Fundamentals of Artificial Intelligence

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

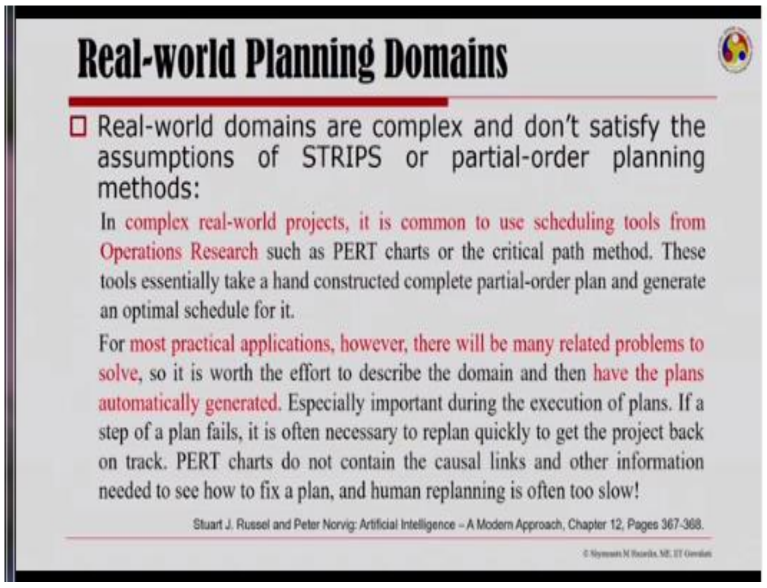
Lecture-24
Practical planning and Acting

Welcome to fundamentals of artificial intelligence, we are discussing planning today but with a slightly different perspective. So far planning and the algorithms of planning that we have looked at we had made certain simplifying assumptions that really do not hold in the practical world certain things like incomplete information or incorrect information needs to be handled by a planner for a real world application.

On the other hand, a planner can also look at a given plan to be made up of a number of sub plans and given a library of plans in principle it may be possible for the planner to get these sub plans and recreate the required plan. In this lecture today, we will particularly look at 2 algorithms that exploit such construction of plans one we call the hierarchical task network planning and in the second we would look at conditional planning.

Now real world planning domains as I have been emphasizing are complex and they do not satisfy the simplifying assumptions of STRIPS and the partial order planning that we have been discussing so far.

(Refer Slide Time: 02:34)



Real-world Planning Domains

- Real-world domains are complex and don't satisfy the assumptions of STRIPS or partial-order planning methods:

  In complex real-world projects, it is common to use scheduling tools from Operations Research such as PERT charts or the critical path method. These tools essentially take a hand constructed complete partial-order plan and generate an optimal schedule for it.

  For most practical applications, however, there will be many related problems to solve, so it is worth the effort to describe the domain and then have the plans automatically generated. Especially important during the execution of plans. If a step of a plan fails, it is often necessary to replan quickly to get the project back on track. PERT charts do not contain the causal links and other information needed to see how to fix a plan, and human replanning is often too slow!

  Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 12, Pages 367-368.

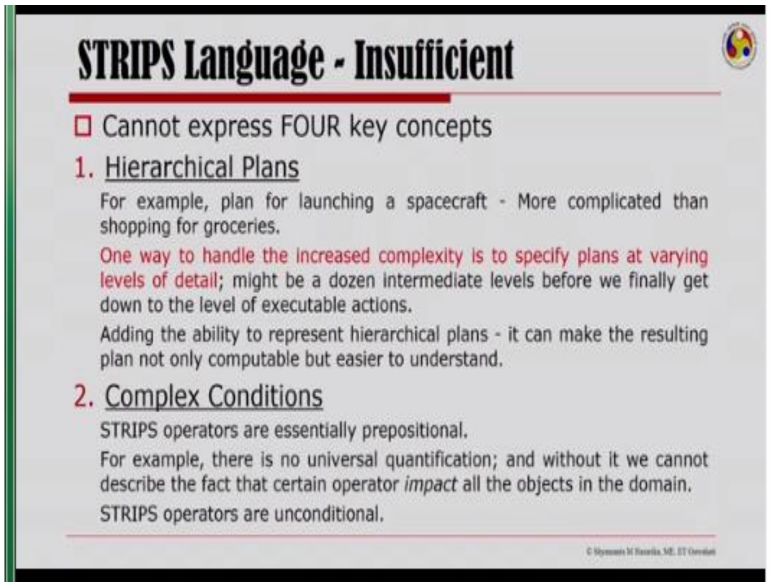
© Shyamanta M Hazarika, ME, IIT Guwahati

In fact here I would like to quote a paragraph from the Russell and Narvik book pages 367, 368 wherein it is said that in complex real-world projects it is common to use scheduling tools such as PERT charts or critical path method. Now these tools essentially take a hand constructed complete partial-order plan and generate an optimal schedule for it. It is because of the very fact that real-world domains are complex, for most practical applications however there may be related problems to solve.

And therefore it is what to describe the domain and then have the plans automatically generated. Now it is often necessary that if a plan fails one can replan quickly to get the project back and the traditional PERT charts do not contain the causal links and other information needed to see how to fix a plan. And therefore these idea of practical planning holds tremendous importance, we will today look at practical planning in terms of what is called the hierarchical task network.

But in order to look at hierarchical task network planning, it is important to realize what are the drawbacks of STRIPS like formulation or to be more precise, why is the STRIPS language insufficient. Now the STRIPS language cannot express a couple of key concepts the first among them is the very hierarchical plans.

(Refer Slide Time: 04:51)



STRIPS Language - Insufficient

- Cannot express FOUR key concepts

1. Hierarchical Plans

   For example, plan for launching a spacecraft - More complicated than shopping for groceries.

   One way to handle the increased complexity is to specify plans at varying levels of detail; might be a dozen intermediate levels before we finally get down to the level of executable actions.

   Adding the ability to represent hierarchical plans - it can make the resulting plan not only computable but easier to understand.

2. Complex Conditions

   STRIPS operators are essentially prepositional.

   For example, there is no universal quantification; and without it we cannot describe the fact that certain operator *impact* all the objects in the domain.

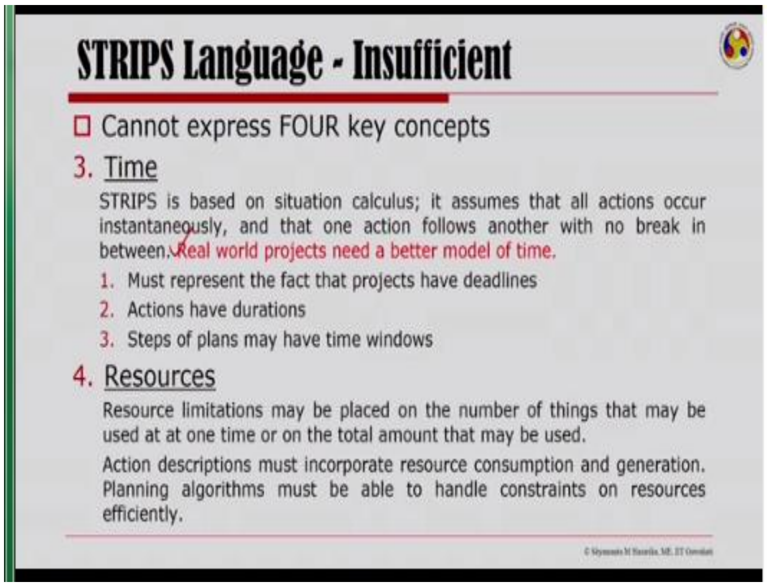
   STRIPS operators are unconditional.

For example I am talking of a plan for launching a spacecraft, now this obviously is more complicated than shopping for groceries are getting a book a cake and some other stuff from the market. One way to handle this increase complexity would be to actually specify plans at varying levels of detail and I may have a number of different levels in which I will generate plans and finally from a very abstract plan.

I would get to a level, where the plan would consist of only executable actions, now this ability to represent hierarchical plans is not there in the STRIPS language. But adding this ability to represent hierarchical plans can make the resulting plan not only computable but really easier to understand. The next key concept that is not possible to be express in STRIPS is complex conditions, now STRIP operators if you remember are essentially propositional.

For example there is no universal quantification and without universal quantification it may not be possible to describe facts that certain operators impact all objects in the domain, STRIPS operators are actually unconditional.

(Refer Slide Time: 06:31)



STRIPS Language - Insufficient

- Cannot express FOUR key concepts

3. Time

   STRIPS is based on situation calculus; it assumes that all actions occur instantaneously, and that one action follows another with no break in between. Real world projects need a better model of time.

   1. Must represent the fact that projects have deadlines
   2. Actions have durations
   3. Steps of plans may have time windows

4. Resources

   Resource limitations may be placed on the number of things that may be used at at one time or on the total amount that may be used.

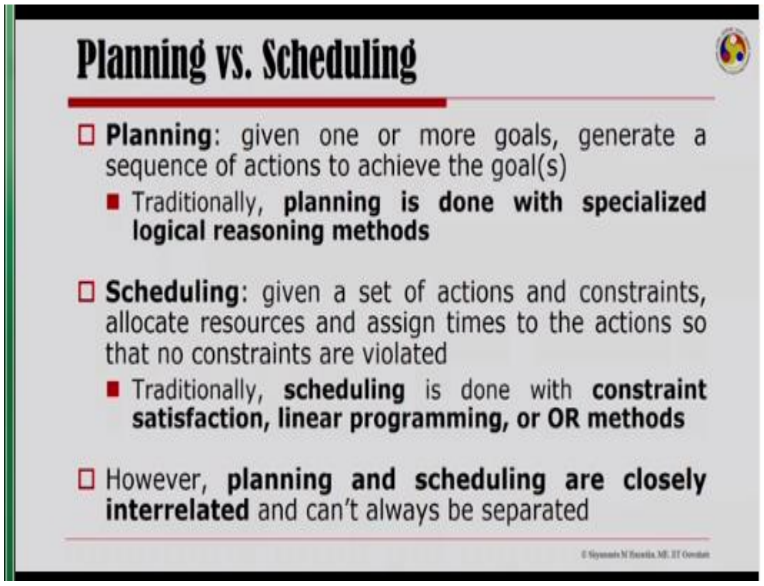
   Action descriptions must incorporate resource consumption and generation. Planning algorithms must be able to handle constraints on resources efficiently.

Now another thing that is not possible to be expressed in STRIPS is time, if you recall STRIPS is based on situation calculus. And it assumes that all actions occur instantaneously and that one action follow another and there is no break. Now this is a very rudimentary model of time any real world project would meet a better model of time and this is one very important concept that needs to be looked at if I have to move these STRIPS like planning algorithms for practical planning.

Now the time model that one need to have must represent the fact that projects have deadlines, actions have durations and steps of plans may have time windows. Another important concept which is not possible in a STRIPS language is representation of resources, now resource limitations are there in the real world and results limitations may be placed on the number of things that may be use at one time or on the total amount that may be use.

In a STRIPS like representation, it is not possible to talk of such resources, now whenever I am talking of action descriptions in a real life scenario. Action descriptions must incorporate resource consumption and resource generation, and planning algorithms must be able to handle constraints on resources efficiently. This could be resources as mundane as money for the grocery planner or it could be very important resource of time itself.

(Refer Slide Time: 08:39)



Planning vs. Scheduling

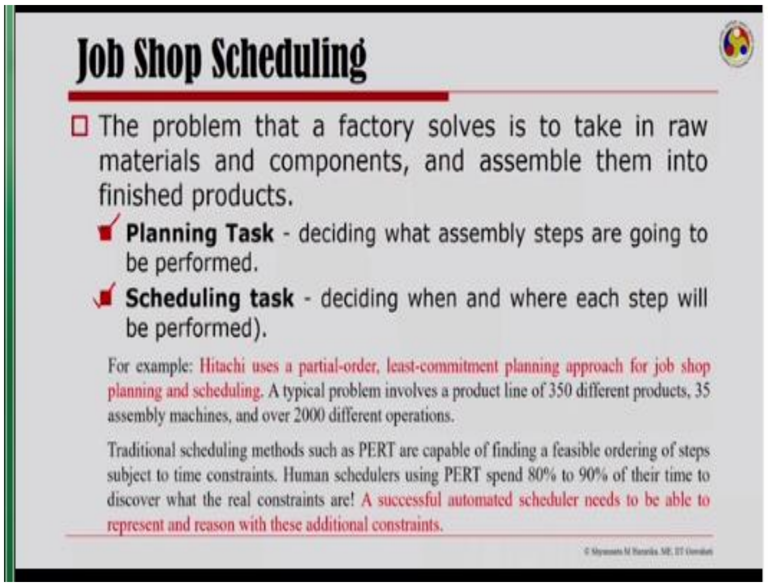
- **Planning**: given one or more goals, generate a sequence of actions to achieve the goal(s)
  - Traditionally, **planning is done with specialized logical reasoning methods**
- **Scheduling**: given a set of actions and constraints, allocate resources and assign times to the actions so that no constraints are violated
  - Traditionally, **scheduling** is done with **constraint satisfaction, linear programming, or OR methods**
- However, **planning and scheduling are closely interrelated** and can't always be separated

© Siyamanta N Hazarika, ME, IIT Guwahati

Now if you look at planning vis-a-vis scheduling then planning as we have seen up till now is about having one or more goals and then generating a sequence of actions to achieve the goal. So traditionally planning is done with specialized logical reasoning methods, whereas scheduling is about having a set of actions and constraints and allocating resources and assigning time frames to the actions.

So that no constraints are violated, traditionally scheduling is done with constraint satisfaction, linear programming or other operation research methods. However planning and scheduling are closely interrelated and in the real world scenario one needs to realize thus it is very difficult to have them in 2 separate compartments, one needs to look at planning and scheduling together.

(Refer Slide Time: 09:50)


Job Shop Scheduling

- The problem that a factory solves is to take in raw materials and components, and assemble them into finished products.
  - **Planning Task** - deciding what assembly steps are going to be performed.
  - **Scheduling task** - deciding when and where each step will be performed).

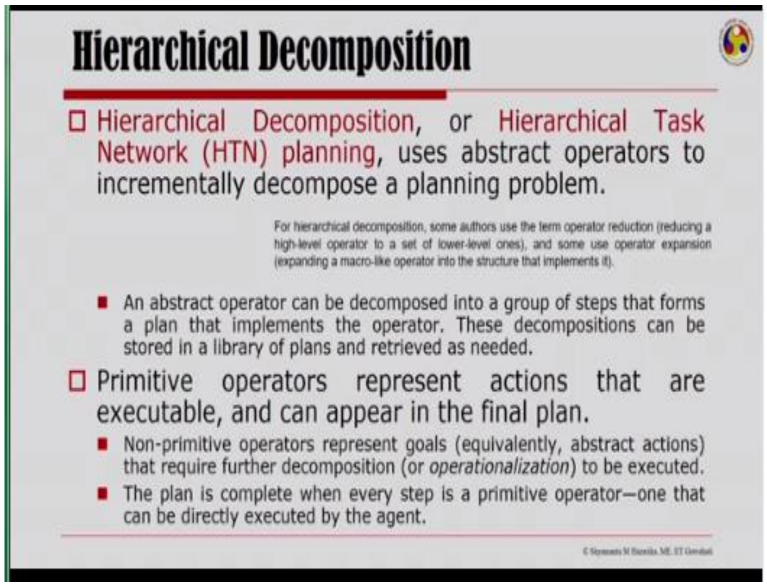
For example: Hitachi uses a partial-order, least-commitment planning approach for job shop planning and scheduling. A typical problem involves a product line of 350 different products, 35 assembly machines, and over 2000 different operations.

Traditional scheduling methods such as PERT are capable of finding a feasible ordering of steps subject to time constraints. Human schedulers using PERT spend 80% to 90% of their time to discover what the real constraints are! A successful automated scheduler needs to be able to represent and reason with these additional constraints.

So we will talk of a job shop scheduling problem, the problem that a factory solves is to take in raw materials and components and assemble them into finished products. So for such a factory, the planning task is about deciding what assembly steps are going to be perform, and the scheduling task is about deciding when and where each step will be performed. Now just for our own interest it is important to know that companies like Hitachi use a partial order, least commitment, planning approach for job shop planning and scheduling.

And in Hitachi a typical problem involves a product line of some 350 different products, 35 assembly machines and over 2000 different operations. Traditional scheduling methods such as PERT are capable of finding a feasible ordering of steps subject to time constraints. And this is where a automated scheduling algorithm is required, where it would represent and reason with additional constraints over and above those that are therefore.

(Refer Slide Time: 11:28)



Hierarchical Decomposition

- Hierarchical Decomposition, or Hierarchical Task Network (HTN) planning, uses abstract operators to incrementally decompose a planning problem.

  For hierarchical decomposition, some authors use the term operator reduction (reducing a high-level operator to a set of lower-level ones), and some use operator expansion (expanding a macro-like operator into the structure that implements it).

  - An abstract operator can be decomposed into a group of steps that forms a plan that implements the operator. These decompositions can be stored in a library of plans and retrieved as needed.
- Primitive operators represent actions that are executable, and can appear in the final plan.
  - Non-primitive operators represent goals (equivalently, abstract actions) that require further decomposition (or *operationalization*) to be executed.
  - The plan is complete when every step is a primitive operator—one that can be directly executed by the agent.

© Shyamanta M Hazarika, ME, IIT Guwahati

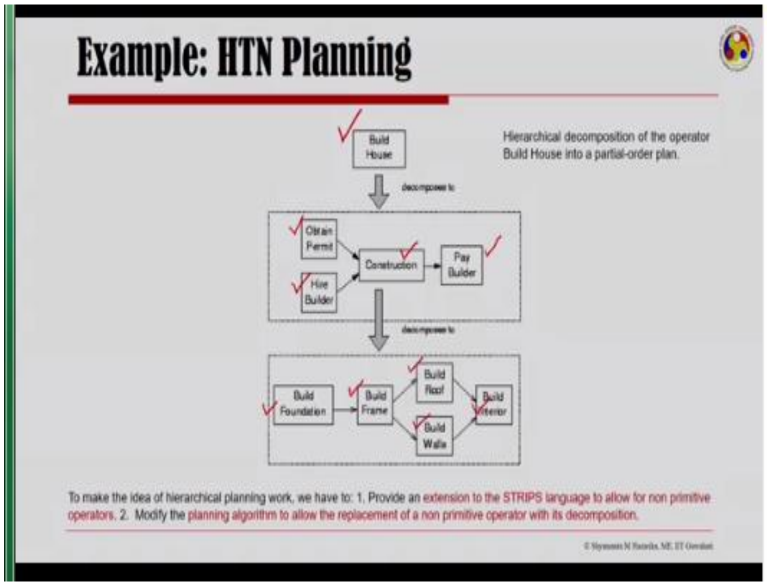
So we will look at hierarchical decomposition here today, hierarchical decomposition or hierarchical task network planning is when you look at a given task at an abstract level. And then you can look at that task being made up of sub tasks and go on decomposing this, until you are able to get down to actions. And then you have a plan involving the actions, so here you use abstract operators to incrementally decompose a planning problem.

For hierarchical decomposition, there are a number of terms that people use some authors use the term operator reduction that is about reducing a high level operator to a set of low level ones. And some use operator expansion that is look at expanding a macro like operator into the structure, that implements a that is at the level of the actions. An abstract operator in a hierarchical task network planning can be decomposed into a group of steps that forms a plan and that implements the operator.

Now as I was telling you in the very beginning of this lecture, these decompositions then possibly could be stored in a library of plans and then one could retrieve them as needed to actually construct the complete plan. Primitive operators in such a hierarchical task network actually represent actions that are executable and as you can make out now, as I keep on decomposing the different operators.

I finally reach to the final level of the plan which are the primitive operators, so in a sense the non primitive operators represent goals that is abstract actions that require further decomposition. And the plan is complete when every step is a primitive operator, one that can be directly executed by the agent.

(Refer Slide Time: 14:09)



Example: HTN Planning

Hierarchical decomposition of the operator Build House into a partial-order plan.

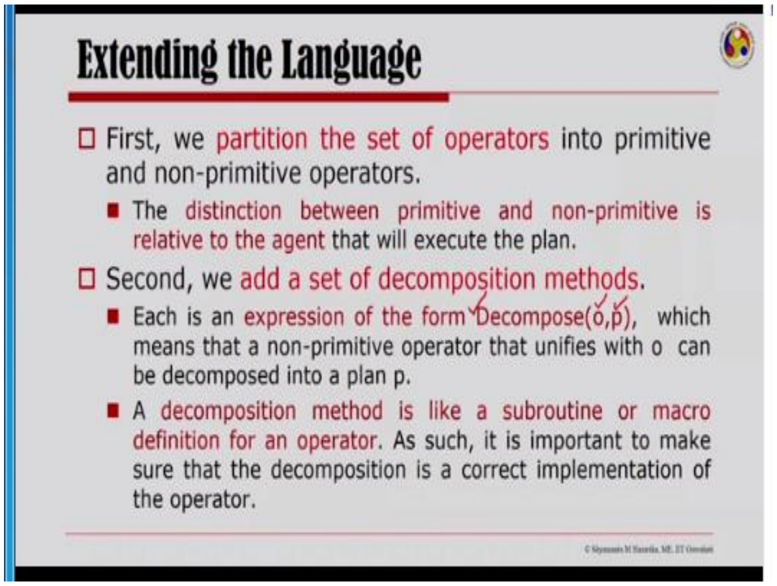
To make the idea of hierarchical planning work, we have to: 1. Provide an extension to the STRIPS language to allow for non primitive operators. 2. Modify the planning algorithm to allow the replacement of a non primitive operator with its decomposition.

Now let us look at what we mean by the hierarchical task network planning for plan to build a house. Now if I am talking of a plan to build a house, my plan could be as abstract as build house and the hierarchical decomposition of the operator now, which is the most abstract operator build house could be to decompose this into a partial order plan, where I am thinking of obtaining the permit and hiring a builder thereafter doing construction and finally paying the builder.

Now as you can see this plan that I am trying to build the most abstract operator that I could think of was build house and I could decompose it into construction and obtaining permit hiring builder and paying the builder. Now a little thought will let you see that the construction operator could again be decompose to it is sub plans like build a foundation, build a frame. And then I can think of building the roof and the walls and thereafter building the interior.

So to make the idea of hierarchical planning work, we need to provide an extension to the STRIPS language. Now this extension needs to allow non primitive operators and I should be able to modify the planning algorithm, to allow the replacement of non primitive operators with it is decomposition.

(Refer Slide Time: 16:01)



Extending the Language

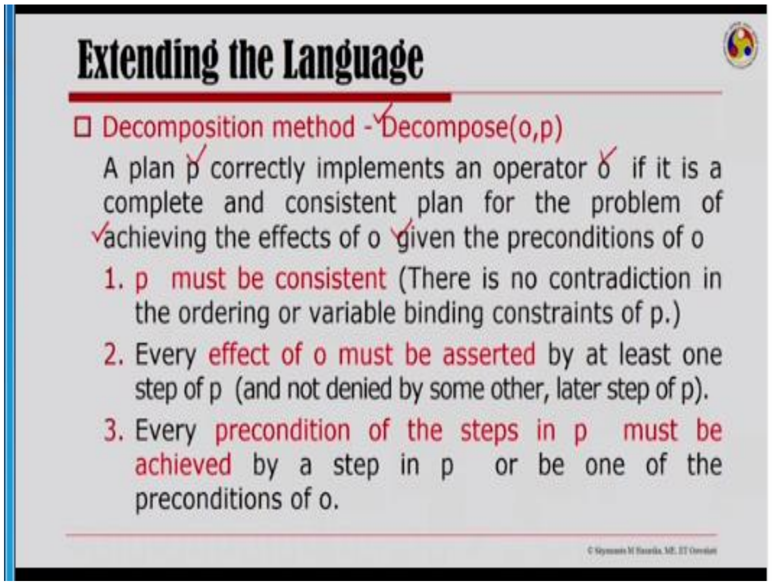
- First, we partition the set of operators into primitive and non-primitive operators.
  - The distinction between primitive and non-primitive is relative to the agent that will execute the plan.
- Second, we add a set of decomposition methods.
  - Each is an expression of the form Decompose(o,p), which means that a non-primitive operator that unifies with o can be decomposed into a plan p.
  - A decomposition method is like a subroutine or macro definition for an operator. As such, it is important to make sure that the decomposition is a correct implementation of the operator.

© Shyamanta M Hazarika, ME, IIT Guwahati

So in terms of extending the language first we would need to partition the set of operators that we have to deal with and the partition could be into primitive and non primitive operators. Now the distinction between primitive and non primitive operators is actually relative to the agent that will execute the plan. Second we would need to add a set of decomposition methods, now these decomposition methods would be some expression of the form decompose o, p which means that I have a non primitive operator that unifies with o and I can decompose into a plan p.

Now a decomposition method is actually something like a substitution for the original operator in terms of it is sub routine or you can look at it as a macro definition for an operator. So as such it is really important to make sure that the decomposition is a correct implementation of the operator.

(Refer Slide Time: 17:24)



Extending the Language

- Decomposition method - Decompose(o,p)

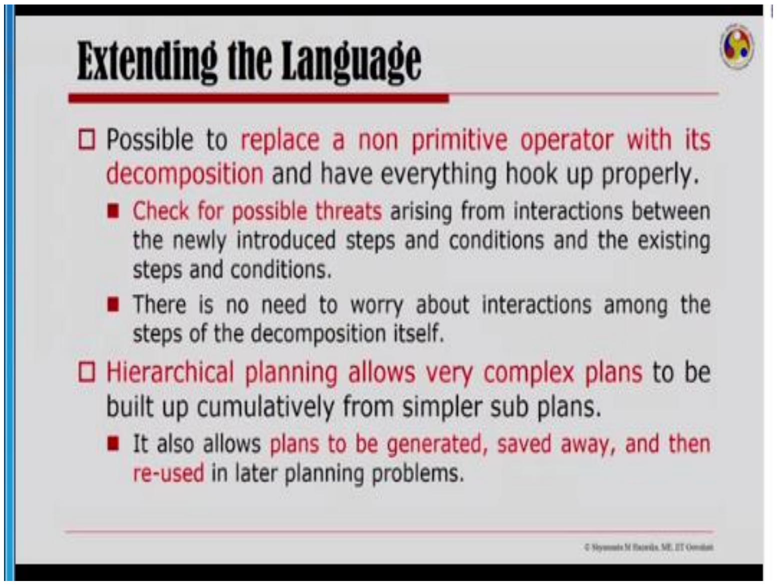
  A plan p correctly implements an operator o if it is a complete and consistent plan for the problem of achieving the effects of o given the preconditions of o

  1. p must be consistent (There is no contradiction in the ordering or variable binding constraints of p.)
  2. Every effect of o must be asserted by at least one step of p (and not denied by some other, later step of p).
  3. Every precondition of the steps in p must be achieved by a step in p or be one of the preconditions of o.

So the decomposition method that we are talking of has an operator o and a plan p needs to correctly implement that operator. Now this decomposition that I have, I end up with having a plan, now a plan p will correctly implement an operator o, if it is complete and consistent plan for the problem of achieving the effects of o. That means that first when I am looking for the plan to replace the operator, I should be looking for a plan that achieves the effects of the operator given the preconditions of the operator.

And thereafter I should look for if p is consistent, now p being consistent means that there is no contradiction in the ordering or variable binding constraints of p and every effect of o must be asserted by at least one step of p and not denied by some other step or later step of . Finally every precondition of the steps in p must be achieved by a step in p or p one of the preconditions of o. Then I say that the plan p is complete and consistent for the operator o.

(Refer Slide Time: 19:01)

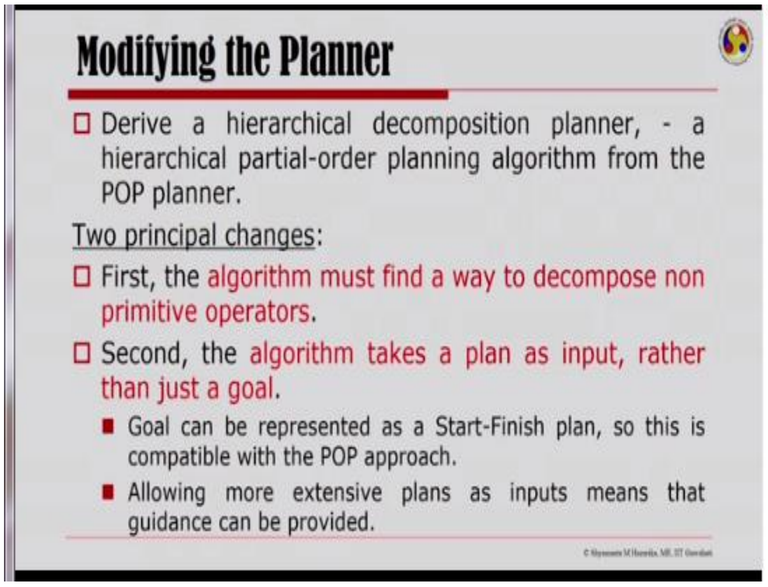

Extending the Language

- Possible to replace a non primitive operator with its decomposition and have everything hook up properly.
  - Check for possible threats arising from interactions between the newly introduced steps and conditions and the existing steps and conditions.
  - There is no need to worry about interactions among the steps of the decomposition itself.
- Hierarchical planning allows very complex plans to be built up cumulatively from simpler sub plans.
  - It also allows plans to be generated, saved away, and then re-used in later planning problems.

Now it is possible to replace a non primitive operator with it is decomposition and have everything hook up properly. All one needs to do is check for possible trends arising from interactions between the newly introduce steps and conditions. Now there could be interactions among the steps of the decomposition itself but then one need not worry about these interactions for if I am talking of replacing a non primitive operator with a decomposition.

We will be actually looking at putting the whole plan together and therefore we need not worry about interactions among steps of the decomposition itself. Hierarchical planning allows very complex plans to be built up from simpler sub plans. And as I was referring to it allows plans to be generated and saved in a library of plans, so that it can be used and reused later in the planning problem.

(Refer Slide Time: 20:12)



Modifying the Planner

- Derive a hierarchical decomposition planner, - a hierarchical partial-order planning algorithm from the POP planner.

Two principal changes:

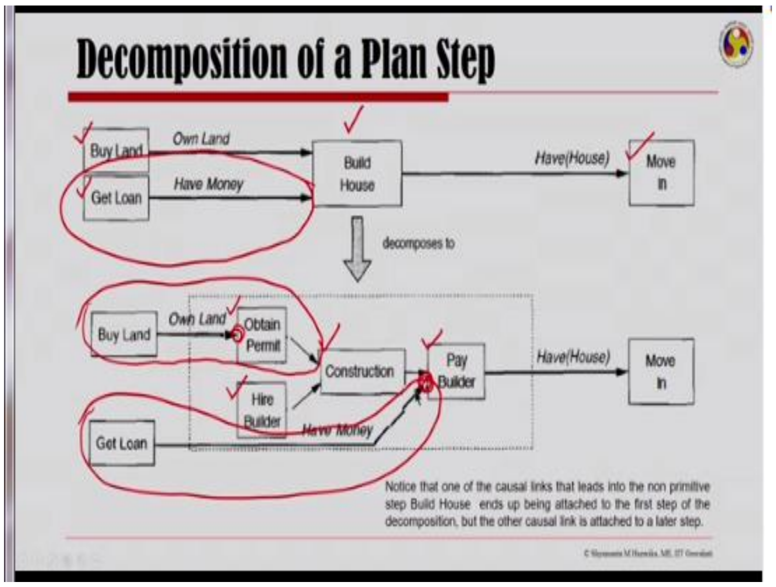
- First, the algorithm must find a way to decompose non primitive operators.
- Second, the algorithm takes a plan as input, rather than just a goal.
  - Goal can be represented as a Start-Finish plan, so this is compatible with the POP approach.
  - Allowing more extensive plans as inputs means that guidance can be provided.

s

Now in terms of modifying the planner, a hierarchical decomposition planner is a hierarchical partial-order planning algorithm. So one needs to look at how to get to depth from the partial-order planner 2 principle changes are required. First, the algorithm itself must find a way to decompose non primitive operators, and second it must take a plan as input rather than just a goal, now if you remember goal can be represented as a start finish plan.

So this is compatible with the partial-order planning approach allowing more extensive plans as input means that the guidance can be provided and with these 2 modifications a hierarchical decomposition planner could be arrived at from the partial-order planner.

(Refer Slide Time: 21:17)



Decomposition of a Plan Step

Notice that one of the causal links that leads into the non primitive step Build House ends up being attached to the first step of the decomposition, but the other causal link is attached to a later step.

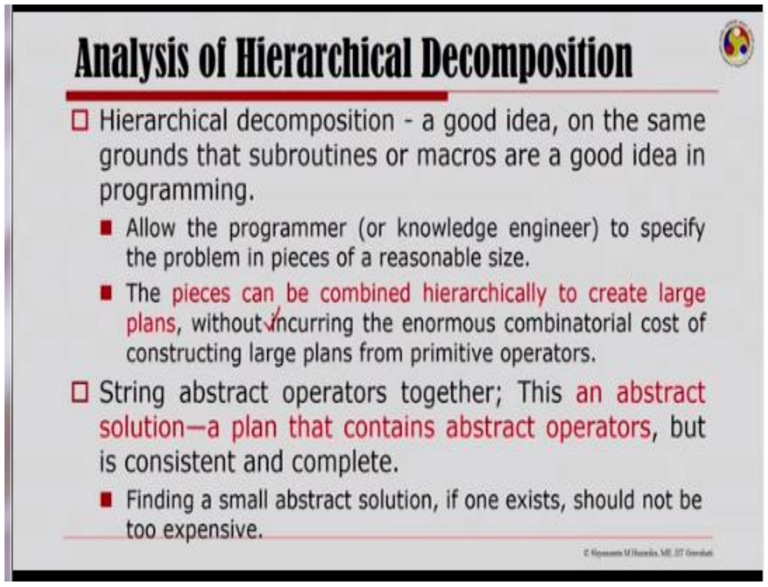
So here is the decomposition of a plan step and how you go about making all things fall in it is place. So if you remember we were talking of having an operator called build house which is an abstract operator, in order to have that one needs to own land and have money. So in order to own land you need to buy land and in order to have money you need to get a loan. And once you have build house you would have a house, where you can move in.

Now the build house is an abstract operator and could be decomposed to a partial-order plan like I could think of obtaining a permit and think of hiring a builder and having construction being done paid a builder and then move in to the house. But now if you notice that one of the causal links to the non primitive step here, this causal link which was coming from get loan to the operator build house, I have attached to the next later state of the decomposition.

Here rather than to the front of the decomposition, as I have done the causal link for obtaining permit to the front of the decomposition. So that is what I meant when I said that when we are decomposing a abstract operator into it is more lower sub plans. Then the causal links will fall in it is own place and here one of the causal links that lead to the non primitive build house ends up being attached to the first step of the decomposition.

And the other one is actually at a later step, so this causal link gets attached here and the other causal link gets attached here.

(Refer Slide Time: 23:44)



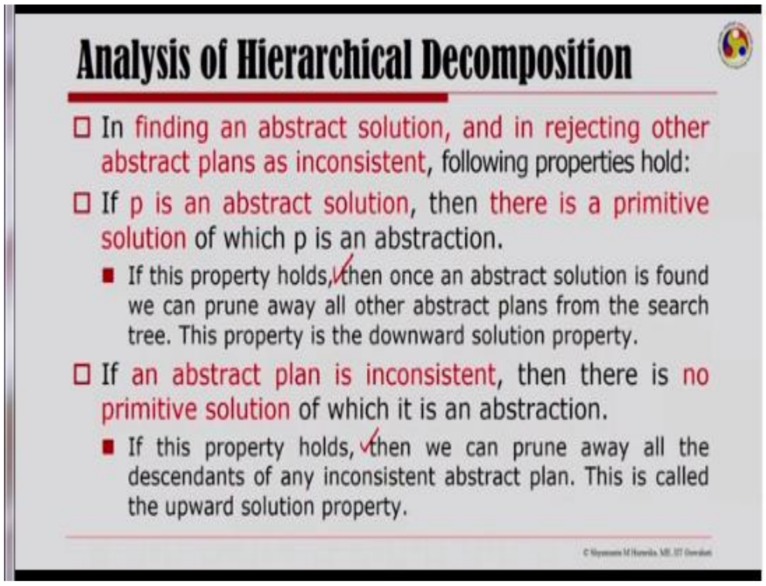
Analysis of Hierarchical Decomposition

- Hierarchical decomposition - a good idea, on the same grounds that subroutines or macros are a good idea in programming.
  - Allow the programmer (or knowledge engineer) to specify the problem in pieces of a reasonable size.
  - The pieces can be combined hierarchically to create large plans, without incurring the enormous combinatorial cost of constructing large plans from primitive operators.
- String abstract operators together; This an abstract solution—a plan that contains abstract operators, but is consistent and complete.
  - Finding a small abstract solution, if one exists, should not be too expensive.

The hierarchical decomposition is a good idea, on the same grounds that subroutines or macros are good ideas in programming. And it is interesting that the pieces that can be combined hierarchically to create large plans can be done without incurring enormous combinatorial cost of constructing large plans from primitive operators. Now if I was thinking of only getting to a very large plan in terms of very primitive operators.

The idea of using hierarchical decomposition to create large plans from breaking these non primitives 1 step at a time can be done more efficiently. Now the idea is how does 1 string abstract operators together, an abstract solution to the planning problem is what would contain the abstract operators. Now once we have looked at an abstract operation finding an abstract solution if one exists should not be too expensive.

(Refer Slide Time: 25:00)



Analysis of Hierarchical Decomposition

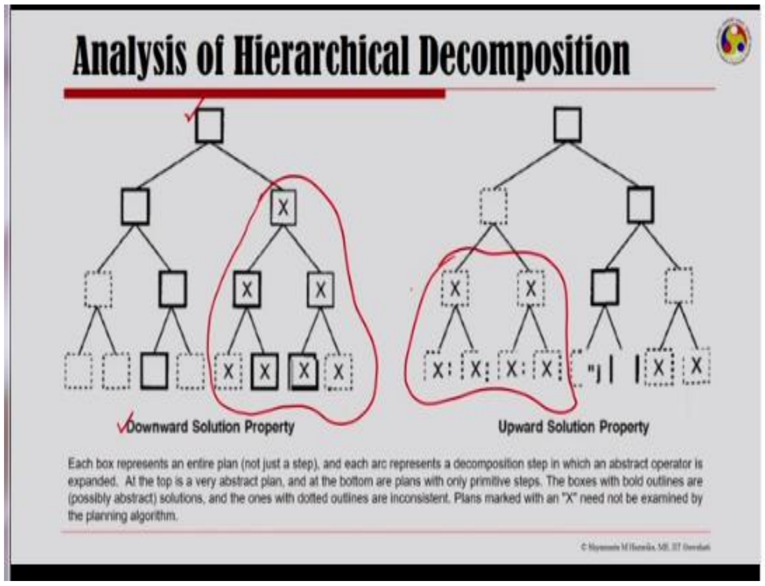
- In finding an abstract solution, and in rejecting other abstract plans as inconsistent, following properties hold:
- If p is an abstract solution, then there is a primitive solution of which p is an abstraction.
  - If this property holds, then once an abstract solution is found we can prune away all other abstract plans from the search tree. This property is the downward solution property.
- If an abstract plan is inconsistent, then there is no primitive solution of which it is an abstraction.
  - If this property holds, then we can prune away all the descendants of any inconsistent abstract plan. This is called the upward solution property.

© Shyamanta M Hazarika, ME, IIT Guwahati

Finding an abstract solution and then rejecting other abstract plans as inconsistent the following properties hold. If p is an abstract solution then there is a primitive solution of which p is an abstraction, so if this property holds then once an abstract solution is found we can prune away all other abstract plans from the search tree, this property is the downward solution property. If an abstract plan is inconsistent then there is no primitive solution of which it is an abstraction.

Now if this property holds then we can prune away descendants of any inconsistent abstract plan and this is called the upward solution property.

(Refer Slide Time: 25:55)



Analysis of Hierarchical Decomposition

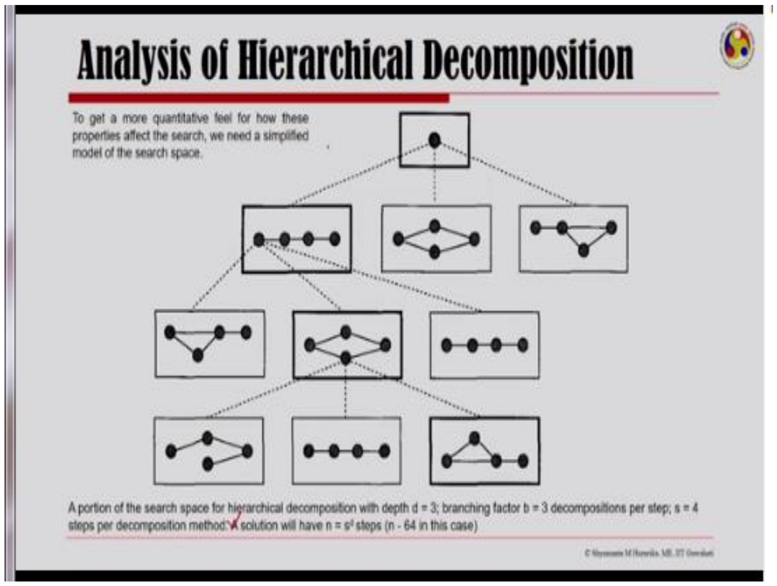
Each box represents an entire plan (not just a step), and each arc represents a decomposition step in which an abstract operator is expanded. At the top is a very abstract plan, and at the bottom are plans with only primitive steps. The boxes with bold outlines are (possibly abstract) solutions, and the ones with dotted outlines are inconsistent. Plans marked with an "X" need not be examined by the planning algorithm.

© Shyamanta M Hazarika, ME, IIT Guwahati

So here is the downward solution property, in here each box that I have shown represents an entire plan and each arc represents a decomposition step in which an abstract operator is expanded. At the top we have the abstract plan and at the bottom are plans with only primitive steps, now the boxes that I have with bold outlines are possibly abstract solutions and dotted outlines are inconsistent.

So the plans which are marked with an X need not be examined by a planning algorithm any more. Now we can see that here we have the downward solution property, where we see that given a abstract solution and we can prune away all other abstract plan from the search tree and in the upward solution property we can prune away all descendants of an inconsistent plan, so here we could prune away all of this.

(Refer Slide Time: 27:21)



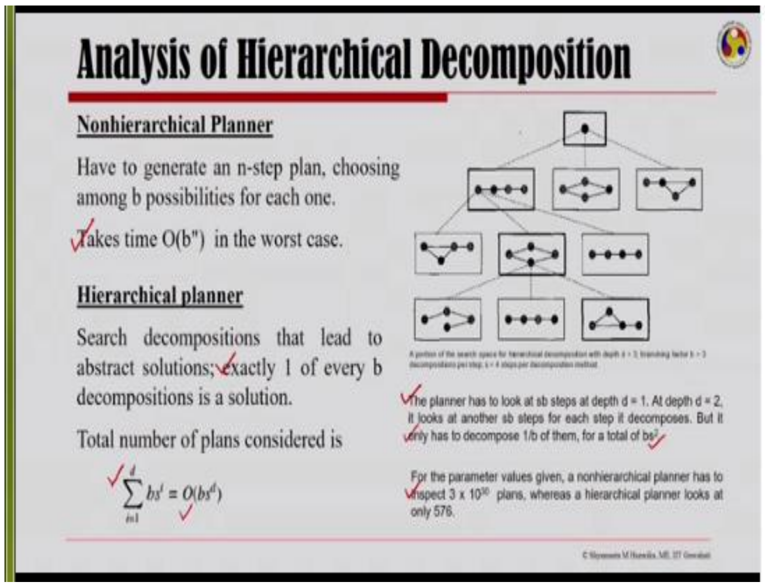
Analysis of Hierarchical Decomposition

To get a more quantitative feel for how these properties affect the search, we need a simplified model of the search space.

A portion of the search space for hierarchical decomposition with depth d = 3; branching factor b = 3 decompositions per step; s = 4 steps per decomposition method: A solution will have $n = s^d$ steps (n - 64 in this case)

To get a more qualitative feel for how these properties affect the search, we need a simplified model of the search space. So here I show a portion of the search space for hierarchical decomposition with depth d = 3 branching factor of 3 decompositions per step and 4 steps per decomposition method, then a solution will have s to the power d steps and in our case n is 64.

(Refer Slide Time: 27:58)



Analysis of Hierarchical Decomposition

Nonhierarchical Planner

Have to generate an n-step plan, choosing among b possibilities for each one.

Takes time $O(b^n)$ in the worst case.

Hierarchical planner

Search decompositions that lead to abstract solutions; exactly 1 of every b decompositions is a solution.

Total number of plans considered is

$$\sum_{i=1}^{d} bs^i = O(bs^d)$$

A portion of the search space for hierarchical decomposition with depth d = 3, branching factor b = 3 decompositions per step, s = 4 steps per decomposition method

The planner has to look at sb steps at depth d = 1. At depth d = 2, it looks at another sb steps for each step it decomposes. But it only has to decompose 1/b of them, for a total of bs^2.

For the parameter values given, a nonhierarchical planner has to inspect 3×10^{30} plans, whereas a hierarchical planner looks at only 576.

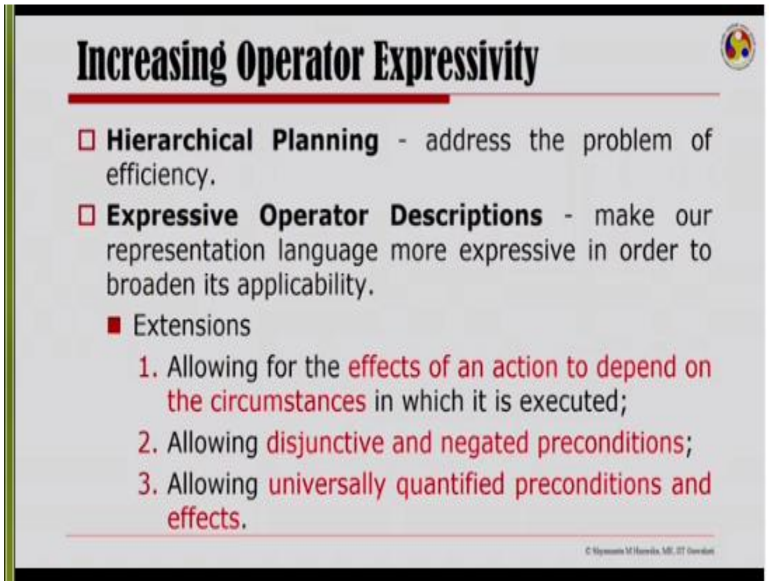
© Shyamanta M Hazarika, ME, IIT Guwahati

Now if it is a non hierarchical planner, then I have to generate these n-step plan choosing among b possibilities for each one of them. And then this would take some time order of b to the power n in the worst case, whereas if it is a hierarchical planner then the search decomposition would lead to abstract solutions . We can think of exactly 1 of every b decomposition is a solution and therefore the total number of plans to be considered would be summation of sum b s to the power d.

Now the planner has to look at if you see s b steps at depth 1, at depth 2 it looks at s b steps for each it is s square b square, however it only has to decompose 1 b of them, so the total is b s square. And therefore the total number of plans considered is of the order of b s to the power d. Now for the parameters given here, if I was talking of a non hierarchical planner, the non-hierarchical planner would have to inspect some 3 into 10 to the power 30 plans.

Whereas a hierarchical planner would look at only 576 plans and this is what is a tremendous saving for hierarchical task network planning.

(Refer Slide Time: 29:45)

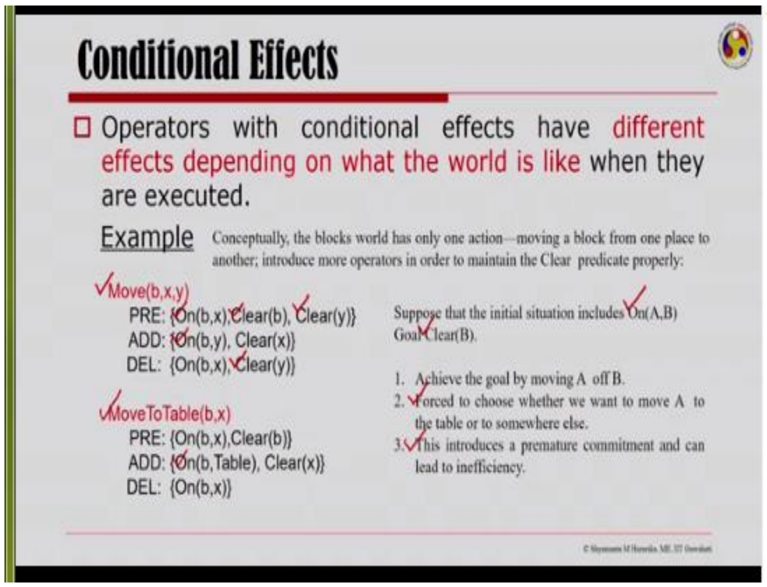


Increasing Operator Expressivity

- **Hierarchical Planning** - address the problem of efficiency.
- **Expressive Operator Descriptions** - make our representation language more expressive in order to broaden its applicability.
  - Extensions
    1. Allowing for the effects of an action to depend on the circumstances in which it is executed;
    2. Allowing disjunctive and negated preconditions;
    3. Allowing universally quantified preconditions and effects.

Now few other possibilities in terms of STRIPS language representation itself is about increasing the operator expressivity. Hierarchical planning addresses the problem of efficiency, whereas expressive operator descriptions would make our representation language more expressive, in order to broaden it is applicability. And we can have 3types of extensions, 1 allowing for the effects of an action to depend on the circumstances in which it is executed, 2 allowing disjunctive and negated preconditions and 3 allowing universally quantified preconditions and effects.

(Refer Slide Time: 30:39)



Conditional Effects

- Operators with conditional effects have different effects depending on what the world is like when they are executed.

Example Conceptually, the blocks world has only one action—moving a block from one place to another; introduce more operators in order to maintain the Clear predicate properly:

✓Move(b,x,y)
PRE: {On(b,x),Clear(b), Clear(y)}
ADD: {On(b,y), Clear(x)}
DEL: {On(b,x),Clear(y)}

✓MoveToTable(b,x)
PRE: {On(b,x),Clear(b)}
ADD: {On(b,Table), Clear(x)}
DEL: {On(b,x)}

Suppose that the initial situation includes On(A,B)
Goal Clear(B).

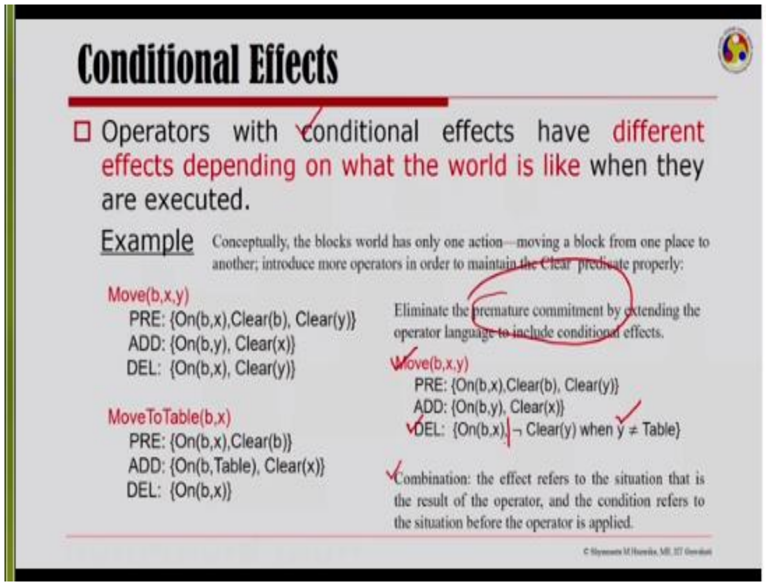
1. Achieve the goal by moving A off B.
2. Forced to choose whether we want to move A to the table or to somewhere else.
3. This introduces a premature commitment and can lead to inefficiency.

Now conditional effects is about operators which have different effects depending on what the world is like when they are executed. For example conceptually if we are talking of the blocks world, then it has only one action it is about moving a block from one place to another. And we introduce more operators in the blocks world, in order to maintain the clear predicate properly, for example I have 2 operators here 1 move b x, y that is move b from x onto y.

And here it is about moving b from on top of x to the table, here when I say on b, x, I mean that b is on and there is nothing in top of b and there is nothing in top of y. So I could move b to y and when I move b to y, I no longer have y clear. Whereas when I move b to the table I do not need to worry about the table being clear or not. Now suppose that the initial situation includes A on B and the goal is to have clear B then I achieve the goal by moving a of B.

Now somehow I am force to choose, whether we want to move A to the table or to somewhere else and this you should be able to realize introduce as premature commitment and can lead to inefficiency.

(Refer Slide Time: 32:43)



Conditional Effects

- Operators with conditional effects have different effects depending on what the world is like when they are executed.

Example Conceptually, the blocks world has only one action—moving a block from one place to another; introduce more operators in order to maintain the Clear predicate properly:

```
Move(b,x,y)
    PRE: {On(b,x),Clear(b), Clear(y)}
    ADD: {On(b,y), Clear(x)}
    DEL: {On(b,x), Clear(y)}

MoveToTable(b,x)
    PRE: {On(b,x),Clear(b)}
    ADD: {On(b,Table), Clear(x)}
    DEL: {On(b,x)}
```

Eliminate the premature commitment by extending the operator language to include conditional effects.

```
Move(b,x,y)
    PRE: {On(b,x),Clear(b), Clear(y)}
    ADD: {On(b,y), Clear(x)}
    DEL: {On(b,x), ¬ Clear(y) when y ≠ Table}
```

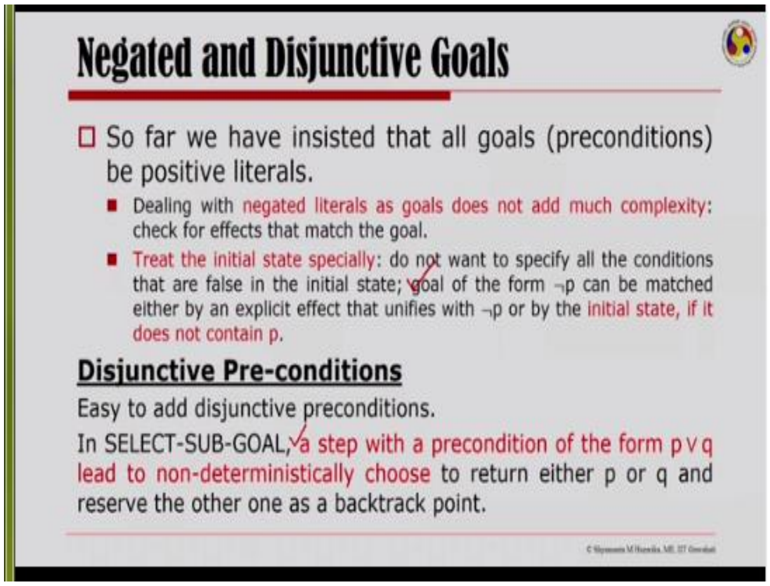
Combination: the effect refers to the situation that is the result of the operator, and the condition refers to the situation before the operator is applied.

© Shyamanta M Hazarika, ME, IIT Guwahati

On the other hand if I combine these 2 operators somehow using a conditional effects that is I still talk of moving b from x onto y and add somewhere here that I do not need to have the fact that b is no longer clear to be in the delete list when y is not the table. Then I can see that the effects refer to the situation that the result of the operator and the condition refers to the situation before the operator is applied.

So I can eliminate premature commitment, how did I eliminate the premature commitment I eliminated the premature commitment by bringing in the conditional effects. So instead of just adding that once I move b onto y, y becomes not clear and I should have it in delete list but then I said that why still remains clear if I only need to have it in the delete list if y is not the table. And therefore I need to have it in the delete list that says that y is no longer clear.

(Refer Slide Time: 34:14)



Negated and Disjunctive Goals

- So far we have insisted that all goals (preconditions) be positive literals.
  - Dealing with negated literals as goals does not add much complexity: check for effects that match the goal.
  - Treat the initial state specially: do not want to specify all the conditions that are false in the initial state; goal of the form ¬p can be matched either by an explicit effect that unifies with ¬p or by the initial state, if it does not contain p.

Disjunctive Pre-conditions

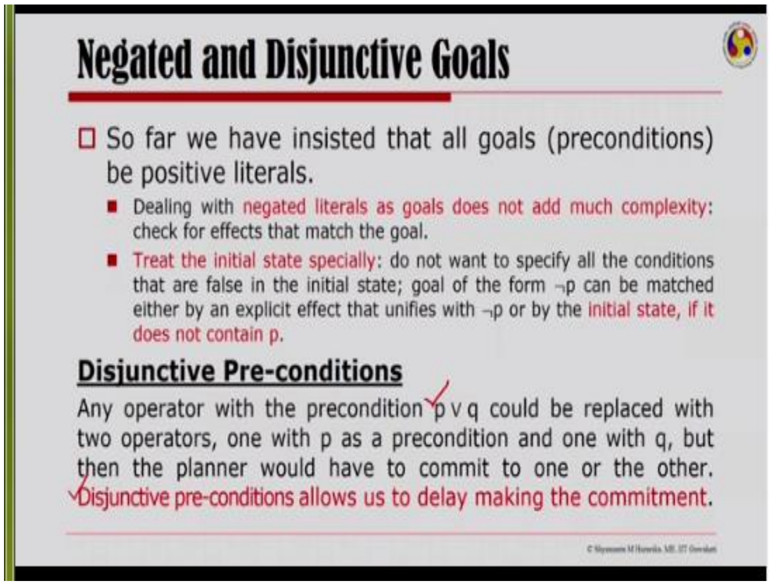
Easy to add disjunctive preconditions.

In SELECT-SUB-GOAL, a step with a precondition of the form p ∨ q lead to non-deterministically choose to return either p or q and reserve the other one as a backtrack point.

And then I have negated and disjunctive goals, so far we have insisted that all goals preconditions be positive literals. Now dealing with negative or negative literals as goals does not add much complicity, I just check for effects that match the goal. So treat the initial states specially do not want to specify all conditions that are false. So I should have goal of the form not p either by an explicit effect that unifies with not p or by the initial state if it does not contain p.

So we have disjunctive preconditions it is easy to add disjunctive preconditions a step with precondition of the form p or q will lead to non-deterministically choose to return either p or q and reserve the other as a backtrack point.

(Refer Slide Time: 35:13)



Negated and Disjunctive Goals

- So far we have insisted that all goals (preconditions) be positive literals.
  - Dealing with negated literals as goals does not add much complexity: check for effects that match the goal.
  - Treat the initial state specially: do not want to specify all the conditions that are false in the initial state; goal of the form ¬p can be matched either by an explicit effect that unifies with ¬p or by the initial state, if it does not contain p.

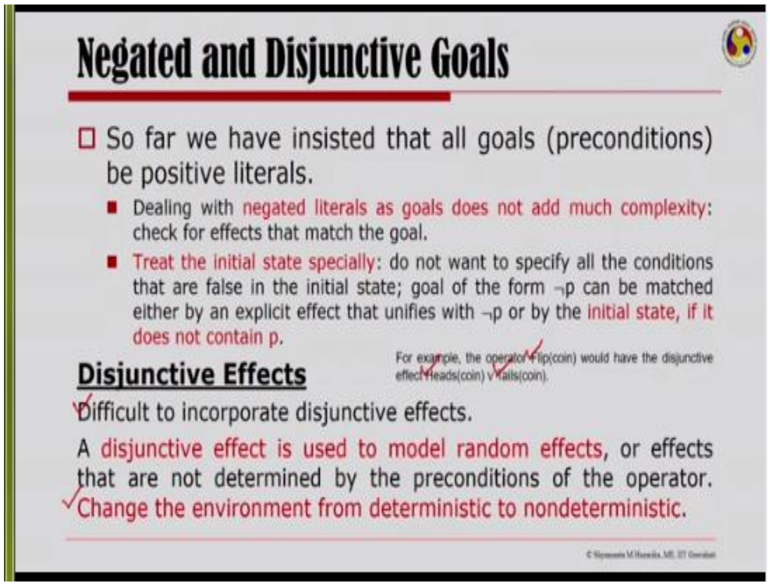
Disjunctive Pre-conditions

Any operator with the precondition p ∨ q could be replaced with two operators, one with p as a precondition and one with q, but then the planner would have to commit to one or the other. Disjunctive pre-conditions allows us to delay making the commitment.

© Shyamanta M Hazarika, ME, IIT Guwahati

Now any operator that has precondition p or q could be replaced with 2 operators one with p as a precondition and one with q. But that the planner would have to commit to one, so disjunctive precondition allows us to delay making the commitment.

(Refer Slide Time: 35:38)



Negated and Disjunctive Goals

- So far we have insisted that all goals (preconditions) be positive literals.
  - Dealing with negated literals as goals does not add much complexity: check for effects that match the goal.
  - Treat the initial state specially: do not want to specify all the conditions that are false in the initial state; goal of the form ¬p can be matched either by an explicit effect that unifies with ¬p or by the initial state, if it does not contain p.

Disjunctive Effects

For example, the operator flip(coin) would have the disjunctive effect heads(coin) ∨ tails(coin).

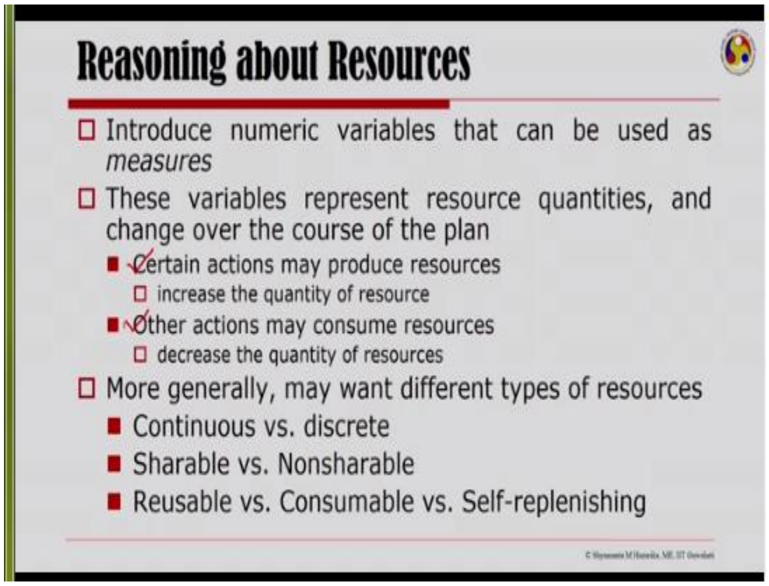
Difficult to incorporate disjunctive effects.

A disjunctive effect is used to model random effects, or effects that are not determined by the preconditions of the operator. Change the environment from deterministic to nondeterministic.

© Shyamanta M Hazarika, ME, IIT Guwahati

Similarly we could have disjunctive effects and now the disjunctive effects are a case like if I have an operator call flip coin. Then I would have either heads or tails and I should be able to express that effects as a disjunction. Now it is difficult to incorporate disjunctive effects, a disjunctive effect is used to model random effects or effects that are not determined by the preconditions. Now change the environment from deterministic to non-deterministic and therefore it is very difficult to incorporate disjunctive effects.

(Refer Slide Time: 36:23)

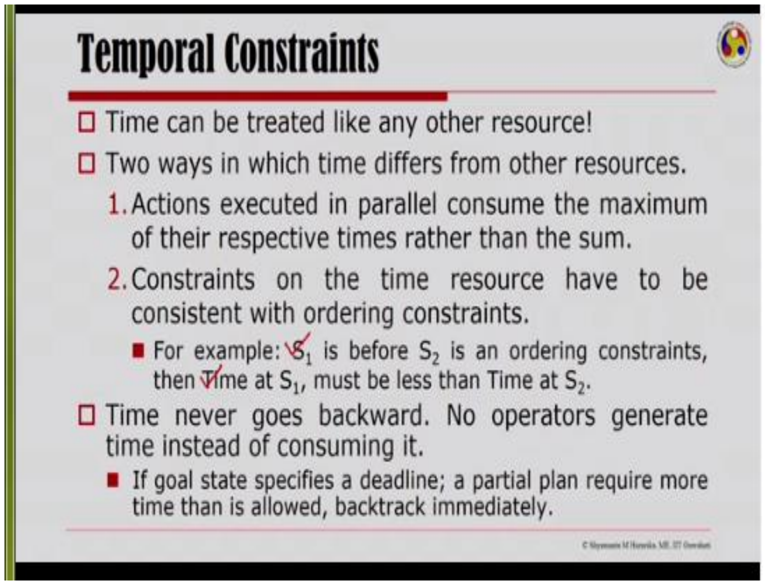


Reasoning about Resources

- Introduce numeric variables that can be used as *measures*
- These variables represent resource quantities, and change over the course of the plan
  - Certain actions may produce resources
    - increase the quantity of resource
  - Other actions may consume resources
    - decrease the quantity of resources
- More generally, may want different types of resources
  - Continuous vs. discrete
  - Sharable vs. Nonsharable
  - Reusable vs. Consumable vs. Self-replenishing

I would also be able to introduce numeric values that can be used as measures, now these variables represent resource quantities and change over the course of the plan. Certain actions may produce resources and therefore increase the quantity other accents may consume resources and therefore decrease the quantity of the resource. More generally we may want different type of resources that our planners should be able to handle like continuous versus discrete, shareable versus non sharable, reusable versus consumable versus self-replenishing.

(Refer Slide Time: 37:06)



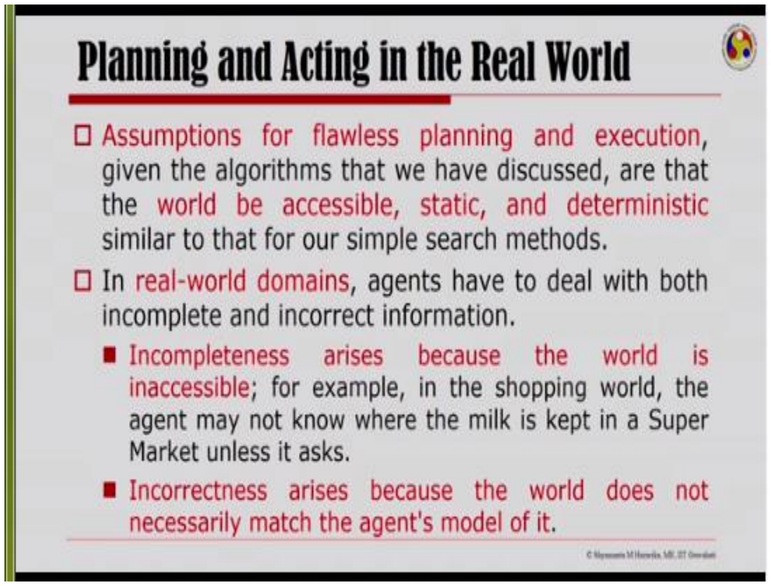
Temporal Constraints

- Time can be treated like any other resource!
- Two ways in which time differs from other resources.
  1. Actions executed in parallel consume the maximum of their respective times rather than the sum.
  2. Constraints on the time resource have to be consistent with ordering constraints.
     - For example: S_1 is before S_2 is an ordering constraints, then Time at S_1, must be less than Time at S_2.
- Time never goes backward. No operators generate time instead of consuming it.
  - If goal state specifies a deadline; a partial plan require more time than is allowed, backtrack immediately.

Now time can also be any other resource, 2 ways in which time differs from other resources is that when I execute actions in parallel then the maximum of the respective times is what I need to consider rather than the sum and the constraints on the time resource have to be consistent with the ordering constraint, that is if I have S1 before S2. Then time at S1 must be less than time at S2, time never goes backward.

So no operator generate time instead of consuming it and that is what we need to take care when we are talking of time constraints.

(Refer Slide Time: 37:52)



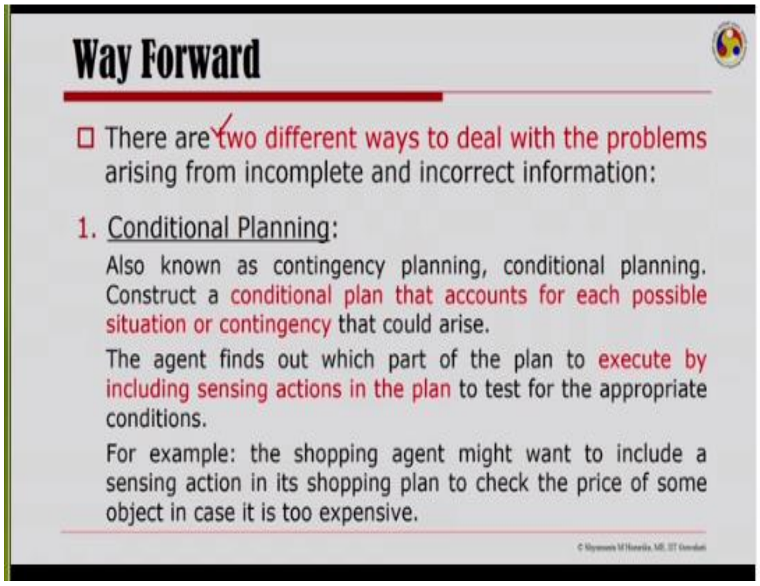
Planning and Acting in the Real World

- Assumptions for flawless planning and execution, given the algorithms that we have discussed, are that the world be accessible, static, and deterministic similar to that for our simple search methods.
- In real-world domains, agents have to deal with both incomplete and incorrect information.
  - Incompleteness arises because the world is inaccessible; for example, in the shopping world, the agent may not know where the milk is kept in a Super Market unless it asks.
  - Incorrectness arises because the world does not necessarily match the agent's model of it.

Now planning and acting in the real world we have assumptions for flawless planning and execution given the algorithms that we have discussed so far. And these assumptions are that the world is accessible, static and deterministic, similar to that for our simple search methods. In real world agents have to deal with both incomplete and incorrect information, so incompleteness arises because the world is inaccessible.

For example in the shopping world the agent may not know where the milk is kept in a supermarket unless it asks. And incorrectness arises because the world does not necessarily match the agents model of it.

(Refer Slide Time: 38:40)


Way Forward

- There are two different ways to deal with the problems arising from incomplete and incorrect information:

1. Conditional Planning:

   Also known as contingency planning, conditional planning. Construct a conditional plan that accounts for each possible situation or contingency that could arise.

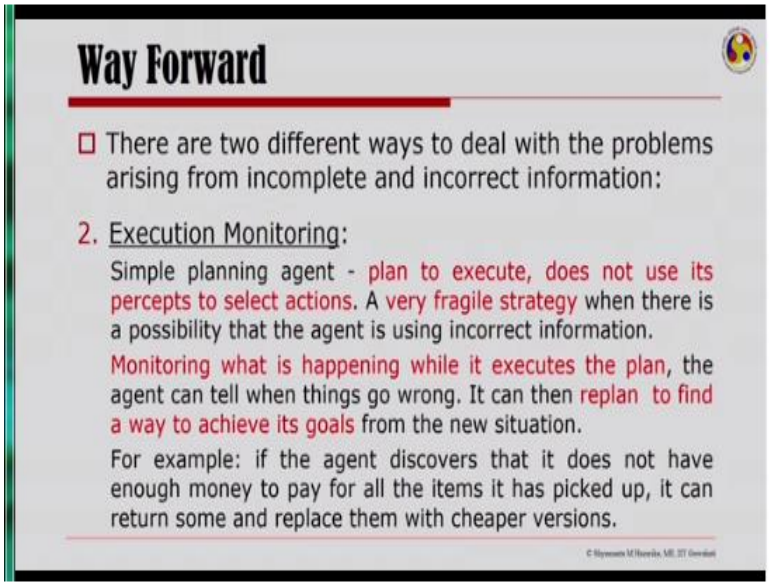
   The agent finds out which part of the plan to execute by including sensing actions in the plan to test for the appropriate conditions.

   For example: the shopping agent might want to include a sensing action in its shopping plan to check the price of some object in case it is too expensive.

So the way forward under such scenarios is that we can have 2 different ways to deal with this problem, one called conditional planning this is also known as contingency planning, in where we construct a conditional plan that accounts for each possible situation or contingency that could arise. The agent finds out which part of the plan to execute by including some sensing actions in it is plan to test for the appropriate conditions.

For example, the shopping agent might want to include a sensing action in it is shopping plan to check the price of some object in case it is too expensive.

(Refer Slide Time: 39:27)


Way Forward

- There are two different ways to deal with the problems arising from incomplete and incorrect information:

2. Execution Monitoring:

   Simple planning agent - plan to execute, does not use its percepts to select actions. A very fragile strategy when there is a possibility that the agent is using incorrect information.

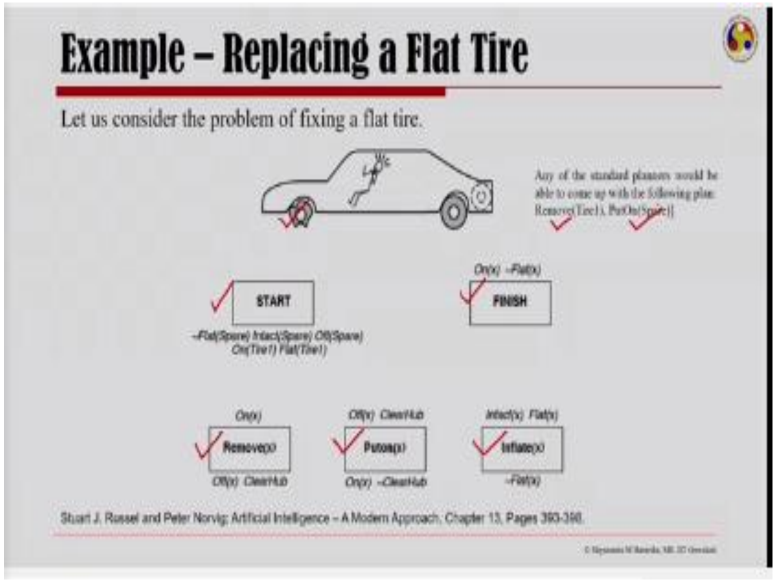
   Monitoring what is happening while it executes the plan, the agent can tell when things go wrong. It can then replan to find a way to achieve its goals from the new situation.

   For example: if the agent discovers that it does not have enough money to pay for all the items it has picked up, it can return some and replace them with cheaper versions.

And there is another way which is called execution monitoring, in execution monitoring you have a simple planning agent and now it uses it is percept to select action. Earlier in a simple planning agent with a plan to execute there was no possibility of using percepts to select action. So that was a very fragile strategy, monitoring what is happening while it executes the plan is what the execution monitoring means it can then replan to find a way to achieve it is goal from the new situation.

And for example the agent discovers that it does not have enough money to pay for all the items it has picked up, it can return some and replace them with a cheaper version this is called execution monitoring and replan.

(Refer Slide Time: 40:22)



Example – Replacing a Flat Tire

Let us consider the problem of fixing a flat tire.

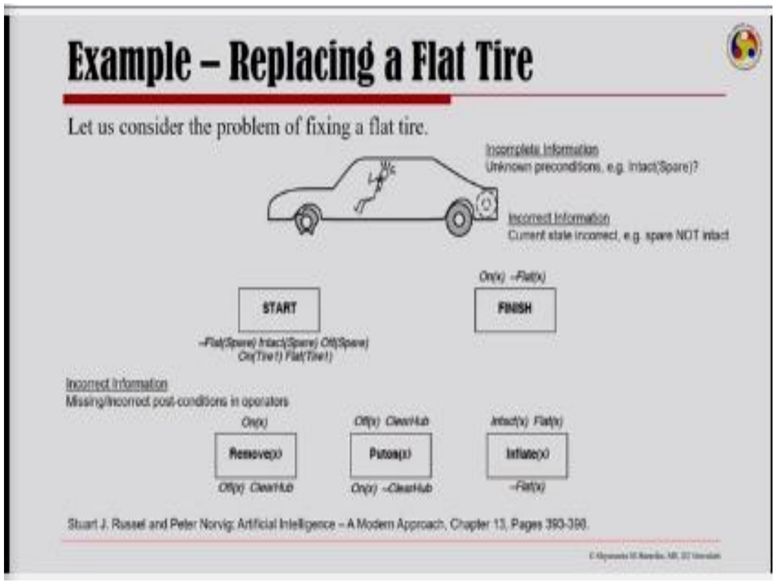
Any of the standard planners would be able to come up with the following plan: [Remove(Tire1), PutOn(Spare)]

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

Now let us look at one example to finish our discussion today about replacing a flat tire. Now this example is from Russell and Norvig artificial intelligence a modern approach, chapter 13 pages 393 to 398. Here the problem is to fix a flat tire, so at the start the spare tire is not flat, I have an intake spare and my tire 1, that I have is flat I want to go and have a situation where the tire is no longer flat.

Now I have operations that I can remove the tire or I can put it on the hub again or I can inflate the tire. Now any of the standard planners would be able to come up with the following plan it would talk of removing the tire and putting on the spare.

(Refer Slide Time: 41:30)



Example – Replacing a Flat Tire

Let us consider the problem of fixing a flat tire.

Incomplete Information
Unknown preconditions, e.g. Intact(Spare)?

Incorrect Information
Current state incorrect, e.g. spare NOT intact

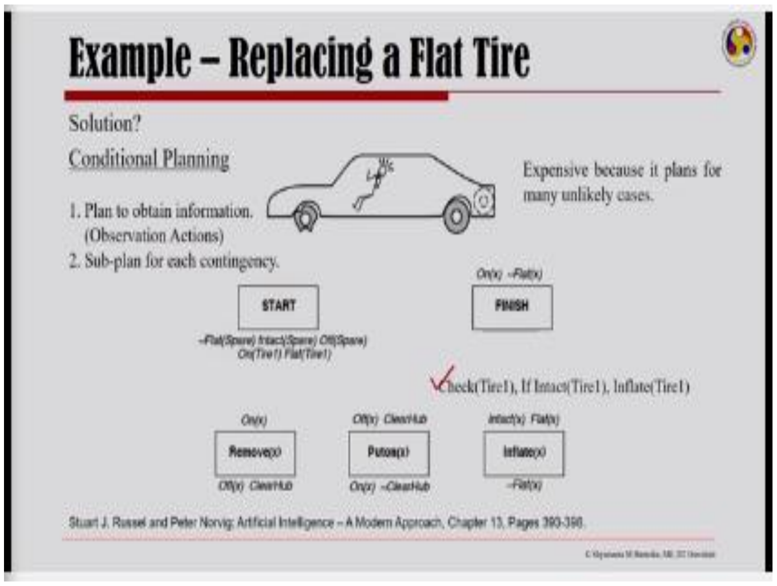
Incorrect Information
Missing/incorrect post-conditions in operators

On(x) ¬Flat(x)
START
FINISH
¬Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)
On(x)
Remove(x)
Off(x) ClearHub
Off(x) ClearHub
Puton(x)
On(x) ¬ClearHub
Intact(x) Flat(x)
Inflate(x)
¬Flat(x)

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 390-398.

But under incorrect information that the current state is incorrect, example we know that the spare is not intake or we have incomplete information, that is unknown preconditions. Such as that is the spare intake or not or even incorrect information like missing incorrect post conditions in operators, how do we go about planning.

(Refer Slide Time: 42:01)



Example – Replacing a Flat Tire

Solution?

Conditional Planning

1. Plan to obtain information.
   (Observation Actions)
2. Sub-plan for each contingency.

Expensive because it plans for many unlikely cases.

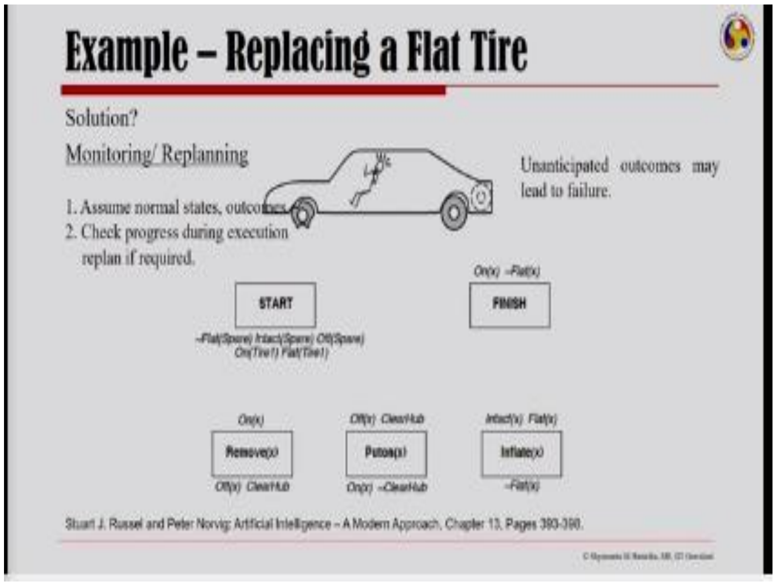
✓Check(Tire1), If Intact(Tire1), Inflate(Tire1)

On(x) ¬Flat(x)
START
FINISH
¬Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)
On(x)
Remove(x)
Off(x) ClearHub
Off(x) ClearHub
Puton(x)
On(x) ¬ClearHub
Intact(x) Flat(x)
Inflate(x)
¬Flat(x)

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 390-398.

So the solution for planning in such scenarios is to go for conditional planning, that is you plan to obtain information from certain observation actions and then you sub plan for each contingency. And like you could look for saying I would love to check tire 1 if tire 1 is intake I will inflate, so this is a conditional plan. Now it is expensive because it plans for many unlikely cases.

(Refer Slide Time: 42:39)



Example – Replacing a Flat Tire

Solution?

Monitoring/ Replanning

1. Assume normal states, outcomes.
2. Check progress during execution
   replan if required.

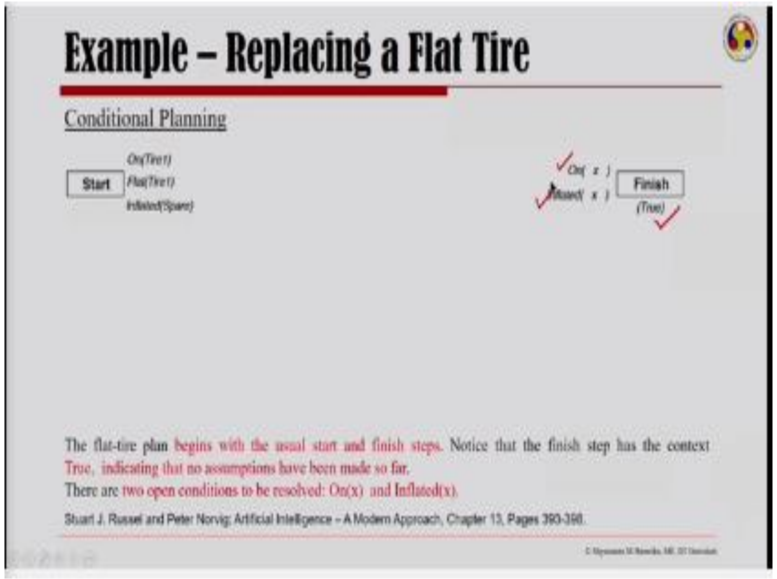
Unanticipated outcomes may lead to failure.

On(x) ¬Flat(x)
FINISH
START
¬Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)
On(x)
Remove(x)
Off(x) ClearHub
Off(x) ClearHub
Puton(x)
On(x) ¬ClearHub
Intact(x) Flat(x)
Inflate(x)
¬Flat(x)

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

On the other hand you could think of monitoring or replanning where you assumed normal states outcomes and you check progress during execution and replant if required. This will lead to failure if I come across unanticipated outcomes.

(Refer Slide Time: 42:58)



Example – Replacing a Flat Tire

Conditional Planning

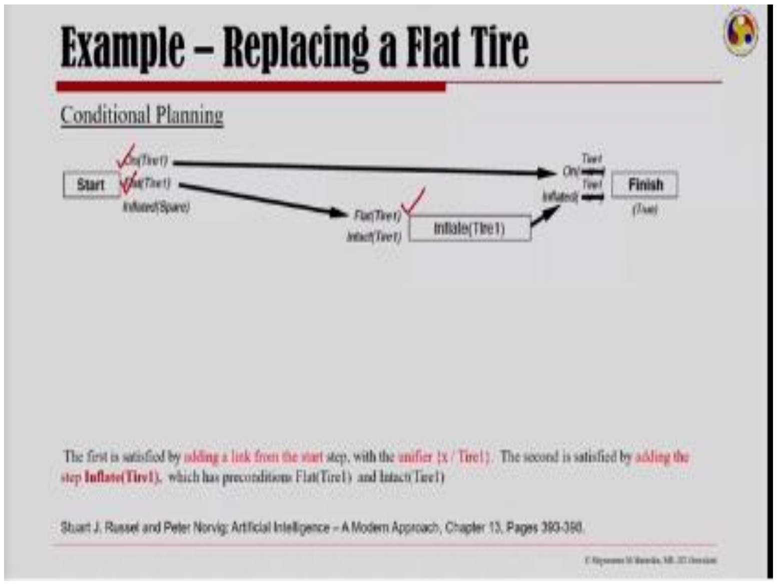
Start
On(Tire1)
Flat(Tire1)
Inflated(Spare)
On(x)
Inflated(x)
Finish
(True)

The flat-tire plan begins with the usual start and finish steps. Notice that the finish step has the context True, indicating that no assumptions have been made so far.
There are two open conditions to be resolved: On(x) and Inflated(x).

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

Now let us look at our first solution of replacing a flat tire that is conditional planning. So the flat tire plan begins with the usual start and finish steps, notice that the finish step has a context true here which indicates that no assumption has been made so far. And now we have 2 open conditions to be resolved on an inflated.

(Refer Slide Time: 43:32)



Start
On(Tire1)
Flat(Tire1)
Inflated(Spare)
Flat(Tire1)
Intact(Tire1)
Inflate(Tire1)
On(Tire1, Tire1)
Inflated(Tire1)
Finish
(True)

Example – Replacing a Flat Tire

Conditional Planning

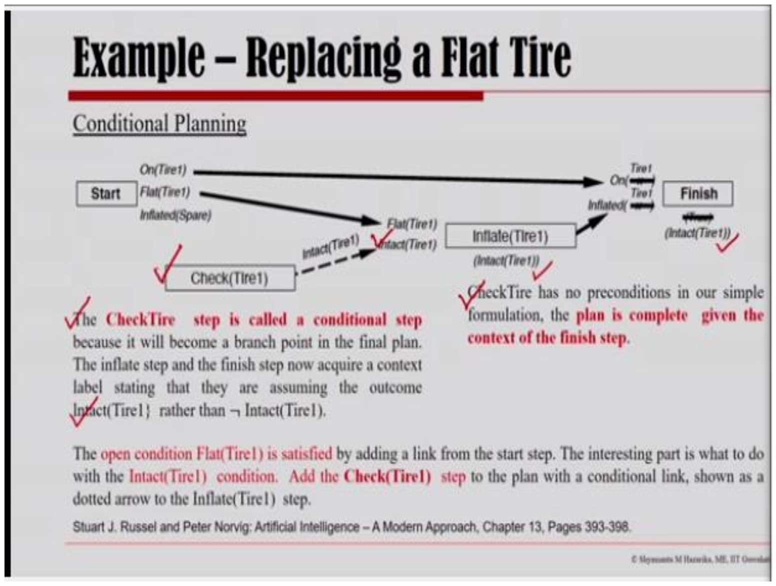
The first is satisfied by adding a link from the start step, with the unifier {x / Tire1}. The second is satisfied by adding the step **Inflate(Tire1)**, which has preconditions Flat(Tire1) and Intact(Tire1)

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

© Shyamanta M Hazarika, ME, IIT Guwahati

And the first is satisfied by adding a link from the start step with the unifier x for tire 1. The second is satisfied by adding the step inflate tire 1 which has preconditions that the tire is flat.

(Refer Slide Time: 43:58)



Start
On(Tire1)
Flat(Tire1)
Inflated(Spare)
Check(Tire1)
Intact(Tire1)
Flat(Tire1)
Intact(Tire1)
Inflate(Tire1)
(Intact(Tire1))
On(Tire1, Tire1)
Inflated(Tire1)
Finish
~~(True)~~
(Intact(Tire1))

Example – Replacing a Flat Tire

Conditional Planning

The **CheckTire step is called a conditional step** because it will become a branch point in the final plan. The inflate step and the finish step now acquire a context label stating that they are assuming the outcome Intact(Tire1} rather than ¬ Intact(Tire1).

CheckTire has no preconditions in our simple formulation, the **plan is complete given the context of the finish step.**

The open condition Flat(Tire1) is satisfied by adding a link from the start step. The interesting part is what to do with the Intact(Tire1) condition. Add the **Check(Tire1)** step to the plan with a conditional link, shown as a dotted arrow to the Inflate(Tire1) step.

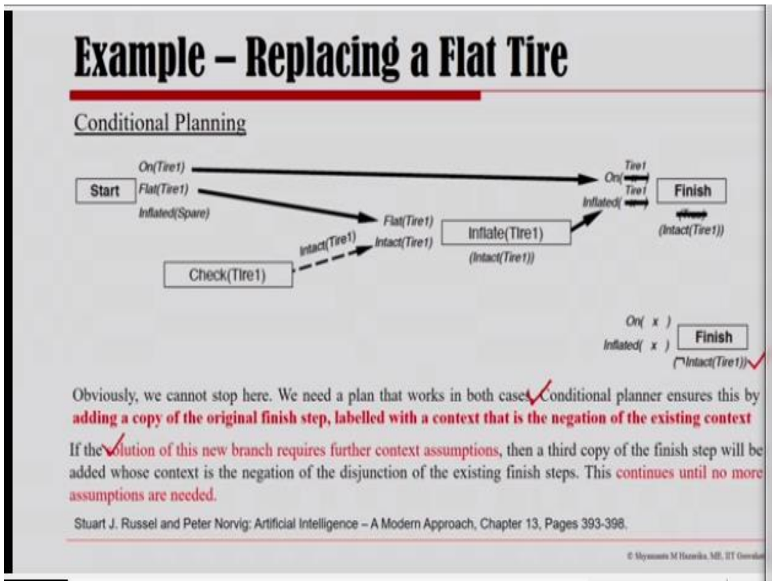
Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

© Shyamanta M Hazarika, ME, IIT Guwahati

But interesting is that it also says that the tire is intact, now the open condition flat is satisfied by adding a link from the start step for the intact tire 1 condition, I need to add the check tire 1 step which is a conditional link shown as a dotted arrow here. And this is called a conditional step because it will become a branch point in the final plan and the inflate step and the finished step acquire a context which is this plan will work only under the context that the tire 1 is intact.

And they are assuming the outcome to be intact tire 1 rather than not intact tire 1, now check tire has no preconditions in our simple formulation. So the plan is complete but this plan is complete given the context of the finish step.

(Refer Slide Time: 45:03)



Example – Replacing a Flat Tire

Conditional Planning

Start
On(Tire1)
Flat(Tire1)
Inflated(Spare)
Flat(Tire1)
Intact(Tire1)
Intact(Tire1)
Check(Tire1)
Inflate(Tire1)
(Intact(Tire1))
On(Tire1)
Inflated(Tire1)
Finish
(Intact(Tire1))
On(x)
Inflated(x)
Finish
(¬Intact(Tire1))

Obviously, we cannot stop here. We need a plan that works in both cases. Conditional planner ensures this by **adding a copy of the original finish step, labelled with a context that is the negation of the existing context**

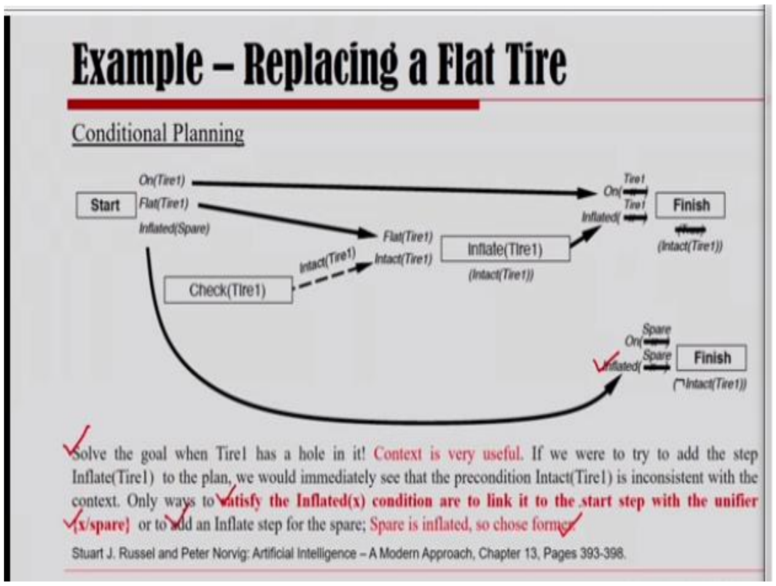
If the solution of this new branch requires further context assumptions, then a third copy of the finish step will be added whose context is the negation of the disjunction of the existing finish steps. This continues until no more assumptions are needed.

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

Obviously we cannot stop here we need a plan that works in both case, if tire 1 is intact or if there is a whole entire one and you cannot inflate it any longer. So the conditional plan ensures this by adding a copy of the original finish step but now labelled with a context which is opposite of the original context so it labeled is as not intact tire 1. And one needs to remember if this new branch that has come if the solution of this new branch requires further context assumptions.

Then I will keep on adding a copy of the finish step but for this problem the only context being intact tire 1 and not intact tire 1 this is good enough.

(Refer Slide Time: 45:52)



Example – Replacing a Flat Tire

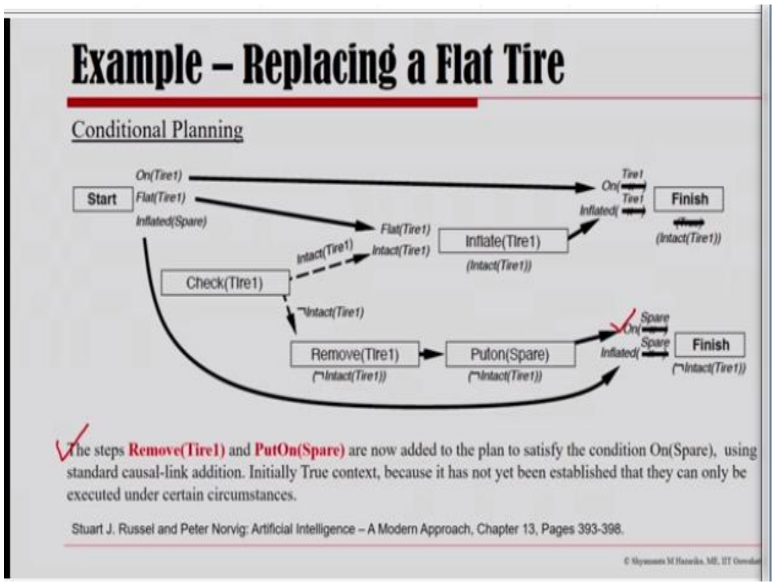
Conditional Planning

Solve the goal when Tire1 has a hole in it! Context is very useful. If we were to try to add the step Inflate(Tire1) to the plan, we would immediately see that the precondition Intact(Tire1) is inconsistent with the context. Only ways to satisfy the Inflated(x) condition are to link it to the start step with the unifier {x/spare} or to add an Inflate step for the spare; Spare is inflated, so chose former.

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

So now we continue this plan, this very fact that tire 1 has a hole in it still I can plan and only way to satisfy the inflated x condition now, is by looking at the start by having spare being replaced for the x. And with this unifier I can add an inflate step for the spare, now the spare is inflated. So you satisfy the inflated x condition either by looking at a substitution of x for spare or add an inflate step for the spare, now spare is inflated, so you choose the former.

(Refer Slide Time: 46:44)



Example – Replacing a Flat Tire

Conditional Planning

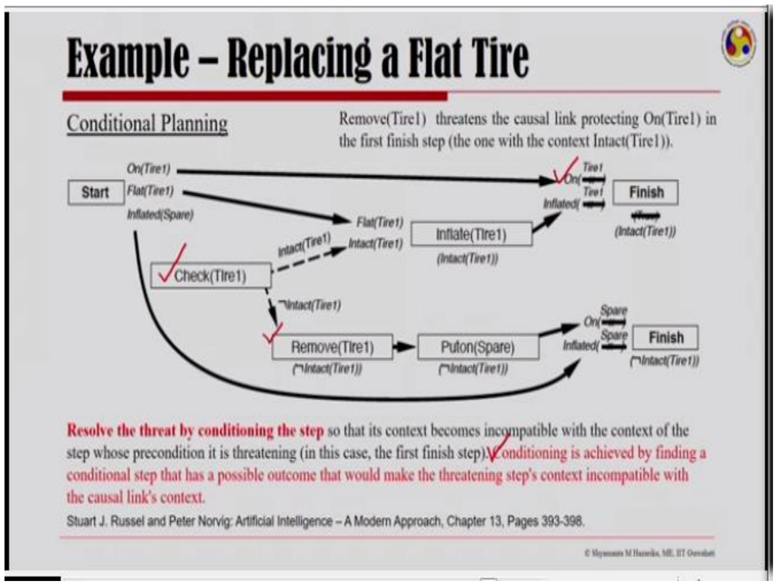
The steps Remove(Tire1) and PutOn(Spare) are now added to the plan to satisfy the condition On(Spare), using standard causal-link addition. Initially True context, because it has not yet been established that they can only be executed under certain circumstances.

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

© Shyamanta M Hazarika, ME, IIT Guwahati

And you have now only to add the 2 conditions that you remove the tire and put on the spare, the steps remove tire and put on the spare are now added to the plan to satisfy the condition on spare.

(Refer Slide Time: 47:05)



Example – Replacing a Flat Tire

Conditional Planning

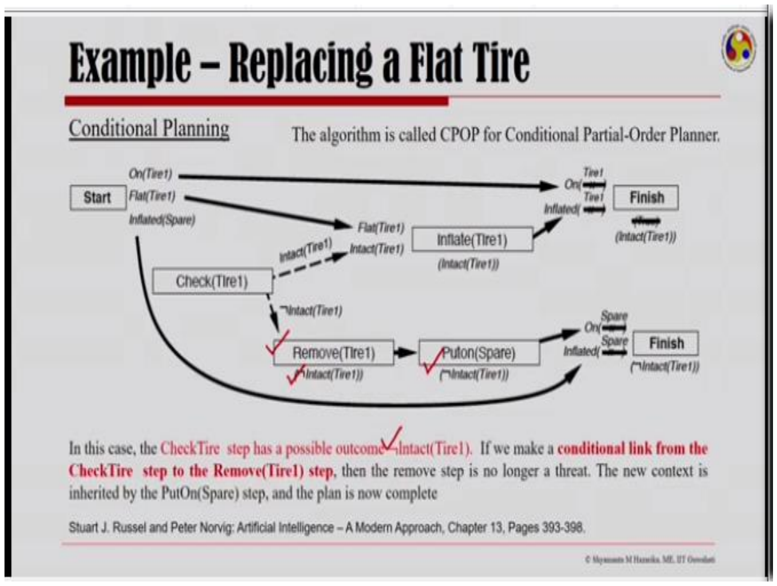
Remove(Tire1) threatens the causal link protecting On(Tire1) in the first finish step (the one with the context Intact(Tire1)).

Resolve the threat by conditioning the step so that its context becomes incompatible with the context of the step whose precondition it is threatening (in this case, the first finish step). Conditioning is achieved by finding a conditional step that has a possible outcome that would make the threatening step's context incompatible with the causal link's context.

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

However when I am doing this one needs to realize that the remove tire 1 actually threatens a causal link which is on tire 1 in the finish step and this I need to take care of, I resolve the threat by conditioning that step. So that it is context becomes incompatible, so conditioning is achieved here by finding a conditional step that has possible outcome that would make the threatening steps context incompatible. So we check and see that this step which is about check tire 1 can have a possible outcome which is not intact tire 1.

(Refer Slide Time: 47:47)



Example – Replacing a Flat Tire

Conditional Planning

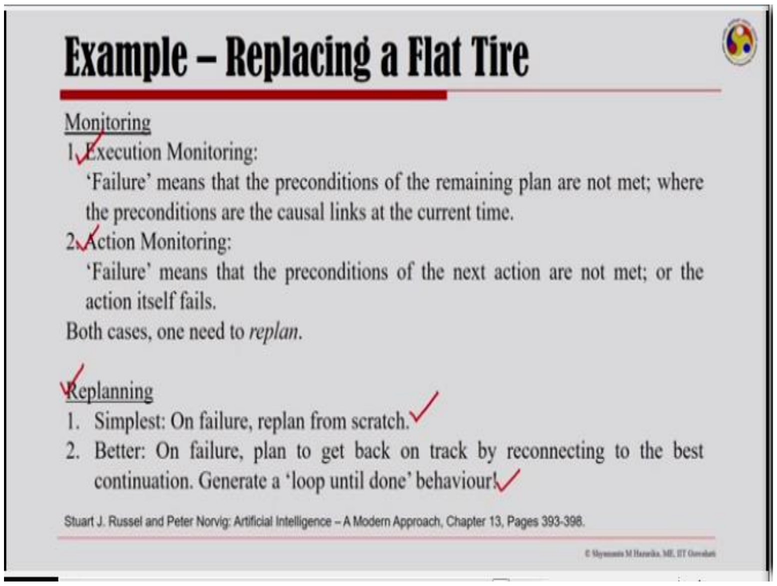
The algorithm is called CPOP for Conditional Partial-Order Planner.

In this case, the CheckTire step has a possible outcome ¬Intact(Tire1). If we make a **conditional link from the CheckTire step to the Remove(Tire1) step**, then the remove step is no longer a threat. The new context is inherited by the PutOn(Spare) step, and the plan is now complete

Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

So we make a conditional link from the check intake tire step to the remove tire step and then this remove tire step is no longer a trap. Because it has now got this not intact tire 1 as it is context and the new context is inherited by the put on spare and the plan is now complete. Now this algorithm that we have just now looked at is called the conditional partial order planner and is a good example of conditional planning.

(Refer Slide Time: 48:36)



Example – Replacing a Flat Tire

Monitoring

1. Execution Monitoring:
   'Failure' means that the preconditions of the remaining plan are not met; where the preconditions are the causal links at the current time.
2. Action Monitoring:
   'Failure' means that the preconditions of the next action are not met; or the action itself fails.

Both cases, one need to *replan*.

Replanning

1. Simplest: On failure, replan from scratch.
2. Better: On failure, plan to get back on track by reconnecting to the best continuation. Generate a 'loop until done' behaviour!

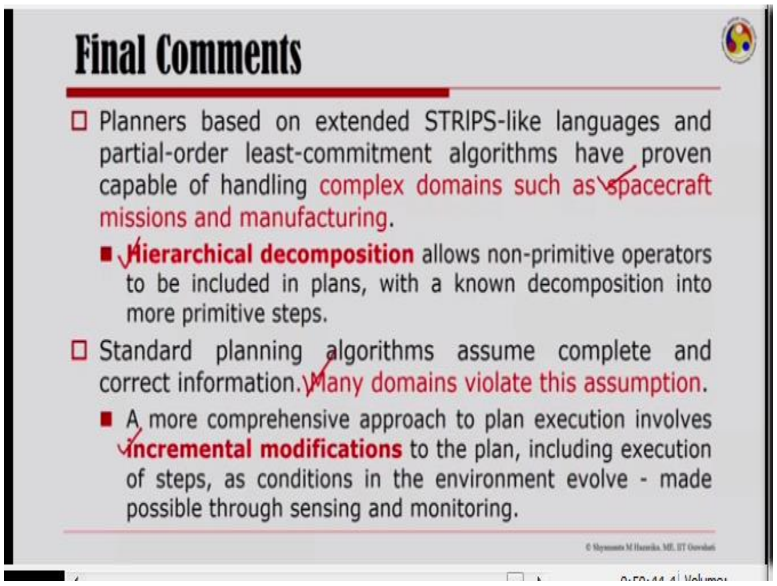
Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 13, Pages 393-398.

© Shyamanta M Hazarika, ME, IIT Guwahati

There is another solution to replacing a flat tire as we have been discussing earlier it is about monitoring. So execution monitoring failure would mean that the precondition of the remaining plan are not made and when the preconditions are causal links at the current time. And action monitoring would mean, failure means the preconditions of the next action are not meet or the action itself fails, so both cases one needs to replan.

Now in replan we can go by a very simple algorithm that on every failure you replan from scratch or better on failure you plan to get back on track by reconnecting to the best continuation and you generate a loop until down behavior.

(Refer Slide Time: 49:29)



Final Comments

- Planners based on extended STRIPS-like languages and partial-order least-commitment algorithms have proven capable of handling complex domains such as spacecraft missions and manufacturing.
  - **Hierarchical decomposition** allows non-primitive operators to be included in plans, with a known decomposition into more primitive steps.
- Standard planning algorithms assume complete and correct information. Many domains violate this assumption.
  - A more comprehensive approach to plan execution involves **incremental modifications** to the plan, including execution of steps, as conditions in the environment evolve - made possible through sensing and monitoring.

© Shyamanta M Hazarika, ME, IIT Guwahati

Now to finally conclude our lecture on practical planning and action, we have seen that planners based on extended STRIPS-like language and partial-order least commitment algorithms are capable of handling complex domains. These domains are as complex as spacecraft missions and manufacturing and hierarchical decomposition allows non-primitive operations to be included in plans with a known decomposition into more primitive steps.

Standard planning algorithms assume complete and correct information, many domains violate this assumption. A more comprehensive approach is one that we have looked at called conditional planning and it could involve incremental modification to the plan including execution of steps as conditions in the environment evolve which is made possible through sensing and monitoring.

Now these were about planning, actions, in our next lecture we will look at sequential decision problems where we need to plan not for a single action but for a sequence of actions thank you.