Fundamentals of Artificial Intelligence

Prof. Shyamanta M Hazarika
Department of Mechanical Engineering
Indian Institute of Technology Guwahati

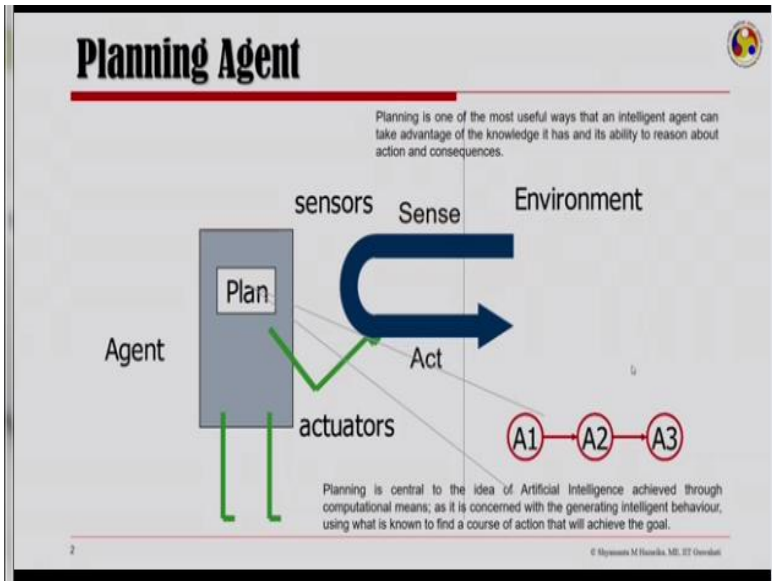
Lecture 21
Introduction to Planning

Welcome to fundamentals of artificial intelligence, in this module we would look at planning, within problem solving state phase search. We saw that an agent can consider the consequence of a sequence of actions even before acting to arrive at the best first move. Knowledge representation and reasoning on the other hand allows explicit representation of a state as well as actions and ensures that an agent is able to succeed in a complex and in accessible environment.

That is too difficult for the problem solving agent, we combine these 2 ideas to arrive at the planning agent. Planning in it is most abstract form can be seen as problem solving, planning is problem solving with the agent using belief about it is action and consequences of the actions to get to a solution by searching through a abstract space of plans. Planning algorithms are special purpose theorem provers and they are able to arrive at a plan by using the actions as certain axioms.

In this lecture we would look at first the planning problem and then we would look at a variant of first order logic called situation calculus, which can use an inference mechanism to arrive at a plan. Thereafter we would introduce STRIPS of formalism in which planning problems can be expressed and a plan can be arrived at.

(Refer Slide Time: 03:19)



Planning Agent

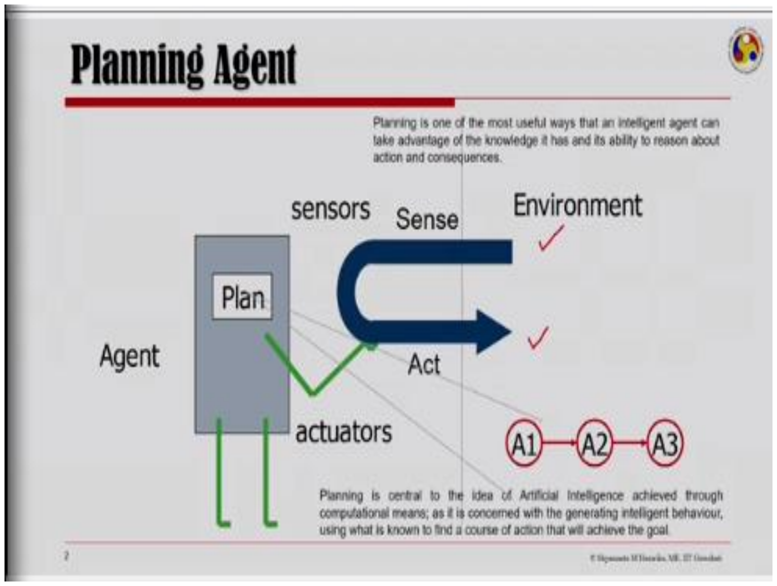
Planning is one of the most useful ways that an intelligent agent can take advantage of the knowledge it has and its ability to reason about action and consequences.

Planning is central to the idea of Artificial Intelligence achieved through computational means; as it is concerned with the generating intelligent behaviour, using what is known to find a course of action that will achieve the goal.

But first let us look at what is a planning agent, in order to understand a planning agent we look at what is called the sense, plan and act sequence. So planning is one of the most useful ways that an intelligent agent can take advantage of the knowledge it has and it is ability to reason about action and consequences. Here we have shown an agent which has both sensors and actuators, so the senses sense the environment and through the actuators it acts.

It decides about the plan through a sense, plan, act cycle, actions can be seen as a sequence of certain activities it does on the environment. Now planning is central to the idea of artificial intelligence wherever AI is achieved through computational means. Because it is concerned with generating intelligent behavior using what is known to find a course of action that will achieve the goal.

(Refer Slide Time: 04:50)



Planning Agent

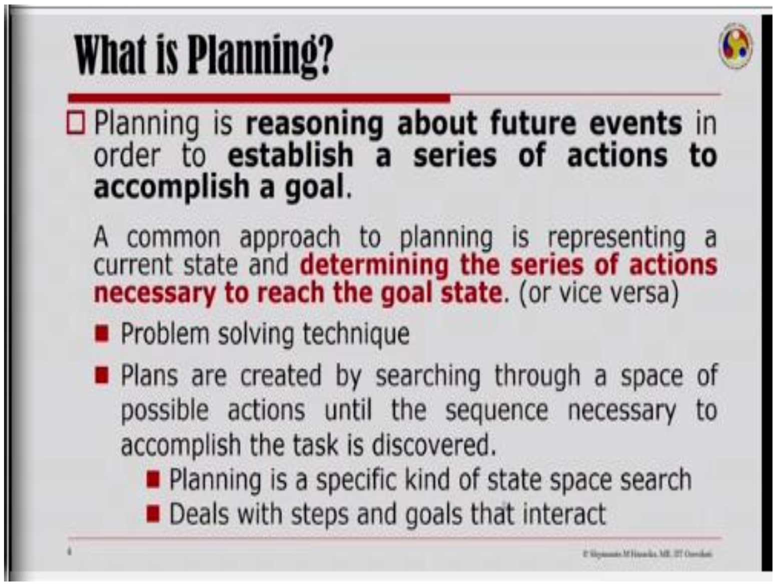
Planning is one of the most useful ways that an intelligent agent can take advantage of the knowledge it has and its ability to reason about action and consequences.

Planning is central to the idea of Artificial Intelligence achieved through computational means; as it is concerned with the generating intelligent behaviour, using what is known to find a course of action that will achieve the goal.

Now the planning agent is very similar to a problem solving agent, for all it needs to do is achieve it is goal through a plan which could be looked at as a problem to arrive at the goal. Now in spite of the similarity the planning agent and the problem solving agent has a huge difference, this difference comes from the very fact that when I am talking of a planning agent I have to somehow think of representing it is goals, states and actions and use explicit logical representation.

Now these use of explicit logical representation enables the planner to direct it is deliberations much more sensibly. Before we proceed further let us try to understand what we mean by planning, when we are talking of planning, planning is reasoning about future events. So as I can establish a series of actions to accomplish a goal.

(Refer Slide Time: 06:25)



What is Planning?

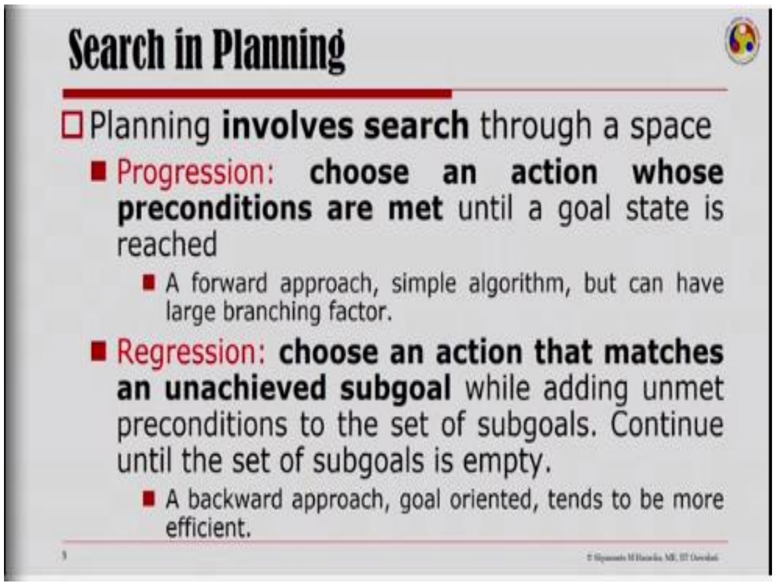
- Planning is **reasoning about future events** in order to **establish a series of actions to accomplish a goal**.

  A common approach to planning is representing a current state and **determining the series of actions necessary to reach the goal state**. (or vice versa)
  - Problem solving technique
  - Plans are created by searching through a space of possible actions until the sequence necessary to accomplish the task is discovered.
    - Planning is a specific kind of state space search
    - Deals with steps and goals that interact

A common approach to planning is about representing the current state and then determining the series of actions necessary to reach the goal state, therefore it could be thought of as a problem solving technique. Plans, on the other hand are created by searching through a space of possible actions until I arrive at a sequence necessary to accomplish the task. And therefore planning could be thought of as a specific kind of state space search.

It deals with steps and goals that it need to interact, coming back to the planning agent we could now have a better feel of how it acts, it senses the environment and does decides which state it is. Under that state it needs to know which actions it need to perform and move forward towards the goal and therefore it interacts with the environment via acting,

(Refer Slide Time: 07:46)



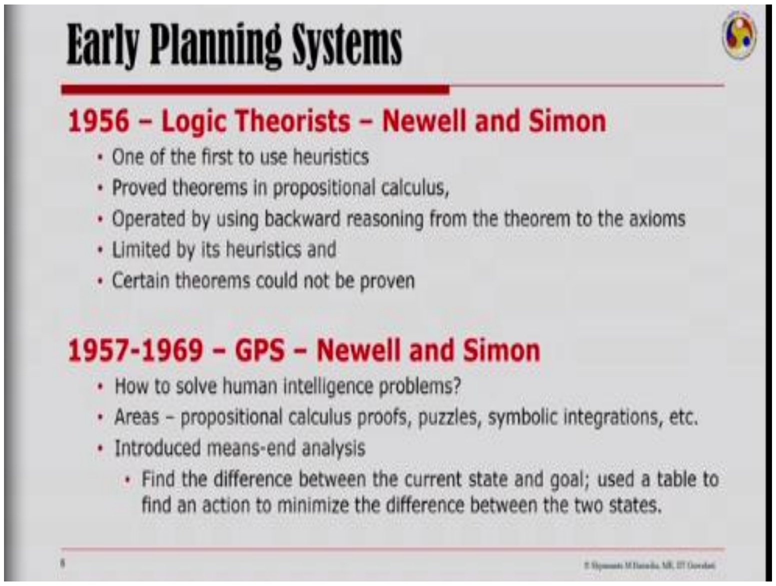
Search in Planning

- Planning **involves search** through a space
  - Progression: **choose an action whose preconditions are met** until a goal state is reached
    - A forward approach, simple algorithm, but can have large branching factor.
  - Regression: **choose an action that matches an unachieved subgoal** while adding unmet preconditions to the set of subgoals. Continue until the set of subgoals is empty.
    - A backward approach, goal oriented, tends to be more efficient.

Planning, as we can make out from this short discussion involves search through a space, and the search could be either of 2 ways you choose an action whose preconditions are met until a goal state is reached. Such a planning is saved to be progression planning, this is a forward approach, whereas we could have regression planning in which case we choose an action that matches an unachieved sub goal while adding the unmet preconditions to the set of sub goals.

And this we continue until the set of sub goals is empty, as you can see regression planning is a backward approach which is goal oriented and tends to be more efficient.

(Refer Slide Time: 08:45)



Early Planning Systems

1956 – Logic Theorists – Newell and Simon

- One of the first to use heuristics
- Proved theorems in propositional calculus,
- Operated by using backward reasoning from the theorem to the axioms
- Limited by its heuristics and
- Certain theorems could not be proven

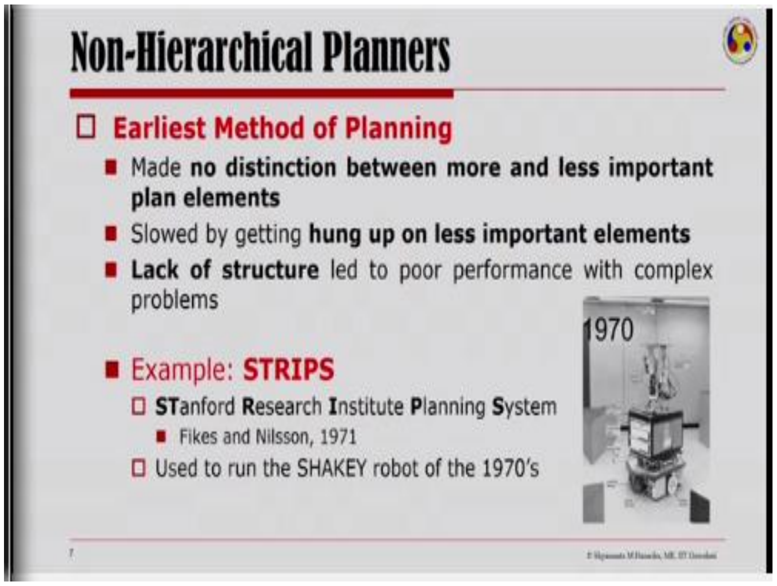
1957-1969 – GPS – Newell and Simon

- How to solve human intelligence problems?
- Areas – propositional calculus proofs, puzzles, symbolic integrations, etc.
- Introduced means-end analysis
  - Find the difference between the current state and goal; used a table to find an action to minimize the difference between the two states.

Now just a couple of mentions on the early planning systems, the first mention should be made of the logic theory test by Newell and Simon in 1956. These were one of the first to use problem domain information and they could prove theorems in propositional calculus. The logic theorists operated by using backward reasoning from the theorem to the exams. Then around 57 we had the general problem solver by a Newell and Simon.

These looked at questions of how to solve human intelligent problems, looked at propositional calculus proofs, puzzles, symbolic integrations etc. The general problem solver actually introduced means end analysis, find the difference between the current state and the goal, used a table to find an action to minimize the difference between the 2 states and thus come up with a plan.

(Refer Slide Time: 09:57)

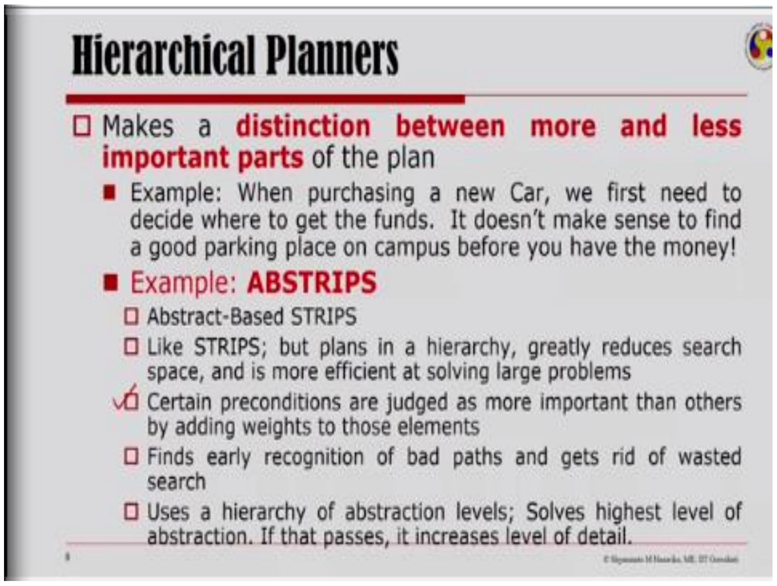


Non-Hierarchical Planners

- Earliest Method of Planning
  - Made **no distinction between more and less important plan elements**
  - Slowed by getting **hung up on less important elements**
  - **Lack of structure** led to poor performance with complex problems
  - Example: **STRIPS**
    - **ST**anford **R**esearch **I**nstitute **P**lanning **S**ystem
      - Fikes and Nilsson, 1971
    - Used to run the SHAKEY robot of the 1970's

The earliest methods of planning made no distinction between more or less important plan elements and the lack of structure led to poor performance. One of the earliest non hierarchical planners was STRIPS, the Stanford Research Institute Planning System. We will look at STRIPS in more detail during the course of this lecture.

(Refer Slide Time: 10:27)



Hierarchical Planners

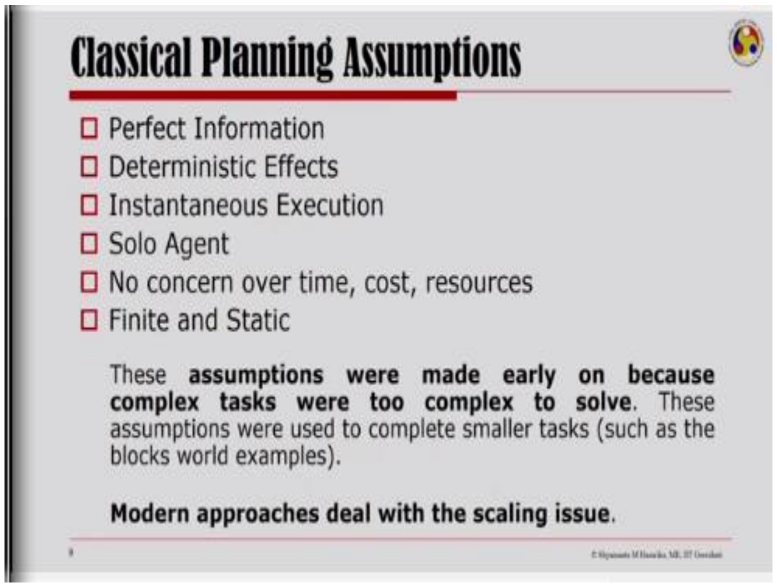
- Makes a **distinction between more and less important parts** of the plan
  - Example: When purchasing a new Car, we first need to decide where to get the funds. It doesn't make sense to find a good parking place on campus before you have the money!
  - Example: **ABSTRIPS**
    - Abstract-Based STRIPS
    - Like STRIPS; but plans in a hierarchy, greatly reduces search space, and is more efficient at solving large problems
    - ✓ Certain preconditions are judged as more important than others by adding weights to those elements
    - Finds early recognition of bad paths and gets rid of wasted search
    - Uses a hierarchy of abstraction levels; Solves highest level of abstraction. If that passes, it increases level of detail.

Hierarchical planners on the other hand made a distinction between more or less important parts of the plan. Now let us take a minute to understand what we mean by important and non important parts of the plan. For example you are thinking of purchasing a new car under a such a scenario one needs to decide where to get the funds. While planning it does not make much sense to find a good parking place on campus for your car before you have the money to buy it.

So non hierarchical planners could not make that distinction, for them every element of the plan was equally important. Hierarchical planners on the other hand made a distinction between more and less important parts of the plan. And one of the first hierarchical planners is the abstract base STRIPS, it is like STRIPS but it plans in hierarchy. So that greatly reduces search space and is much more efficient at solving large problems.

Now what it means by able to distinguish between more or less important parts of the plan is that certain preconditions are just as more important than others by adding weights to those elements. So early recognition of bad paths and getting rid of wasted search is a criterion of such planners. Use a hierarchy of abstraction levels, that is such planners solve the highest level of abstraction once that passes it increases level of detail.

(Refer Slide Time: 12:19)


Classical Planning Assumptions

- Perfect Information
- Deterministic Effects
- Instantaneous Execution
- Solo Agent
- No concern over time, cost, resources
- Finite and Static

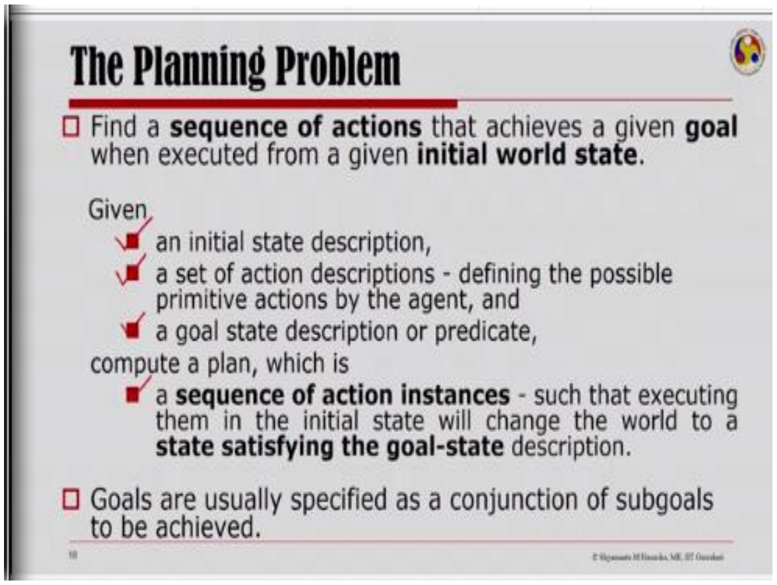
These **assumptions were made early on because complex tasks were too complex to solve**. These assumptions were used to complete smaller tasks (such as the blocks world examples).

Modern approaches deal with the scaling issue.

There are certain assumptions under classical planning, and here is their list we assume perfect information deterministic effects is what is assumed under classical planning. And the classical planning any action is assume to be instantaneous, that is execution is instantaneous for a given action. Such planners are single agent and these agents not concerned about time, cost or resources and the environment is finite and static.

Now these assumptions look very strong and one needs to remember that these were made early on because complex tasks were too complex to solve. So these assumptions were used to complete smaller tasks such as the blocks world examples. Modern approaches to planning deal with scaling issues which we will cover in the next lecture.

(Refer Slide Time: 13:24)



The Planning Problem

- Find a **sequence of actions** that achieves a given **goal** when executed from a given **initial world state**.

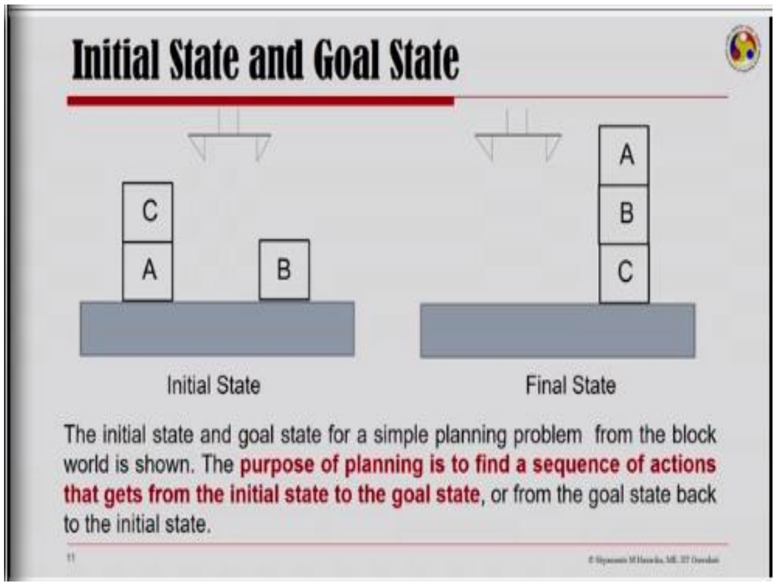
  Given
  - an initial state description,
  - a set of action descriptions - defining the possible primitive actions by the agent, and
  - a goal state description or predicate,

  compute a plan, which is
  - a **sequence of action instances** - such that executing them in the initial state will change the world to a **state satisfying the goal-state** description.
- Goals are usually specified as a conjunction of subgoals to be achieved.

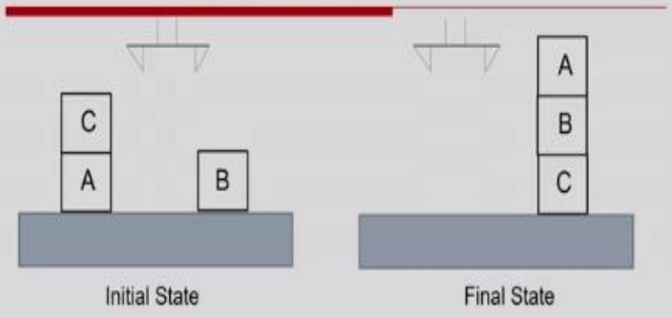
Now let us try to understand and define the planning problem, the planning problem as we have been mentioning is about finding a sequence of actions that achieves a given goal, when executed from a given initial world. So what we have is an initial state description thereafter we know the actions possible, a set of action descriptions which define the possible primitive actions by the agent and a goal state description.

And when we say we need to compute a plan, we mean that it must come up with a sequence of action instances. Such that executing them in the initial state will change the world to a state satisfying the goal state description. Now the goals are usually specified as a conjunction of sub goals to be achieved.

(Refer Slide Time: 14:26)



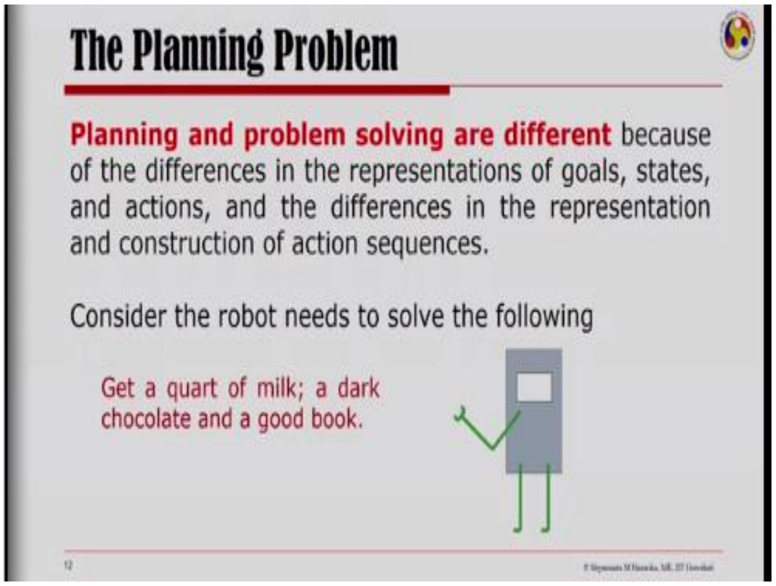
Initial State and Goal State



The initial state and goal state for a simple planning problem from the block world is shown. The **purpose of planning is to find a sequence of actions that gets from the initial state to the goal state**, or from the goal state back to the initial state.

So here on your screen is the initial state of a blocks world problem, now the purpose of planning is to find a sequence of action that gets me from the initial state to the goal state that is shown on your right. So the idea is to pick up block C, put it on the table, pick up B put it on C, pick up a and then put it on B to arrive at the final state which is A, B, C stacked one on top of the other.

(Refer Slide Time: 15:07)

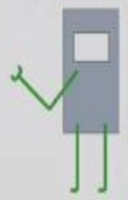


The Planning Problem

Planning and problem solving are different because of the differences in the representations of goals, states, and actions, and the differences in the representation and construction of action sequences.

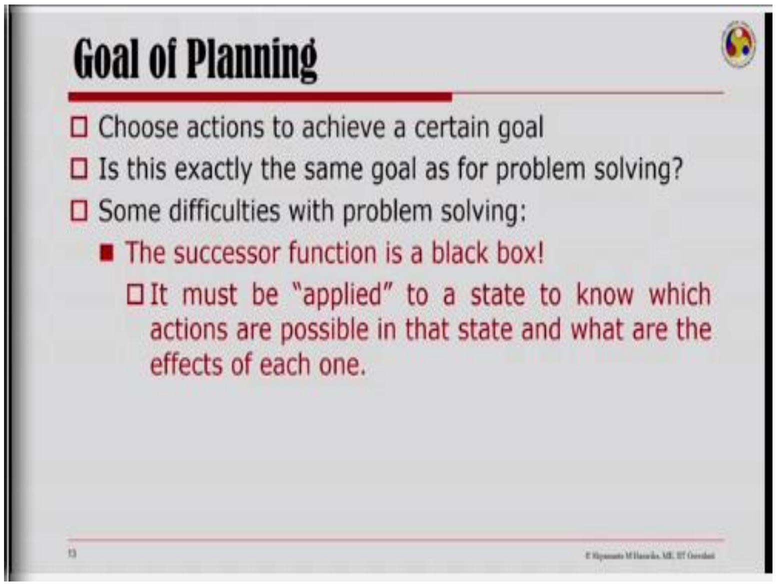
Consider the robot needs to solve the following

Get a quart of milk; a dark chocolate and a good book.



Now even if planning at it is most abstract level can be seen as problem solving, one needs to understand that planning and problem solving are different. And this difference comes because of the representation of the goals, states and actions. And of course the differences in the representation and construction of the action sequences. Now consider the robot shown on your screen on the right and imagine that it needs to solve the following problem, it is asked to get a quart of milk, a dark chocolate and a good book.

(Refer Slide Time: 15:57)

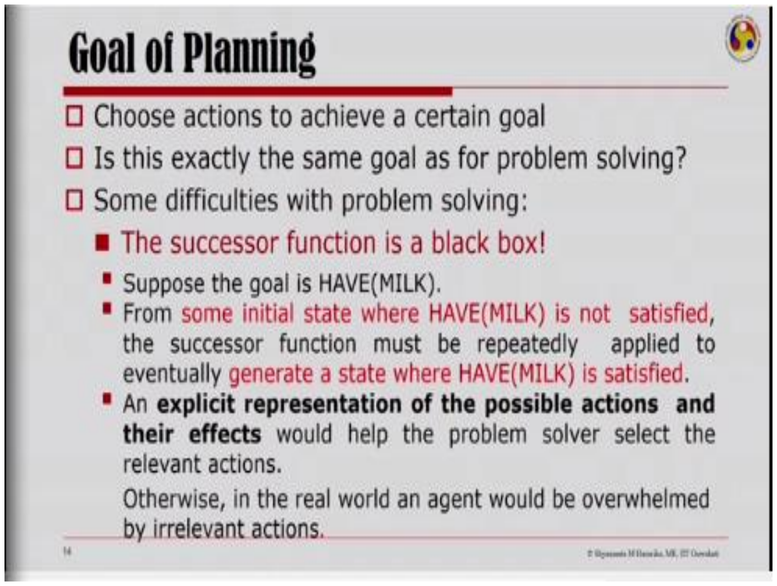


Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The successor function is a black box!
    - It must be "applied" to a state to know which actions are possible in that state and what are the effects of each one.

Now the goal of planning for that agent is to choose actions to achieve a certain goal, is this exactly the same goal as for problem solving. Now we need to understand that there are some difficulties with problem solving. The successor function is a black box, it must be applied to a state to know which actions are possible in that state and what are the effects of each one.

(Refer Slide Time: 16:26)



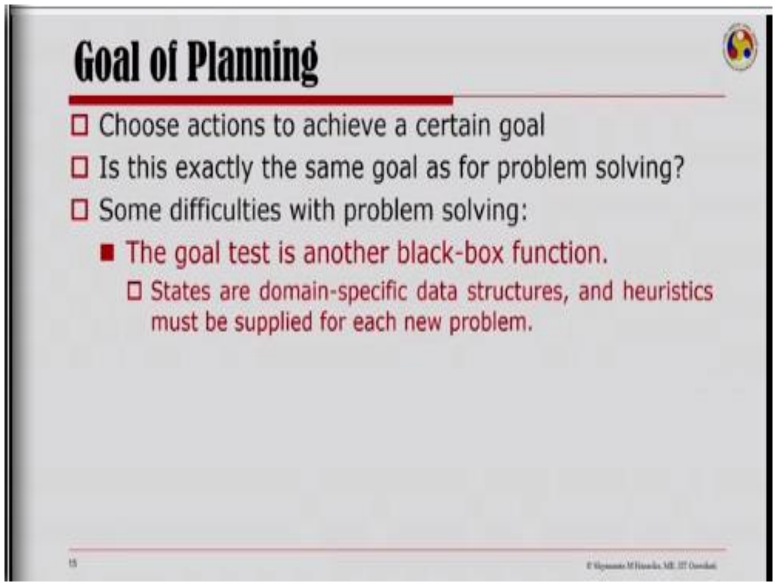
Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The successor function is a black box!
    - Suppose the goal is HAVE(MILK).
    - From some initial state where HAVE(MILK) is not satisfied, the successor function must be repeatedly applied to eventually generate a state where HAVE(MILK) is satisfied.
    - An **explicit representation of the possible actions and their effects** would help the problem solver select the relevant actions.

      Otherwise, in the real world an agent would be overwhelmed by irrelevant actions.

Suppose now on the other hand for the robot here the goal is to have milk. For some initial state where have milk is not satisfied, the successor function must be repeatedly applied to eventually generate a state where have milk is satisfied. An explicit representation of the possible actions and their effects would help the problem solver select the relevant action and this is what is done in planning and missing in problem solving. Otherwise if that is not done in the real world an agent would be overwhelmed by irrelevant actions.

(Refer Slide Time: 17:17)

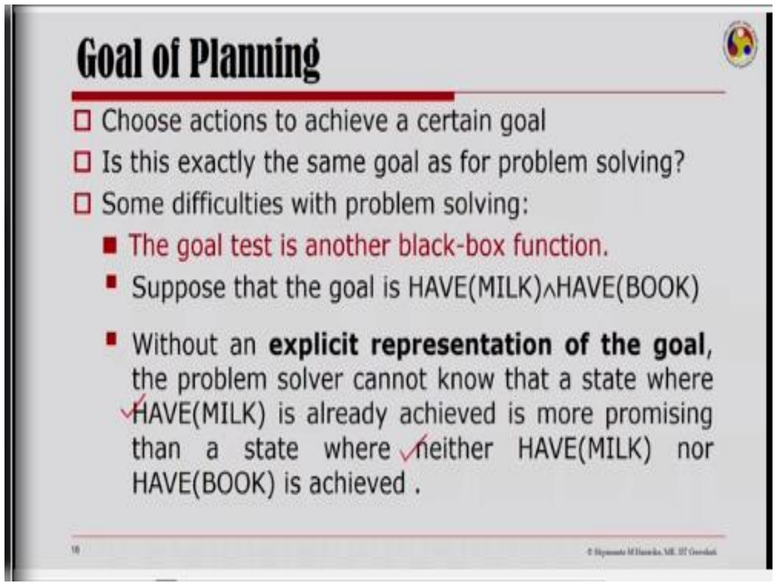


Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal test is another black-box function.
    - States are domain-specific data structures, and heuristics must be supplied for each new problem.

Now on the other hand if you look at the goal test, the goal test is another black box function. States are domain specific data structures and heuristic must be supplied for each new problem.

(Refer Slide Time: 17:32)

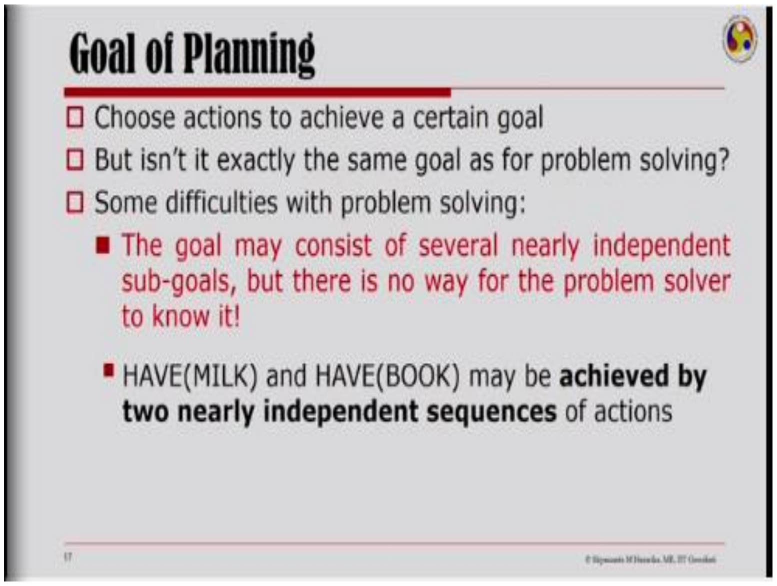


Goal of Planning

- Choose actions to achieve a certain goal
- Is this exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal test is another black-box function.
  - Suppose that the goal is HAVE(MILK)$\wedge$HAVE(BOOK)
  - Without an **explicit representation of the goal**, the problem solver cannot know that a state where HAVE(MILK) is already achieved is more promising than a state where neither HAVE(MILK) nor HAVE(BOOK) is achieved .

On the other hand, for the robot suppose the goal is have milk and have book, without an explicit representation of the goal the problem solver cannot know that a state where have milk is already achieved, this is more promising than a state where neither have milk nor have book is achieved. So give an explicit representation of the goal it is possible for a planning agent to know which is a more promising state and this is not possible for a simple problem solving agent.

(Refer Slide Time: 18:18)



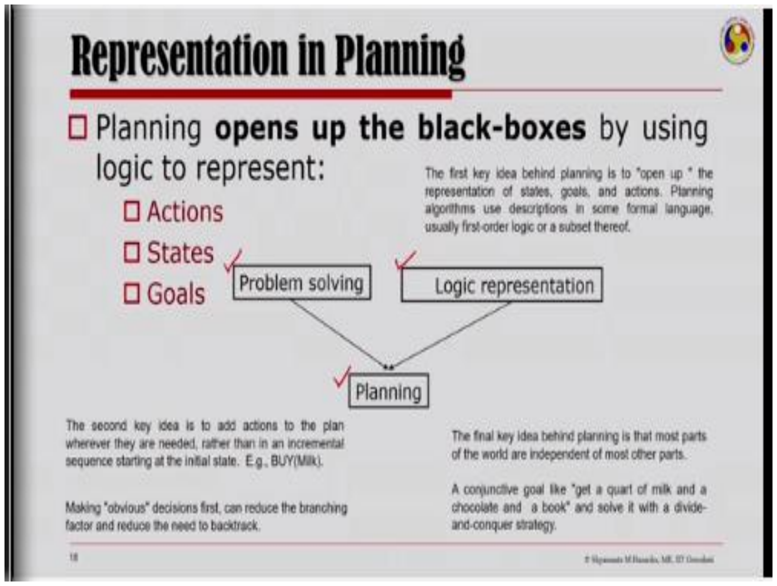
Goal of Planning

- Choose actions to achieve a certain goal
- But isn't it exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal may consist of several nearly independent sub-goals, but there is no way for the problem solver to know it!
  - HAVE(MILK) and HAVE(BOOK) may be **achieved by two nearly independent sequences** of actions

17

Now the goal may consist of several nearly independent sub goals, this is not possible to be known to the problem solving agent at all. Whereas when we are talking of the planning agent have milk and have book may be achieved by 2 nearly independent sequence of actions. And this we can have somehow encoded in the representation schema of the action.

(Refer Slide Time: 18:50)



Representation in Planning

- Planning **opens up the black-boxes** by using logic to represent:
  - Actions
  - States
  - Goals

The first key idea behind planning is to "open up " the representation of states, goals, and actions. Planning algorithms use descriptions in some formal language, usually first-order logic or a subset thereof.

$$\begin{tikzcd}[column sep=small] \checkmark\ \boxed{\text{Problem solving}} \arrow[dr] & & \checkmark\ \boxed{\text{Logic representation}} \arrow[dl] \\ & \checkmark\ \boxed{\text{Planning}} & \end{tikzcd}$$

The second key idea is to add actions to the plan wherever they are needed, rather than in an incremental sequence starting at the initial state. E.g., BUY(Milk).

Making "obvious" decisions first, can reduce the branching factor and reduce the need to backtrack.

The final key idea behind planning is that most parts of the world are independent of most other parts.

A conjunctive goal like "get a quart of milk and a chocolate and a book" and solve it with a divide-and-conquer strategy.

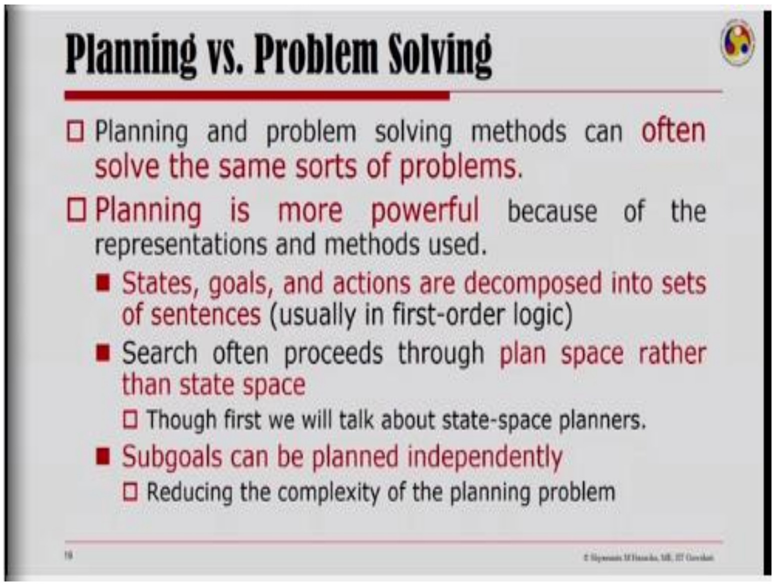
18

So representation in planning actually opens up the black boxes by using logic to represent the action, states and goals. This is the first idea behind planning, the first idea behind planning is to open up the representation. And the planning algorithms use descriptions in some formal language usually first order logic or subsets thereof. So if you look at planning, planning is about problem solving and logic representation being brought together.

The second idea is to add actions to the plan whenever they are needed rather than in an incremental sequence starting at the initial state. For example in order to have, have milk satisfied I can include an action sequence buy milk when it is required. Now making such obvious decisions first can reduce the branching factor and reduce the need to backtrack. The final key idea behind planning is that most parts of the world are independent of most other parts.

So a conjunctive goal like get a quart of milk and a chocolate and a book can be solved by divide and conquer strategy. That is by generating 3 goals out of this, to get a quart of milk, to get it chocolate and to get a book.

(Refer Slide Time: 20:38)



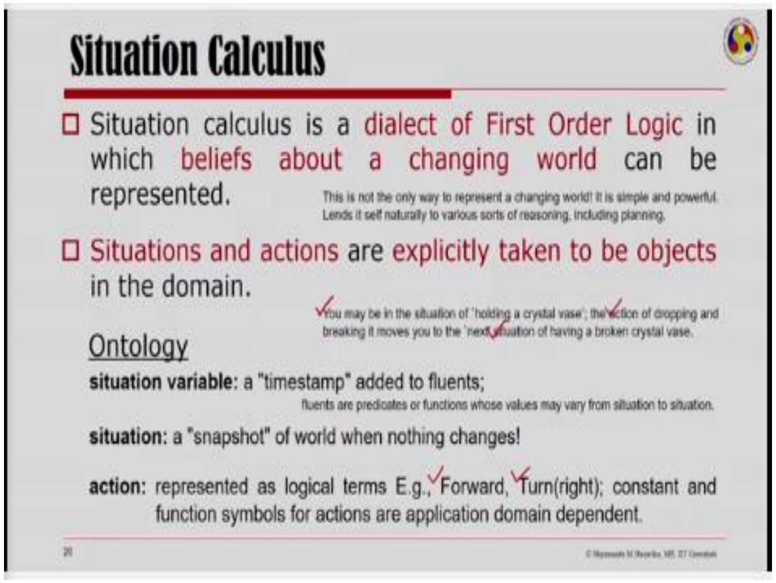
Planning vs. Problem Solving

- Planning and problem solving methods can often solve the same sorts of problems.
- Planning is more powerful because of the representations and methods used.
  - States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
  - Search often proceeds through plan space rather than state space
    - Though first we will talk about state-space planners.
  - Subgoals can be planned independently
    - Reducing the complexity of the planning problem

Planning and problem solving, as we saw can often solve the same sort of problems. However as highlighted here planning is more powerful because of the representation and the methods used. The representation of the states, goals and actions which are decomposed into set of sentences usually in first order logic. And then the search in planning proceeds through plan space rather than state space. So we are searching over an abstract space of plans rather than the state space, though during the course of our discussion here we will also talk about state space planners.

And the most important thing to realize is that sub goals can be planned independently, reducing the complexity of the planning problem.

(Refer Slide Time: 21:42)



Situation Calculus

- Situation calculus is a dialect of First Order Logic in which beliefs about a changing world can be represented.

  This is not the only way to represent a changing world! It is simple and powerful. Lends it self naturally to various sorts of reasoning, including planning.

- Situations and actions are explicitly taken to be objects in the domain.

  You may be in the situation of 'holding a crystal vase'; the action of dropping and breaking it moves you to the 'next' situation of having a broken crystal vase.

Ontology

situation variable: a "timestamp" added to fluents;

fluents are predicates or functions whose values may vary from situation to situation.

situation: a "snapshot" of world when nothing changes!

action: represented as logical terms E.g., Forward, Turn(right); constant and function symbols for actions are application domain dependent.

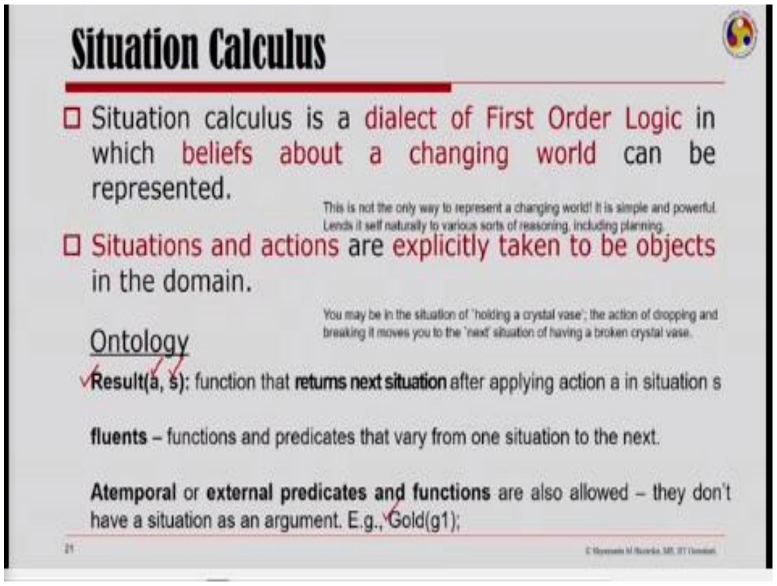
20 [illegible]

We will now talk of a situation calculus which is a dialect of first order logic in which we can express beliefs about a changing world. Now, one needs to realize that this is not the only way to represent a changing world. However situation calculus is a simple and powerful way to represent a world that is changing. And it lends itself naturally to various sorts of reasoning including planning.

Situations and actions are explicitly taken to be objects in the domain of situation calculus, you may be in the situation of holding a crystal vase. Now the action of dropping and breaking it moves you to the next situation of having a broken crystal vase, this is what we try to represent in situation calculus. So the ontology is about having a situation variable, a timestamp added to fluents, fluents are predicates or functions whose value may vary from situation to situation, I then have a situation which is a snapshot of world when nothing changes.

Like in this example there are 2 situations, one where you are holding a crystal vase and another when you are having a broken crystal vase. These are snapshots of the world when nothing changes, in between you have the action of dropping the vase and breaking it which is when the world changes. So action is represented in logical terms, so I could have actions as already you could figure out the action of dropping it and breaking it or you could have other actions like the robot moves forward, the robot takes a right turn so on and so forth. Constant and function symbols for actions are application domain dependent.

(Refer Slide Time: 24:06)



Situation Calculus

- Situation calculus is a dialect of First Order Logic in which beliefs about a changing world can be represented.

This is not the only way to represent a changing world! It is simple and powerful. Lends it self naturally to various sorts of reasoning, including planning.

- Situations and actions are explicitly taken to be objects in the domain.

You may be in the situation of 'holding a crystal vase'; the action of dropping and breaking it moves you to the 'next' situation of having a broken crystal vase.

Ontology

Result(a, s): function that **returns next situation** after applying action a in situation s

fluents – functions and predicates that vary from one situation to the next.

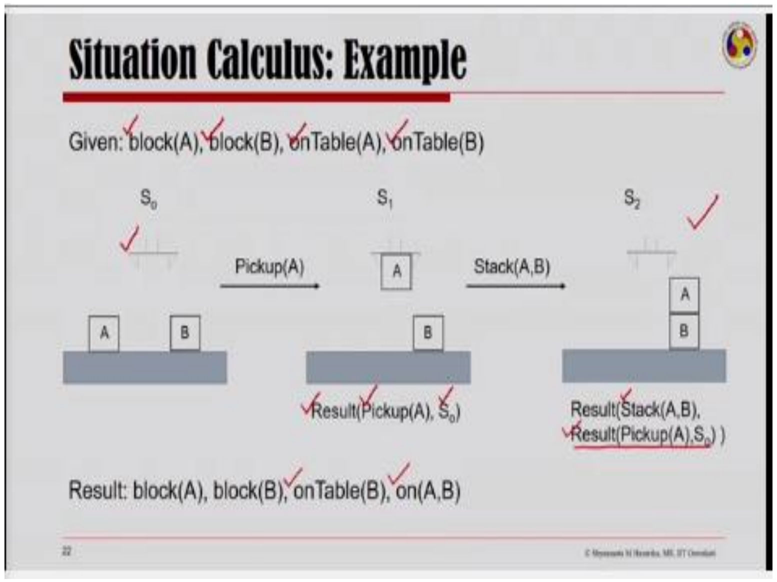
Atemporal or **external predicates and functions** are also allowed – they don't have a situation as an argument. E.g., Gold(g1);

21

Now there is a function that returns the next situation after applying action a in situation s and that is written as result a, s where a is the action and s is the situation and this function returns the next situation. So in our example of holding a crystal vase, dropping it and then the next situation of having a broken crystal vase could be best represented by s being the situation where I am holding a crystal vase and a being the action of dropping and breaking it.

So this function would return the next situation of having a broken crystal vase, fluents has already mentioned our functions and predicates that vary from one situation to the next. I could also have a temporal or external predicates and functions, that do not have a situation as an argument like gold g1, so g1 is where gold is and things like that.

(Refer Slide Time: 25:21)


Situation Calculus: Example

Given: block(A), block(B), onTable(A), onTable(B)

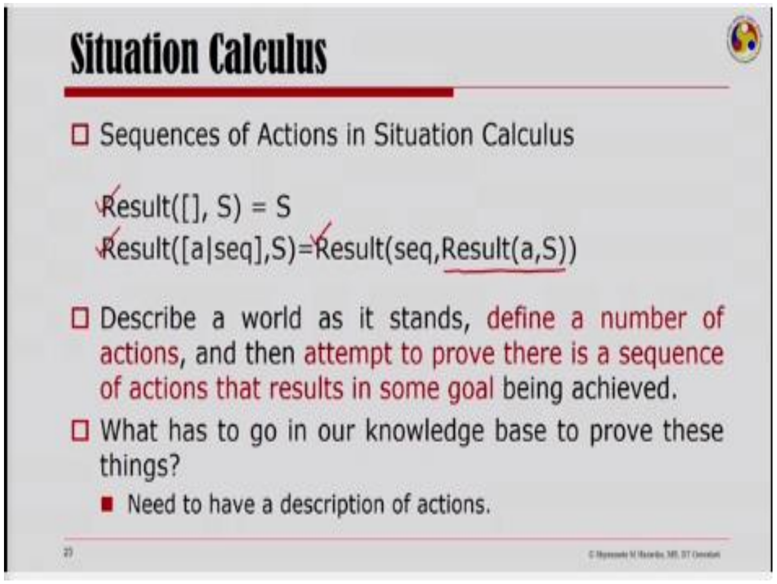
Result: block(A), block(B), onTable(B), on(A,B)

So here is a situation calculus example for the blocks world problem, on the left you have your first situation S0 and you could see that I have a block A and I have block B. A is on the table and B is on the table, kindly note that I could have added more predicates here to say that the hands of the robot are free. And I could have added a proposition saying hands free, what I want to show here is that I have a situation which could be expressed in terms of certain predicates and functions that I define.

And then I have a neck situation here which is S1, so this situation could be about holding A and B on the table thereafter I could have a third situation where A is on B. So from moving from S0 to S1 I would need an action like pick up A, moving from S1 to S2 would require A action like stack A on B. And in terms of the results function I could write situation S1 as the result of the action of picking up A in the situation S0, so that is situation S1.

And I could write S2 as the result of stacking A on B from a situation that results from picking up A in a situation S0. So now as you can see this, I can embed the result function as I want because this situation here is S1 and the action of stacking A on B. And then on S1 gives me S2 but instead of writing S1 here explicitly I could express it using the results function. So finally after these 3 situations, I have the final situation here which is about block A and block B, block B being on the table and A being on top of B.

(Refer Slide Time: 28:20)



Situation Calculus

- Sequences of Actions in Situation Calculus

  Result([], S) = S
  Result([a|seq],S)=Result(seq,$\underline{\text{Result(a,S)}}$)

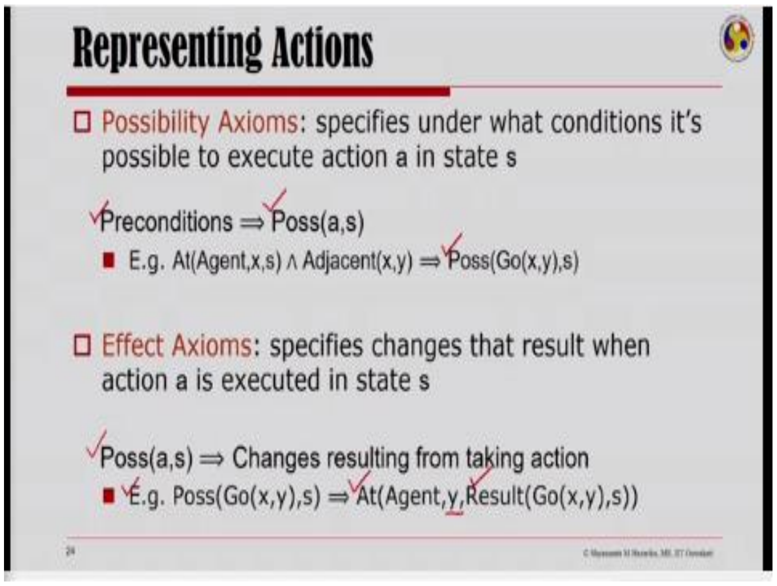
- Describe a world as it stands, define a number of actions, and then attempt to prove there is a sequence of actions that results in some goal being achieved.
- What has to go in our knowledge base to prove these things?
  - Need to have a description of actions.

23

Now this is where you need to take a minute and realize, what is the sequence of actions in situation calculus look like. So here when I have results and the action sequence is empty, I remain at the same situation S. But if I have an X action sequence which is a followed by the whole sequence. Then I can write it as the result of having the sequence and the situation that results out of a on S and this is what we did in the previous example just now.

Describing a world as it is tense I define in number of actions and then I attempt to prove there is a sequence of actions that results in some goal being achieved, this is what is done in situation calculus. Now in order to do that what has to go in our knowledge base to prove these things is that we need to have a description of the actions.

(Refer Slide Time: 29:31)



Representing Actions

- Possibility Axioms: specifies under what conditions it's possible to execute action a in state s

  Preconditions ⇒ Poss(a,s)
  - E.g. At(Agent,x,s) ∧ Adjacent(x,y) ⇒ Poss(Go(x,y),s)

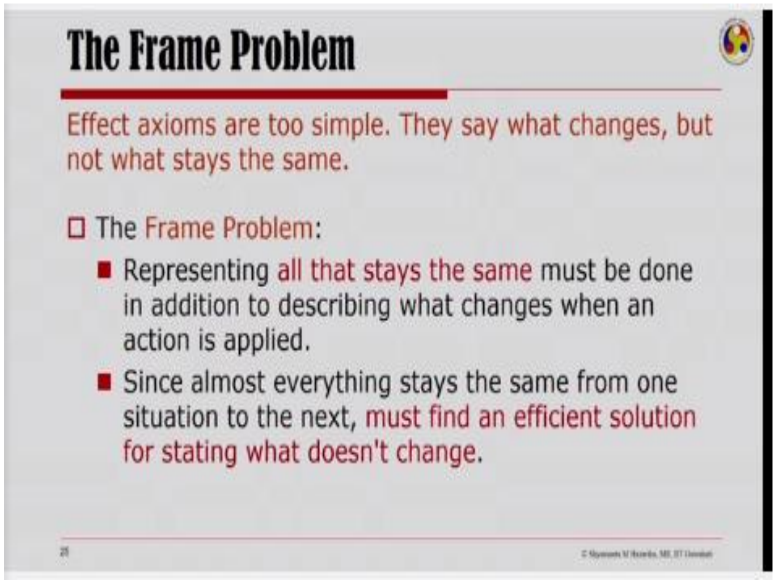
- Effect Axioms: specifies changes that result when action a is executed in state s

  Poss(a,s) ⇒ Changes resulting from taking action
  - E.g. Poss(Go(x,y),s) ⇒ At(Agent,y,Result(Go(x,y),s))

Now representing actions in situation calculus uses 2 axioms, 1 the possibility axiom specifies under what condition it is possible to execute action a in state s. So it gives a series of preconditions under which it is possible to have action a in situation s. Like the agent could be at x in situation s and x and y are adjacent so it must be possible for the agent to go to y. The other axiom we have is about the effects, it specifies changes that results when action a is executed in state s.

For example I am told that it is possible to do action a in state s, then I should be able to write down the changes resulting from taking that action and that is the effect axiom. So if I say here that it is possible to go to x from y under situation s, then as a result of that the effects axiom should be able to state that the agent is at y as a result of that action in situation s.

(Refer Slide Time: 31:03)



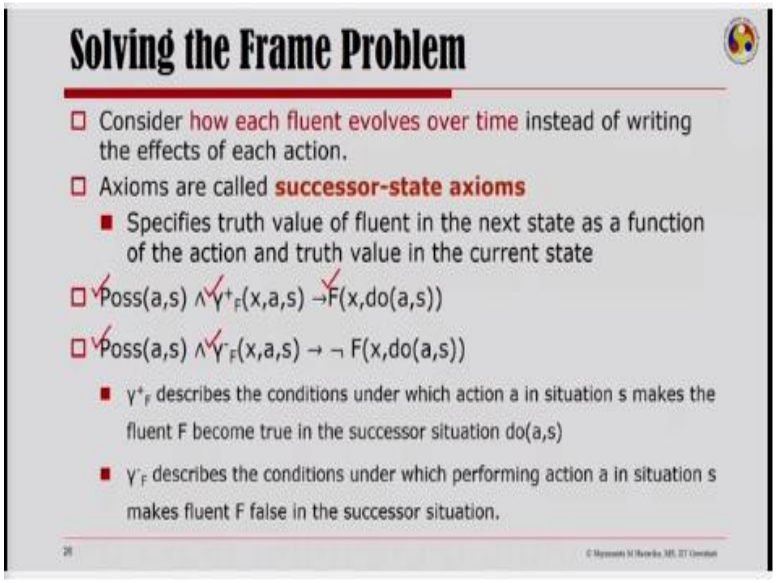
The Frame Problem

Effect axioms are too simple. They say what changes, but not what stays the same.

- The Frame Problem:
  - Representing all that stays the same must be done in addition to describing what changes when an action is applied.
  - Since almost everything stays the same from one situation to the next, must find an efficient solution for stating what doesn't change.

So effects axioms are too simple, they say what changes but not what stays the same. Now this is a very important problem of how much of the things that do not change do you want to cover in the axioms that you want to write. So the frame problem is about representing all that stays the same this must be done in addition to describing what changes when an action is applied. Since almost everything stays the same from one situation to the next we must find an efficient solution for stating what does not change.

(Refer Slide Time: 31:47)



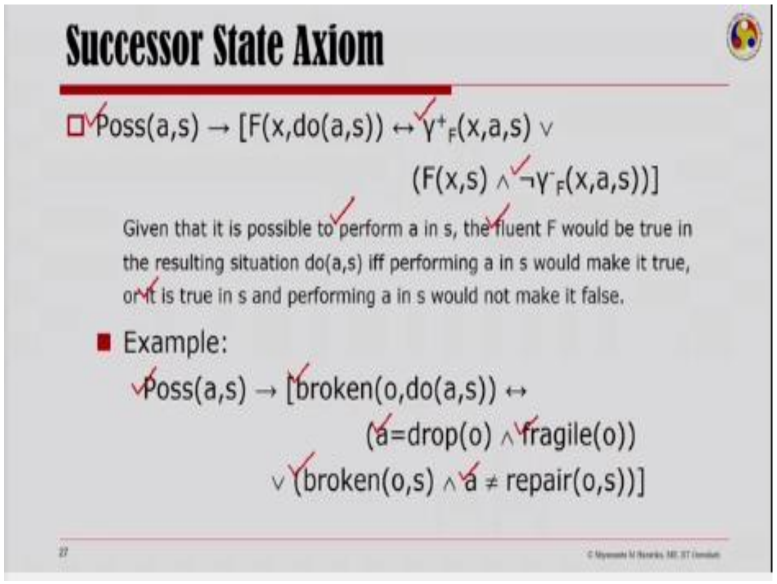
Solving the Frame Problem

- Consider how each fluent evolves over time instead of writing the effects of each action.
- Axioms are called **successor-state axioms**
  - Specifies truth value of fluent in the next state as a function of the action and truth value in the current state
- ✓$Poss(a,s) \land$ ✓$\gamma^{+}_{F}(x,a,s) \rightarrow$ ✓$F(x,do(a,s))$
- ✓$Poss(a,s) \land$ ✓$\gamma^{-}_{F}(x,a,s) \rightarrow \neg F(x,do(a,s))$
  - γ^{+}_{F} describes the conditions under which action a in situation s makes the fluent F become true in the successor situation do(a,s)
  - γ^{-}_{F} describes the conditions under which performing action a in situation s makes fluent F false in the successor situation.

So consider how each fluent evolves over time instead of writing the effects of each action that is what is one solution to the frame problem, so this is called the successor state axioms. The successor state axiom specifies truth values of fluent in the next state as a function of the action and the truth value in the current state. So here is what is the successor state axiom saying I have the possibility of action a in state s and this fluent becomes true.

So I have to list here all the conditions under which a in situation s makes the fluent become true in the successor situation. And I should also describe all the conditions under which the fluent becomes false in the successor situation. This is called the positive effects and this is called the negative effects and this is how you need to write the successor state axioms.

(Refer Slide Time: 33:12)



Successor State Axiom

- $Poss(a,s) \rightarrow [F(x,do(a,s)) \leftrightarrow \gamma^{+}_{F}(x,a,s) \vee (F(x,s) \wedge \neg\gamma^{-}_{F}(x,a,s))]$

  Given that it is possible to perform a in s, the fluent F would be true in the resulting situation do(a,s) iff performing a in s would make it true, or it is true in s and performing a in s would not make it false.

- Example:

  $Poss(a,s) \rightarrow [broken(o,do(a,s)) \leftrightarrow (a=drop(o) \wedge fragile(o)) \vee (broken(o,s) \wedge a \neq repair(o,s))]$

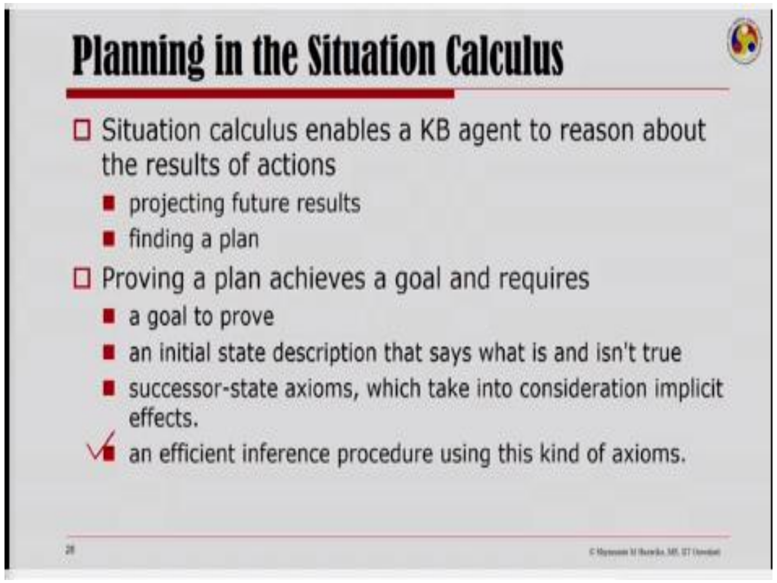
27

Let us take an example to understand this, so here I have the possibility of having an action a in situation s, so my fluent F would become true on doing a in s. And this would be all the positive actions or I would just have F and the situation and I would have all the negative actions not being there. So given that it is possible to perform a in s the fluent F would be true in the resulting situation if performing a in s would make it true or it is true in s and performing a in s would not make it false, that is what I am saying in the successor state axiom.

So let us take this example and see what we mean by here, I have an action a in s and let us say that the object o, I want to do an action in s. Now if the action is about dropping o and I know o is fragile, then it is the condition that it would be broken. And it would be broken in s and not repairable is what I am saying. So here what I am saying is or it was the case that it was already broken in s without the action being done and the action was not about it being repaired.

Because if it was broken in s and the action was about it being repaired then I would not have this being broken. So the action is about dropping, I would break it what does not change is not being covered in terms of explicit facts. But what is covered is the fact that it would have remained broken without even the action, if the action was not about being it repaired. So the successor state axiom is talking about performing an action in s and the fluent F be true in the resulting situation or it is already true in s and performing a would not make it false.

(Refer Slide Time: 35:56)

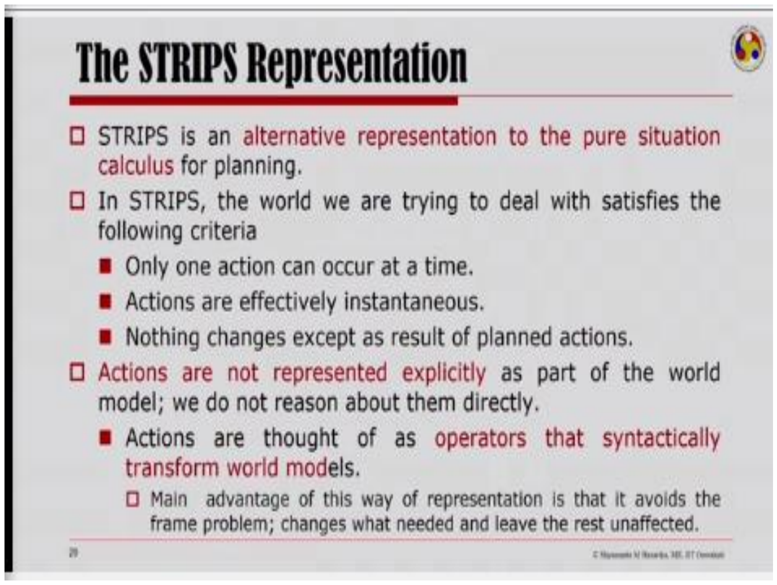


Planning in the Situation Calculus

- Situation calculus enables a KB agent to reason about the results of actions
  - projecting future results
  - finding a plan
- Proving a plan achieves a goal and requires
  - a goal to prove
  - an initial state description that says what is and isn't true
  - successor-state axioms, which take into consideration implicit effects.
  - an efficient inference procedure using this kind of axioms.

Now planning in situation calculus enables a knowledge base agent to reason about the results of action by projecting the future results and finding a plan and it achieves a goal, but these are the requirements. So proving a plan is about achieving a goal and requires a goal to prove an initial state description that says what is and is not true. Successor state axioms, which take into consideration implicit effects and an efficient inference procedure using this kind of axioms.

(Refer Slide Time: 36:42)



The STRIPS Representation

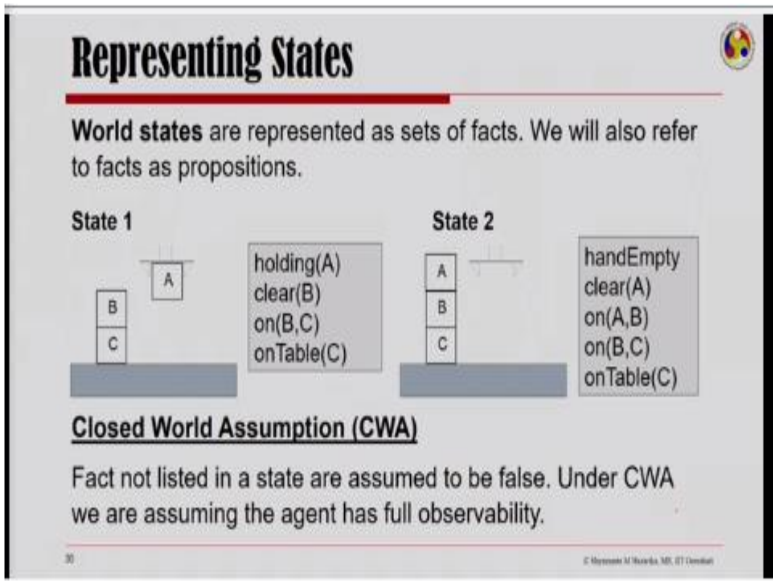
- STRIPS is an alternative representation to the pure situation calculus for planning.
- In STRIPS, the world we are trying to deal with satisfies the following criteria
  - Only one action can occur at a time.
  - Actions are effectively instantaneous.
  - Nothing changes except as result of planned actions.
- Actions are not represented explicitly as part of the world model; we do not reason about them directly.
  - Actions are thought of as operators that syntactically transform world models.
    - Main advantage of this way of representation is that it avoids the frame problem; changes what needed and leave the rest unaffected.

29

Now let us look at the STRIPS representation which is an alternate representation to the pure situation calculus for planning. In STRIPS, the world we are trying to deal with satisfies the following 3 criterias only one action can occur at a time. Actions are effectively instantaneous nothing changes except as result of planned actions. Now actions are not represented explicitly as part of the world model and we do not reason about them directly in STRIPS.

Actions are thought of as operators that syntactically transform the world models, now the main advantage of this way of representation is that it avoids the frame problem, changes what needed and leave the rest unaffected.

(Refer Slide Time: 37:48)



Representing States

World states are represented as sets of facts. We will also refer to facts as propositions.

State 1

holding(A)
clear(B)
on(B,C)
onTable(C)

State 2

handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)

Closed World Assumption (CWA)

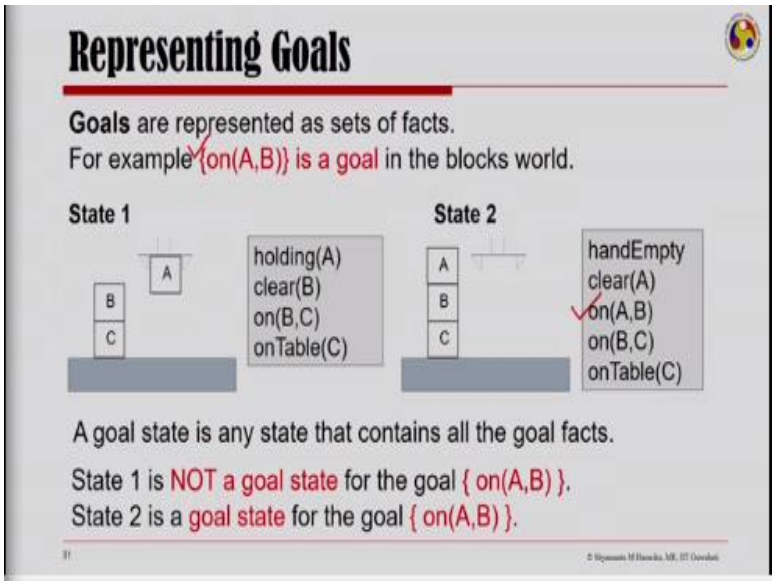
Fact not listed in a state are assumed to be false. Under CWA we are assuming the agent has full observability.

30

So world states are represented as set of facts, we will also refer to facts as propositions here. So on your screen on your left is a particular state, I call it state 1 and it is represented by the following facts. That C is on the table, B is on C, there is nothing on top of B, so I say it clear B and the robot is holding A. So these 4 propositions of holding A, clear B, B on C and C on the table is what describes state 1.

State 2 on the other hand if you see is about hand being empty, nothing on top of A, so clear A, A on B, B on C and C on the table. We have in STRIPS what is called a closed world assumption, that is the facts that are not listed in a state are assumed to be false. So under closed world assumption we are assuming that the agent has full observability.

(Refer Slide Time: 39:18)



Representing Goals

Goals are represented as sets of facts.
For example {on(A,B)} is a goal in the blocks world.

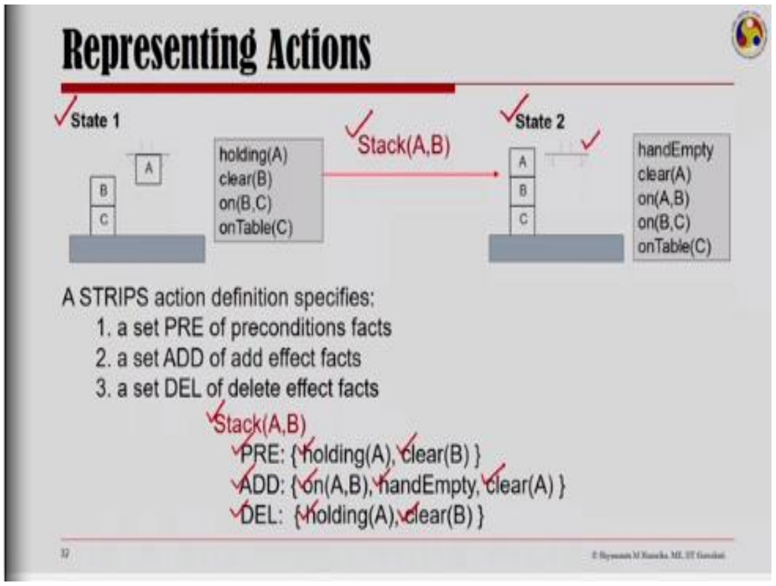
A goal state is any state that contains all the goal facts.

State 1 is NOT a goal state for the goal { on(A,B) }.
State 2 is a goal state for the goal { on(A,B) }.

Goals in STRIP are represented as set of facts again, for example if I say on A, B is a goal in the blocks world that I am describing. Then here on A, B is not there whereas we can see on A, B in state 2. So state 1 is not a goal state for the goal on A, B whereas state 2 is a goal state for the goal on A, B. Goals are represented as set of facts for example A on B is a goal in the blocks world and we could see A on B as part of the description for state 2 and not in the description in state 1.

So state 1 is not a goal state for the goal A on B whereas state 2 is a goal state for the goal A on B.

(Refer Slide Time: 40:28)



Representing Actions

State 1
A
B
C
holding(A)
clear(B)
on(B,C)
onTable(C)
Stack(A,B)
State 2
A
B
C
handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)

A STRIPS action definition specifies:

1. a set PRE of preconditions facts
2. a set ADD of add effect facts
3. a set DEL of delete effect facts

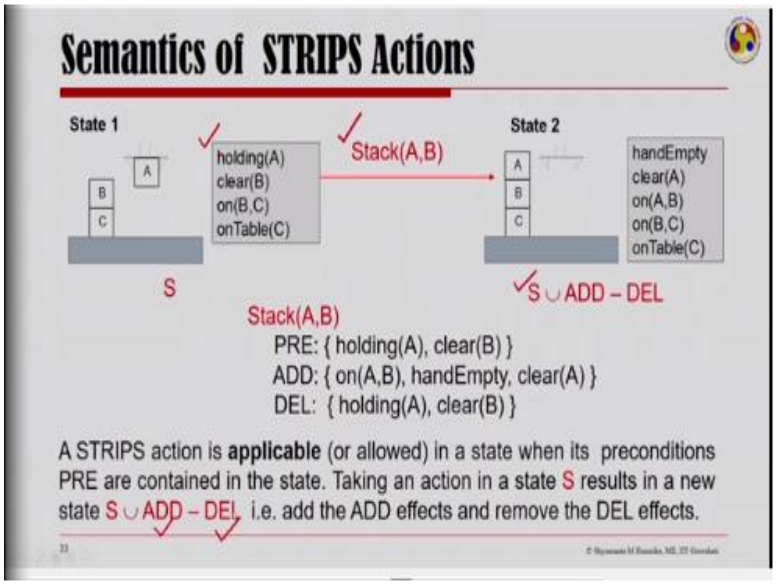
Stack(A,B)
PRE: { holding(A), clear(B) }
ADD: { on(A,B), handEmpty, clear(A) }
DEL: { holding(A), clear(B) }

Actions on the other hand are defined in terms of a set of preconditions, a set of add effects and a set of delete effects. Like to continue a discussion from state 1 to state 2, I could come by performing an action of stacking A on B. Now the action of stack A on B is defined in terms of 3 sets, the precondition set is a set of precondition facts that is conditions that must hold for the action to be possible.

So if I am talking of stacking A on B, I must be holding A and there must not be anything on B, so this is my precondition facts. Once the action is done what is the new scenario that has emerged is shown by the add effects, facts. The set add here lists what are the new propositions that will become true. So once I stack A on B, A on B would become true and the robot will no longer be holding A, so I will have hand empty.

And of course I will generate a new proposition being nothing in top of A which is clear A. Now if I was maintaining a database of facts then I could clearly see that on taking this action of stating A on B, I should no longer have holding A and clear B being true, so that must become false. And one way of doing this is by saying that these facts need to be deleted, so they are the delete effect facts.

(Refer Slide Time: 42:57)



Semantics of STRIPS Actions

State 1
A
B
C
holding(A)
clear(B)
on(B,C)
onTable(C)
S
Stack(A,B)
State 2
A
B
C
handEmpty
clear(A)
on(A,B)
on(B,C)
onTable(C)
S ∪ ADD – DEL

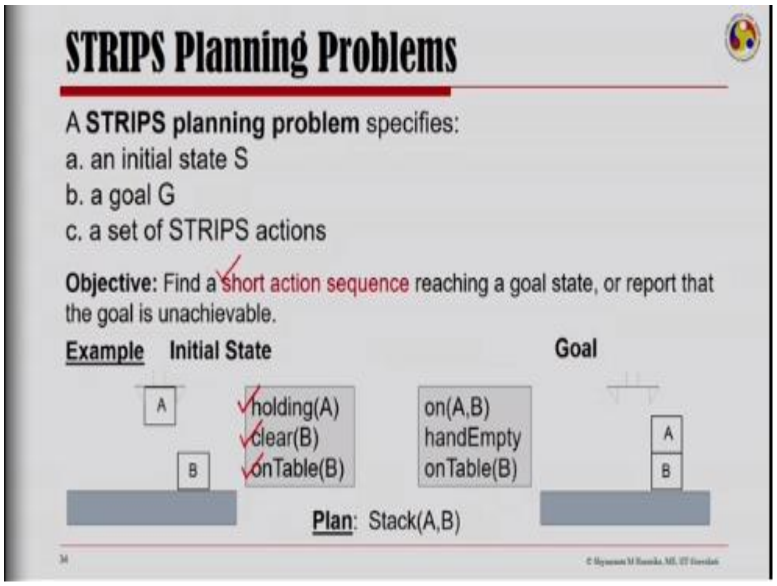
Stack(A,B)
PRE: { holding(A), clear(B) }
ADD: { on(A,B), handEmpty, clear(A) }
DEL: { holding(A), clear(B) }

A STRIPS action is **applicable** (or allowed) in a state when its preconditions PRE are contained in the state. Taking an action in a state S results in a new state S ∪ ADD – DEL i.e. add the ADD effects and remove the DEL effects.

In terms of semantics of the STRIPS action, I have state 1 which is S here holding A, clear B, on B, C and on table C. And I have the action stack A, B giving rise to state 2 which is about hand empty clear A, on A, B, on B, C and on table C. So in terms of the preconditions, the preconditions are to be checked here and final state description is about taking the union of the add and removing the del of the action.

So I would add ADD to this set of propositions describing the state and I will subtract the delete effects from the description of that particular state. So a STRIPS action is applicable in a state when it is preconditions are contained in that state and taking an action in state S results in a new state which is I add the ADD effects and remove the delete effects facts.

(Refer Slide Time: 44:18)



STRIPS Planning Problems

A **STRIPS planning problem** specifies:

a. an initial state S

b. a goal G

c. a set of STRIPS actions

Objective: Find a short action sequence reaching a goal state, or report that the goal is unachievable.

Example **Initial State** **Goal**

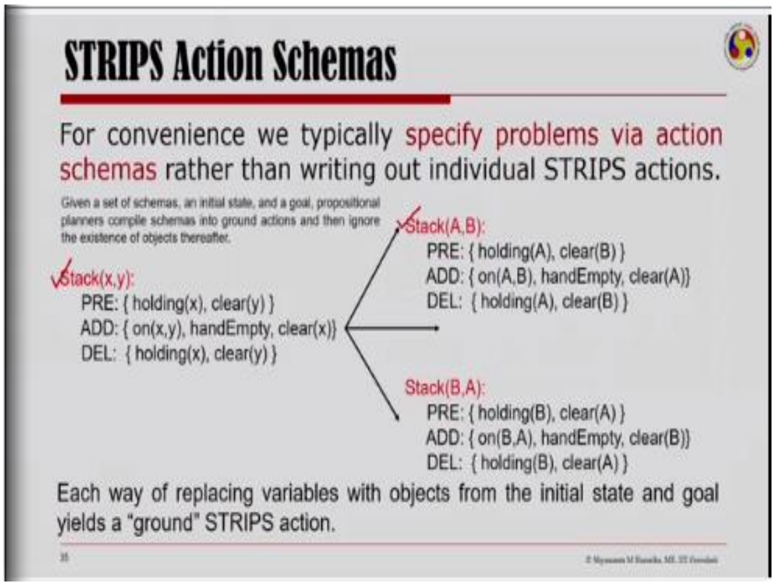
holding(A)	on(A,B)
clear(B)	handEmpty
onTable(B)	onTable(B)

Plan: Stack(A,B)

So a STRIPS planning problem specifies an initial state, a goal state, a set of STRIPS actions. And the objective is to find a short action sequence reaching a goal state or report that the goal is unachievable. So here is an example I have the initial state where the robot is holding A and B is on the table. So I have holding A, clear B, on table B and the goal is when A is on B and the hand is empty and B is on the table.

So here the plan is just stack A, B because that is the short action sequence that will take me from the initial state to the goal state.

(Refer Slide Time: 45:16)



STRIPS Action Schemas

For convenience we typically specify problems via action schemas rather than writing out individual STRIPS actions.

Given a set of schemas, an initial state, and a goal, propositional planners compile schemas into ground actions and then ignore the existence of objects thereafter.

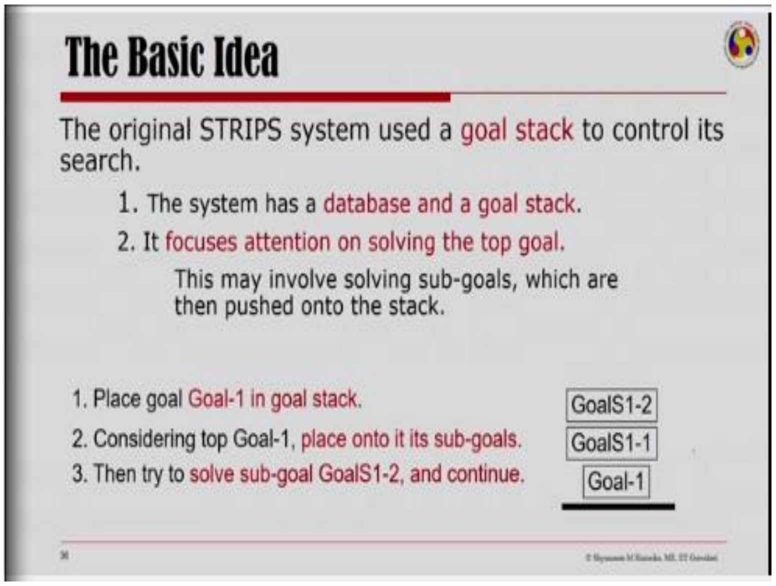
```
Stack(x,y):
    PRE: { holding(x), clear(y) }
    ADD: { on(x,y), handEmpty, clear(x)}
    DEL:  { holding(x), clear(y) }

Stack(A,B):
    PRE: { holding(A), clear(B) }
    ADD: { on(A,B), handEmpty, clear(A)}
    DEL:  { holding(A), clear(B) }

Stack(B,A):
    PRE: { holding(B), clear(A) }
    ADD: { on(B,A), handEmpty, clear(B)}
    DEL:  { holding(B), clear(A) }
```

Each way of replacing variables with objects from the initial state and goal yields a "ground" STRIPS action.

For convenience we typically specify problems via action schemas rather than writing out individual STRIP actions. What it means is, like instead of saying stack A, B any action would be written in terms of an action schema, I would say stack x, y. And then given a set of schemas an initial state and a goal propositional planners with then compile the schemas into ground actions. Like I could then say stake A, B which is about holding A and clear B and adding that after stack A, B is done I would have on A, B and hand empty and clear A.

And I would be doing deleting the holding A and clear B, or I could have another action from that schema which is about stacking B on A. So each way of replacing the variables with objects from the initial state and the goal yields a ground STRIPS action.

(Refer Slide Time: 46:32)



The Basic Idea

The original STRIPS system used a goal stack to control its search.

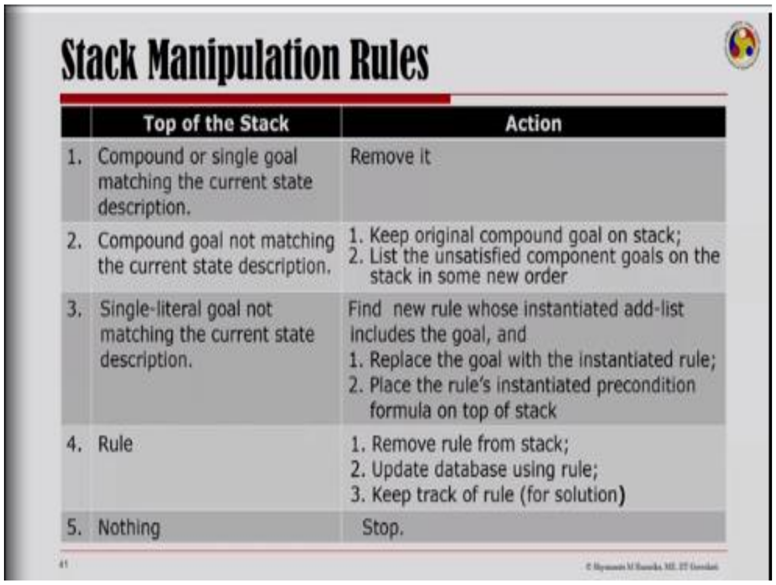
1. The system has a database and a goal stack.
2. It focuses attention on solving the top goal.

   This may involve solving sub-goals, which are then pushed onto the stack.

1. Place goal Goal-1 in goal stack.
2. Considering top Goal-1, place onto it its sub-goals.
3. Then try to solve sub-goal GoalS1-2, and continue.

The original STRIP system use a goal state to control it is search, now the system as a data base and a goal stack, it focuses attention on solving the top goal. This may involve solving sub goals which are then pushed onto the stack, like I have the goal 1 in the goal stack. And if I have goal 1 and suppose goal 1 has sub goals 1 and 2 then I would place onto the stack the sub goals first and tries to solve the sub goals and continue.

(Refer Slide Time: 47:22)



Stack Manipulation Rules

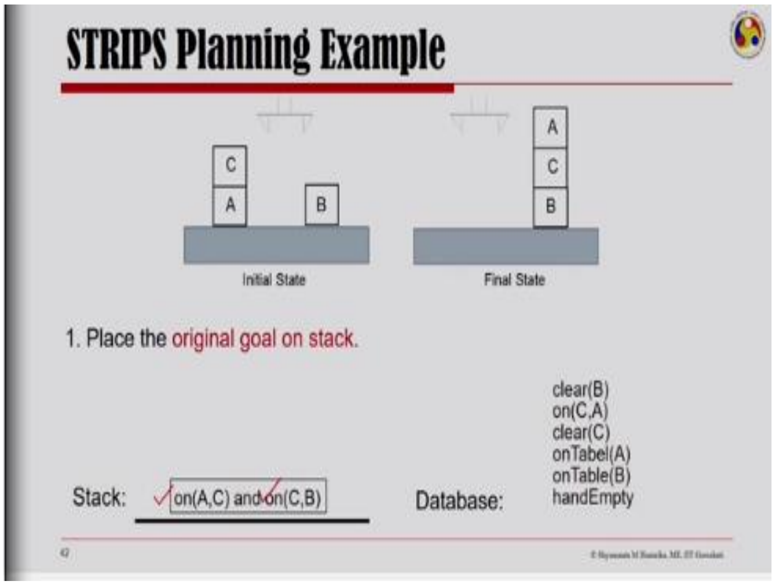
	Top of the Stack	Action
1.	Compound or single goal matching the current state description.	Remove it
2.	Compound goal not matching the current state description.	1. Keep original compound goal on stack; 2. List the unsatisfied component goals on the stack in some new order
3.	Single-literal goal not matching the current state description.	Find new rule whose instantiated add-list includes the goal, and 1. Replace the goal with the instantiated rule; 2. Place the rule's instantiated precondition formula on top of stack
4.	Rule	1. Remove rule from stack; 2. Update database using rule; 3. Keep track of rule (for solution)
5.	Nothing	Stop.

Now there are a couple of stack manipulation rules that need to be quickly looked at before I look at the STRIPS problem to be solved. So the first rule says that if I have a compound or single goal matching the current state description, then I would remove it from the stack. If I have a compound goal not matching the current state description then the idea is to keep the original compound goal on the stack and list the unsatisfied component goals on the stack in some new order.

If there is a single literal goal not matching the current state description then you got to find a new rule whose instantiated at list includes the goal. And you replace the goal with the instantiated rule, place the rules instantiated precondition formula on top of the stack. And if the top of the stack is a rule you remove the rule from the stack update the database using the rule. Because you know once you have a rule it will have some add effects facts (()) (48:39) and delete effect facts which you need to look at the database and update.

And then you need to keep track of the rule to remember that it is part of the solution, and if your stack has nothing on it this is where you stop and you have a plan.

(Refer Slide Time: 49:02)



STRIPS Planning Example

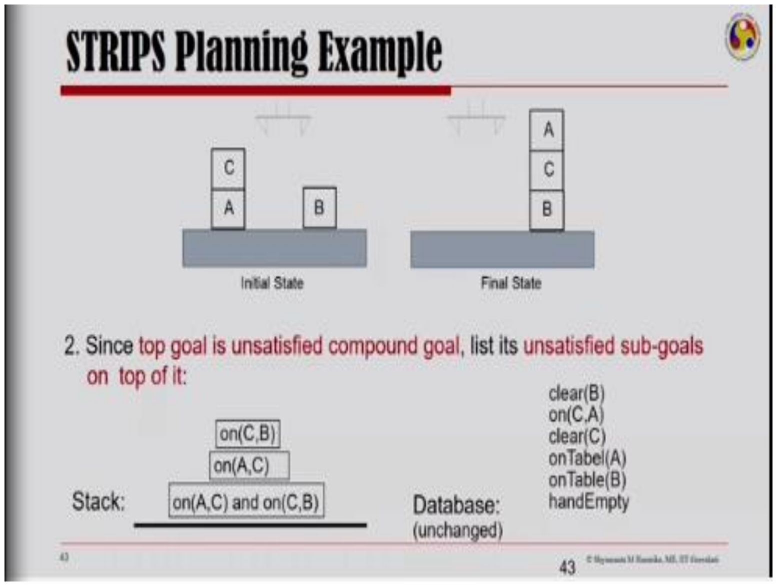
1. Place the original goal on stack.

Stack: on(A,C) and on(C,B)

Database:
clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

So let us look at a simple STRIPS planning example as an illustration here. So I have the initial state here which is C is on A and B is on the table and finally I want to arrive at B on the table C one B and A on C. So I first place the original goal on the stack which is A on C and C on B, now that is the original goal under stack.

(Refer Slide Time: 49:34)



STRIPS Planning Example

2. Since top goal is unsatisfied compound goal, list its unsatisfied sub-goals on top of it:

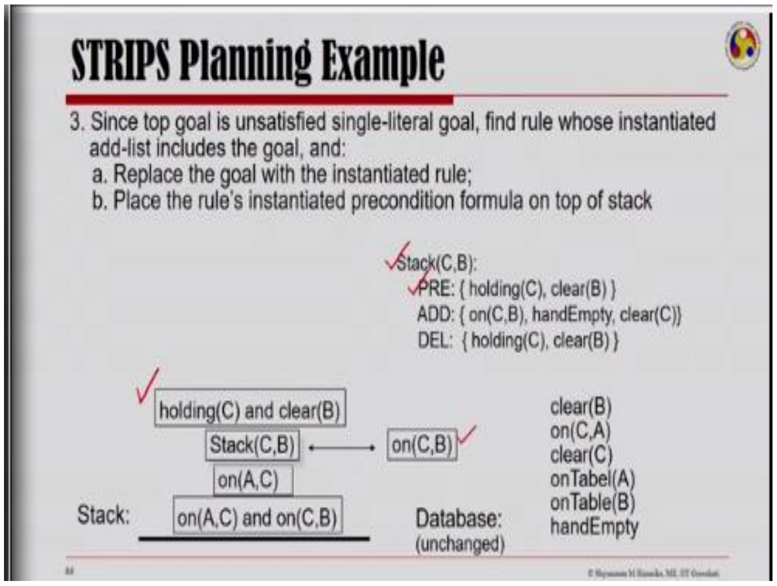
Stack:
on(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)
clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

43

This goal is unsatisfied compound goal, so I have to list it is unsatisfied sub goals and put it on top of each other. And I need to remember that to start with my database is about the initial state which is I have a clear B, I have C on A, I have C being clear, A is on the table, B is on the table and the hands of the robots are empty.

(Refer Slide Time: 50:10)



STRIPS Planning Example

3. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
   a. Replace the goal with the instantiated rule;
   b. Place the rule's instantiated precondition formula on top of stack

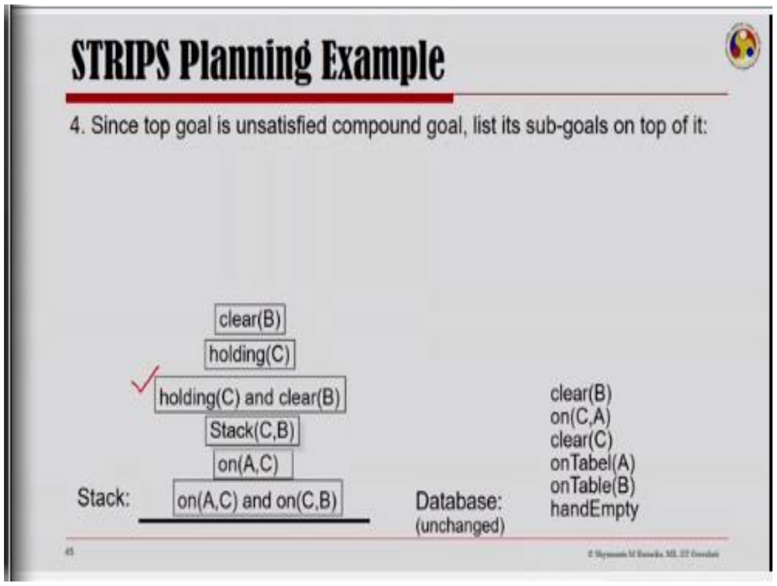
✓Stack(C,B):
✓PRE: { holding(C), clear(B) }
ADD: { on(C,B), handEmpty, clear(C)}
DEL: { holding(C), clear(B) }

Stack: ✓ holding(C) and clear(B) / Stack(C,B) ⟵⟶ on(C,B) ✓ / on(A,C) / on(A,C) and on(C,B)

Database: (unchanged) clear(B), on(C,A), clear(C), onTabel(A), onTable(B), handEmpty

Now since the top goal here is an unsatisfied single literal goal, I need to find a rule whose instantiated at list includes the goal and replace the goal with the instantiated rule. So I place the rules instantiated precondition formula on top of the stack. So the rule that we are looking for is the rule of stacking C on B and it has the preconditions that you must be holding C and B must be clear, so that is what is placed here on top of the stack.

(Refer Slide Time: 50:53)



STRIPS Planning Example

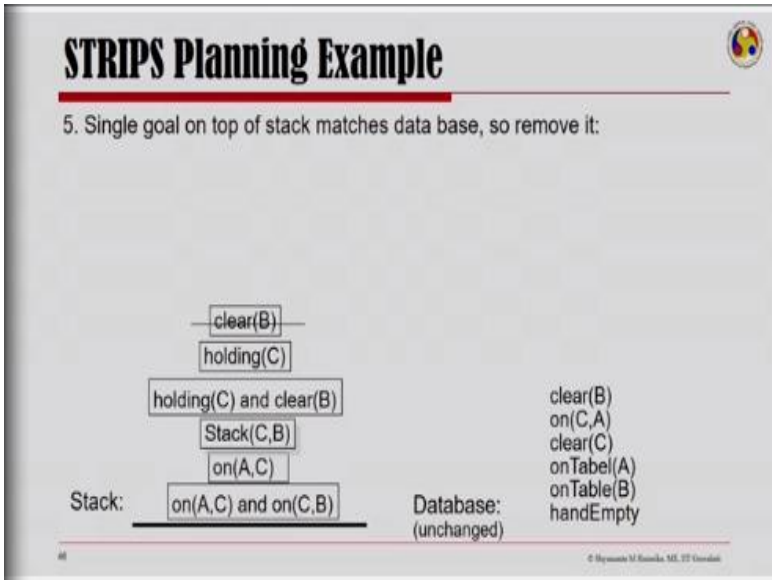
4. Since top goal is unsatisfied compound goal, list its sub-goals on top of it:

Stack: clear(B) / holding(C) / ✓ holding(C) and clear(B) / Stack(C,B) / on(A,C) / on(A,C) and on(C,B)

Database: (unchanged) clear(B), on(C,A), clear(C), onTabel(A), onTable(B), handEmpty

And now when you look at the top of the stack this is a compound goal, so I will list it is sub goals one on top of other, holding C and clear B.

(Refer Slide Time: 51:10)



STRIPS Planning Example

5. Single goal on top of stack matches data base, so remove it:

Stack:

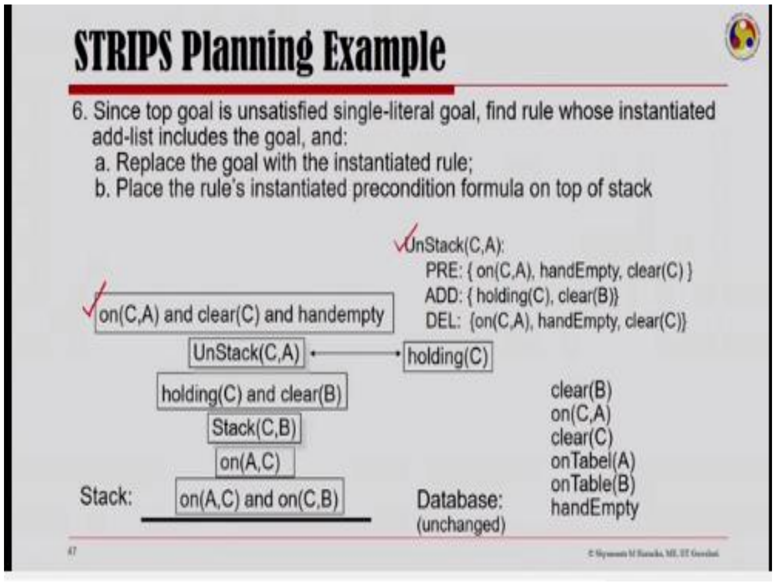
~~clear(B)~~
holding(C)
holding(C) and clear(B)
Stack(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

Now this is interesting because clear B if you see is a single goal on top of the stack which matches the database. And so according to the stack manipulation rules that I have described little while ago, I will just remove this from the stack.

(Refer Slide Time: 51:29)



STRIPS Planning Example

6. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
   a. Replace the goal with the instantiated rule;
   b. Place the rule's instantiated precondition formula on top of stack

✓UnStack(C,A):
PRE: { on(C,A), handEmpty, clear(C) }
ADD: { holding(C), clear(B)}
DEL: {on(C,A), handEmpty, clear(C)}

Stack:

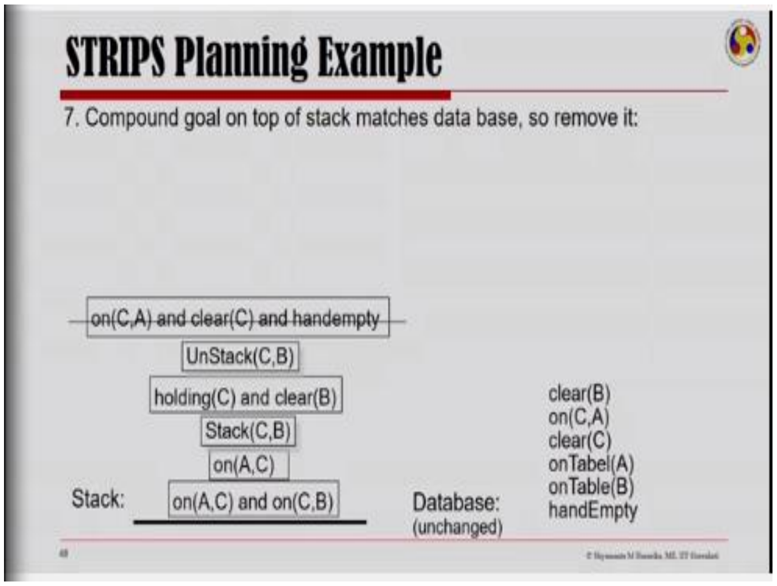
✓on(C,A) and clear(C) and handempty
UnStack(C,A) ← holding(C)
holding(C) and clear(B)
Stack(C,B)
on(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

Now I have holding C there, so this is an unsatisfied single literal goal, so I will find a rule whose instantiated add-list includes the goal and replace this with the instantiated precondition formula. So this is about unstacking C and A, I have the precondition on C, A, hands empty and clear, so this is what I put it here.

(Refer Slide Time: 51:59)



STRIPS Planning Example

7. Compound goal on top of stack matches data base, so remove it:

~~on(C,A) and clear(C) and handempty~~

UnStack(C,B)

holding(C) and clear(B)

Stack(C,B)

on(A,C)

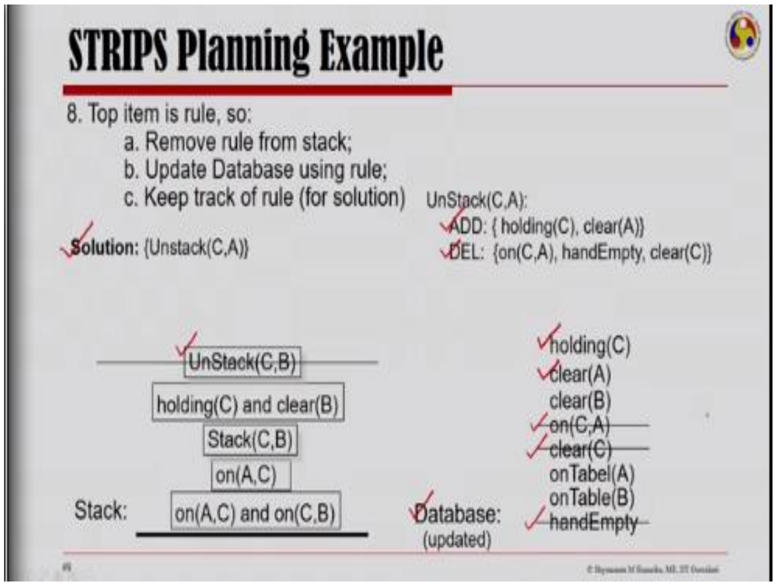
Stack: on(A,C) and on(C,B)

Database: (unchanged)

clear(B)
on(C,A)
clear(C)
onTabel(A)
onTable(B)
handEmpty

And next I see that the compound goal on the top of the stack matches the database, so I simply remove it, I now have unstack C, B.

(Refer Slide Time: 52:11)



STRIPS Planning Example

8. Top item is rule, so:
   a. Remove rule from stack;
   b. Update Database using rule;
   c. Keep track of rule (for solution)

Solution: {Unstack(C,A)}

UnStack(C,A):
ADD: { holding(C), clear(A)}
DEL: {on(C,A), handEmpty, clear(C)}

~~UnStack(C,B)~~

holding(C) and clear(B)

Stack(C,B)

on(A,C)

Stack: on(A,C) and on(C,B)

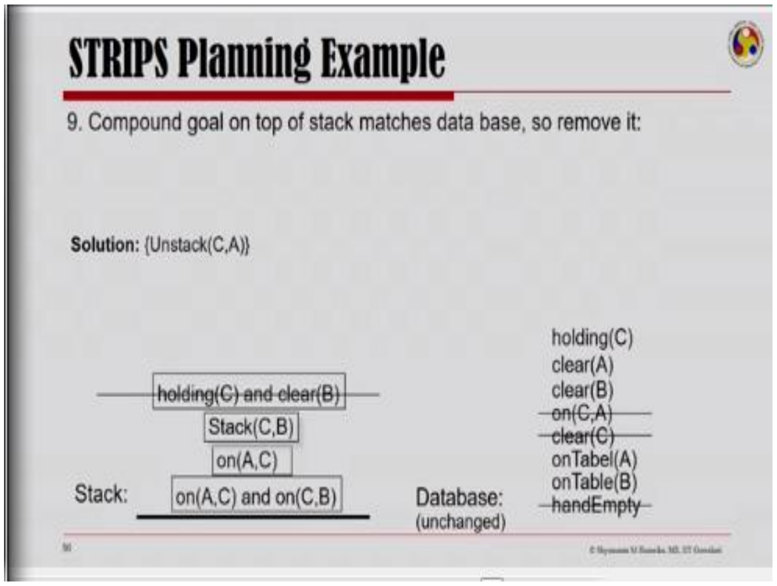
Database: (updated)

holding(C)
clear(A)
clear(B)
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
~~handEmpty~~

So I want to get to the top item but the top item is a rule now, so if I remove the rule from the stack, I need to update the database and I need to keep track of it for the solution. So when I am trying to move this rule out ,I see that it has ADD list and a delete list. So things to be added I add it to my database which is holding C and clear A, so I update my database. And I have to remember this for my solution before I move it out, so I remember it from my solution here.

And as I remember it from my solution I remove things from the delete list in my database, so I update my database.

(Refer Slide Time: 53:07)



STRIPS Planning Example

9. Compound goal on top of stack matches data base, so remove it:

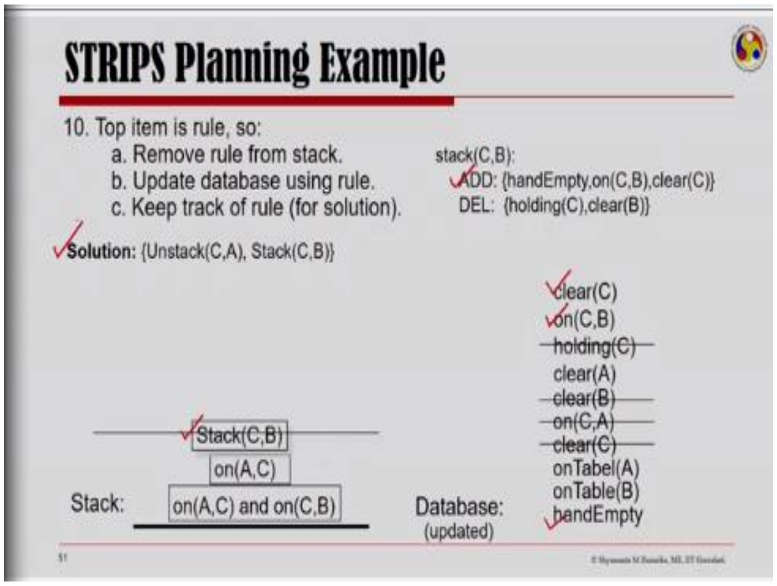
Solution: {Unstack(C,A)}

~~holding(C) and clear(B)~~
Stack(C,B)
on(A,C)
Stack: on(A,C) and on(C,B)

Database: (unchanged)
holding(C)
clear(A)
clear(B)
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
~~handEmpty~~

Once this is done the compound goal and the top of the stack matches the database, so I simply remove it.

(Refer Slide Time: 53:14)



STRIPS Planning Example

10. Top item is rule, so:
  a. Remove rule from stack.
  b. Update database using rule.
  c. Keep track of rule (for solution).

stack(C,B):
✓ADD: {handEmpty,on(C,B),clear(C)}
DEL: {holding(C),clear(B)}

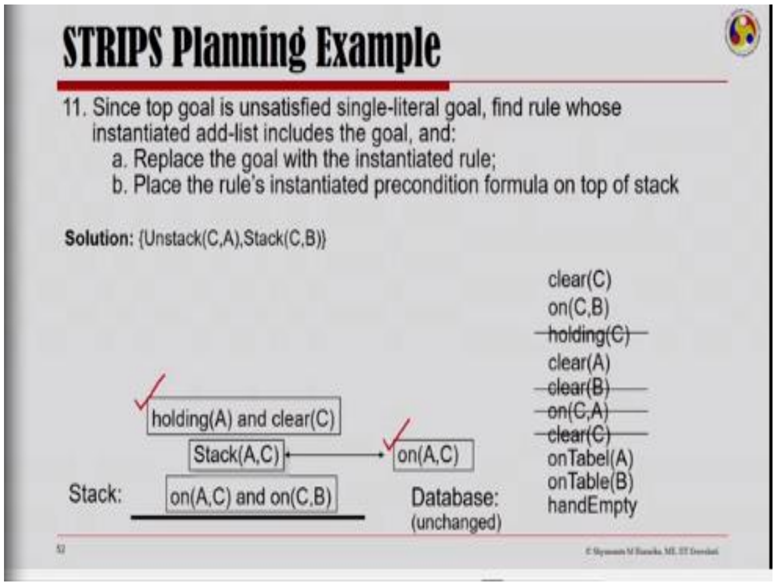
✓**Solution:** {Unstack(C,A), Stack(C,B)}

~~✓Stack(C,B)~~
on(A,C)
Stack: on(A,C) and on(C,B)

Database: (updated)
✓clear(C)
✓on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
✓handEmpty

And then, I now have again a rule on top of the database, now in order to bring it and make it part of thus solution, I follow the steps that I did for unstuck. That means I ADD things on the add-list into my database, so I add clear C, I add on C, B, I add hands empty and I delete holding C and clear B from the database and makes (()) (53:47) B part of the solution here.

(Refer Slide Time: 53:51)



STRIPS Planning Example

11. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
    a. Replace the goal with the instantiated rule;
    b. Place the rule's instantiated precondition formula on top of stack

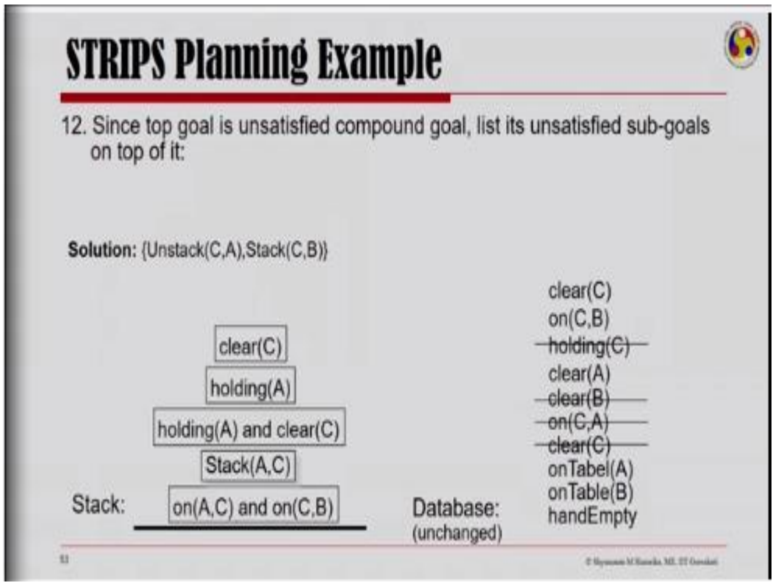
Solution: {Unstack(C,A),Stack(C,B)}

Stack:
- ✓ holding(A) and clear(C)
- Stack(A,C) ⟷ ✓ on(A,C)
- on(A,C) and on(C,B)

Database: (unchanged)
clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

Then I have now the top goal which is a single literal goal, I find a rule whose instantiated add-list includes the goal. So that rule is about stack, so I get that rule written here it is preconditions written as top of the stack.

(Refer Slide Time: 54:15)



STRIPS Planning Example

12. Since top goal is unsatisfied compound goal, list its unsatisfied sub-goals on top of it:

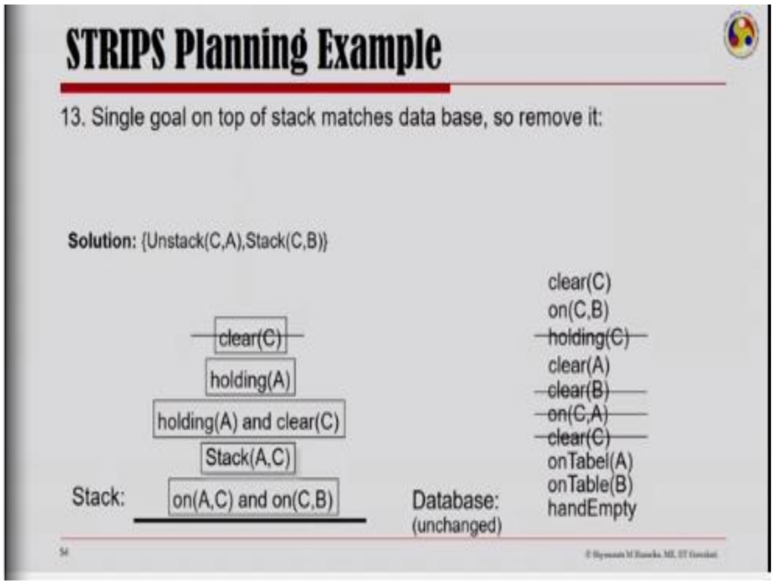
Solution: {Unstack(C,A),Stack(C,B)}

Stack:
- clear(C)
- holding(A)
- holding(A) and clear(C)
- Stack(A,C)
- on(A,C) and on(C,B)

Database: (unchanged)
clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

And now this is an unsatisfied compound goal, so I will break it up into 2 and stack it here on top of one another and I have clear C now.

(Refer Slide Time: 54:31)



STRIPS Planning Example

13. Single goal on top of stack matches data base, so remove it:

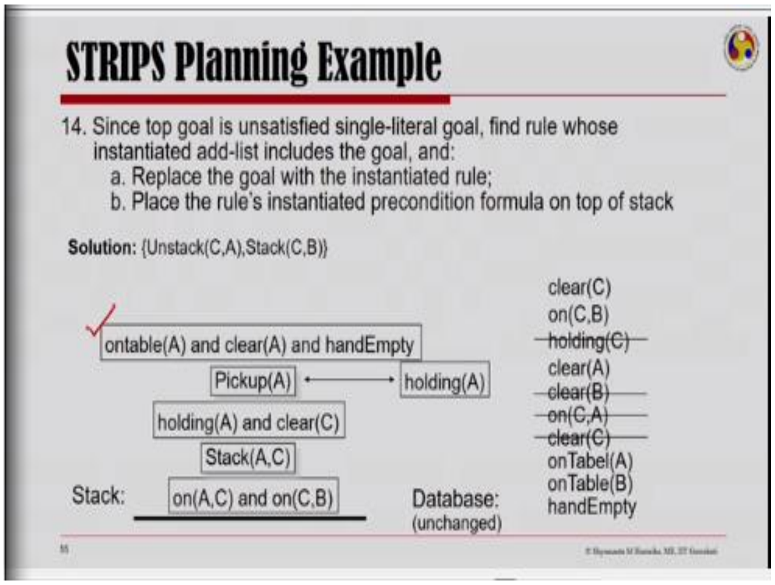
Solution: {Unstack(C,A),Stack(C,B)}

Stack:
~~clear(C)~~
holding(A)
holding(A) and clear(C)
Stack(A,C)
on(A,C) and on(C,B)

Database: (unchanged)
clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

Now you could see that clear C is a single goal on top of the stack which matches the database, so I remove clear C from the top of the stack.

(Refer Slide Time: 54:43)



STRIPS Planning Example

14. Since top goal is unsatisfied single-literal goal, find rule whose instantiated add-list includes the goal, and:
a. Replace the goal with the instantiated rule;
b. Place the rule's instantiated precondition formula on top of stack

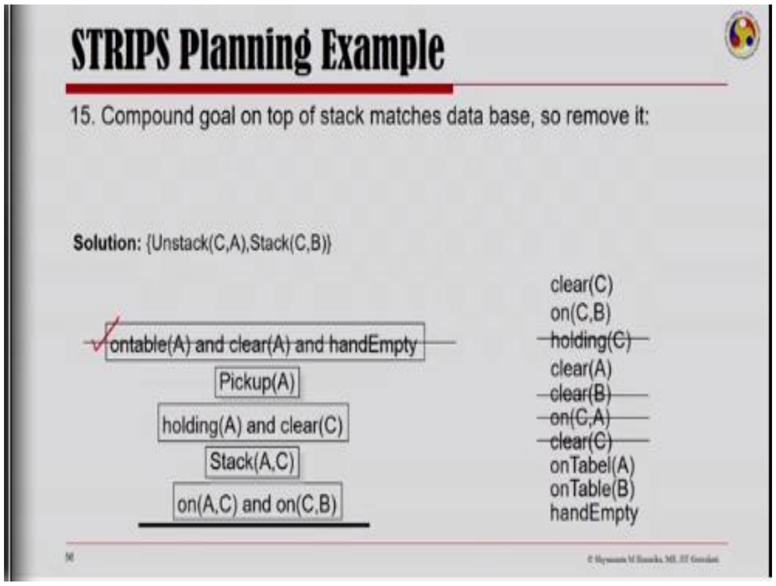
Solution: {Unstack(C,A),Stack(C,B)}

Stack:
✓ ontable(A) and clear(A) and handEmpty
Pickup(A) ⟷ holding(A)
holding(A) and clear(C)
Stack(A,C)
on(A,C) and on(C,B)

Database: (unchanged)
clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

And then I have holding A, now this is an unsatisfied single literal goal, I find the rule which satisfies this, this is about pick up a I give it is precondition here. So this is about on table A, clear A and hand empty.

(Refer Slide Time: 55:08)



STRIPS Planning Example

15. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B)}

~~ontable(A) and clear(A) and handEmpty~~

Pickup(A)

holding(A) and clear(C)

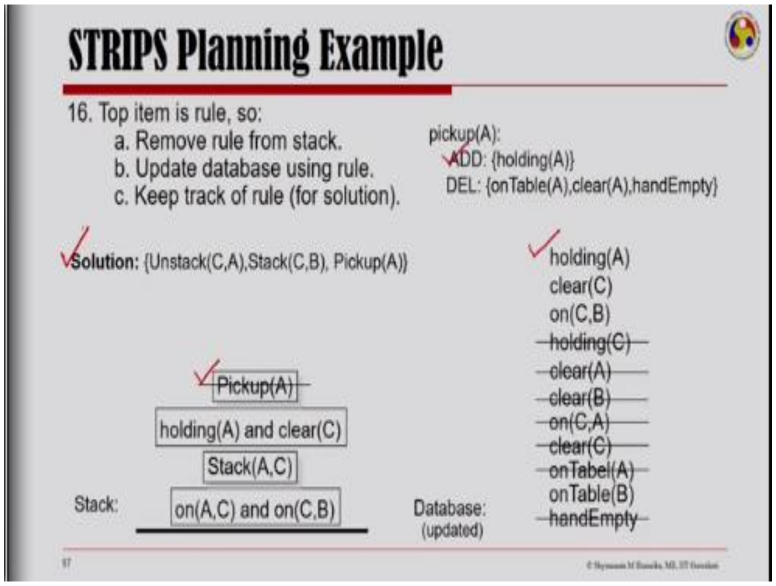
Stack(A,C)

on(A,C) and on(C,B)

clear(C)
on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
onTabel(A)
onTable(B)
handEmpty

Once I have that, I have now a compound goal on top of the stack that matches the database because if you see I have on table A, clear A and hand empty. So I have on table A, clear A and hand empty, so I can remove that.

(Refer Slide Time: 55:30)



STRIPS Planning Example

16. Top item is rule, so:
    a. Remove rule from stack.
    b. Update database using rule.
    c. Keep track of rule (for solution).

pickup(A):
ADD: {holding(A)}
DEL: {onTable(A),clear(A),handEmpty}

Solution: {Unstack(C,A),Stack(C,B), Pickup(A)}

Stack:

~~Pickup(A)~~

holding(A) and clear(C)

Stack(A,C)

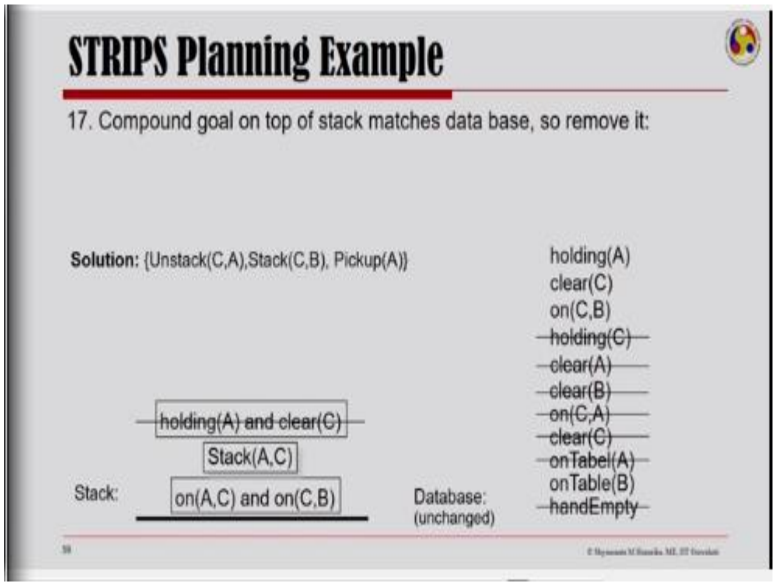
on(A,C) and on(C,B)

Database: (updated)

holding(A)
clear(C)
on(C,B)
~~holding(C)~~
~~clear(A)~~
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
~~handEmpty~~

Once I remove this, I have the top element which is a rule I will add the add-list of the rule to my database which is holding A. I will delete things from the delete list which is on table A, clear A and hands empty, I will remove this rule from here and make it part of the solution.

(Refer Slide Time: 55:58)



STRIPS Planning Example

17. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B), Pickup(A)}

Stack:

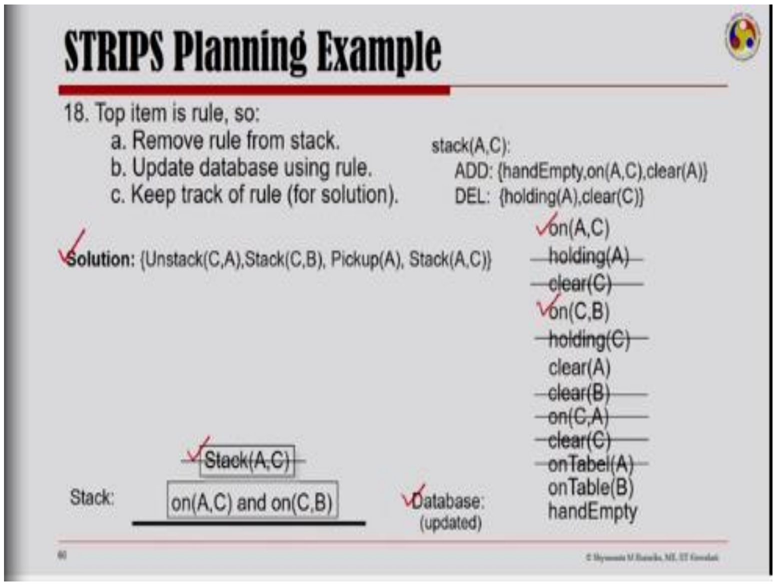
~~holding(A) and clear(C)~~
Stack(A,C)
on(A,C) and on(C,B)

Database: (unchanged)

holding(A)
clear(C)
on(C,B)
~~holding(C)~~
~~clear(A)~~
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
~~handEmpty~~

So I have the solution there, the compound goal on top of the stack matches database, so I remove it and then I have holding A, clear C being removed.

(Refer Slide Time: 56:10)



STRIPS Planning Example

18. Top item is rule, so:
    a. Remove rule from stack.
    b. Update database using rule.
    c. Keep track of rule (for solution).

stack(A,C):
ADD: {handEmpty,on(A,C),clear(A)}
DEL: {holding(A),clear(C)}

✓**Solution:** {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

Stack:

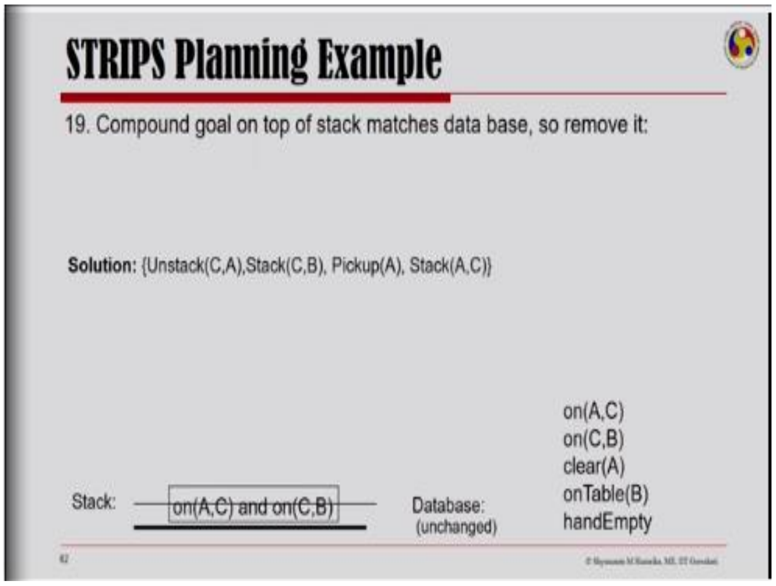
✓~~Stack(A,C)~~
on(A,C) and on(C,B)

✓Database: (updated)

✓on(A,C)
~~holding(A)~~
~~clear(C)~~
✓on(C,B)
~~holding(C)~~
clear(A)
~~clear(B)~~
~~on(C,A)~~
~~clear(C)~~
~~onTabel(A)~~
onTable(B)
handEmpty

Now I have the top item which is a rule, I follow the 4 previous steps that is I update the database, I make it part of the solution and then remove it from the stack, I have on A, C and on C, B there which if you see is part of the database here, so I am in a position to remove this.

(Refer Slide Time: 56:45)



STRIPS Planning Example

19. Compound goal on top of stack matches data base, so remove it:

Solution: {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

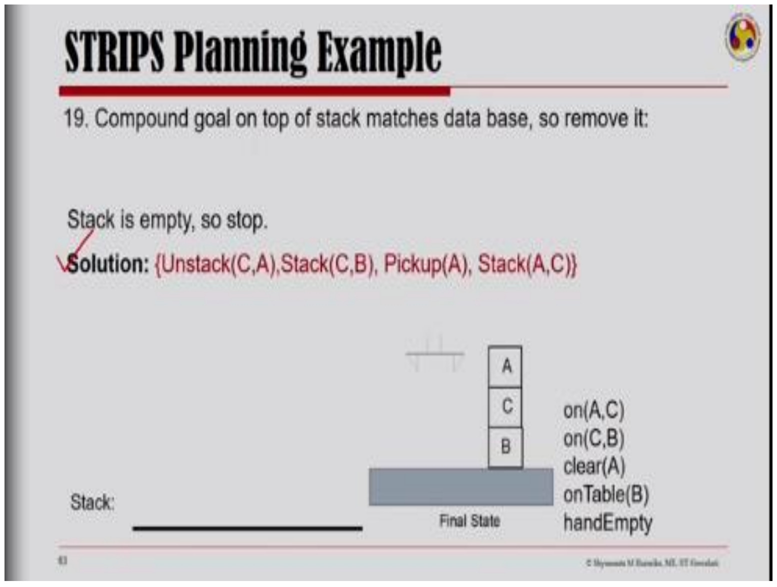
Stack: ~~on(A,C) and on(C,B)~~

Database: (unchanged)

on(A,C)
on(C,B)
clear(A)
onTable(B)
handEmpty

But before that let me clean it off first and show you that what I have here is a compound goal that matches with the database and therefore I am in a position to move it out.

(Refer Slide Time: 56:59)



STRIPS Planning Example

19. Compound goal on top of stack matches data base, so remove it:

Stack is empty, so stop.

Solution: {Unstack(C,A),Stack(C,B), Pickup(A), Stack(A,C)}

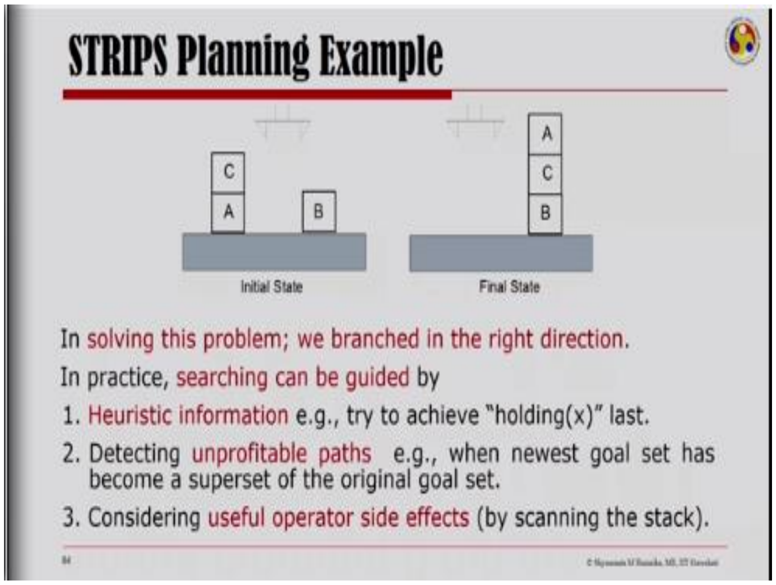
Stack:

A
C
B
Final State

on(A,C)
on(C,B)
clear(A)
onTable(B)
handEmpty

And I have an empty stack, so once I have an empty stack, I have the solution, so the solution to the problem is about unstacking C, A and the unstacking C on B, picking up A and then stacking A on top of C, so that is the final solution that we get.

(Refer Slide Time: 57:21)


C
A
B
Initial State
A
C
B
Final State

STRIPS Planning Example

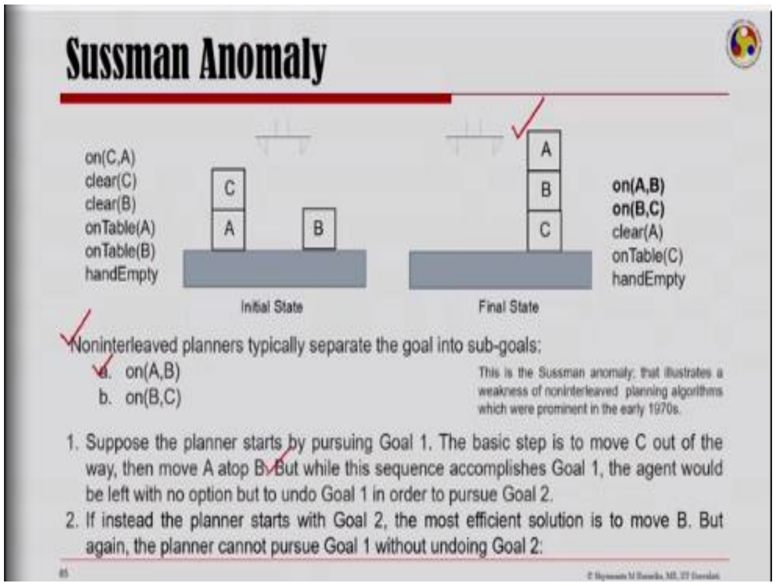
In solving this problem; we branched in the right direction.

In practice, searching can be guided by

1. Heuristic information e.g., try to achieve "holding(x)" last.
2. Detecting unprofitable paths e.g., when newest goal set has become a superset of the original goal set.
3. Considering useful operator side effects (by scanning the stack).

So the STRIPS planning example, in this example while we were solving this problem we branched in the right direction. In practice searching can be guided by heuristic information, so you could try holding x last or you could detect unprofitable paths when newest goal set has become a superset of the original goal set. Or you could consider useful operator side effects by scanning the stack.

(Refer Slide Time: 57:58)


C
A
B
Initial State
A
B
C
Final State

Sussman Anomaly

on(C,A)
clear(C)
clear(B)
onTable(A)
onTable(B)
handEmpty

on(A,B)
on(B,C)
clear(A)
onTable(C)
handEmpty

Noninterleaved planners typically separate the goal into sub-goals:

a. on(A,B)
b. on(B,C)

This is the Sussman anomaly; that illustrates a weakness of noninterleaved planning algorithms which were prominent in the early 1970s.

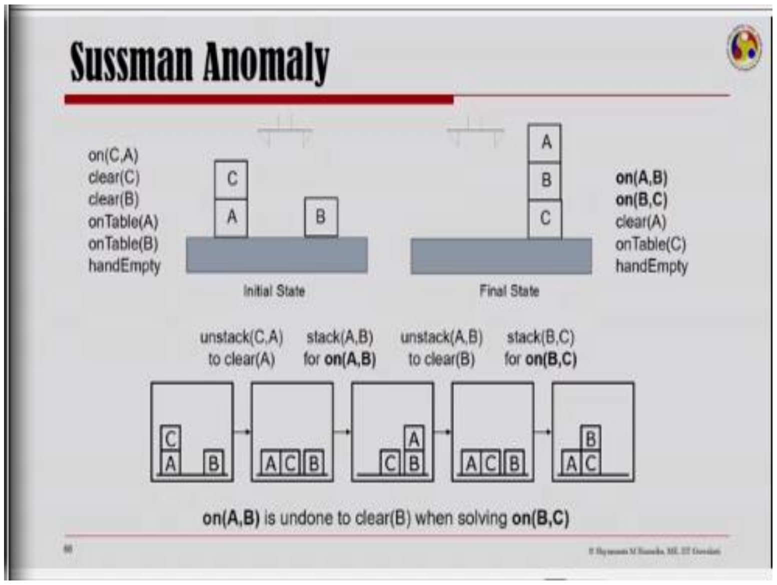
1. Suppose the planner starts by pursuing Goal 1. The basic step is to move C out of the way, then move A atop B. But while this sequence accomplishes Goal 1, the agent would be left with no option but to undo Goal 1 in order to pursue Goal 2.
2. If instead the planner starts with Goal 2, the most efficient solution is to move B. But again, the planner cannot pursue Goal 1 without undoing Goal 2:

Now let us slightly change our goal state, if you remember we had stacked B,C and A. Instead of that if we think of stacking A, B, C on top of one another then what happens is that non interleaf planners would separate these into 2 sub goals of A and B and B and C. Now this is called a

Sussman anomaly, because if the planner starts to look at goal 1, the basic step is to move C out of the way and move A atop B.

But while this sequence accomplishes goal 1, the agent will be left with no option but to undo goal 1 in order to pursue goal 2. Now if instead the planner starts with goal 2, the most efficient solution is to move B but again the planner in that case cannot pursue goal 1 without undoing goal 2. So even if I have separated it out into 2 goals satisfying one goal and then looking for the other goal would force me to undo the first goal, and this anomaly illustrates a weakness of non interleaf planning algorithms.

(Refer Slide Time: 59:30)



Sussman Anomaly

on(C,A)
clear(C)
clear(B)
onTable(A)
onTable(B)
handEmpty

Initial State

Final State

on(A,B)
on(B,C)
clear(A)
onTable(C)
handEmpty

unstack(C,A) to clear(A)

stack(A,B) for **on(A,B)**

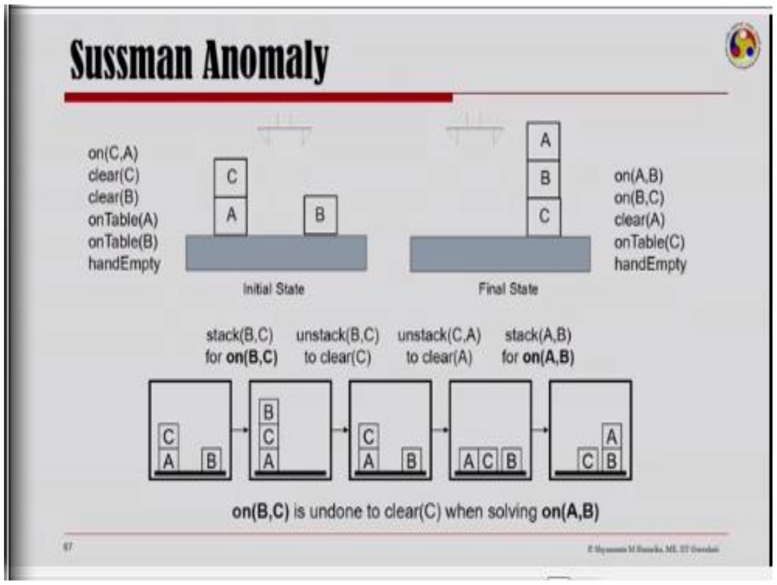
unstack(A,B) to clear(B)

stack(B,C) for **on(B,C)**

on(A,B) is undone to clear(B) when solving **on(B,C)**

So undo on A, B to clear B when solving on B, C, so let us look at that, so here is our start state. We pick up C and put it on the table and then we pick up A and put it on B but now when we have to do the other goal, then I would have to put C back on the table, this is one way of looking at goal 1 and goal 2.

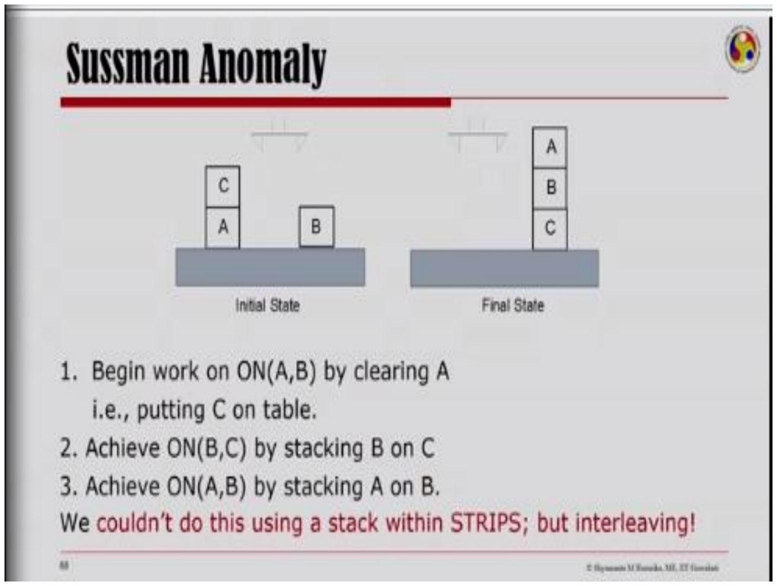
(Refer Slide Time: 1:00:04)


Sussman Anomaly

on(B,C) is undone to clear(C) when solving **on(A,B)**

And then if you look at the other way doing B, C first, so you would love to do B, C so you will stack B on top of C. But then if you want to reach the other goal of A on B you have to again unstack C, A to clear A, that means you have to take B down again and undo that goal.

(Refer Slide Time: 1:00:30)


Sussman Anomaly

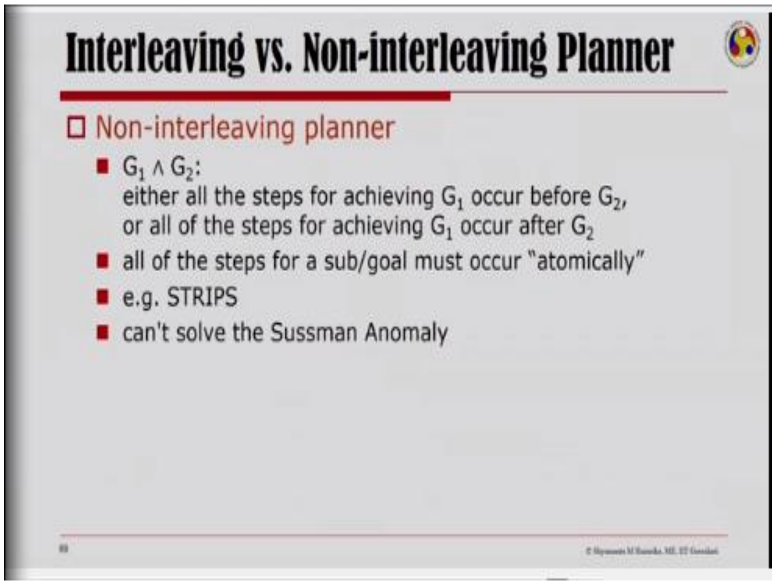
Initial State
Final State

1. Begin work on ON(A,B) by clearing A i.e., putting C on table.
2. Achieve ON(B,C) by stacking B on C
3. Achieve ON(A,B) by stacking A on B.

We couldn't do this using a stack within STRIPS; but interleaving!

So you begin work on A, B by clearing A, that is putting C on table achieve on B, C by stacking B on C, achieve on A, B by stacking A on B. We could not do this using a stack within STRIPS and this requires interleaving.

(Refer Slide Time: 1:00:52)



Interleaving vs. Non-interleaving Planner

- Non-interleaving planner
  - $G_1 \wedge G_2$:
    either all the steps for achieving G_1 occur before G_2,
    or all of the steps for achieving G_1 occur after G_2
  - all of the steps for a sub/goal must occur "atomically"
  - e.g. STRIPS
  - can't solve the Sussman Anomaly

Interleaving planners is what we will look at in our next lecture, thank you.