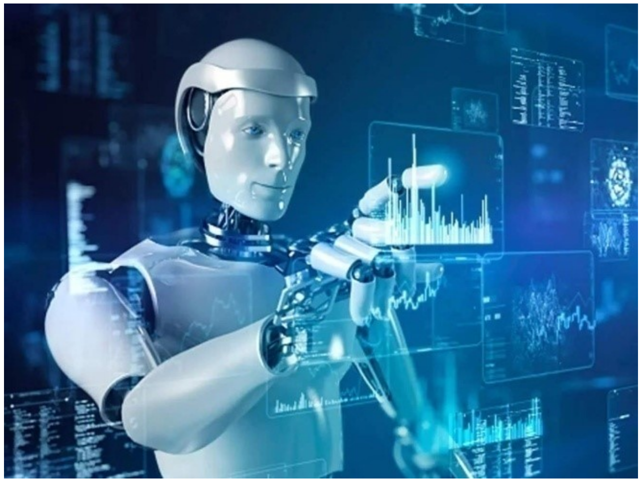
# SNS COLLEGE OF TECHNOLOGY

**(An Autonomous Institution)**
**COIMBATORE-35**
**Accredited by NBA-AICTE and Accredited by NAAC – UGC with A+ Grade**
**Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai**

## ARTIFICIAL INTELLIGENCE
## UNIT 3

## TOPIC: Planning with State Space Search

# TOPIC OUTLINE



Introduction to planning
Logic: A General Idea
Planning with State-Space Search

# Planning with State Space Search in AI

- State-space search is a fundamental approach in AI planning where an agent explores different possible states of the world to find a sequence of actions that leads to a goal state.
- It represents planning as a search problem, where each state represents a configuration of the world, and actions define transitions between states.

# Key Concepts of State-Space Search in Planning

**State:** A representation of the current situation in the world.

**Initial State:** The starting point of the agent.

**Goal State:** The desired outcome that the agent aims to reach.

**Operators (Actions):** Defined transformations that transition the agent from one state to another.

**Search Tree (or Graph):** A structure where nodes represent states and edges represent actions.

**Path Cost:** A numerical value that quantifies the cost of reaching a state.

# Planning with State-Space Search

The most straightforward approach of planning algorithm, is state-space search

*Forward state-space search (Progression)*

*Backward state-space search (Regression)*

The **descriptions of actions** in a planning problem, and specify both **preconditions and effects**

It is possible to **search in both direction**: either forward from the initial state or backward from the goal

We can also use the explicit action and goal representations, to derive effective heuristics automatically.

# Types of State-Space Search

**Forward (Progression) Search:**

Starts from the **initial state** and applies actions to move towards the **goal state**.

•Searches the state space by simulating actions one by one.

•Used in algorithms like **Breadth-First Search (BFS)** and *A Search**.

Example: A robot starting at (0,0) in a grid world, trying to reach (3,3).

# Backward (Regression) Search:

•Starts from the **goal state** and searches backwards to find an initial state that leads to it.
•Instead of applying actions, it tries to determine what states could lead to the goal.
•Efficient when there are fewer possible goal states than initial states.

**Bidirectional Search:**
•Combines **forward** and **backward** search to meet in the middle, reducing search time.
•Common in large state spaces where searching in both directions speeds up the process.

# Problem formulation

## Problem Formulation for Progression

- Initial state:
  - Initial state of the planning problem
- Actions:
  - Applicable to the current state.
  - First actions' preconditions are satisfied, Successor states are generated
  - Add positive literals to add list and negative literals to delete list.
- Goal test:
  - Whether the state satisfies the goal of the planning
- Step cost:
  - Each action is 1 (assumed)

# Contd…

## Progression

- From initial state, search forward by selecting operators whose preconditions can be unified with literals in the state
- New state includes positive literals of effect; the negated literals of effect are deleted
- Search forward until goal unifies with resulting state
- This is forward state-space search using STRIPS operators

# Contd…

## Example : Transportation of air cargo between airports.

$$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(JFK) \land Airport(SFO))$$

$$Goal(At(C_1, JFK) \land At(C_2, SFO))$$

$Action(Load(c, p, a),$
- PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
- EFFECT: $\neg At(c, a) \land In(c, p))$

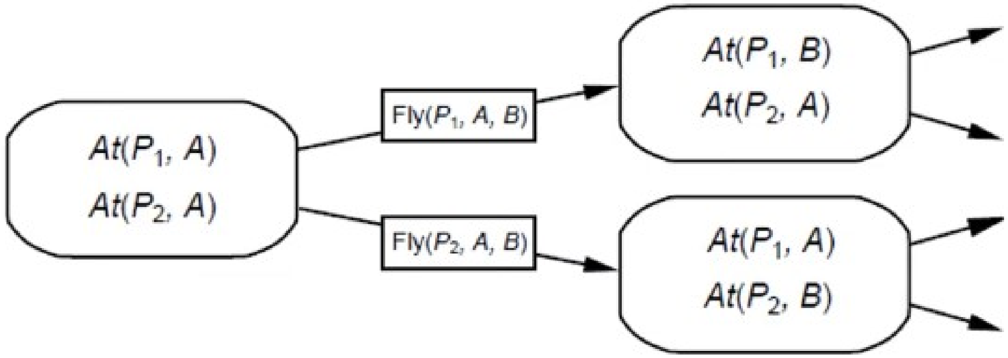
$Action(Unload(c, p, a),$
- PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
- EFFECT: $At(c, a) \land \neg In(c, p))$

$Action(Fly(p, from, to),$
- PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
- EFFECT: $\neg At(p, from) \land At(p, to))$

# Contd…

## Forward (progression) state-space search

- Starting from the initial state and using the problem's actions to search forward for the goal state.



# Contd…

## FSSS Algorithm

- (1) compute whether or not a state is a goal state,
- (2) find the set of all actions that are applicable to a state, and
- (3) compute a successor state, that is the result of applying an action to a state
- Algorithm takes as input the statement P = (O, s0, g) of a planning problem P. (O contains a list of actions)
- If P is solvable, then Forward-search(O, s0, g) returns a solution plan; otherwise it returns failure.

# Contd…

## Algorithm for FSSS

1. Forward-search(O, s0, g)
2. s ← s0
3. π ← the empty plan
4. loop
   1. if s satisfies g then return π
   2. applicable ← {a | a is a ground instance of an operator in O, and precond(a) is true in s}
   3. if applicable = Ø then return failure
   4. Non-deterministically choose an action a ∈ applicable
   5. s ← γ(s, a)
   6. π ← π.a

# Regression

## Problem Formulation for Regression

- Initial state:
  - Initial state of the planning problem
- Actions:
  - Applicable to the current state.
  - First actions' preconditions are satisfied, Successor states are generated
  - Add positive literals to add list and negative literals to delete list.
- Goal test:
  - Whether the state satisfies the goal of the planning
- Step cost:
  - Each action is 1 (assumed)

# Regression

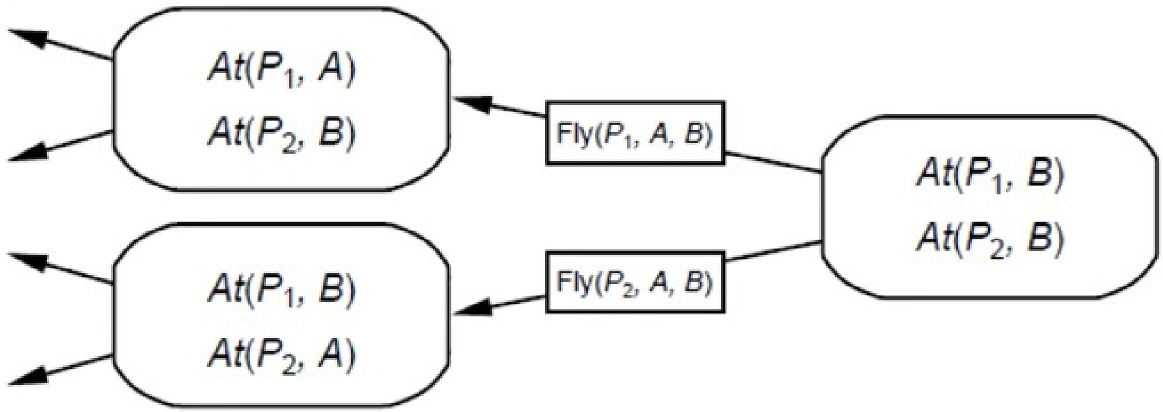
## Regression

- The goal state must unify with at least one of the positive literals in the operator's effect
- Its preconditions must hold in the previous situation, and these become subgoals which might be satisfied by the initial conditions
- Perform backward chaining from goal
- Again, this is just state-space search using STRIPS operators

# Regression

## Backward (regression) state-space search:

- Backward state search starting from the goal state(s)
- Using the inverse of the actions to search backward for the initial state.



# Regression

## BSSS Algorithm

1. Backward-search(*O*, *s*0, *g*)
2. π the empty plan
3. loop
   1. if *s*0 satisfies *g* then return π
   2. *applicable* ← {*a* | *a* is a ground instance of an operator in *O* that is relevant for *g*}
   3. if *applicable* = Ø then return failure
   4. Non-deterministically choose an action *a* ϵ *applicable*
   5. π ←*a*. π
   6. *g* ← $\gamma$-1(*g*, *a*)

# Regression

## Heuristics for State-Space Search

- How to find an admissible heuristic estimate?
  - Distance from a state to the goal?
  - Look at the effects of the actions and at the goals and guess how many actions are needed
- NP-hard
- Relaxed problem
- Subgoal independence assumption:
- The cost of solving a conjunction of subgoals, is approximated by the sum of the costs of solving each subgoal independently

# Regression

## Relaxation Problem

- Idea: removing all preconditions from the actions
- Which implies, the number of steps required to solve a conjunction of goals, and the number of unsatisfied goals
  - There may be two actions,
  - Some actions deletes the goal literal achieved by the other actions
  - some action may achieve multiple goals
- Combining relaxation with subgoal $\rightarrow$ exact # of unsatisfied goals


ARTIFICIAL INTELLIGENCE
RISE OF ROBOTS
THANKS