ARTIFICIAL INTELLIGENCE
UNIT-4
Part-1

Dr. R.Seeta Sireesha
Associate Professor,
Department of Computer Science and Engineering
GVPCE (A), Madhurawada.

UNIT - 4 (Part-1)

PLANNING : INTRODUCTION, PLANNING PROBLEM, THE LANGUAGE OF PLANNING PROBLEMS, PLANNING WITH STATE SPACE SEARCH, PARTIAL ORDERING PLANNING, CONDITIONAL PLANNING

INTRODUCTION

What is planning?

- The planning problem in Artificial Intelligence is about the decision making performed by intelligent creatures like robots, humans, or computer programs when trying to achieve some goal.
- **“The task of coming up with a sequence of actions that will achieve a goal is known as *planning”***.
- Everything we humans do is with a definite goal in mind, and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.

Planning..

1. We have some ***Operators***.
2. We have a ***Current state.***
3. We have a ***Goal State.***
4. We want to know:

How to arrange the operators to reach the Goal State from Current State.

Planning..

- Planning can be ***Classical or Non-classical***.
- In case of **Classical Planning**, the environment is **fully observable, deterministic, static and discrete**,
- whereas in case of **Non-classical Planning**, the environment is **partially observable** (i.e. the entire state of the environment is not visible at a given instant) or **non-deterministic** (or stochastic, i.e. the current state and chosen action cannot completely determine the next state of the environment).

What is Problem solving agent?

- A problem-solving agent is an AI system that is designed to solve problems by searching through a space of possible solutions.
- These agents typically receive input in the form of a problem statement, and use algorithms or heuristics to explore different possible states and actions until they find a solution that satisfies certain criteria or constraints.
- The search process can be guided by various types of knowledge, including rules of thumb, heuristics, or domain-specific expertise.

Drawbacks of problem solving agents

- **Search space explosion**: A problem-solving agent may get stuck in suboptimal solutions or local optima when the *problem space grows too large for the agent to search through all possible solutions in a reasonable amount of time*
- **Lack of flexibility**: The agents may not be able to adapt to changes in the environment.
- **Limited ability to reason**: The agents typically rely on pre-defined rules or heuristics to guide their search, and may not be able to reason more abstractly about the problem or generate creative solutions.
- **Limited ability to handle uncertainty**: These agents may struggle to deal with uncertainty or incomplete information.

Key differences of problem solving agents and planning agents

	Problem solving agents	Planning agents
Representation of problems	These agents typically represent problems as a search space of possible actions and states, and use search algorithms to find a solution.	These agents represent problems as a set of states and actions, and use reasoning and decision-making to generate a plan of action to achieve a goal.
Type of knowledge used	Problem-solving agents rely on domain-specific knowledge about the problem being solved, such as heuristics or rules of thumb.	Planning agents, on the other hand, rely on more general knowledge about the domain, such as causal relationships and constraints.
Efficiency	Problem solving agents are searching through a large space of possible actions	Planning agents are typically more efficient as they can generate plans of action directly
Ability to handle uncertainty	Problem solving agents cannot handle uncertainty and incomplete information	Planning agents are generally better equipped to handle uncertainty and incomplete information

Example

Imagine you want to **bake a cake**.

A brute force search algorithm would have to examine all possible combinations of ingredients, measurements, and cooking times to find the recipe that results in a delicious cake. This would be incredibly time-consuming and inefficient.

A planning agent, on the other hand, "can start with the goal of baking a cake and use knowledge about what ingredients and techniques are needed to achieve that goal". For example, the agent might know that a cake requires flour, sugar, eggs, and baking powder, and that those ingredients need to be mixed together and baked in an oven. With this knowledge, the agent can generate a specific plan of action that will result in the goal of baking a cake being achieved in a much more efficient manner.

Planning problem

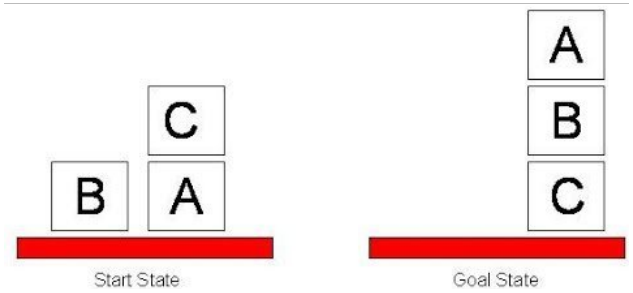
- A planning problem in AI is a task that involves generating a sequence of actions to achieve a specific goal or set of goals.
- It involves identifying the actions that can be taken to solve the problem, the conditions that must be satisfied for each action to be executed, and the constraints that limit the set of possible actions

A planning problem typically consists of the following components:

1. Initial state
2. Goal state
3. Actions
4. Preconditions
5. Effects
6. Constraints

Example - Blocks world planning

The block world consists of:



-> a flat surface such as a table top
-> an adequate set of identical blocks which are identified by letters
-> the blocks can be stacked one upon another
->there is a robot arm that can manipulate the blocks
-> the robot can hold one block at a time and only one block can be moved at a time
-> Any number of blocks can be on the table

PLANNING ALGORITHMS

- Representation of planning problems - states, actions, and goals - should make it possible for planning
- Algorithms are nothing but logical structure of the problem
- To define an efficient algorithm, language is very important
- **STRIPS language - the language of classical planner**

The language of planning problems - STRIPS

- STRIPS (Standford Research Institute Problem Solver) is an automated planning technique used to find a goal, by executing domain from the initial state.
- With STRIPS, we can first derive the world (initial and goal state) by providing objects, actions, precondition, and effects.
- To describe the world, we used to two categories of terms
  - States - initial and goal states
  - Action schema - objects, actions, preconditions and effects
- Once the world is described, then provide a problem set.
- STRIPS can then search all possible states, starting from the initial one, executing various actions, until it reaches the goal.

Planning domain definition language

- A common language for writing STRIPS domain and problem set, is the planning Domain Definition language (PDDL)
- In PDDL most of the codes are English words, so that it can be clearly read and well understood.
- It is relatively easy approach to writing simple AI planning problems.

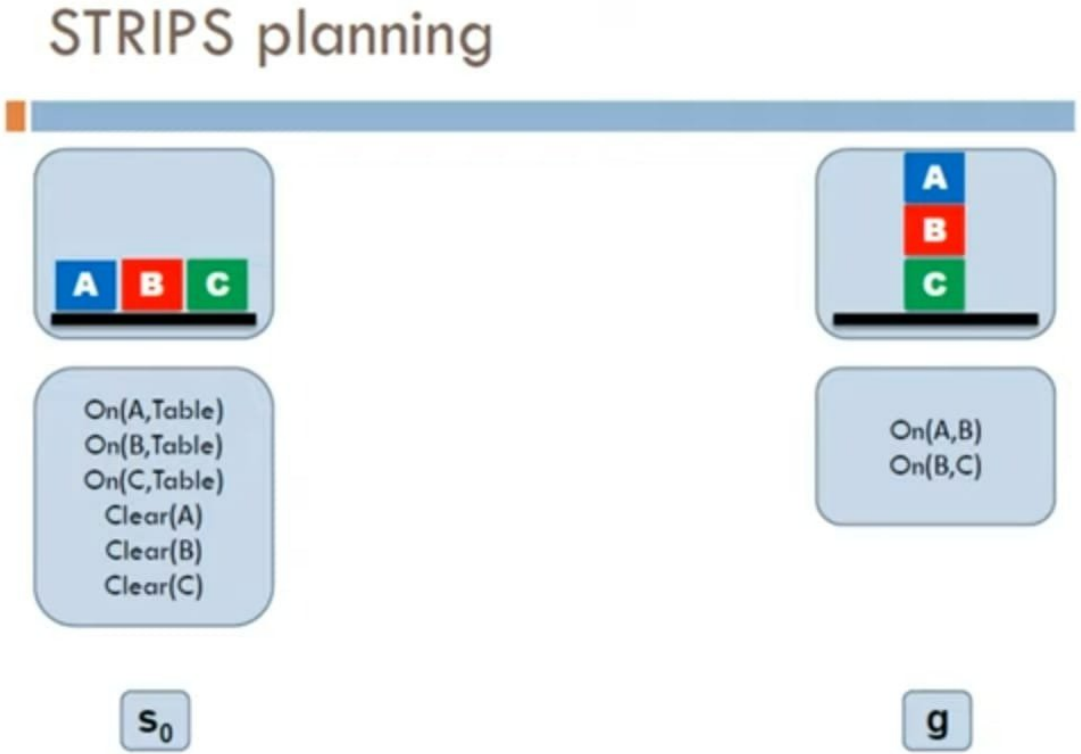
STRIPS - States, Goals and Actions

- States: conjunctions of ground, function-free, and positive literals, such as At(Home) ^ Have(Banana)
- To describe states, Closed-world assumption is used (the world model contains everything, the agent needs to know: there can be no surprise)
- Goals: conjunctions of literals, may contain variables (existential), goal may represent more than one state
  - E.g. At(Home) ^ ¬Have(Bananas)
  - E.g. At(x) ^ Sells(x, Bananas)
- Actions: preconditions that must hold before execution and the effects after execution

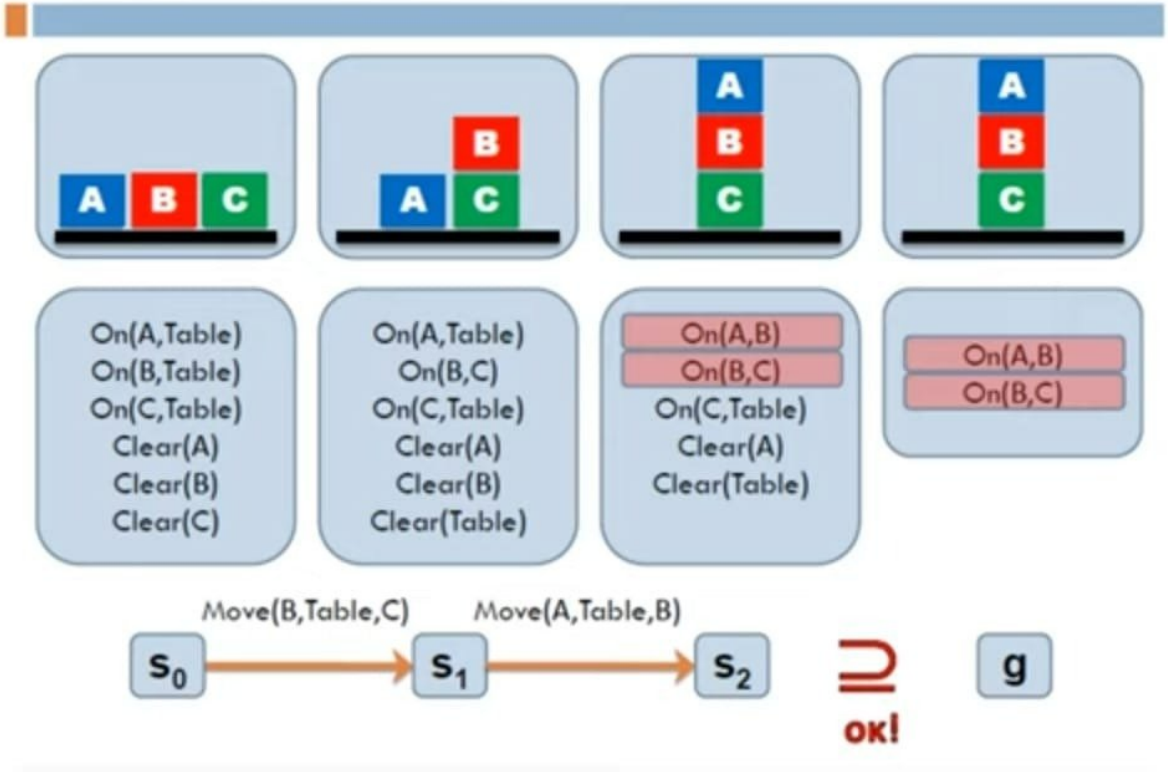
STRIPS Action Schema

- An action schema includes:
  - Action name & parameter list (variables)
  - Precondition: a conjunction of function-free positive literals. The action variables must also appear in precondition.
  - Effect: a conjunction of function-free literals (positive or negative)
- Add-list: positive literals
- Delete-list: negative literals
- Example:
  - Action: Buy (x)
  - Precondition: At (p), Sells (p, x)
  - Effect: Have(x)

Blocks world planning using STRIPS



STRIPS planning



Air Cargo Transport problem

The air cargo transport problem is a classic example of a logistics problem, which involves transporting cargo from one place to another using airplanes. The problem can be stated as follows:

- There are several airplanes, airports, and cargo packages.
- Each package needs to be transported from its current location to a specified destination airport.
- Each plane can carry multiple packages, and each package can be loaded and unloaded at different airports.
- The goal is to find a sequence of actions that will transport all packages to their respective destinations, while minimizing the total time and resources required.

To solve this problem using AI planning techniques, we need to represent the problem in a formal language, such as the STRIPS language.

Air Cargo Transport problem

In the air cargo transport problem, the objects include the **planes, airports, and cargo packages**.

- The predicates include **At(obj, loc)** to represent the location of an object,
- **In(obj, plane)** to represent that an object is inside a plane,
- **Cargo(obj)** to represent that an object is a cargo package,
- **Plane(obj)** to represent that an object is a plane, and
- **Airport(obj)** to represent that an object is an airport.

-> The operators include **Load(plane, cargo, loc)** to load a cargo package onto a plane at an airport,
-> **Unload(plane, cargo, loc)** to unload a cargo package from a plane at an airport, and
-> **Fly(plane, from, to)** to fly a plane from one airport to another.
Each operator has preconditions that must be satisfied before it can be applied, and effects that update the state of the problem after it is applied.

Representing Air Cargo Transport problem in STRIPS

```
DOMAIN AirCargoTransport
  OBJECTS
    Plane1, Plane2 - planes
    JFK, SFO, ATL - airports
    P1, P2, P3 - packages
  END OBJECTS
```

```
PREDICATES
  At(obj, loc) - object is located at location
  In(obj, plane) - object is inside the plane
  Cargo(obj) - object is a cargo
  Plane(plane) - object is a plane
  Airport(loc) - object is an airport
  Load(plane, cargo, loc) - load cargo onto the plane at location
  Unload(plane, cargo, loc) - unload cargo from the plane at location
  Fly(plane, from, to) - fly the plane from one airport to another
END PREDICATES
```

```
INITIAL STATE
  At(P1, JFK)
  At(P2, SFO)
  At(P3, ATL)
  At(Plane1, JFK)
  At(Plane2, SFO)
  Cargo(P1)
  Cargo(P2)
  Cargo(P3)
  Plane(Plane1)
  Plane(Plane2)
  Airport(JFK)
  Airport(SFO)
  Airport(ATL)
END INITIAL STATE
```

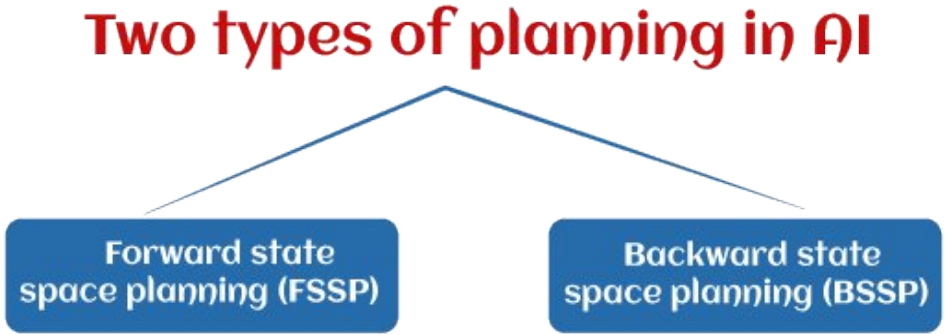
```
GOAL
  At(P1, SFO)
  At(P2, JFK)
  At(P3, JFK)
END GOAL
```

```
OPERATORS
  // Load cargo onto plane at airport
  Load(p, c, a)
    PRECONDITIONS
      At(c, a)
      At(p, a)
      Plane(p)
      Cargo(c)
      Airport(a)
    EFFECTS
      In(c, p)
      !At(c, a)
  END
```

```
// Unload cargo from plane at airport
Unload(p, c, a)
  PRECONDITIONS
    In(c, p)
    At(p, a)
    Plane(p)
    Cargo(c)
    Airport(a)
  EFFECTS
    At(c, a)
    !In(c, p)
END
```

```
  // Fly the plane from one airport to another
  Fly(p, from, to)
    PRECONDITIONS
      At(p, from)
      Plane(p)
      Airport(from)
      Airport(to)
    EFFECTS
      !At(p, from)
      At(p, to)
  END
END OPERATORS
```

Types of planning



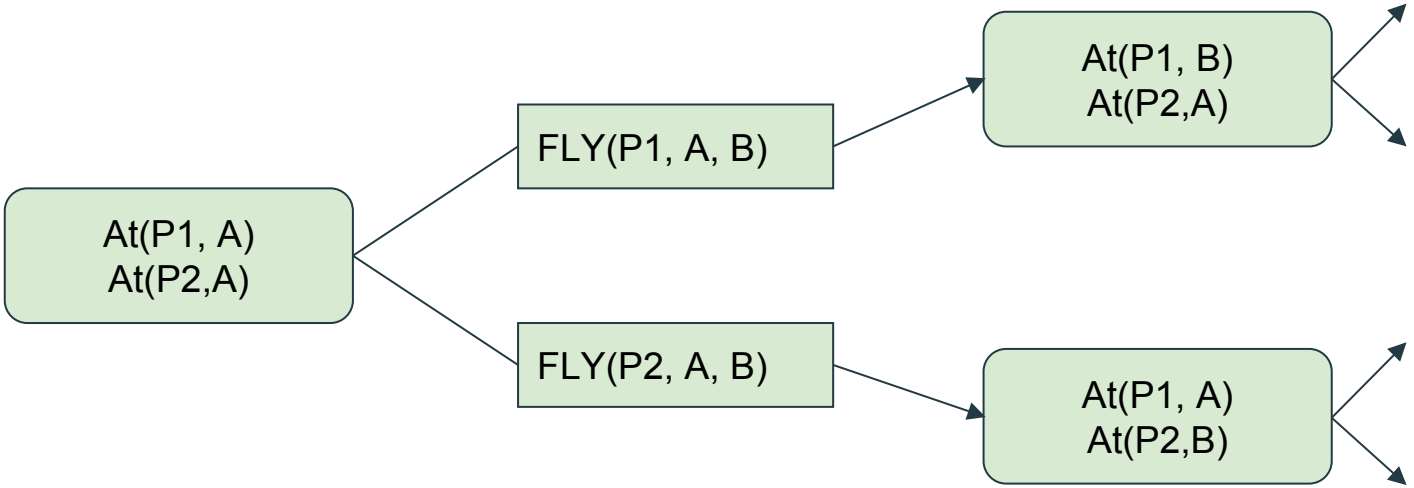
1. **Forward state space planning**: It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S', called a ***progression***. It continues until we reach the target position
2. **Backward state space planning**:In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called ***regression***

Forward state space planning

It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S', called a ***progression***. It continues until we reach the target position. It uses STRIPS representation.

- **Initial state**: start state
- **Actions**: Each action has a particular precondition to be satisfied before the action can be performed and an effect that the action will have on the environment.
- **Goal state**: To check if the current state is the goal state or not.
- **Step cost**: Cost of each step which is assumed to be 1.

Example



Forward state space search planning:

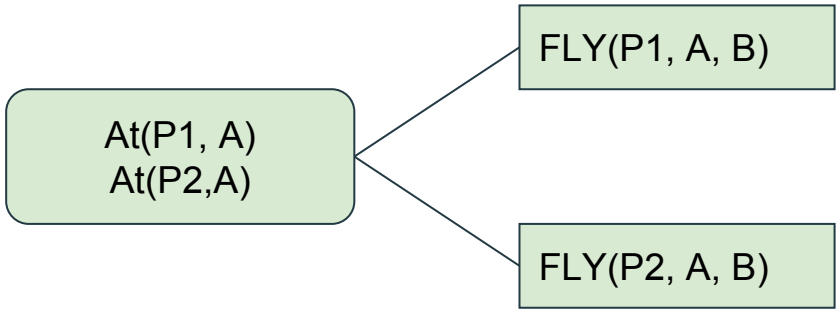
Initially,

At(P1, A)
At(P2,A)

Two planes P1, P2 are at airport A. Now perform an action to move the planes to desired airport.

Forward state space planning:

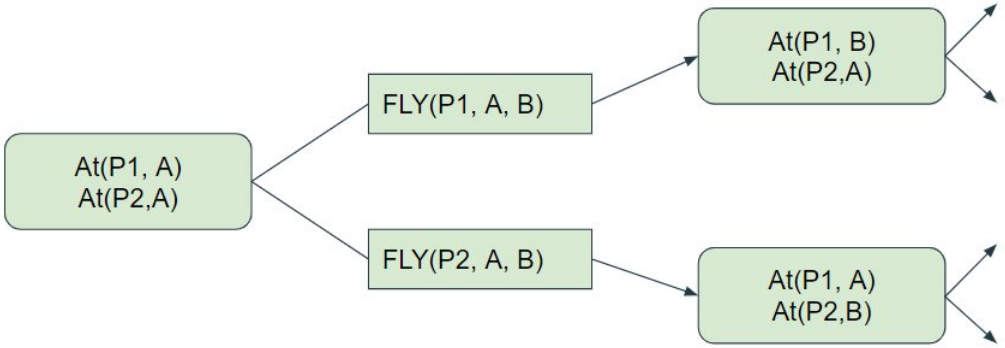
After performing an action:



Here two actions performed, 1st action is to fly the plane P1 from A to B and then 2nd action is to fly the plane P2 from A to B. so wet get two goal states for the two actions

Forward state space search:

The final goal,



- For the 1st action, plane P1 reached airport B(At(P1,B)) and plane P2 remains same at airport A(At(P2, A)).
- For the 2nd action, plane P2 reached airport B(At(P2,B)) and plane P1 remains same at airport A(At(P1, A)).

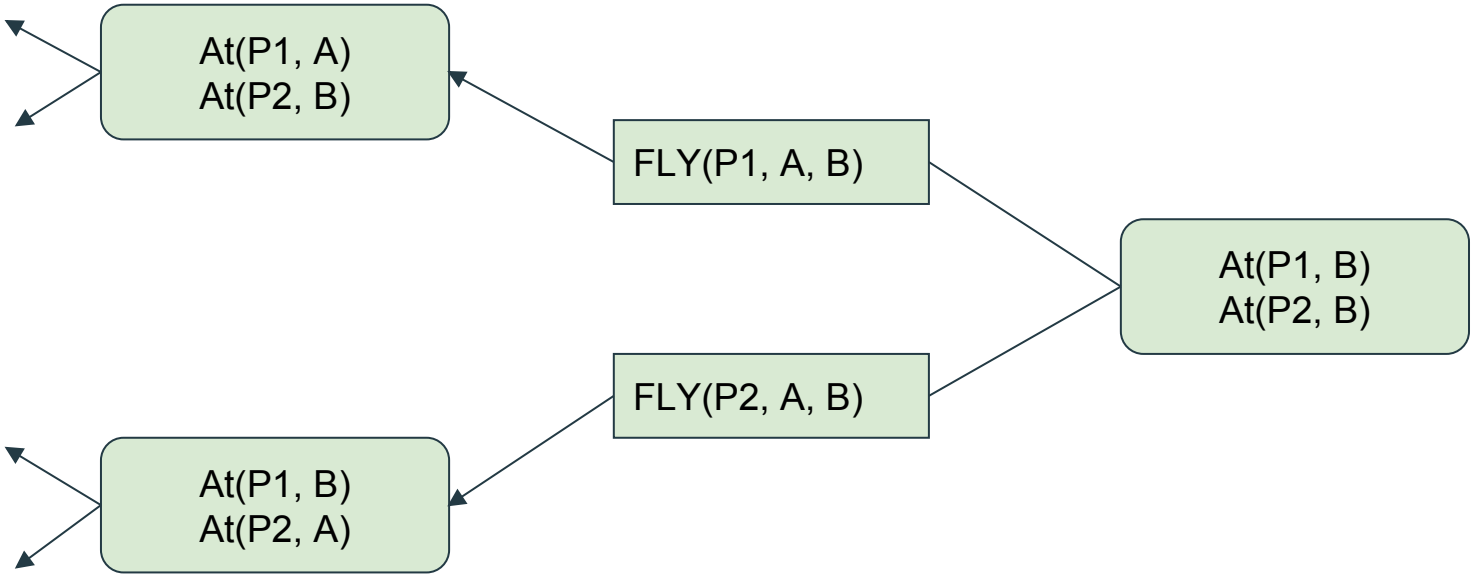
Drawbacks in Forward state space planning

- Forward state space search does not address the irrelevant action problem and this approach quickly bogs down without a good heuristic.

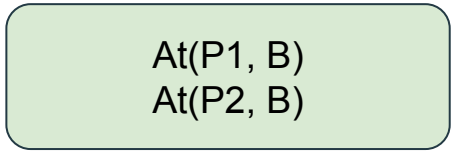
Backward state space planning

1. It is also called as Regression.
2. It uses STRIPS representation.
3. The problem formulation is similar to that of FSSS and consists of the initial state, actions, goal test and step cost.
4. In BSSS, the searching starts at the goal, checks if it is the initial state. If not, it applies the inverse of the actions to produce sub goals until start state is reached.

Example

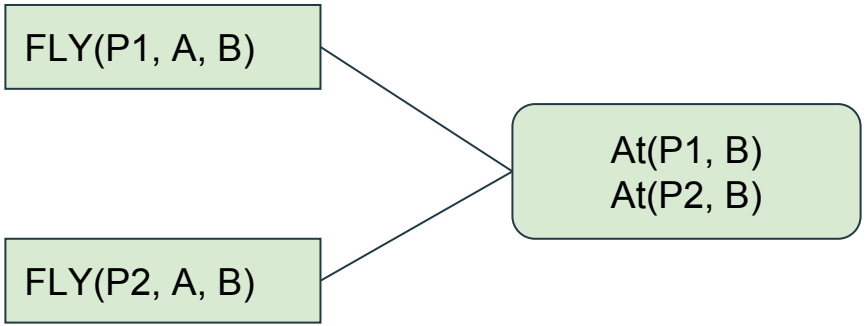


Backward state space planning:



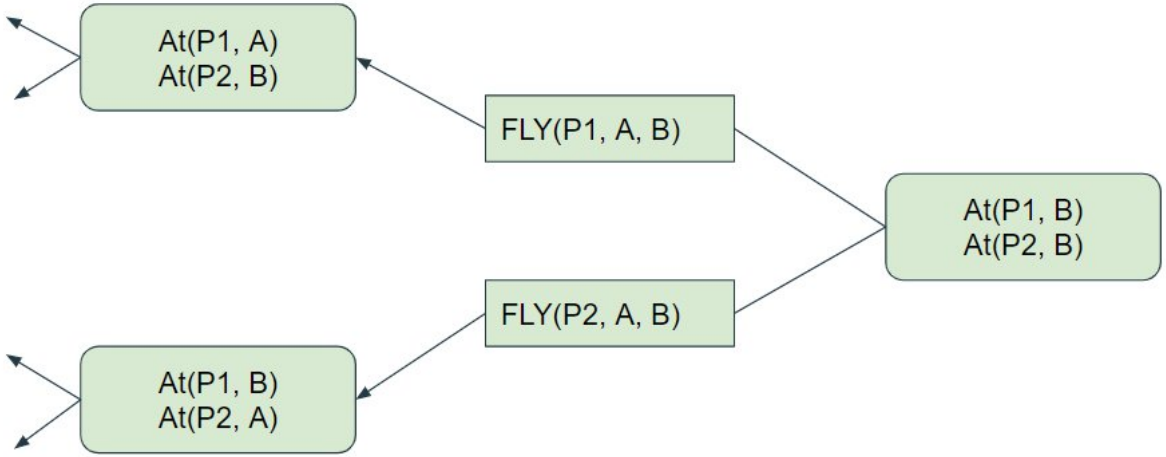
Initially, we are at goal (i.e airport B). now find out actions using which we reached the goal state.

Backward state space planning:



Now, we found the actions responsible for reaching goal state, Based on these actions, explore the initial states.

Backward state space planning:
Getting initial states,



So now we derived initial states from the actions (i.e., arrows pointing from actions to initial states).

Advantages of Backward state space planning

- It considers only relevant actions (i.e, only the conditions leading to reach goal are taken into account, since it works from backward.

Heuristic for state space search

- A heuristic function estimates the distance from a state to the goal; in STRIPS planning, the cost of each action is 1, so the distance is the number of actions.
- The basic idea is to look at the effects of the actions and at the goals that must be achieved and to guess how many actions are needed to achieve all the goals.
- Finding the exact number is NP hard, but it is possible to find reasonable estimates most of the time without too much computation. We might also be able to derive an admissible heuristic—one that does not overestimate.

Heuristic for state space search (Cont..)

There are two approaches

1. Relaxed problem:
    a. relaxed problem" refers to a simplified version of the original problem that may involve relaxing some constraints or assumptions.
    b. The goal of creating a relaxed problem is to make it easier to solve than the original problem, while still providing useful information about the original problem.
    c. Gives an admissible heuristic for the original problem

Heuristic for state space search (Cont..)

2. Subgoal independence:

- It include divide and conquer algorithm.
- Under certain assumptions, we can estimate the total cost of solving a set of subgoals by adding up the costs of solving each subgoal independently.
- This is a heuristic approach that is often used when the subgoals are relatively independent of each other and can be solved in parallel or without much interaction.
- The heuristics described here can be used in either the progression or the regression direction.

Partial order planning

Total order planning

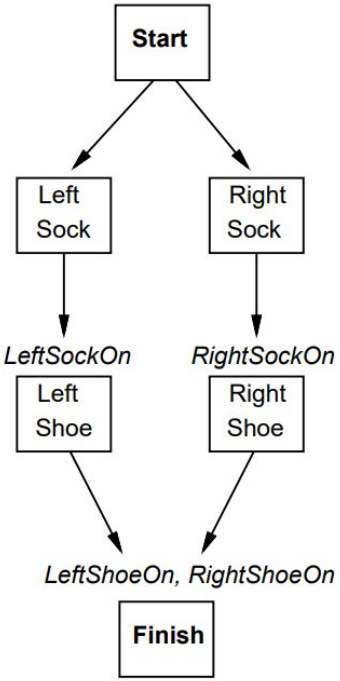
- Any planner that maintain a partial solution as a totally ordered list of steps found so far is called as total order planner / linear planner.
- Forward / Backward state space searches are forms of totally ordered plan search.
  - Explores only strictly linear sequence of actions, directly connected to the start or goal.
  - Cannot take advantage of problem decomposition.

Partial order planning (POP)

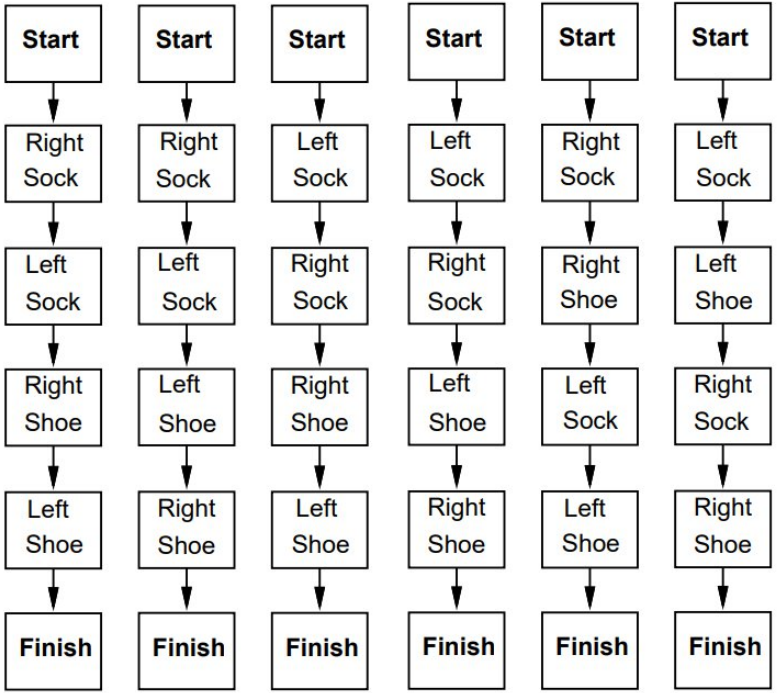
- Works on several subgoals independently.
- The subgoals are completed with subplans
- The sub plans are combined in any order.
- If we only represent partial order constraints on steps, then we have a partial order planner.
- In this case, we specify set of temporal constraints between pairs of steps of the form S1<S2 meaning that step S1 comes before, but not necessarily immediately before step S2.

POP Example: Putting on a pair of Shoe

Partial Order Plan:



Total Order Plans:



Example solution

- Goal(RightShoeOn ^ LeftShoeOn)
- Init()
- Action: RightShoe
  - PRECOND: RightSockOn
  - EFFECT: RightShoeOn
- Action: RightSock
  - PRECOND: None
  - EFFECT: RightSockOn
- Action:LeftShoe
  - PRECOND: LeftSockOn
  - EFFECT: LeftShoeOn
- Action: LeftSock
  - PRECOND: None
  - EFFECT: LeftSockOn

How to define partial order plan?

- Set of actions:
  - These are the steps of plan.
  - For e.g.: Set of Actions = {Start, Rightsock, Rightshoe ,Leftsock, Leftshoe, Finish}
- Set of ordering constraints/preconditions:
  - Preconditions are considered as ordering constraints.(i.e. without performing action “x” we cannot perform action “y”)
  - For e.g.: Set of ordering = {Right-sock <right-shoe; left-sock<left-shoe} that is in order to wear shoe first we should wear a sock

How to define partial order plan?

- Set of causal links: Action A achieves effect “E” for action B

  - You can understand that if you buy an apple its effect can be eating an apple and the precondition of eating an apple is cutting apple.
  - For e.g. Set of Causal Links = {Right-sock-> Right-sock-on -> Right-shoe}
- Set of open preconditions:

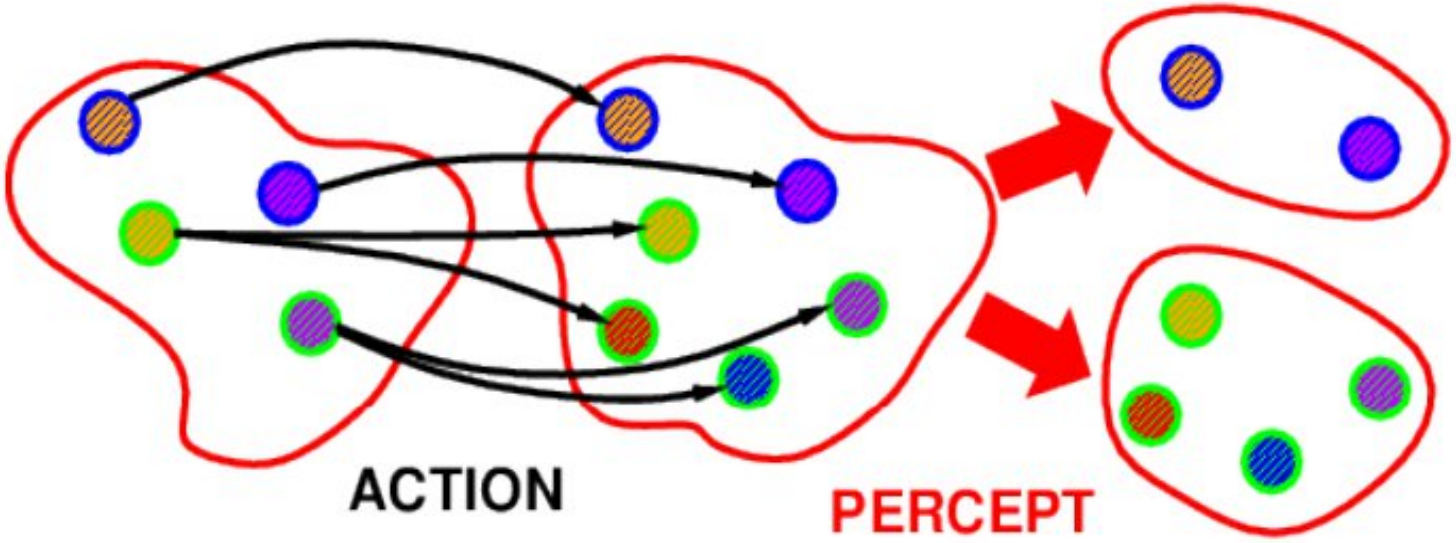
$$RightSock \xrightarrow{RightSockOn} RightShoe$$

  - Preconditions are called open if it cannot be achieved by some actions in the plan.
  - A consistent Plan doesn’t have cycle of constraints; it doesn’t have conflicts in the causal links and doesn’t have open preconditions so it can provide a solution for POP problem.

Conditional planning

- It deals with uncertainty by inspecting what is happening in the environment at predetermined points in the plan.
- It can take place in fully observable and non-deterministic environments. It will take actions and must be able to handle every outcome for the action taken.
- For instance, If <test-cond> then plan A else plan B.
- In case of a vacuum cleaner problem, If At Left ^ Clean then Right else Suck.

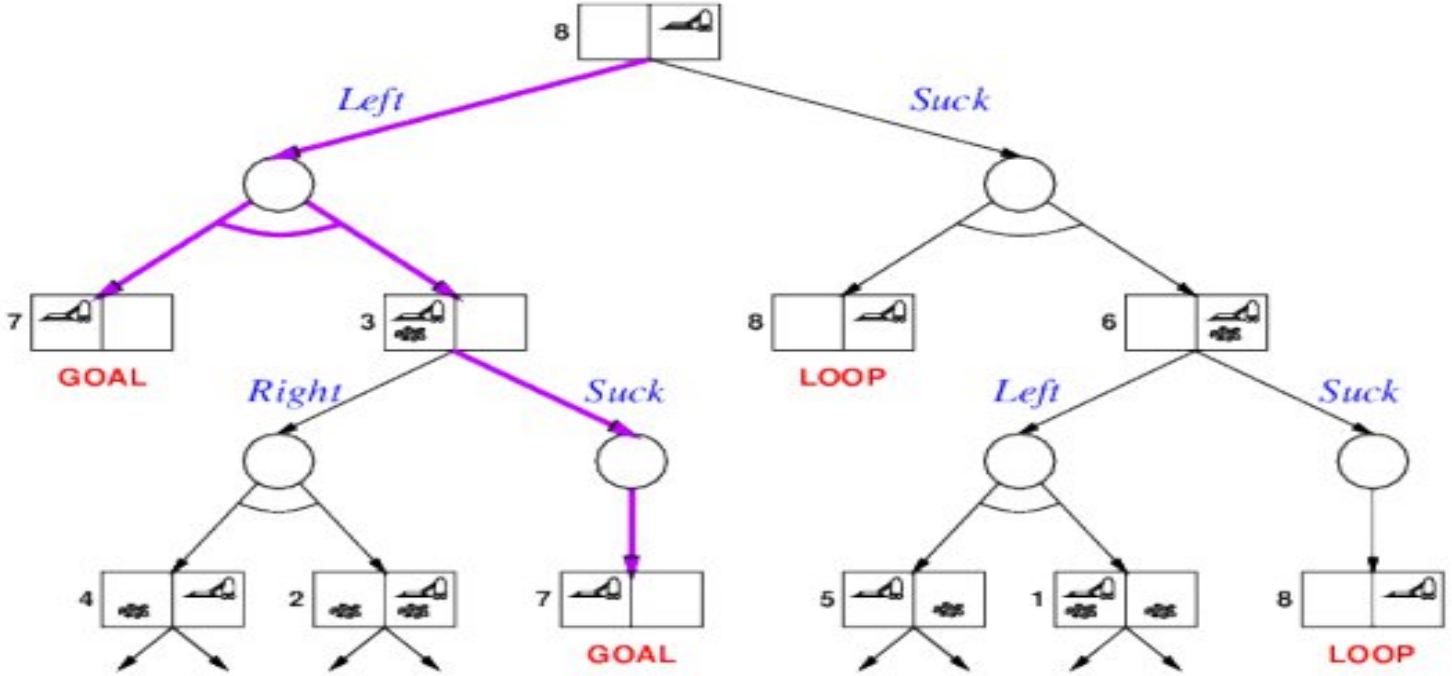
- If the world is nondeterministic or partially observable
  then percepts usually *provide information*,
  i.e., *split up* the belief state



- Conditional plans check (any consequence of KB +) percept
- $[\ldots, \textbf{if } C \textbf{ then } Plan_A \textbf{ else } Plan_B, \ldots]$
- Execution: check C against current KB, execute "then" or "else"
- Need *some* plan for *every* possible percept
  - game playing: *some* response for *every* opponent move
  - backward chaining: *some* rule such that *every* premise satisfied
- AND–OR tree search (very similar to backward chaining algorithm)

Example

- Double Murphy: sucking or arriving may dirty a clean square



In the “double-Murphy” vacuum world, the plan is:

[
  Left,
  if $AtL \wedge CleanL \wedge CleanR$
    then [ ]
    else *Suck*
]

Ex) "Fixing a flat tire"

(1) Possible operators

- Op(ACTION:Remove(x),
  PRECOND:On(x),
  EFFECT:Off(x) $\wedge$ ClearHub(x) $\wedge$ $\neg$On(x))
- Op(ACTION:PutOn(x),
  PRECOND:Off(x) $\wedge$ ClearHub(x),
  EFFECT:On(x) $\wedge$ $\neg$ClearHub(x) $\wedge$ $\neg$Off(x))
- Op(ACTION:Inflate(x),
  PRECOND:Intact(x) $\wedge$ Flat(x),
  EFFECT:Inflated(x) $\wedge$ $\neg$Flat(x))

(2) goal

- On(x) $\wedge$ Inflated(x)

(3) Initial conditions

- Inflated(Spare) $\wedge$ Intact(Spare) $\wedge$ Off(Spare) $\wedge$ On($Tire_1$) $\wedge$ Flat($Tire_1$)

(4) Initial plan

- [Remove($Tire_1$), PutOn(Spare)]

- The initial plan is good if there is no Intact($Tire_1$). But, if $Tire_1$ is intact, only the inflation is needed
- Conditional step

  If(<condition>,<ThenPart>,<ElsePart>,)

- If(Intact($Tire_1$),[Inflate($Tire_1$)],[Remove($Tire_1$), PutOn(Space)])
- Sensing Action

  $\forall x,s$ Tire(x) $\Rightarrow$
  KnowsWhether("Intact(x)",Result(CheckTire(x),s))

  In our action schema format

  - Op(ACTION:CheckTire(x),
    PRECOND:Tire(x),
    EFFECT:KnowsWhether("Intact(x)"))

- An algorithm for generating conditional plans
  - ex) Example:”Fixing a flat tire”
- Context :Contexts of steps are essential for keeping track of which steps can establish or violate the preconditions

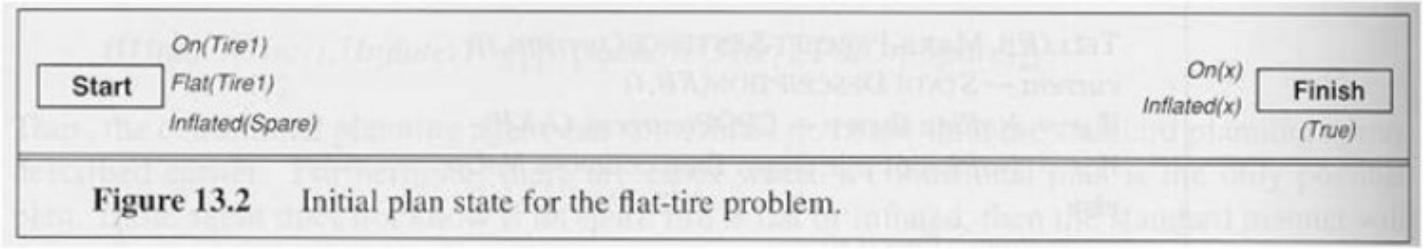


Figure 13.2 Initial plan state for the flat-tire problem.

- Two open conditions to be resolved
  - On(x)
  - Inflated(x)
- Introduce operator
  - Inflate($Tire_1$)
  - preconditions Flat($Tire_1$) and Intact($Tire_1$)

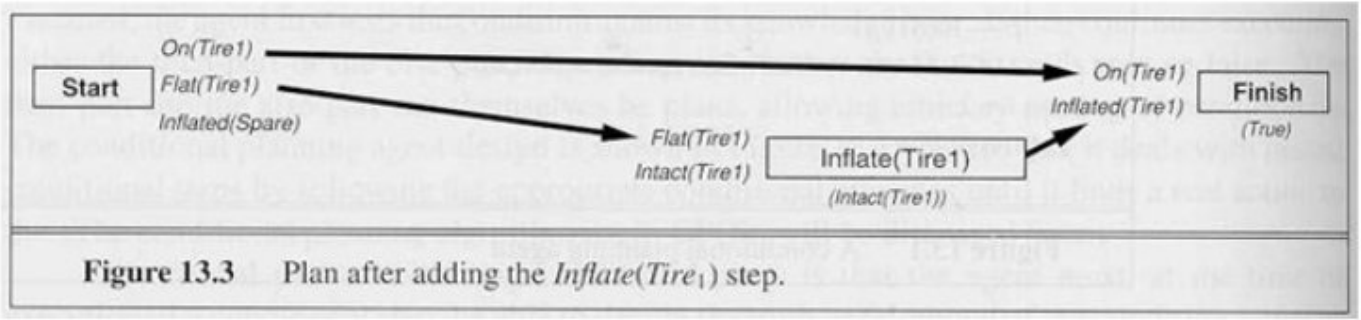


Figure 13.3 Plan after adding the *Inflate*($Tire_1$) step.

- Precondition Intact($Tire_1$) ?
  - There is no action that can make it satisfied
- But the action CheckTire(x) allows us to know the truth value of the preconditon ⇒ conditional step : Sensing action
- We add the CheckTime step to the plan with a conditional link :dotted arrow

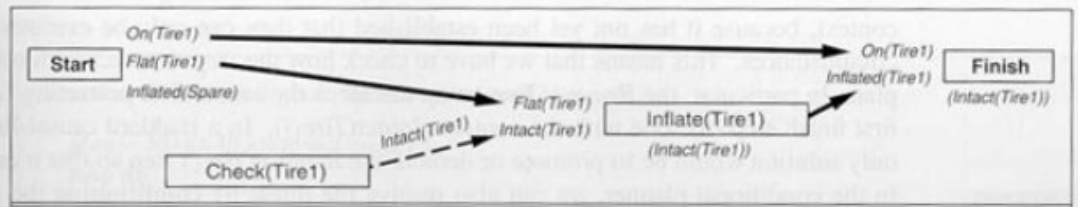


Figure 13.4 The plan to inflate $Tire_1$ will work provided that it is intact.

- We add steps for the case where Tire1 is not intact: another Finish action

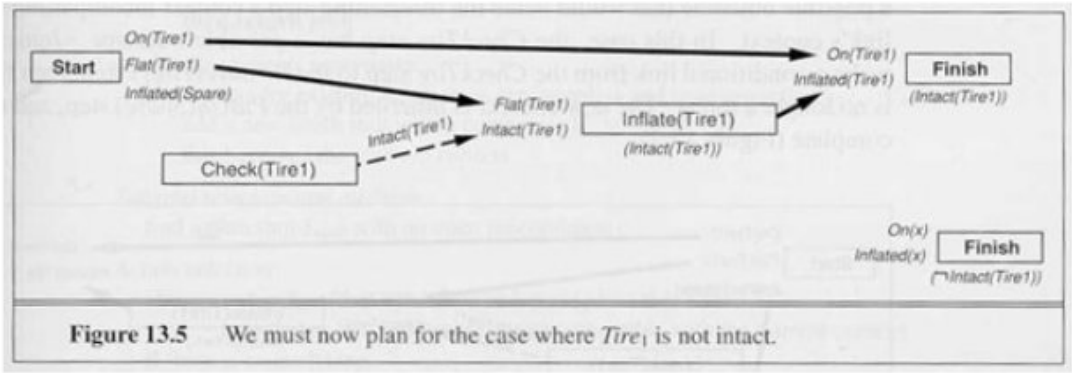


Figure 13.5 We must now plan for the case where $Tire_1$ is not intact.

- If we add Inflate($Tire_1$) to the new Finish step, the precondition Intact($Tire_1$) is inconsistent with ¬Intact($Tire_1$). Therefore, we link the start step to Inflated step ⇒ Context is useful, here.

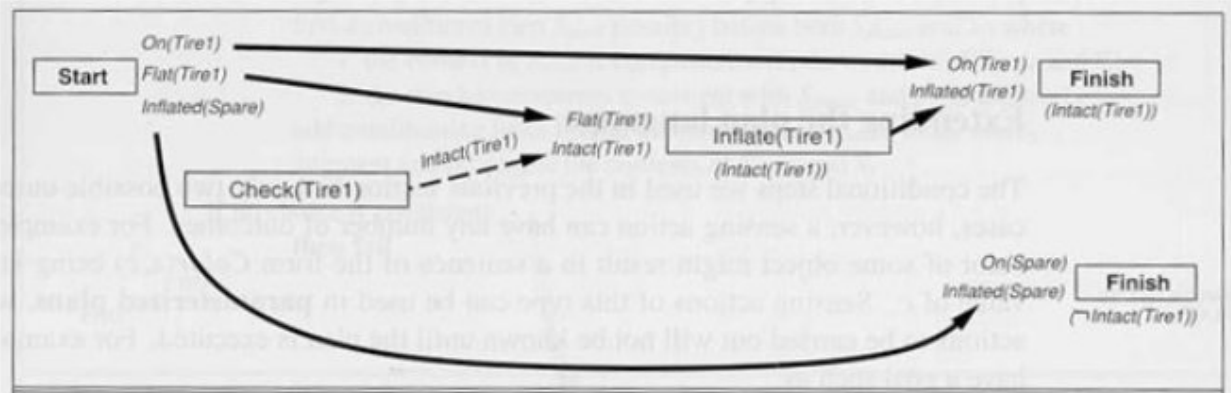


Figure 13.6 When $Tire_1$ is not intact, we must use the spare instead.

- We add Remove($Tire_1$), PutOn(Spare) to satisfy the condition On(Spare)
- But, the Remove(Tire1) can threaten the causal link protecting On($Tire_1$)
- We resolve the threat by conditioning the step so that its context becomes incompatible with the context of precondition it is threatening.
  In the example, CheckTire can give ¬Intact($Tire_1$)
- If we link from CheckTire to Remove($Tire_1$), then the Remove is no longer a threat

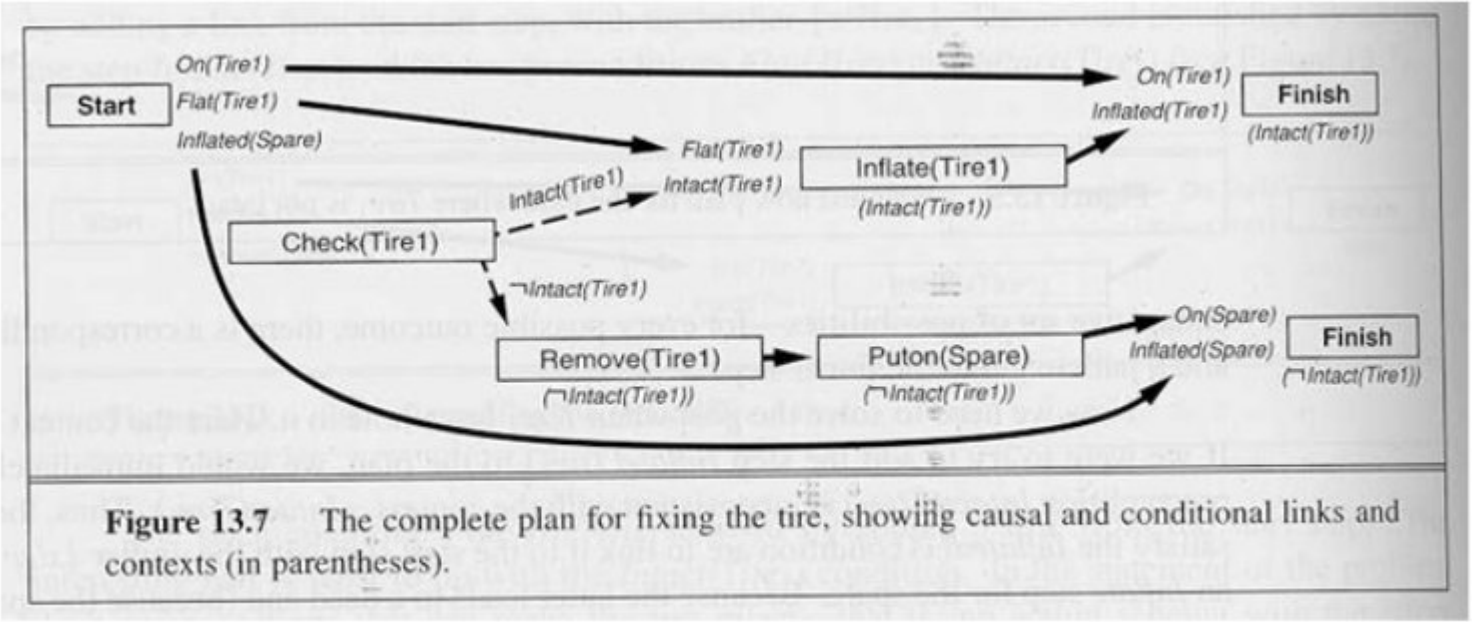


Figure 13.7 The complete plan for fixing the tire, showing causal and conditional links and contexts (in parentheses).