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

Lecture No- 17
Introduction to Planning

We are at the middle of the course right now. In the first part, we have studied some of the major parts of AI, namely the search algorithms which enable us to do problem solving. And then detection techniques or first order logic, etc., which helps us to represent problems in a declarative way and solve them using detective approaches.

(Refer Slide Time: 01:14)

Introduction to Planning

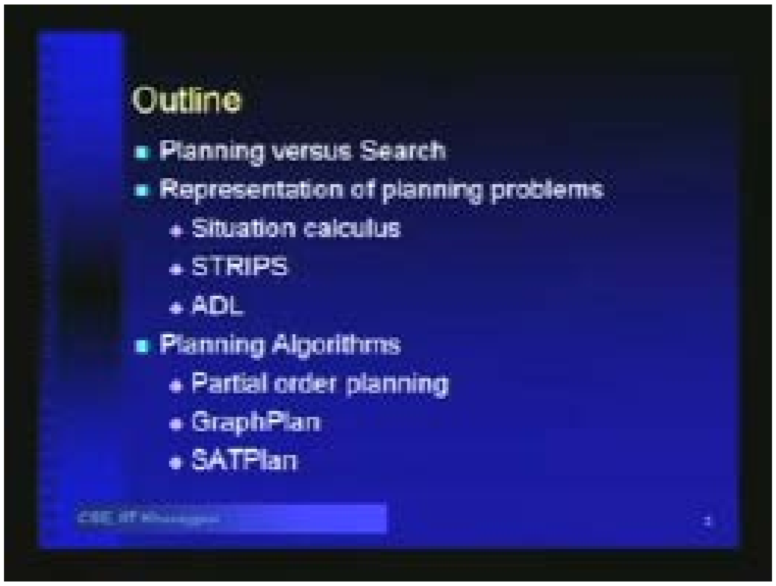
Course: CS40022
Instructor: Dr. Pallab Dasgupta

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

now there Hereafter, we will see several variants of these and specifically targeted to certain types of applications. What people have- see, these are the 2 original things that were done in AI; namely, the ability of doing automated problem solving and the ability to do deduction. And then, when people actually wanted to apply these techniques to different problems, they found that these are too generic to handle problems of very large complexity and of very specific types. So, people came up with different problem solving techniques, specifically targeted toward those domains.

The first area that we are going to study will come under the area of planning. Planning is a category of problems which are essentially search problems, but because they have certain specific features, we will model them in a different way and we will have dedicated algorithms for solving them. There are variants of such algorithms, but there are certain specific features of them. In this particular topic, these are the things that we are going to do. Firstly, we will just have a small discussion on planning versus search. So, how does planning problems differ from generic search problems?

(Refer Slide Time: 02:48)



Outline

- Planning versus Search
- Representation of planning problems
  - Situation calculus
  - STRIPS
  - ADL
- Planning Algorithms
  - Partial order planning
  - GraphPlan
  - SATPlan

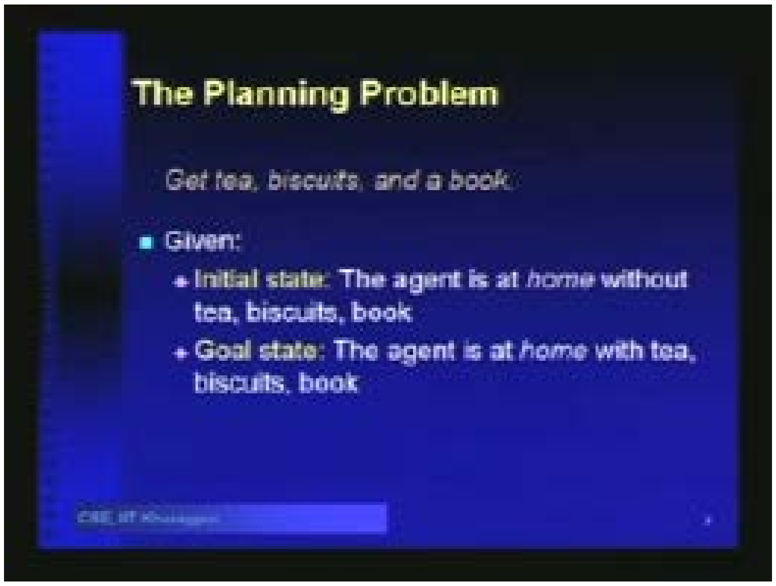
CSE, IIT Kharagpur

2

Then, we will look at 3 different syntaxes for representing planning problem. These are not as powerful as first order logic, but they are good enough to represent the kind of problems that we have in planning. And then, we will study some planning algorithms- specifically, partial order planning, which is quite a- I mean, which was developed several years ago, and graph plan and SAT plan, which are more recent technique, which will which came up in the 90s, okay? In this, let us start with an example of what is a planning problem.

the problem Suppose we are asked Suppose we have an agent and we want to tell the agent that get tea, biscuits and a book, right? And we will also define the set of actions that the agent can take; we will define what is the initial state of the agent, and the final state should be a state where we have tea, biscuits and a book. So, the agent has faced all 3 for us.

(Refer Slide Time: 04:03)



The Planning Problem

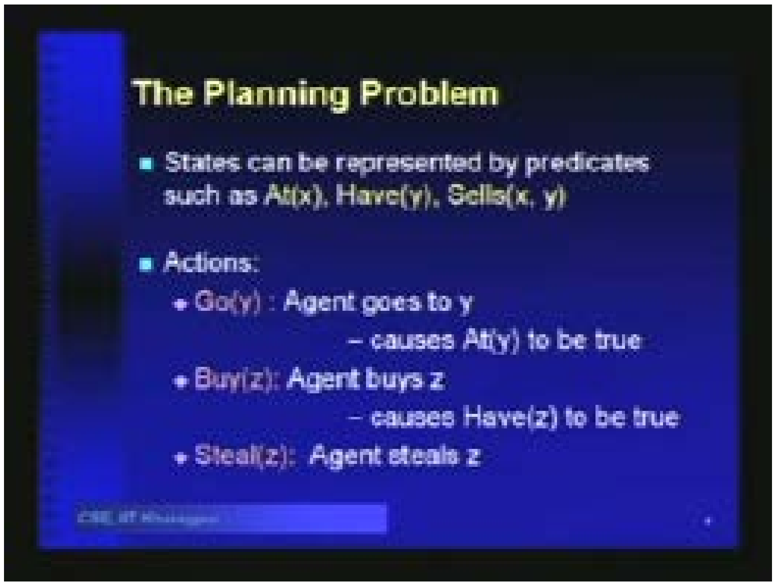
Get tea, biscuits, and a book

- Given:
  - Initial state: The agent is at *home* without tea, biscuits, book
  - Goal state: The agent is at *home* with tea, biscuits, book

CSE, IIT Kharagpur

So, the initial state is the agent is at home without tea, biscuits and book. Home here is symbolic. We can define anything to be home and the goal state is the agent is at home with tea, biscuits and book, right? States can be represented by predicates such as at x, indicating that the agent is at a specific position x.

(Refer Slide Time: 05:18)



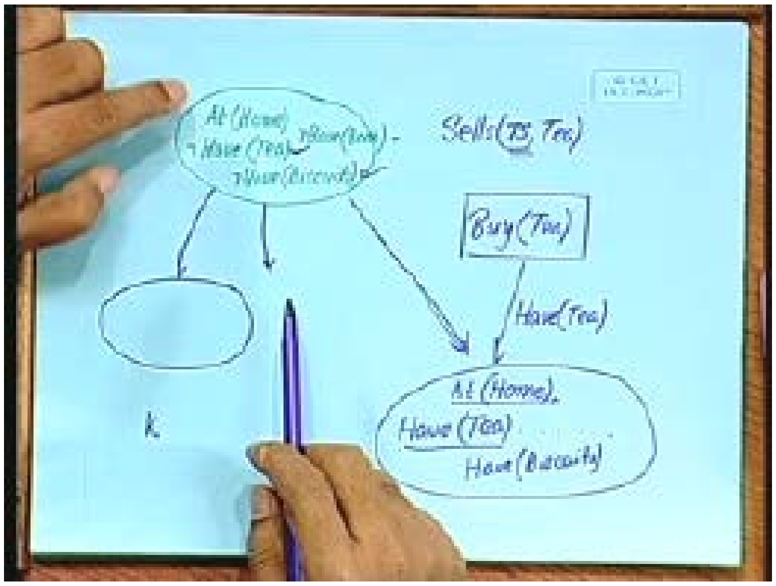
The Planning Problem

- States can be represented by predicates such as At(x), Have(y), Sells(x, y)
- Actions:
  - Go(y) : Agent goes to y
    - causes At(y) to be true
  - Buy(z): Agent buys z
    - causes Have(z) to be true
  - Steal(z): Agent steals z

CSE, IIT Kharagpur

have y means that the agent has y and sells xy means that x is some location which sells the item y. We can have actions like go y, the result of which will be that the agent goes to y and because it causes the agent to go to y, the net effect will be that the predicate at y will become true as a result of this action. Then, you can have buy z and the result of buy z is that we will have z; causes have z to be true, and we can also have steal z, which is another way of achieving the predicate have z. Now, what is the search problem here? The search problem is that we start from the initial state, which says that at home, not have tea, not have biscuits, not have books, right? And then, we have the set of actions.

(Refer Slide Time: 10:55)



If we will go back to state space search kind of framework, then our initial state iS1 where we have at home; then, we have not have book and not have biscuits. And the goal state, which we want to reach, is at home, and we have all these things- have tea, have all of the 3. And we have to find out a path from here to here. And what are our actions? The actions are our state transition operators. For example, if we say that buy tea- if we add this action buy tea, then the effect of this action will be that we will have have tea. But in order to have tea, we must go somewhere which sells tea.

Suppose we have another action; we have suppose we are given that sells, say, tea stall tea, right? Firstly, our action should be go to tea stall, right? So, each action, if I apply any action on a given state, it is going to take me to another state. And how are the states defined? States are defined in terms of the set of predicates that are true in the state. If I have, say, k propositional k propositions like this, then potentially, the number of states could be 2 to the power of k, and each action is taking me from 1 step to another. And our objective is to find out a sequence of actions which takes us from the start state to the goal state, so it is purely a search problem.

But there are certain differences with a generic search problem. For example, there could be some sub-goals for which the ordering in which in the the order in which we get then may not be important. For example, we have to get tea and we have to get biscuits. Both of these might be available in the tea stall. Whether we buy tea first or whether we buy biscuits first is immaterial. Both will be valid solutions. And because of such partial orders in a planning problem, the number of solutions which are valid can be quite large. So, it is true that I have to reach the the final state from the start state, but there can be many ways of achieving this, and those ways can differ in in terms of the sequence in which I achieve the sub-goals, like have tea, have book and have biscuit.

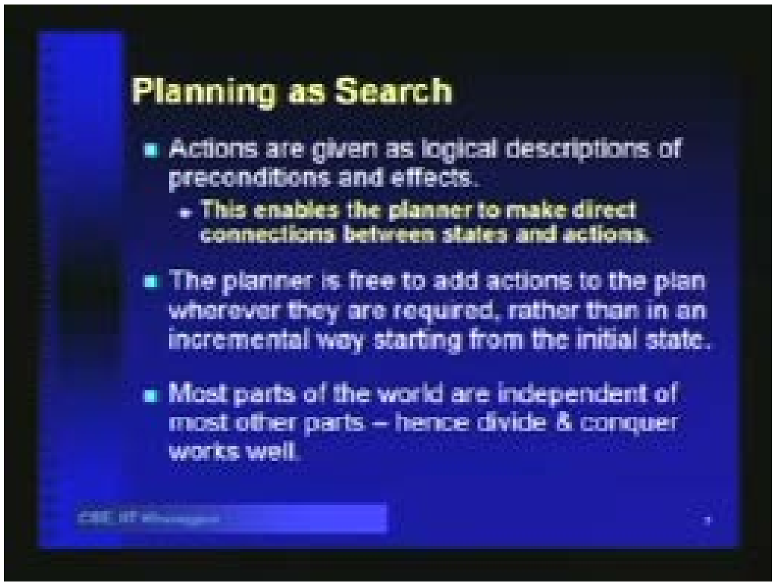
If we can have algorithms which exploit this fact, that there are many different ways of solving the problem and there is actually a partial order between the solutions, then we can have solutions or algorithms, which will be more efficient than a normal search algorithm in terms of complexity. The main differences between planning and search are that actions are given as logical descriptions of preconditions and effects. So, in a state- in a- typically, when we are talking about state transition operators, I mean; here, not all operators are applicable in all states, right?

There are pre-conditions of every action, right? This enables the planner to make direct connections between states and actions. For example, in the particular case that we were looking at, see, our goal was to achieve at home, have tea, then have biscuits, etc. Now, what we can do is, rather than starting from the start state and trying to enumerate all different actions that are possible here, which would be the way in which state space search would go, we can do a different thing.

We can see, that which is an action which has have tea, has its effect, right? And we see that okay, if you buy tea, then have tea is its effect. Also, if you steal tea, then also, have tea is its effect, right? So, it reduces- we can do go backward also, from the goal, and- beg a pardon? (Student speaking). Yes. We can go backwards to check for every sub-goal which are the actions, which has that- sub-goal has the effect, and search backwards among those possible ones. And typically, the number of the number of actions which will have your sub-goal as the effect is usually much lesser than the number of actions that would apply at a given state.

So, it is more goal directed search backwards, so that is another thing which makes planners different from generic search strategies. And the planner is free to add actions to the plan wherever they are required, rather than in an incremental way, starting from the initial state. This is what I was saying, that you do not necessarily have to start from the initial state and progressively try all possible actions. You can decide which ones will achieve the desired sub-goals for you.

(Refer Slide Time: 12:47)



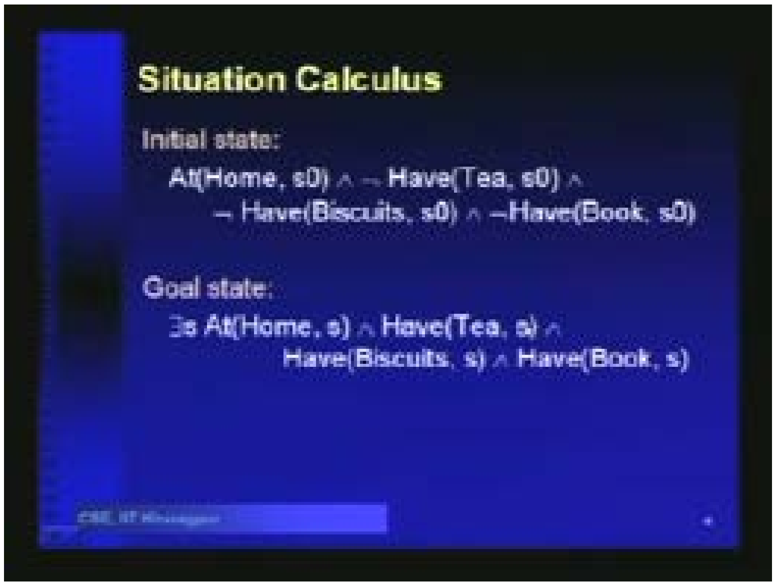
Planning as Search

- Actions are given as logical descriptions of preconditions and effects.
  - This enables the planner to make direct connections between states and actions.
- The planner is free to add actions to the plan wherever they are required, rather than in an incremental way starting from the initial state.
- Most parts of the world are independent of most other parts – hence divide & conquer works well.

The third most important thing, as I have already mentioned, is most parts of the world are independent of most other parts. Hence divide and conquer works very well. So, you will you may often find that the sub-goals that you have to achieve are independent. So, you can just solve them separately, but then, there can be conflicts between these. For example, if you go some place, then at home does not hold any more. If you wanted to use an action which requires at home, then you have to add it before you go to some other place, right? Even if they are not totally independent, you can still have some kind of topological ordering. We will come to the actual algorithms which will help you to decide, that which is the order in which the sub-goals have to be solved.

People initially attempted to represent planning problems through variants of predicate calculus, like first order logic and propositional calculus. And the problem was, okay, we will come to the problem later. Let us first see 1 such calculus, which is called situation calculus. Slides please. So, in situation calculus, we will write the initial state like this. With every predicate, we have an argument called the situation.

(Refer Slide Time: 15:17)



Situation Calculus

Initial state:

$At(Home, s0) \wedge \neg Have(Tea, s0) \wedge \neg Have(Biscuits, s0) \wedge \neg Have(Book, s0)$

Goal state:

$\exists s\ At(Home, s) \wedge Have(Tea, s) \wedge Have(Biscuits, s) \wedge Have(Book, s)$

So, the second argument that we have here is the situation. Now, you can think of these as situations in the sense that they are certain frames. It is apt- the situation is 0, which is the starting situation- we have at home, not have tea, not have biscuits, and not have book. Now, when you go somewhere, the situation will change, right? Any action that you take is going to change your situation. Next is the goal state- the goal state is: there exist some situation, where we are at home, we have tea, have biscuits and have book, all in the same situation. And just like in first order logic, this s and this s are bound. There is a binding from 1 to the other, right?

Then, operators. Now, this iS1 sample operator; for all a, and for all s, all actions, all situations have tea in the result of a s. Now, what is the result of a s? Result of a s is the situation which results from applying the action a, when you are in situation s. So, if I am in situation s, if I apply the action a, then, the new situation is called result of a s, right? So, this is the special predicate. We are saying that okay, if you take an action a and a situation s, then, if you have tea in the resulting action if and only if the action is buy tea and you are in the tea shop in the situation s; in the situation s, we are in the tea shop and we are buying tea.

Or the other scenario is that you already had tea in the situation s, which means that you achieved this much before and you did not drop the tea. Because there can be some actions which will have not have tea as the result, so we are saying that a is no such action. It is not the action drop tea which would have that. Now, see the problem here: so this is is this clear? How this rule is framed? Okay. Now, slides please. Yes, now, the problem that we have here is that, in order to have continuity, in the sense that from situation to situation, lot of things will not change, there will be lot of things which we have already achieved and which will simply be carried forward.

Now, here, we will have to write out those things, like we are having to say that otherwise you had tea and that having tea has carried over to this scenario. You also have to write rules, which says that if you had something and you did not use these actions, then you will still have those some things, and you have to enumerate this over all possible things, and that is quite extensive. And this came up as a problem called the frame problem. The frame problem really made the representation of planning problem very complex, because for everything that you ideally would like to say, that it just does not change, you have to explicitly specify in predicate logic to say that it will not change. Later on, people solved this frame problem by making the default assumption, that whatever changes, that I am not specifying, those predicates will just carry on their previous values.

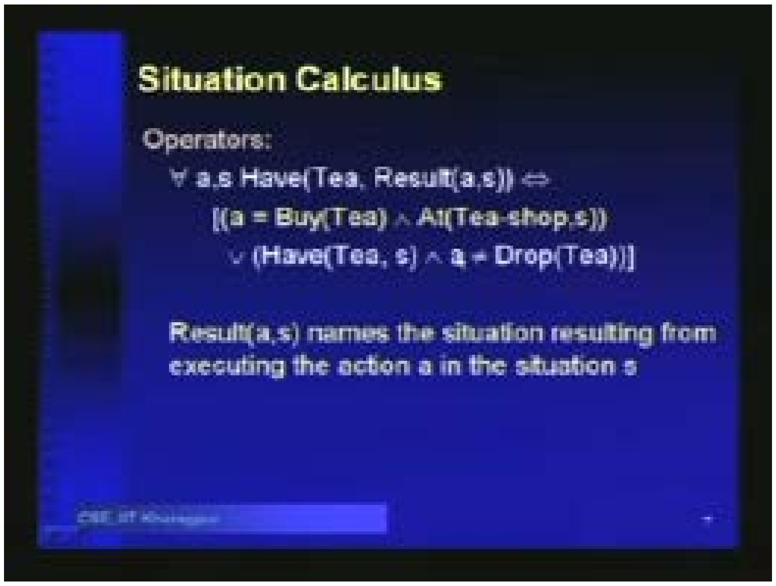
So, now, see, if you consider- all those different possible states that could have been there are now being narrowed down, because we are classifying them based on the changes that that are taking place. It is always if If in your world, very few things are changing at a time, then, it is always easier to model it as changes rather than anything else. That is how we do it for everything- like, even in game playing and things like that, you just see- mention the changes that you have from frame to frame. (Student speaking). Yes. (Student speaking). Drop tea is an action which causes you to drop the tea, so- (Student speaking). No, if you do not have an action- if you do not have any action which causes not have tea, whose effect is not have tea, then this part of the clause will become redundant- you do not need this.

But because we have an action which says drop tea, and the drop tea action has the effect not have tea, so therefore, we have to explicitly write this to indicate that if in the situation s, you have tea and your action is not equal to drop tea, then, in the in the result of a s, you will still have tea. (Student speaking). Beg your pardon? (Student speaking). No, no. It is not a question of- this is an action; you are given a set of actions. The planning problem is, you are given an initial state, you are given a set of actions. So, drop tea was given to you. It iS1 of the actions that was given to you, right?

See, when you are doing a planning, if you knew exactly what actions are necessary, then the planning problem is easy, then the planning problem is only to sequence those actions. But the the but in general, what you will have is, you will have a set of actions and you do not know which sub-set of these actions used in which possible sequences will achieve the goal state. So, there can be some other problem, where dropping the tea might be necessary for something else, right? So, drop tea is there, okay?

Because drop tea is there, when you try to model the predicate have tea in the result of a s- slides please- when you have to model this in this scenario, we have to what we are trying to do is, we are trying to say- when do we have tea in a particular situation, right? This is the rule which tells us that, and because drop tea is there aS1 of the actions given to us, we have to make sure that that is the that is not the 1 which is applied on the state where we have tea. Because then, in the resulting state, we will not have tea, right? That is why we are having to mention all of this.

(Refer Slide Time: 22:08)



Situation Calculus

Operators:

$\forall$ a,s Have(Tea, Result(a,s)) $\Leftrightarrow$
[(a = Buy(Tea) $\wedge$ At(Tea-shop,s))
$\vee$ (Have(Tea, s) $\wedge$ a $\neq$ Drop(Tea))]

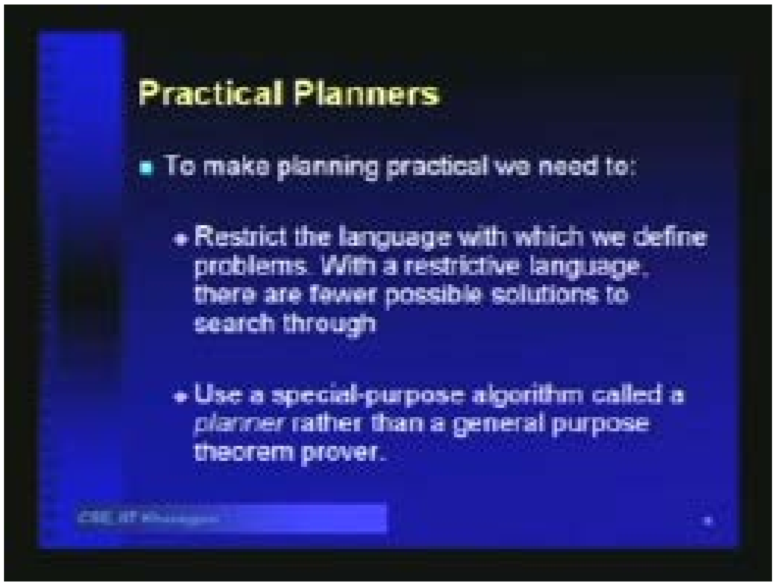
Result(a,s) names the situation resulting from executing the action a in the situation s

CSE, IIT Kharagpur

Now, and that is actually, it shows you what the frame problem is all about. It shows you that there are lot of things that we will have to explicitly specify, if we have to model them as first order logic rules. Later on, we will see some variants of these, where the situation will be replaced by time, so that will bring us to frames which are temporal in nature. Here, we have situations and these situations do not mention anything about time, though we normally tend to believe in terms of time. But here, it is not necessarily, that this situation resulting out of using the action a on s, is something which happens later in time. Your actions could be backwards in time also. I mean, logically, there is no difference, right?

But later on, we will talk about situations where this this situations are progressively going into the future, so those will describe model logics and temporal logics, okay? Now, to make planning practical, we need to do the following: we need to restrict a language with which we define problems with a restrictive language. There are fewer possible solutions to search through. We will see what kind of restrictions are there. And we will use special purpose algorithms, which are generally called planners, rather than a general purpose theorem prover. That a general purpose theorem prover will do backward chaining or forward chaining or resolution to try to prove this- so, it will just do search.

(Refer Slide Time: 24:01)



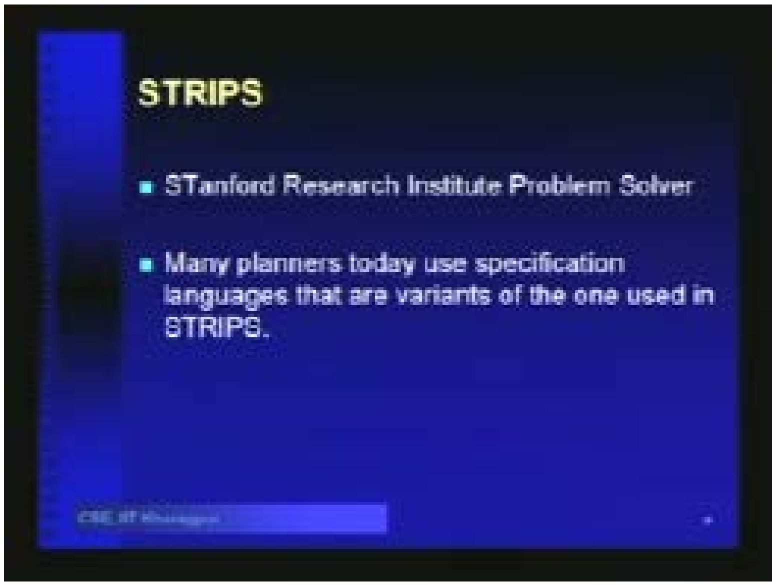
Practical Planners

- To make planning practical we need to:
  - Restrict the language with which we define problems. With a restrictive language, there are fewer possible solutions to search through
  - Use a special-purpose algorithm called a *planner* rather than a general purpose theorem prover.

CSE, IIT Kharagpur

The first language that we will study is called STRIPS- it stands for the STanford Research Institute Problem Solver. The Stanford research institute is a very renowned institute, which has been working, and theorem provers, etc. for several decades.

(Refer Slide Time: 24:19)



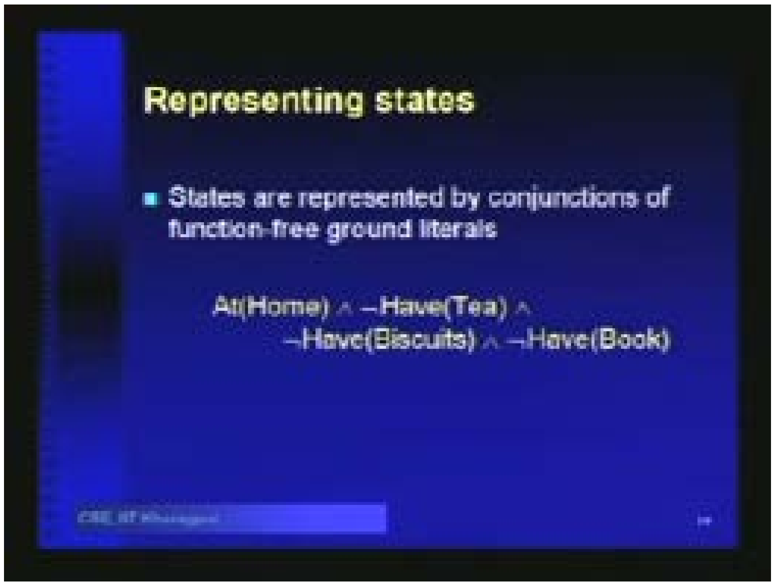
STRIPS

- STanford Research Institute Problem Solver
- Many planners today use specification languages that are variants of the one used in STRIPS.

So, they came up with this language. And this language also solves the frame problem and many planners today use specification languages that are variants of the 1 used in STRIPS. We will see what kinds of variations that have come, but let us first have a look at STRIPS. Now, note the definitions very carefully, because in STRIPS, we are actually restricting the kind of things that we can specify. In first order logic or situation calculus, you have more flexibility. Here, the flexibility is less, but we will see that it helps us in solving the problem better. So, states are represented by conjunctions of function free ground literals.

Now, what is the meaning of this? It says that in a state, you have to have conjunctions of literals. First of all, you cannot have disjunctions of literals. So, you cannot say that not have tea or not have biscuits is a state. not have tea or not have biscuits cannot be a state; it will actually be have to be split up into the states where you have not have tea, have biscuits, not have biscuits, have tea and not have tea and not of biscuits, right? You have to split it up into those 3 states.

(Refer Slide Time: 24:57)



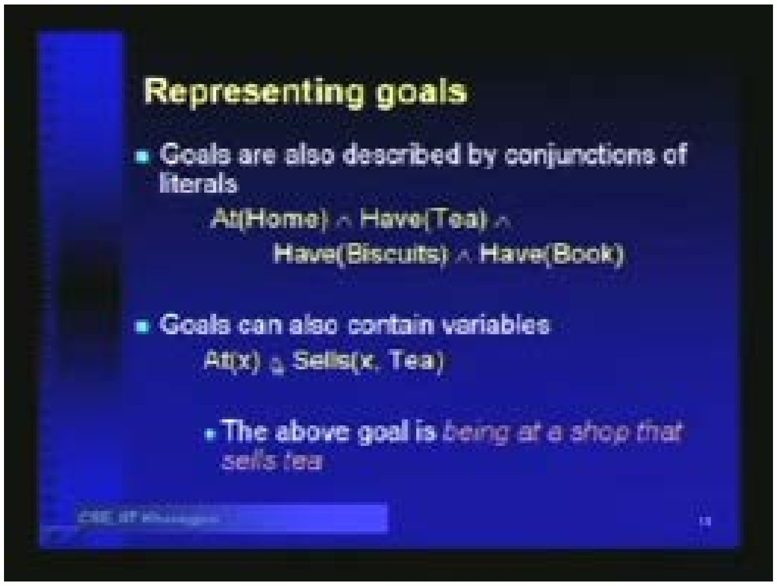
Representing states

- States are represented by conjunctions of function-free ground literals

$$At(Home) \wedge \neg Have(Tea) \wedge \neg Have(Biscuits) \wedge \neg Have(Book)$$

And then, your states are function free ground literals. So, see here- we have the all of these are ground literals, you cannot have functions like at of f of x- we cannot have that. In first order logic, we can have that; here, we cannot have that. See, the idea is that they will- people have designed these languages, so that it is possible to translate a specification into complete propositional logic. They will It should be possible to convert the language to the convert the specification into propositional logic.

(Refer Slide Time: 27:26)



Representing goals

- Goals are also described by conjunctions of literals
  $At(Home) \wedge Have(Tea) \wedge$
  $Have(Biscuits) \wedge Have(Book)$
- Goals can also contain variables
  $At(x) \wedge Sells(x, Tea)$
  - The above goal is *being at a shop that sells tea*

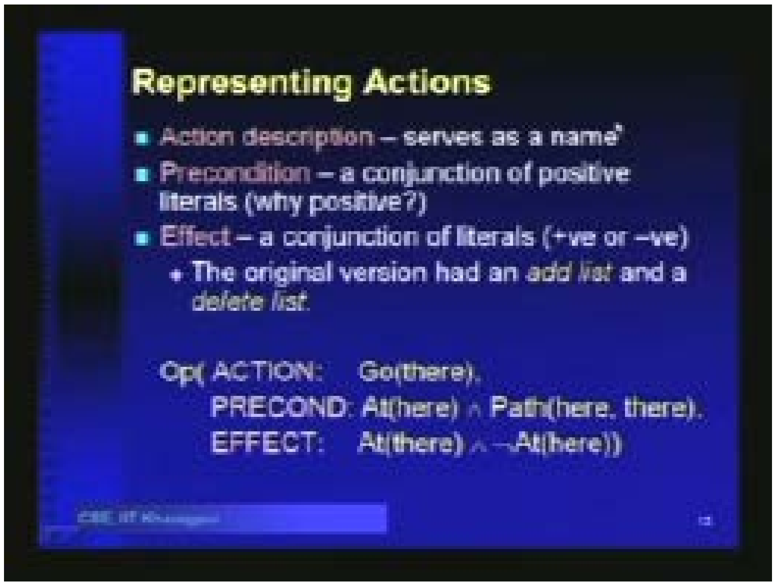
CSE, IIT Kharagpur
13

The number of propositions can be large, but they will convert it into the propositional logic and then use specific search algorithms for working on that. Representing goals- goals are also described by conjunctions of literals- at home, have tea, have biscuits, have book. Goals can also contain variables, like you can have at x and sells x tea, where x is variable, which is binding these 2 predicates. So, the above goal is being at a shop that sells tea, so this part is fine. Now, let us look at actions in actions.

We have an action description which serves as a name index. Then, we have a pre-condition, which is a conjunction of positive literals. Why positive literals? I will come to this. (Student speaking). This is a very important thing. You have to understand why only positive literals are given in the pre-condition okay? I will- just in a moment, I will come back to that. Let me complete the complete syntax of an action. The effect is a conjunction of literals- positive or negative.

The original version of STRIPS had to specify the effect aS2 different lists, called an add list- add list is the set of positive literals- and a delete list, which is a set of negative literals. For example, suppose we want to model the action go to there. There is a variable, the pre-condition is at here, and there is a path from here to there. The effect is at there and not at here, right? Now, let us see. Now, let us address that question, that why do we have positive literals in the pre-condition.

(Refer Slide Time: 29:37)



Representing Actions

- Action description – serves as a name*
- Precondition – a conjunction of positive literals (why positive?)
- Effect – a conjunction of literals (+ve or –ve)
  - The original version had an *add list* and a *delete list*.

```
Op( ACTION:    Go(there),
    PRECOND: At(here) ∧ Path(here, there),
    EFFECT:    At(there) ∧ ¬At(here))
```

CSE, IIT Kharagpur

12

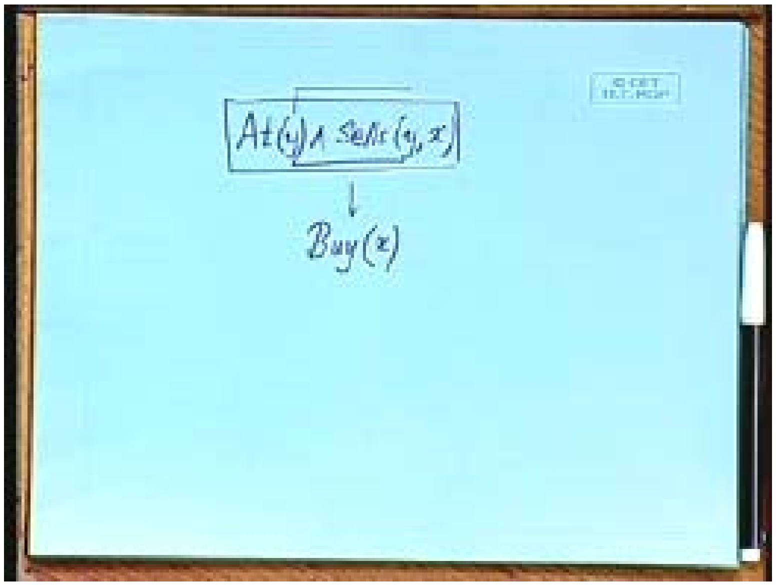
The reason is, recall that when we are talking about knowledge basis; what were we doing? We were deducing facts and putting it into the knowledge base. Any fact which is not yet in the knowledge base is unknown to us; we do not know whether it is true, we do not know whether it is false, right? You remember this- when we were doing deduction, any fact which is not yet there in the knowledge base- we cannot say that it is false, because later on, through some other sequence of deductions, that fact can get added into the knowledge base. Therefore, if you have a pre-condition which has a negative literal, then that the question will be when can we at all use that action.

Now, if you are maintaining only positive literals in your knowledge base, then you will never be able to add that, because in general, in first order logic, if something is not true, then you may not be able to establish at all whether it is true or not, because of the semi decidability problem. So, that action which has the negative as this thing- we do not know when to apply it, right? For positive literals, it is not the case, because if the positive literal is not there right now, we will not apply the action, right now. Eventually, if at some point of time, that literal comes into the knowledge base which is there in the pre-condition of my action, then, I can apply that action at that point of time. Yes or no?

But later on, there have been other planning systems, where we have the ability to have negation there as well, and they will work in a slightly different way; they will use something called negation as failure. And the assumption will be, that anything that you do not have in your knowledge base right now- so, if you cannot deduce something, then you will assume that the negation of that is true, right? That you can do, provided you are working in a propositional domain, because you do not have the decidability problem, right? How do we represent the plan? A plan will consist of a set of planned steps; each step iS1 of the operators of the problem.

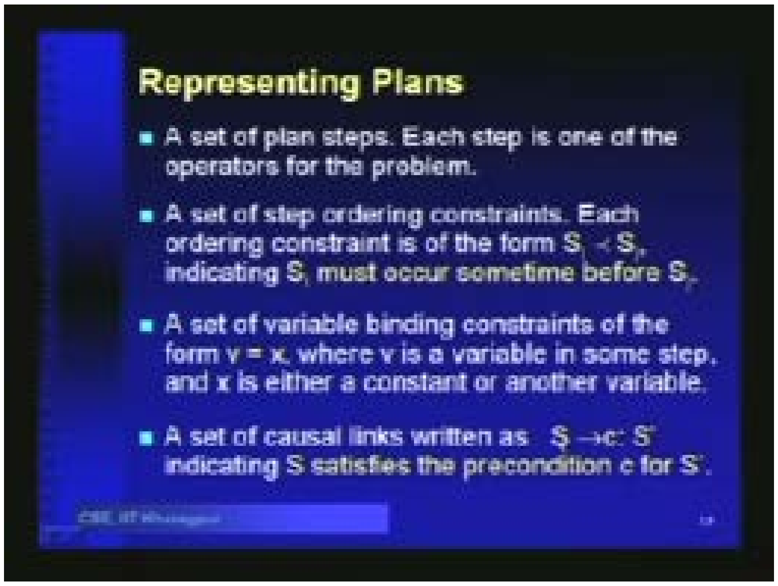
I will give an example to describe this in a moment. We have a set of step ordering constraints. Each ordering constraint is of the form Si precedes Sj, indicating that Si must occur sometime before Sj. We have a set of ordering constraints, okay? The reason that we will require this can be more than 1. 1 is that Si has something in its effect, which is a pre-condition to Sj. Also, we will have something, that if there is some other action which will cause the pre-condition of Sj to disappear, then, we have to do that either before Si or after Sj. That is another kind of constraint that will come up. I will demonstrate with the examples.

(Refer Slide Time: 34:57)



A set of variable binding constraints of the form v equal to x, where v is a variable in some step and x is either a constant or another variable. For example, suppose I want to model the buy action. So, I want to say buy x, and then I can say that the pre-condition of this is at y and sells y x, right? This is the pre-condition; if this pre condition can be met, then I can use the buy action. Now, here, there is this binding.

(Refer Slide Time: 35:33)



Representing Plans

- A set of plan steps. Each step is one of the operators for the problem.
- A set of step ordering constraints. Each ordering constraint is of the form $S_i \prec S_j$, indicating S_i must occur sometime before S_j.
- A set of variable binding constraints of the form v = x, where v is a variable in some step, and x is either a constant or another variable.
- A set of causal links written as $S \xrightarrow{c} S'$ indicating S satisfies the precondition c for S'.

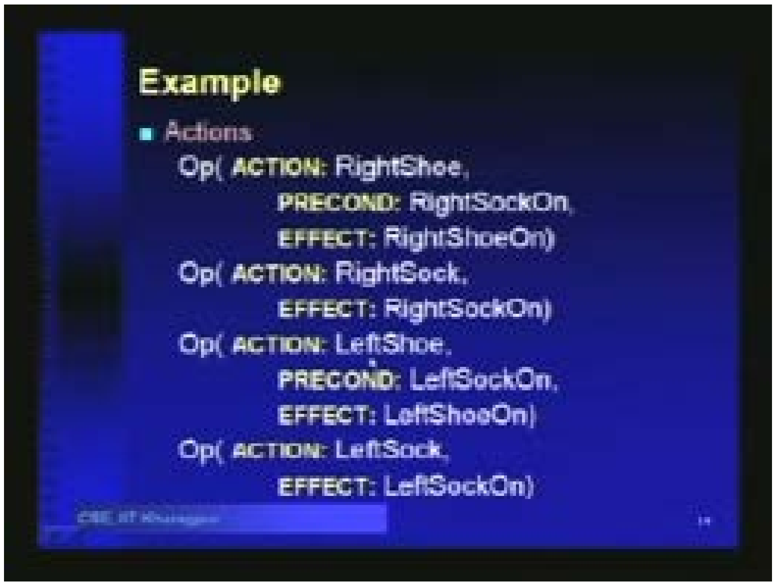
CSE, IIT Kharagpur

If you are at home, and say, x is tea and home does not sell tea, then, this home- the binding of home, if you are at home, then, this is also going to get bound to home. So, sells home tea will not be there in the knowledge base, so, we will not be able to apply this. But if you are at tea stall, and if you are given that sells tea stall tea, then he will be able to buy tea, right, so, that is what we are saying, by the variable binding constraints here, this is simple form of variable binding, but it can get more complex.

And then, we have a set of causal links written as this thing, indicating that S satisfies the pre-condition c for S dash, right? This ordering constraint is more general, this encompasses these as well as some other types, which I will describe later. The causal links are the ones which says that we must do this before this, because this action- this step- is going to create the pre-condition for S steps. Just take an example; we consider the action. What we are trying to do is, we are trying to put on our shoes; we are trying to make the agent put on shoes. So, the actions are to wear the right shoe, to wear the left shoe; but to but in order to wear the shoe, we must first wear the socks.

So, the actions are- slides please, slides please. We have right shoe, so this is the action of wearing the right shoe; pre-condition is right sock on, effect is right shoe on. Action for the right sock, for wearing the right sock is- there is no pre-condition and you just- on applying the right sock action, you just have the right sock on.

(Refer Slide Time: 36:55)



Example

- Actions

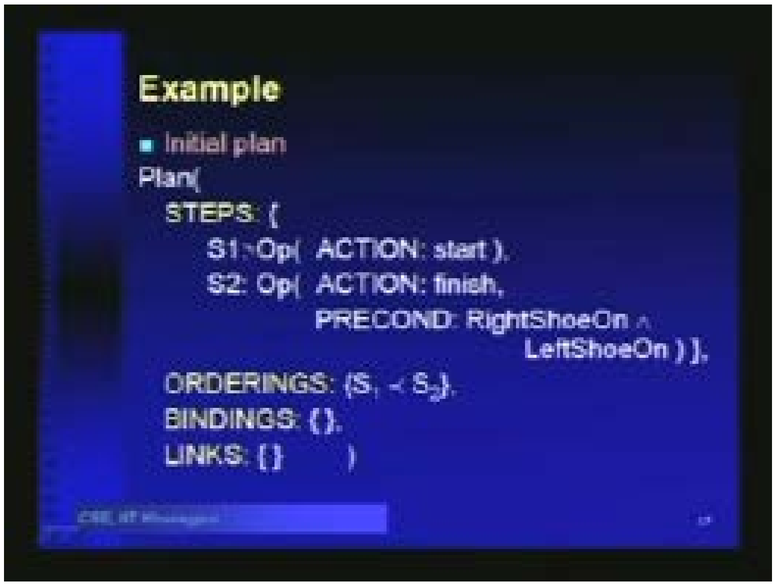
```
Op( ACTION: RightShoe,
        PRECOND: RightSockOn,
        EFFECT: RightShoeOn)
Op( ACTION: RightSock,
        EFFECT: RightSockOn)
Op( ACTION: LeftShoe,
        PRECOND: LeftSockOn,
        EFFECT: LeftShoeOn)
Op( ACTION: LeftSock,
        EFFECT: LeftSockOn)
```

CSE, IIT Kharagpur

14

Similarly, for the left shoe and this thing, right. Okay, now how do we represent the plan? The initial plan is going to look like this: so, firstly, we will have steps now. These steps in the plan are going to get augmented as the planner proceeds.

(Refer Slide Time: 37:25)



Example

- Initial plan

```
Plan(
  STEPS: {
      S1: Op( ACTION: start ),
      S2: Op( ACTION: finish,
                PRECOND: RightShoeOn ∧
                                LeftShoeOn ) },
  ORDERINGS: {S1 < S2},
  BINDINGS: {},
  LINKS: {}      )
```

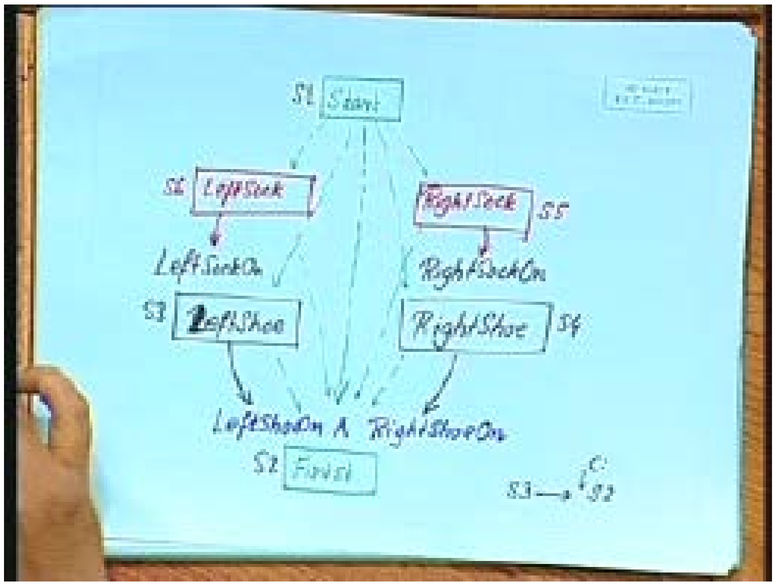
Initially, I have only 2 steps; the start step the start step; has the following feature- it has the action start, there is no pre-condition, there is no post condition. And the second step is the finish step, where the pre-condition is that the right shoe must be on and the left shoe must be on. When these 2 are on, then we can apply the finish step and the application of the finish step is the end of this thing. Orderings is that S1 must precede S2; so, always, the start must precede the finish. We do not have any bindings yet; we do not have any links yet, right?

Now- (Student speaking). S1 can have effects in certain cases; for example, in the previous example, when we are talking about getting tea, biscuits, etc., then, the effect of S1 will be not have tea, not have biscuits and not have books and at home. These are the effects of the start action in that example. In this particular example, the action start has no pre-condition, it does not have any effect. Typically, the start will never have a pre condition and the finish will never have a post condition, but the start can have effects and the finish will always have pre-condition, because otherwise, you can directly achieve the finish. Now, let us see- how do we start from this particular example and build up the plan?

Initially, what we have is, we are given 2 steps. So, we are given start. Can you see this color clearly? We are given the start, and the last step here is finish, and for finish, we have some pre-conditions. Let me write down these pre-conditions. We require left shoe on and we require and-, right? So, what we are going to do is, we first examine the set of the possible actions that we have, and let us go back to what actions we have. Slides please, yes. We have these actions, and we find that the that this action has the effect right shoe on and this action has the effect left shoe on.

What we will do is, we will start by applying- sorry, left shoe here. We add this step into the plan, so, this is the step that we are adding into the plan, and we can add this step also into the plan.This step will produce this as the effect, so this is a casual link this is a casual link that we have to put in, right? So, if you look at the syntax of the plan, then here, these orderings and the links, are the ones which are going to help us here.

(Refer Slide Time: 44:37)



In this case, we are going to put the link from left shoe to finish. So, we this was our S1; this was our S2, so, we will have S3 and S4, and we will put a link from S3 to S2, and the condition here- the condition c that we have here- is left shoe on. Similarly, we will have this causal link, and then, when we look at left shoe as a step, then, we will see that left shoe step has some pre-condition, and what is a pre-condition of left shoe? Left sock on. This has the pre-condition left sock on, and is left sock on in the in the plan already? It is not yet in the plan, therefore, it is a sub-goal which is not yet achieved.

Similarly, the sub-goal right sock on is not yet achieved. Again, we look at the set of steps that are possible, set of actions that are possible, and we find that there is the action right sock and left sock, which is going to give us the desired results. So, we will put these steps into the plan as a S5 and S6, we will call this left sock. This is going to achieve left sock on, this is going to achieve right sock on, and these do not have any pre-conditions. Now, if you look at that plan, what do we have in the plan? In the plan, we have: to start with- slides please- to start with, we had the ordering S1 to S2, and we have added some casual links.

So, if we if I draw it here, then, I have the this is our initial ordering, and we have these things. And also, we will follow 1 genetic ordering scheme. That is, whenever we add an action, we will always put an ordering from the start to that action, and from that action to the finish, because everything should have happen between the start and finish. So, we have all these other co other ordering links also, besides the causal links. Beg pardon? (Student speaking). These are ordering links. Now, ordering links are different from causal links, in the sense that in the ordering links, there is no direct cause effect relationship between the actions.

But we have to put them in a topological order, following the direction of these links, right? And why is that? Suppose I had an action and I wanted to put it before start. I should not be able to do that, regardless of whether start create something, which is used by this or not. So, these are therefore these are ordering links which are not causal related; they are not related to any cause effect relationship- it is just that we want this to happen between start and finish. We will see later on; there are other kinds of orderings also, which are non-causal in nature. And of course, when you have a casual link, that is definitely also a ordering link. Any casual link is also an ordering link, because if this if you want this effect from this action, then, obviously, we have to do a S6 followed by S3, right?

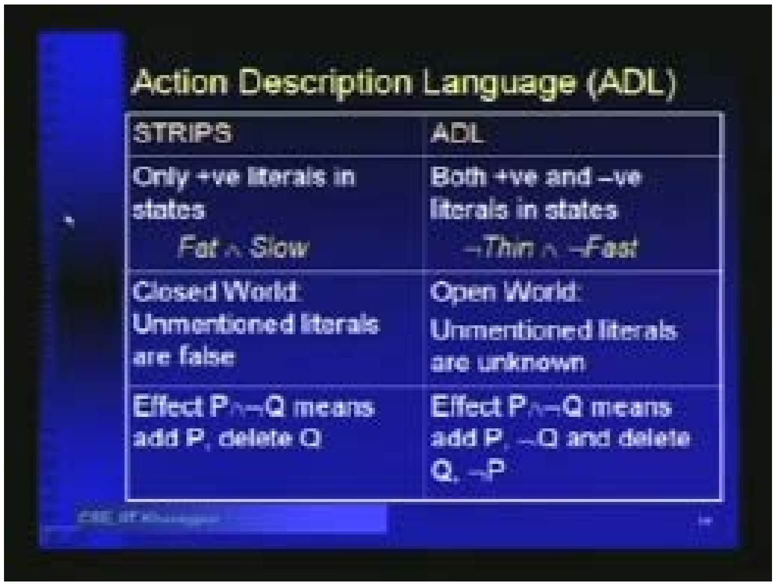
You could also do S6 followed by S5 followed by S3, because the ordering does not prevent you from doing that. You can do S1; S1 has to be done first. Then you can do S6, then you can do S5, then you can do S3; no problem, right? But suppose wearing the right sock is the effect of the taking off the left sock. If supposing, in some peculiar scenario, wearing the right sock means the left sock comes off, then we cannot do S6, followed by S5, followed by S3, and the reason is that, if you did S6 followed by S5, then the pre-condition of S3 will disappear.

In that case, the sequence will have to be S6, followed by S3, followed by S5, or S5 followed, by S6, followed by S3. So, this S5 should be either before S6 or after S3, right? So, those will also add additional orderings, which we will discuss slowly. There are some other constraints of that. But what I want to you to gather from this is that, this initial plan will be like this, and the additions that will take place into the plan are more steps, like S1, S2, S3, S4, right? These steps will be actions taken from the pool of actions that are given to us, and after adding each action, we may have to add some bindings and some links and some orderings. And when do we stop? (Student speaking). Right.

When we have when we will find that each of the pre-conditions of each step has been achieved by some other step in the plan; that is take place when we have the complete plan, right? But when we have the complete plan, what is the solution to the planning problem? Any topologically ordered sequence of actions is a solution to the planning problem, right? So, the objective was the of the planner is to get the plan in this partial order form, and once we have that partial order form, then we will topologically sort the actions, sort the steps, and that sequence of application of those steps will achieve the desired result. Now, quickly, we will have we will check that that the difference between STRIPS and a more recent language called the action description language.

This table just shows the differences between the 2 languages, so, here, in STRIPS, recall that we have only positive literals in steps. And why do we have that? Because remember, that in our knowledge base, we are always storing what is true. We have no information about what is false, but in action description language, we are allowed to have both positive and negative literals and states, which means that inside your knowledge base, you have to main maintain both which is true which is false, and which is unknown.

(Refer Slide Time: 49:35)



Action Description Language (ADL)

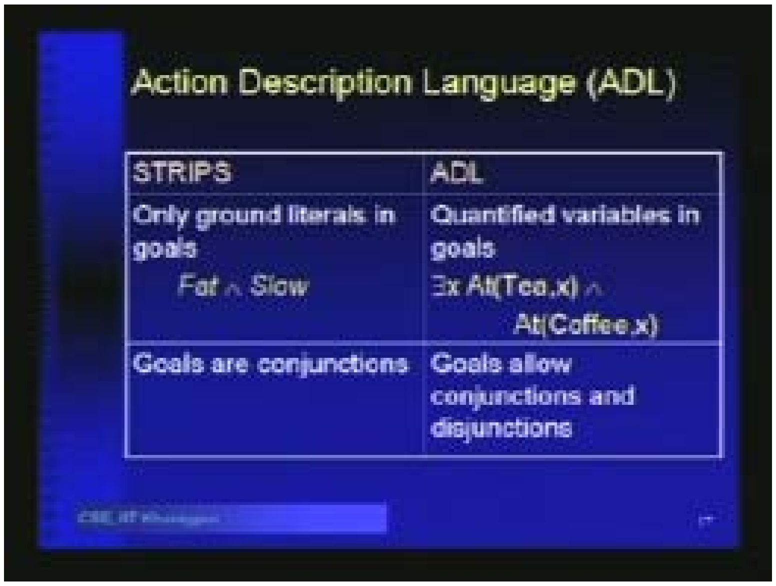
STRIPS	ADL
Only +ve literals in states $Fat \wedge Slow$	Both +ve and –ve literals in states $\neg Thin \wedge \neg Fast$
Closed World: Unmentioned literals are false	Open World: Unmentioned literals are unknown
Effect $P \wedge \neg Q$ means add P, delete Q	Effect $P \wedge \neg Q$ means add P, ¬Q and delete Q, ¬P

And something which is unknown can become true later on, can become false later on, and we have to avoid the problem that something which is now false, later becomes true, or something which is now true, later becomes false. That should not happen, right? That has to be taken care of by any planner, which uses adl as the input. In STRIPS- we have the closed world assumption. Closed world assumption means that whichever literals are not mentioned are assumed to be false. In the sense that anything which is not given right now, if that is there in the peak condition of some action, we cannot apply that action, because as of now, we will treat it as false.

adl makes an open world assumption. It says that mention literals are unknown. In STRIPS, the effect of p and not q- suppose if your effect has p and not q, that means from the knowledge base, you will add p and you will delete q, to get the new situation. From the previous situation, you will add p and you will delete q. Suppose you had had milk and dropped milk, then, the effect of dropping milk will cause have milk to disappear and not have milk to appear. Effect of p and not q here in adl means, add not q and delete q and delete not p.

Because recall, that in this framework, we are storing both positive and negative literals, so we have to do both of them. Next, only ground literals are allowed in goals, in STRIPS, but in adl, we can have- okay, in my version of, strips previously I mentioned that you can have variables also, right? But in the original version of strips, it was interlocked, but adl gives you quantified variables in goals.

(Refer Slide Time: 52:42)



Action Description Language (ADL)

STRIPS	ADL
Only ground literals in goals *Fat* ∧ *Slow*	Quantified variables in goals $\exists x$ At(Tea,x) ∧ At(Coffee,x)
Goals are conjunctions	Goals allow conjunctions and disjunctions

So, you can say there exist x at tea x and at coffee x, so you can use quantifiers also on the variables like this. This is more like first order logic. So, you are trying to find out that whether there is a stall which will sell both tea and coffee. I think I should have written the other way now- at x tea and at x coffee. Anyway, goals are conjunctions but in adl, goals allow conjunctions as well as disjunctions. So, today, we will conclude the lecture at this point of time. In the next class, I will start with the partial order planning algorithm. I have briefly outlined the algorithms with these examples, but we will see the algorithm in details. And then, we will go into the other planning algorithms- more recent algorithms like graph plan and SAT plan.