain of x and generate pa, pb, pc- like that. Internally, it will solve a Boolean satisfiability problem, but the front end can be different, as we shall see.

(Refer Slide Time: 05:10)

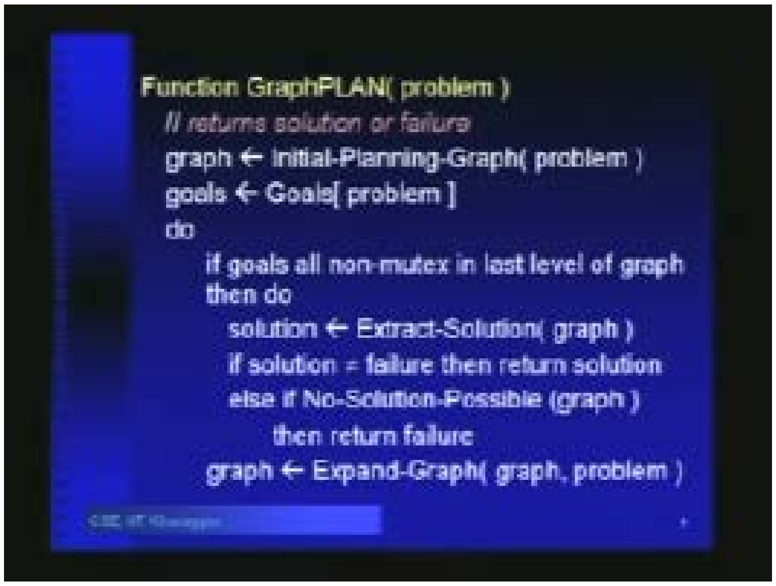


Mutex Literals

- Mutex relation between two literals if:
  - One is the negation of the other, or
  - Each possible pair of actions that could achieve the two literals is mutually exclusive (inconsistent support)

Then, similarly, for mutex literals also, we will have to adopt the same kind of approach. Okay, let us skip through these.

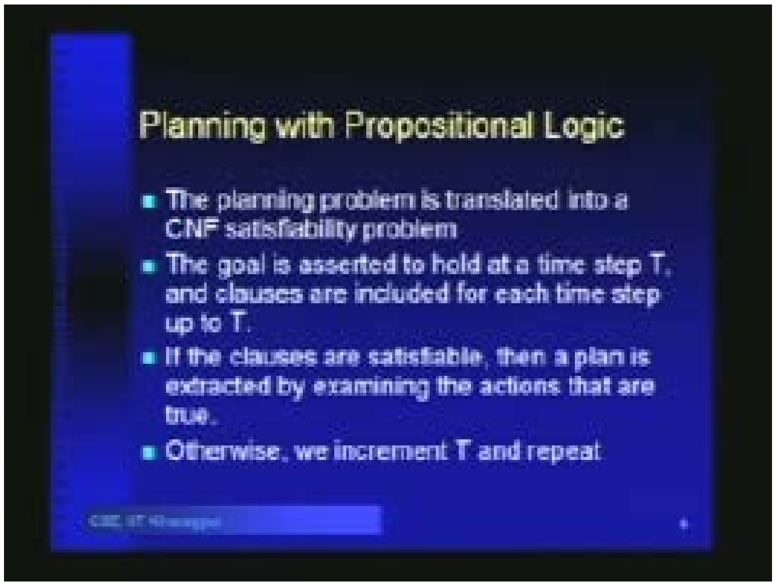
(Refer Slide Time: 05:23)



```
Function GraphPLAN( problem )
  // returns solution or failure
  graph ← Initial-Planning-Graph( problem )
  goals ← Goals[ problem ]
  do
      if goals all non-mutex in last level of graph
      then do
        solution ← Extract-Solution( graph )
        if solution = failure then return solution
        else if No-Solution-Possible (graph )
            then return failure
      graph ← Expand-Graph( graph, problem )
```

Essentially, what we are trying to do is to translate the planning problem into a CNF satisfiability problem. The goal is asserted to hold at a time step T and clauses are included for each time step up to T. So, what we will do is, initially, we will assume T equal to 0 and then try to see whether the goal can be asserted by generating clauses only up to the first time step, which means that the initial set of facts, whether they can actually imply the goal.

(Refer Slide Time: 06:36)



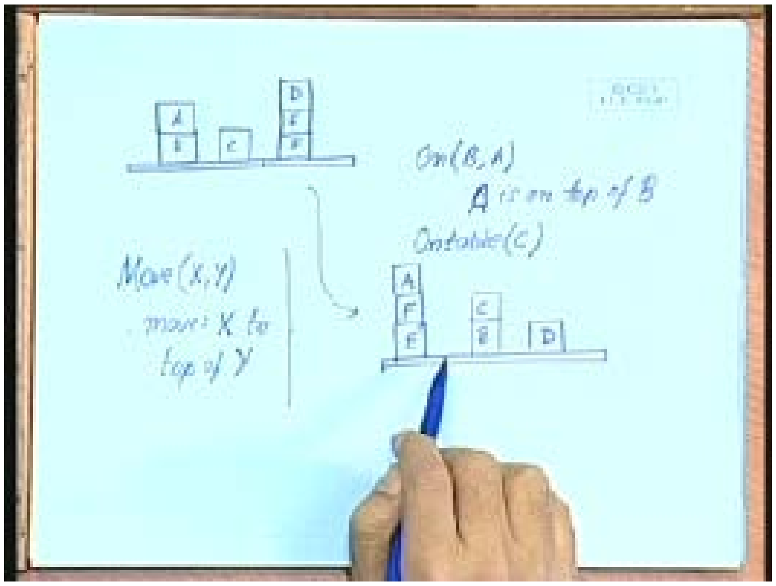
Planning with Propositional Logic

- The planning problem is translated into a CNF satisfiability problem
- The goal is asserted to hold at a time step T, and clauses are included for each time step up to T.
- If the clauses are satisfiable, then a plan is extracted by examining the actions that are true.
- Otherwise, we increment T and repeat

Then, we will increase time- then, we will increase time step by 1. I am trying to see whether within k time steps, I am able to satisfy the goal; if so, then, there is a plan of length k. If the clauses are satisfiable, then, a plan is extracted by examining the actions that are true, otherwise, we increment T and repeat. This is the basic. Let us take an example and elaborate on this. Say, suppose we have the following blocks world problem. Blocks world is a famous example of planning problems, where we have different blocks- like this is A, this is B and they are all on a table.

This blocks world is a classical world for representing planning problems and has been used in variety of domains like robotics, etc. So, the idea is that we have a set of blocks here; I have just 2 for this example, but you can have any k number of block, and there on the table in different configurations. You can have A on top of B and C on top of D and then, you can have another block here with some other things on top of it, and then, so, that is a starting configuration and then you have to reach an ending configuration.

(Refer Slide Time: 10:37)



Suppose I have something like ABC and then DEF and then, maybe the configuration that I want to reach at the end is C on top of B. Then, I have F, E, A on top of F and maybe D lying here. This is the initial, this is the final and the planning is will tell me the sequence of actions that I can do. What are the actions that I have? I can move- which moves now? There is a pre-condition of this action and the pre-condition is that there should be nothing else on top of Y. And also, right, there can be nothing on top of x, so, you can move only 1 block at a time; you cannot pick up D and E together and move it to some other place.

You can pick up only the top most block of every stack and move it, either on top of any of the other blocks or you can put it on the table. This is a set of rules and the set of predicates are things like on B A, indicating that A is on, and we also have a special predicate called ontable C, which says that C is on the table. This is a glimpse of the blocks world, so, in the blocks world, the planning problem will use this sequence of actions and translate the initial state to the final state. Now, let us see how can we model this blocks world problem as SAT problems. To examine this, I will start with 1 example, where the initial state is A. This is the initial state and finally, I will require that A is on the table and B is on top of it. That will be my goal.

Now, let us see how do we model the initial state as a set of clauses. We will write ontable B and I will put 0 here to indicate that this is in the time step 0 and I will have on B A. Is that sufficient? (Students speaking). Yes (Students speaking). Yes, but if there is nothing, then you cannot specify that- what is not on top of A. You cannot specify, but there are other things that we must specify, because see, in the connotation of planning that we have in this world, we know that if B is on the table, then B cannot be on top of it and also we know that if A is on top of B, then A cannot be on the table.

So, we must add clauses like not ontable A and not on- now see, these things should come by default, but when we actually model it in proportional satisfiability, then, we have to explicitly create these clauses. Otherwise, for another domain, this is not automatic, and CNF SAT does not know which domain you are talking about. It just treats them as a set of proportional clauses, so, we have to add these things. Then, yes. (Students speaking). If that is going to come automatically by the proportional satisfiability.

For example, if you have a CNF formula which is on variables x1 x2 x3, the variable x4 which does not exist in the formula is automatically taken care of. So, if there is any variable which does not appear at all in the formula, then, that is fine. We do not have to worry about that, but if there are some variables which are there in the formula, then, we must exactly specify the initial state, otherwise it will represent possible inconsistencies in the domain, like having 2 blocks on top of the same block, kind of thing.

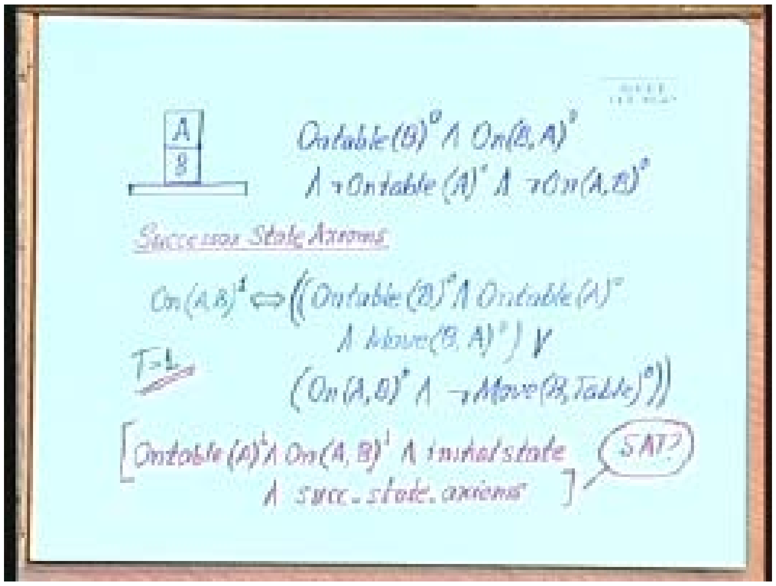
Later, we will see how we can express those general axioms also in the SAT plan scenario, so that explicitly, we do not have to write many things. But I am saying that these clauses will anyway have to be generated internally, whether we specify it externally or not- they will have to be generated internally for this domain. Then, we will have a bunch of successor state axioms, which are going to tell us that if we have this set of clauses, then, what are the things that we can deduce from it. So, these will actually indicate the effect of the different action. Suppose we write that on A B in time step 1 and then we will have to write if and only if. How we can achieve this? This is the effect- let me want and then here, we have to enumerate all the different ways of achieving this, so let us see what are the different ways of achieving that.

One way of achieving this is you have ontable B in the zeroth time step and you have- right? And the action is- this is 1 way of achieving this. So, in the zeroth time step, B was on the table, A was on the table and we moved B to the top of A. This is 1 way of achieving it. Then, another way of achieving this is, we already had on A B in time step 0, and we have the persistence of that. How do we indicate the persistence? We say that do not move- do not move B away from top of A, so, do not not of move, right? And so on- you have to enumerate all the other ways of achieving this.

Let us say that so far, we just add these 2, then what? Yes? (Students speaking). What we are saying is that if we have these clauses at time step 0, then, in time step 1, we will have this. These successor state axioms will generalize for k- the kth time step here and the k plus 1 th time step here. In general, we will have this as k and this as k plus 1, but yes- (Students speaking). See, this says that you already have B on A and you do not move B, then, you still have B on A in time step 1. This says that you already had B on A and you do not move it, then, in the next time step also, you will have B on A.

Now, the successor state axioms can be- (Students speaking). Not- (Students speaking). For this, yes, we have to. In fact, we will see later on, that we will had some other actions, general rules, which will always generate the clauses that whenever A B is on A, then, A cannot beyond B. Those are the additional- so, I am trying to show you that there is a lot of additional work that goes into generating the clauses. Here, for example, when we have this, in general, we will express this successor state axioms in terms of what we can have in the k plus 1th time step; in terms of what we have in the k the time step or previous less than k time steps. But when we actually want to apply this axiom by on the SAT plan instance, then, when initially we are in time step 0, we will check what we can generate in time step 1; when we are being time step 2, we will see what can generate in time step 3, and so on.

(Refer Slide Time: 23:29)



$Ontable(B)^0 \wedge On(B,A)^0$
$\wedge \neg Ontable(A)^0 \wedge \neg On(A,B)^0$

Successor State Axioms

$On(A,B)^1 \Leftrightarrow ((Ontable(B)^0 \wedge Ontable(A)^0$
$\wedge Move(B,A)^0) \vee$
$(On(A,B)^0 \wedge \neg Move(B,Table)^0))$

$T=1$

$[Ontable(A)^1 \wedge On(A,B)^1 \wedge$ initial state
$\wedge$ succ. state. axioms $]$ SAT?

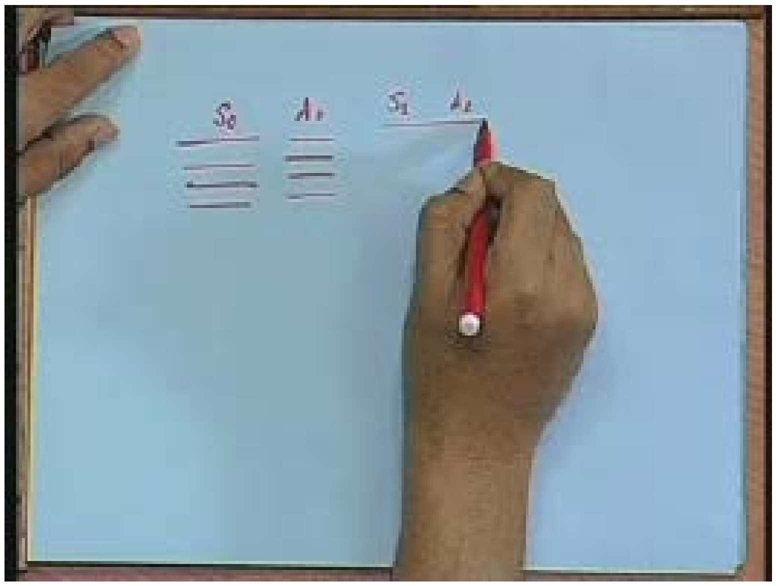
This value of k and k plus 1 will get instantiated as we progress down the iterations. After doing this, after generating the set of successor state axioms and the initial state propositions that are given to us, we will check for the satisfiability of the following thing. We will also represent the goal as a set of clauses, which means that we will check, say, ontable A and on A B. This is our goal. So, what we will try initially? We will try to solve this problem in 1 time step; so, we will say that okay, T is 1, so, if T is 1, then, we will expect that this goal clauses will be deduced in time step 1, so, we will have them as time step 1.

And then, we will see that whether we can and this with the initial state and with the successor state axioms and then, we will check for this whole thing, whether it is satisfiable. If this is satisfiable, then there is an assignment of values to the axioms, such that we are able to deduce the goal now. Is that clear to you? What is that the SAT is trying to assign? What is that the SAT is trying to solve? It is trying to tell us which of these successor state axioms we require, so that we are able to deduce this. So, we take the initial state; we take the set of successor state axioms and we take the set of goal propositions and then together and check for satisfiable. If this is unsatisfiable, then
we clearly do not have a plan yet.

We have to increase T to 2 and proceed forward and again, generate all the classes up to time step 2, create a similar SAT instance like this and again check for satisfiability. This is how SAT plan will work. But there are lot of other things that we need to do- for example, there are certain actions which will require pre-condition. Let me give an example. We will require things like- and an example of this could be that when we have move in some time step 0, then it implies- so, what does this say? That suppose you want to move B on top of A, so, this action can be true provided that on B A is not true.

What does on B A mean? That we have A on top of B, right? If this is true, then, obviously you cannot move B on top of A. This is a pre-condition of the action. For every action, there will be a set of pre-conditions, right? (Students speaking). No, see, what we recall- how we were in graph plan? Do you remember that we had the state S0 which was a set of propositions and then A0 is the set of actions that were taken over that set of propositions? So, our indexes indicate that this is the set of actions which are taken on the initial state, then, that gives us S1 and then actions that are applicable on S1 are given by A1, right?

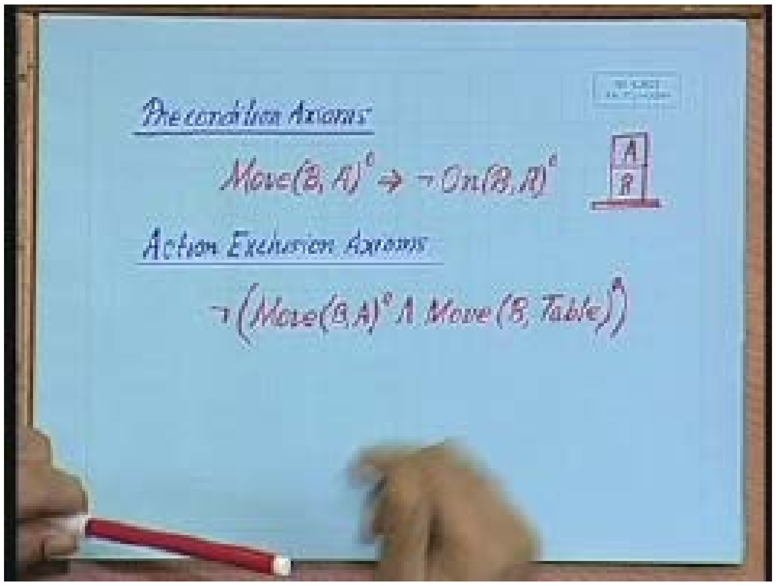
(Refer Slide Time: 26:05)



Similarly, that is why we have this and this, so, if we have this thing 0, then, this action can be applicable on the initial state. It is just a matter of putting the nomenclature of defining these index ones. But you are right, that once we have the not of this, only then, we can apply this action. (Students speaking). If you have on A comma B, means B is on A. (Students speaking). No, this this means move B to top of A. (Students speaking). Because it is already on top. No, in that case, it can be true also; there is no problem, you can just pick it up and put it down back there.

Now, but that condition- see, I think you are confused. What you are saying is what we have written here; this says move B to top of A. We have just written what you have- what you are saying is exactly what we have written here. You have A on top of B, so, you cannot move B to top of A. That is what you wanted to say, right? Okay. In addition to this, there will be a bunch of other things also; like, we will have action exclusion axioms.

(Refer Slide Time: 29:38)



Precondition Axioms

$$Move(B,A)^0 \Rightarrow \neg On(B,A)^0$$

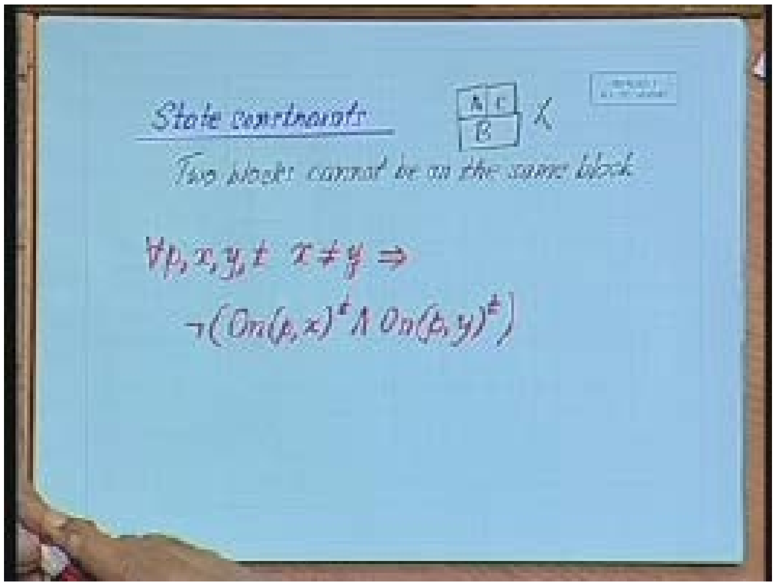
Action Exclusion Axioms

$$\neg\left(Move(B,A)^0 \wedge Move(B,Table)^0\right)$$

These basically tell us which actions are mutex. For example, you can- let me write down 1 action exclusion axiom. I have move B A, and now, let us see what this says; this says, move B to the top of A and in the same time step, move B to table. Now, we cannot move the same thing to 2 different places in the same time step, so therefore, these we have to explicitly specify, that these 2 cannot happen together. So, the action exclusion axiom will say that not of this- in other words, that these 2 actions are mutex- you cannot do both of them.

So, for all pairs of mutex actions, we have to add exclusion axioms like this, which says that if you have this, you cannot have this and so on. And then recall, that we also had exclusion between the literals in a state. How do we express that kind of mutex? Those will be called state constraints.

(Refer Slide Time: 32:49)



State constraints

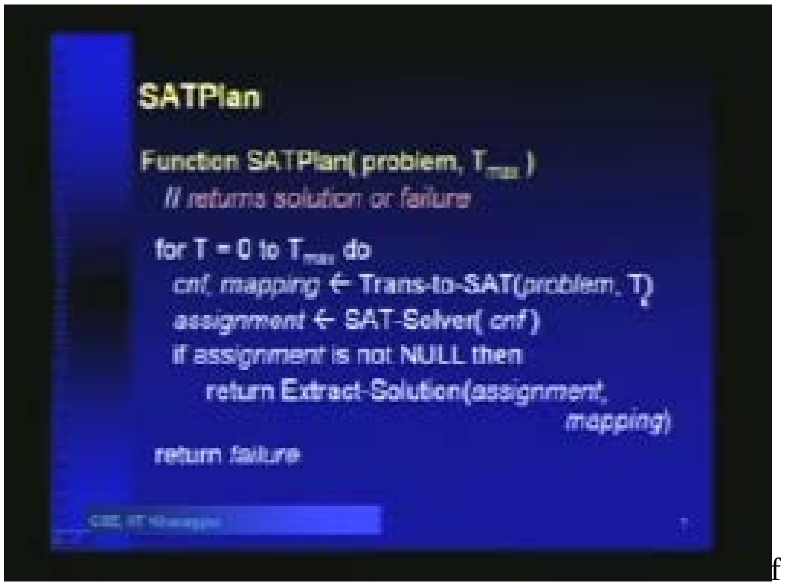
Two blocks cannot be on the same block

$\forall p, x, y, t \; x \neq y \Rightarrow$
$\neg(On(p,x)^t \wedge On(p,y)^t)$

For example, we want to express the fact that 2 blocks, which means that this is not allowed, this is not allowed, all blocks are of the same size; we cannot allow this. How do we express this? We can express this in a kind of first order form, where we say for all p x, y, t; x not equal to y implies, so, we have- what does this say? That suppose in a given time step t, we have x on p and we also have y on p. This is not allowed, so, we have not of this and for all p, x, y, t, but we also have to put this x not equal to y, because both of these x and y can get instantiated to the same block, so, this is a typical state constraint.

In our planning problem, what we need to do is to represent the initial set of states- the set of clauses for the initial state, then the pre-condition axioms, the action exclusion axioms and the state constraints, and then progressively increase the time step and check whether we are able to satisfy the set of goal constraint. This is the SAT plan algorithm; so, we need a t max now. See, this is where we have to be careful, that SAT plan requires a bound on the number of time steps within which you expect to find a solution, right? For t equal to 0 to Tmax do the CNF comma mapping is translate to SAT problem T. What we are effectively doing is, we are generating all the clauses up to time step T, and then, we are calling the SAT solver with the CNF that we have found by this translation.

(Refer Slide Time: 34:01)



SATPlan

```
Function SATPlan( problem, $T_{max}$ )
  // returns solution or failure
  for T = 0 to $T_{max}$ do
    cnf, mapping ← Trans-to-SAT(problem, T)
    assignment ← SAT-Solver( cnf )
    if assignment is not NULL then
      return Extract-Solution(assignment,
                               mapping)
  return failure
```

CSE, IIT Kharagpur 7

f

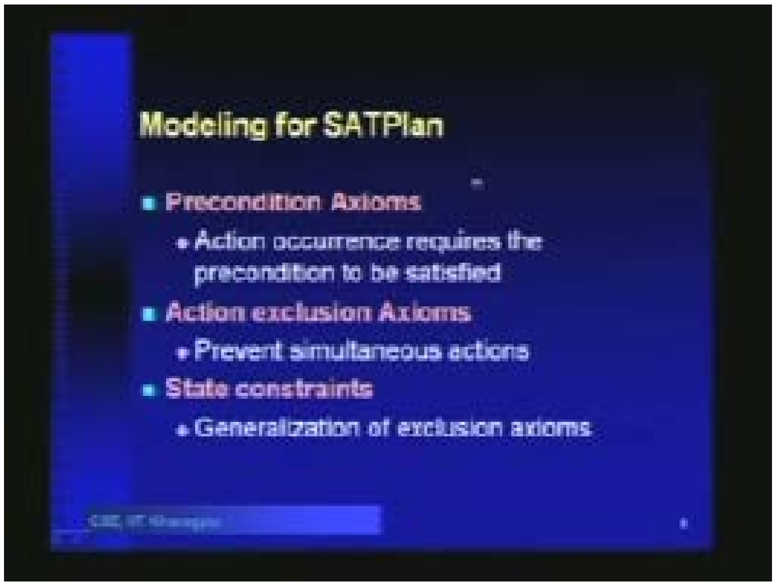
If it finds that the assignment is null, then we still do not have a solution. It is not satisfiable. If it is not null, then, we do have a satisfying assignment. Then, we call the extract solution from the assignment mapping; effectively, what happens is, we see each of our move actions- they are also predicates, like when we have this- text please. We have this- see, this move actions- these are also predicates. If we assign true to this predicate, it means that we are actually applying this axiom, so, at the end of the satisfiability, we check which of these actions have been given the true value and that is the set of actions that are part of the plan.

Once we have the assignment, then we can extract the solution by checking which of the actions have been assigned 1. Otherwise, we return failure, if you are still unable to find the satisfaction within Tmax; we return failure. Now, the question is, what value of Tmax should be used? How should we decide the value of Tmax? What could be the maximum the Tmax can to take place? Recall that we had seen that in the planning graph when unfolded will taper off after finite number of steps.

So, if we could somehow gauge that the diameter of the planning graph, then, we can give the given accurate Tmax, within which we should be getting this thing. So, there has been research on the fundamental problem of determining the diameter of a graph which is specified implicitly. The graph is not given to you as the entire set of nodes, etc. What is given to you is the initial state and a set of state transition relations and without unfolding the graph, can we predict the diameter of the graph?

That is another fundamental problem which people have addressed and attempted to solve and there are different kinds of estimates on the diameter of the graph. We have to find out an appropriate this thing and this thing, but in many cases, we actually have some expectation about the length of the plan. So, people can actually give some higher figure here and then be able to solve the problem. So, we have already discussed pre-condition axioms, action exclusion axioms, state constraints, etc.

(Refer Slide Time: 37:48)

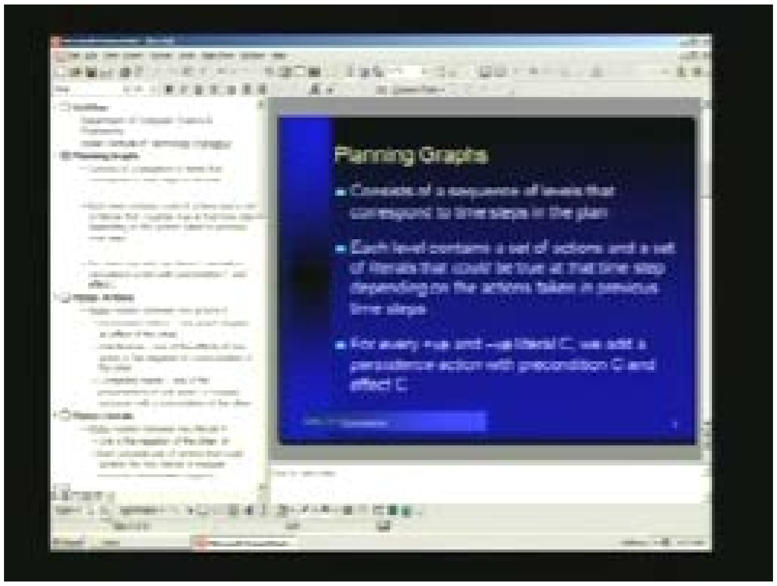


Modeling for SATPlan

- Precondition Axioms
  - Action occurrence requires the precondition to be satisfied
- Action exclusion Axioms
  - Prevent simultaneous actions
- State constraints
  - Generalization of exclusion axioms

With this, we come to the end of the section on SAT plan and we are also at the end of the chapter on planning. Any questions?

(Refer Slide Time: 38:04)



Planning Graphs

- Consists of a sequence of levels that correspond to time steps in the plan
- Each level contains a set of actions and a set of literals that could be true at that time step depending on the actions taken in previous time steps
- For every +ve and ¬ve literal C, we add a persistence action with precondition C and effect C

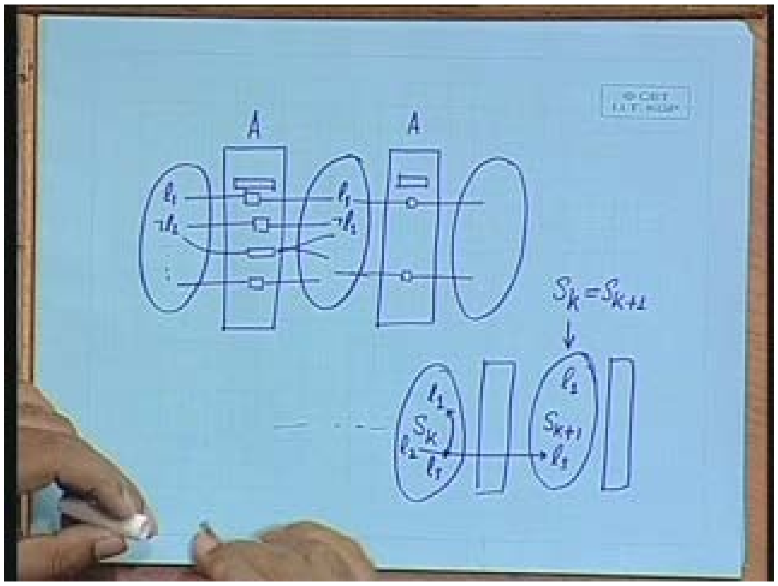
(Students speaking). Okay. Let me- you want to know, your question is- what is the fix point that we were talking about when we talked about expanding the planning graph? It goes on like this, that we have an initial state set of literals, that is the initial state, so, this has a set of literals which are in conjunction. I can have some literal, L1 and not of L2 and so on; these are all part of the initial state. Then, when I apply actions, I want to apply the set of actions on this. Now, see, all actions here will not be applicable, because their pre-condition will not be there in it. I can apply an axiom provided that the pre-condition of that axiom is there in the set. Suppose that requires L1 and not L3 but you do not have not L3 here yet, so, you cannot apply that action.

There is a subset of the axioms that you can apply now. When you do that, then, you will get a new set of literals that are the effect of those actions also, each of these literals will get carried over to the next because of the persistence action. Whatever we have, all of them will get carried over and in addition, because you are applying some actions, some new literals can get generated. Now, recall that the set of literals can contain negations of different literals. For example, you can have L1 and you can also have not L1, but this L1 is generated by some action, this not L1 is generated by some other action. Those 2 actions are mutually exclusive; I cannot take both of them, but I can take either of them. In this set, I am maintaining all possible literals that I could have generated. I cannot generate all of this advance, but I can generate any subset of that which is consistent.

Again, on this literal set, I can again apply the set of axioms and now, some of the actions which previously could not be applied may now become applicable, because the literals which were missing have coming. Again, by applying this set of actions, I will get another set of literals, which is a super set of this, because all of these will get carried on by the persistence and some new ones can get added. In this way, we continue and the claim is that after finite number of iterations, you will find that in 2 successive iterations, this set sk and this set sk plus 1 are the same.

I will find that sk is equal to sk plus 1 and when do I say that these 2 are identical? If they have the same set of literals and they have the same set of mutex links. That is when I say that these 2 are identical. Then, if I apply the set of actions again, I will get the same thing. This is the fix point of the set of actions that I have, so, if up to that point of time also, I have not been able to obtain the set of goals in a non-mutex manner, then, I know that the plan does not exist. (Students speaking). No, because what may happen is you can have 2 literals here- say, L1 and L3, which are mutex.

(Refer Slide Time: 43:36)

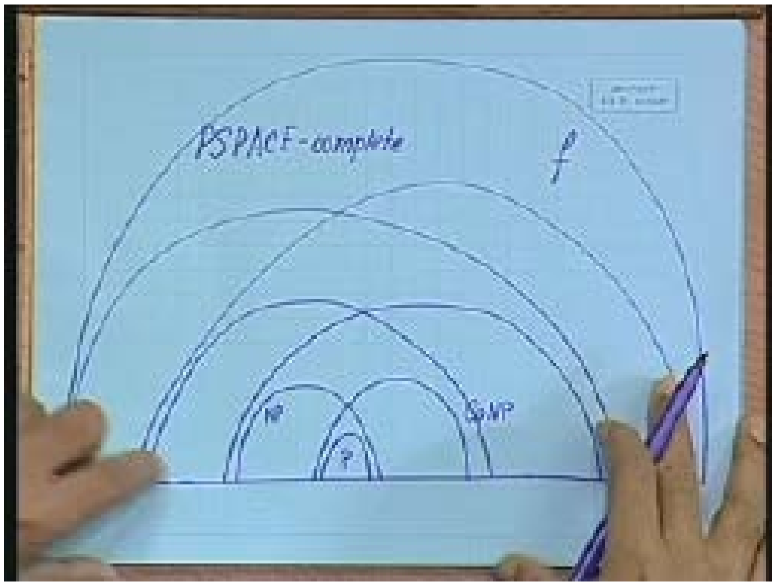


But when you apply the action once again, then, from some other- suppose L2 and L1 were non-mutex, then, you get L3 from L2. Suppose using this action once again, you get now L3 from L2 and because L1 and L2 are non-mutex, so, now in this one, L1 and L3 will be non-mutex. It is possible to have the same set of literals, but the number of mutexes can decrease. There has been some debate on what kind of problems require what kind of planner.

It is still not a problem which is well decided as such, but people have began to understand that what are the kinds of problems can be tackled by which kinds of planners. And the general understanding is that if your planning problem is NP complete, then, graph plan and SAT plan in general, what better, whereas if your planning problem is solvable in polynomial time, then, partial order planning usually finds the solution more quickly, because in SAT plan and graph plan, adding out all those constraints themselves take a lot of time; to create the instance of the SAT takes a lot of time.

For those problems which are polynomially solvable, it is usually the case that POP finds the solution much more quickly; the overheads do not balance. But for hard problems, POP can take a very long time, because it will have to see after it has generated the graph; recall that you have to do extract solutions. It has been shown that the planning problem is PSPACE complete. Do you know what is PSPACE complete? Yes. (Students speaking). Yes, is this harder than NP complete? Well, if p is not equal to NP, then yes, to give you a very quick idea about what this is- see, okay. Firstly, we have a set of problems which are in NP, which means they can be solved by a non-deterministic polynomial time algorithm. And then there is a clause called CoNP, these 2 differ in the sense that these are the duals of the other problem.

(Refer Slide Time: 49:46)



For example, if I ask you that given a formula CNF formula f- is it satisfiable? Then if the answer is yes, it is satisfiable, you can give me a witness quite easily. How can you give me a witness? He will just give me an assignment to the variable value. But if the formula is unsatisfiable, then, how will you prove it to me? How will you prove that the formula is unsatisfiable? It is not easy, right? For the yes answer, I have a poly time verifiable witness; for the no answer, I do not have anything. In the CoNP, it is the other way around; for the no answer, I have a poly time verifiable witness, right?

Now, for example, if I ask- is f valid? Is it true for all instances, all instantiations? If the answer is no, then, you can give me a counter example, but if the answer is yes, you cannot do that. So, while satisfiability is in NP; unsatisfiability or validity is in coNP, and we have p, the set of poly times available in the intersection of this, because we can for poly time solvable problems, the yes can also be proved in polynomial time, the no can also be proved in non-polynomial time. Now then, people say that okay, if I gave you an oracle which could solve NP problems in polynomial time, then, what could you do?

Suppose somebody gave me an oracle which is a black box and solve these problems in a jiffy. Then, we can have problems which are hard to solve with that oracle. So, this is a higher class, so, we can build similarly like NP and CoNP, we can build a similar clause on top on this. So, this is a set of problems which can be solved using a non-deterministic polynomial time algorithm, given an oracle which can solve NP complete NP problems in polynomial time. And this is the co-version of that; they do allow that, so, in this way, we can build a hierarchy.

So, again, we can have something on top of this like this, and we can have something on top of this like this. This is how the hierarchy builds up and this hierarchy will build up forever; this hierarchy is called the polynomial hierarchy and this whole polynomial hierarchy can be solved in polynomial space, so, that is the clause PSPACE that we have. So, when we talk about PSPACE complete, it means that it is harder than NP; harder than NP to the- okay, so it is harder than all these clauses. The planning problem is known to be PSPACE complete, but the generation of the graph up to the depth D can work in polynomial in the depth D.

It is the extract solution part which is hard; the 1 that we totally crossed over is actually the hard problem. Once you have the set of goal propositions in non-mutex form, then, extracting the solution is hard. Because that is hard, so, if your problem is NP complete in nature, then, that part is really difficult to solve. So, algorithms like graph plan and SAT plan which solve them as satisfiability problems tend to work better for NP complete kind of problems.