Introduction to Artificial Intelligence

COMP307
Planning and Scheduling 1:
Classic Planning

Yi Mei

yi.mei@ecs.vuw.ac.nz

Outline

- Why Planning
- What is Planning
- Planning Domain Definition Language (PDDL)
  - State
  - Action
- Planning Algorithms as State-Space Search
  - Forward Search
  - Backward Search

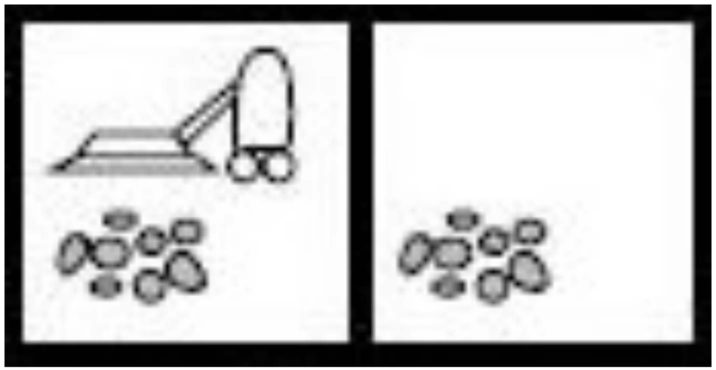
Why Planning

- We make plans (mostly unintentionally) everyday
  - Change clothes
  - Make breakfast
  - Go from one place to another
  - …
- Robots
  - Clean/Housekeeping
  - Delivery
  - Game playing
- Sounds trivial?
  - Computers don't think so
  - World is complex and uncertain

What is Planning

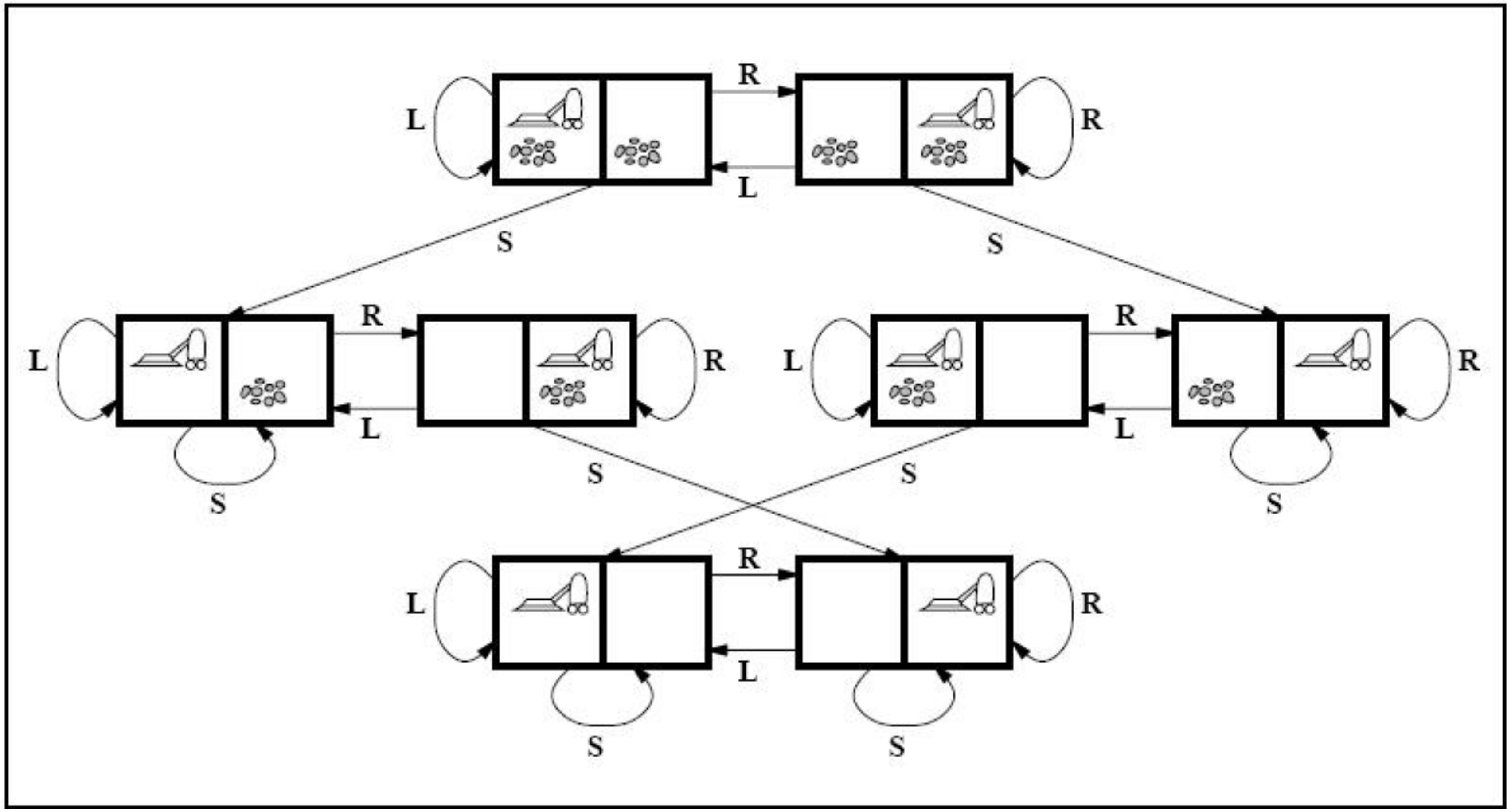
- Find a plan, which is a sequence of actions to achieve the goal state from the initial state.

- Example: a vacuum cleaner’s world
  - Two rooms (Left, Right)
  - Initial state: both rooms dirty, I am in room Left
  - Actions: {Suck, Move to Left, Move to Right}
  - Goal state: both rooms clean



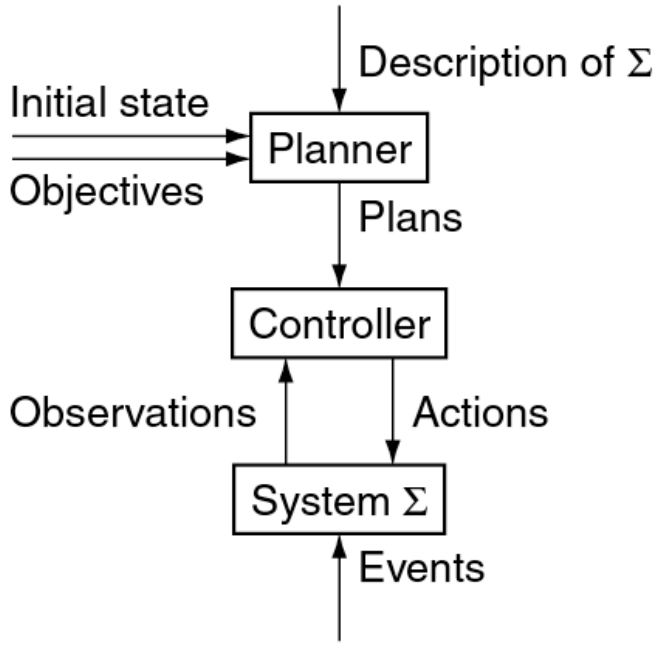
State Space in Planning

- The state space is essentially a graph
- Each node stands for a state
- Each link (directed edge) stands for an action



Conceptual Model

- State-transition systems (discrete-event systems)
- $\Sigma = (S, A, E, \gamma)$
  - $S = \{s_1, s_2, \ldots\}$ is a finite set of states
  - $A = \{a_1, a_2, \ldots\}$ is a finite set of actions
  - $E = \{e_1, e_2, \ldots\}$ is a finite set of events
  - $\gamma : S \times A \times E \rightarrow 2^S$ is a state-transition function
- Represent as a directed graph
- Actions are transitions that are controlled
- Events are transitions that are contingent



- Planner: given Σ, initial state, objective, provide a plan for controller
- Controller: given a state and plan, provide an action

Classical Planning

- **Deterministic**
  - $\gamma: S \times A \rightarrow S$: each state and action leads to a single other state
- **Static**
  - $\Sigma = (S, A, \gamma)$: no contingency event
- **Finite**
  - There are finite number of states and actions
- **Fully observable**
  - We know everything about Σ
- **Restricted goals**
  - Can be specified as an explicit goal state(s)
- **Implicit time**
  - Actions have no duration, instantaneous state transition

Classical Planning

- Problem
  - The environment $\Sigma = (S, A, \gamma)$
  - The initial state s_0
  - The goal state(s) S_g
- Solution (Plan)
  - A sequence of actions $(a_1, a_2, \ldots)$
  - State transitions $(s_1, s_2, \ldots s_k)$, where $s_1 = \gamma(s_0, a_1)$, $s_2 = \gamma(s_0, a_2)$, …, and $s_k \in S_g$ is a goal state

- How to represent the states and actions?
- How to perform the search for a solution efficiently
  - Which search space, which algorithm, and what heuristics and control techniques to use for finding a solution.

Planning Domain Definition Language

- A classic representation for planning

- A **state** is represented as a conjunction of fluents that are ground (no variable) and functionless atoms.
  - Lowercase = variable
  - Capital letters = value
  - Opposite to the style of Probability

- Example
  - $At(x)$ is invalid: not ground and has variable x
  - $\neg Clean(Right)$ is invalid: has the negate function
  - $At(Father(Fred), Sydney)$ is invalid: has the function $Father(Fred)$
  - $At(Left) \wedge Clean(Left)$ is valid

- Closed world assumption: any fluents that are not mentioned are false.
  - $At(Left)$ means Left is not clean, as $Clean(Left)$ is not mentioned

Planning Domain Definition Language

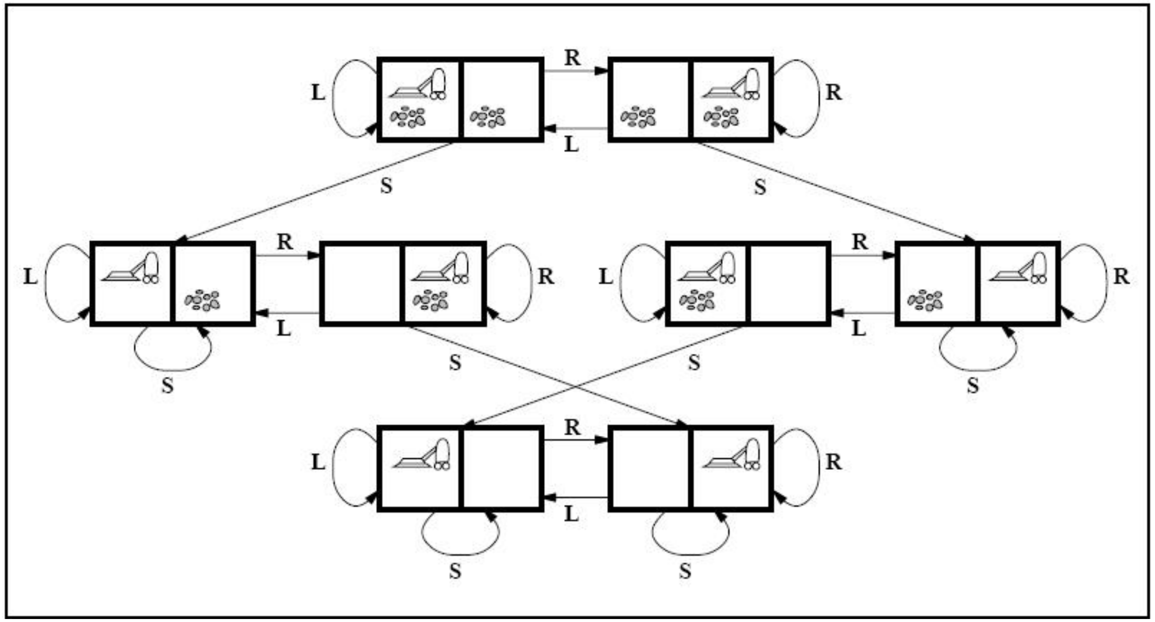
- An **action** consists of an action name, all the variables used, a precondition and an effect.
  - Difference from State: there can be variables in actions

- Example: a plane flies from an airport to another airport
  - $Action(Fly(p, from, to),$
  - PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  - EFFECT: $\neg At(p, from) \wedge At(p, to))$

- **Applicability**: an action a is applicable in state s, if its precondition is satisfied by s
- **Multiple instantiation**: $Fly(NZ410, Auckland, Wellington)$ and $Fly(NZ87, Auckland, HK)$

PDDL in Vacuum Cleaner's World

- $Init(At(Left))$
- $Goal(Clean(Left) \wedge Clean(Right))$
- $Action(MoveLeft(\,),$
-   PRECOND:
-   EFFECT: $At(Left) \wedge \neg At(Right))$
- $Action(MoveRight(\,),$
-   PRECOND:
-   EFFECT: $At(Right) \wedge \neg At(Left))$
- $Action(Suck(x),$
-   PRECOND: $At(x)$
-   EFFECT: $Clean(x))$



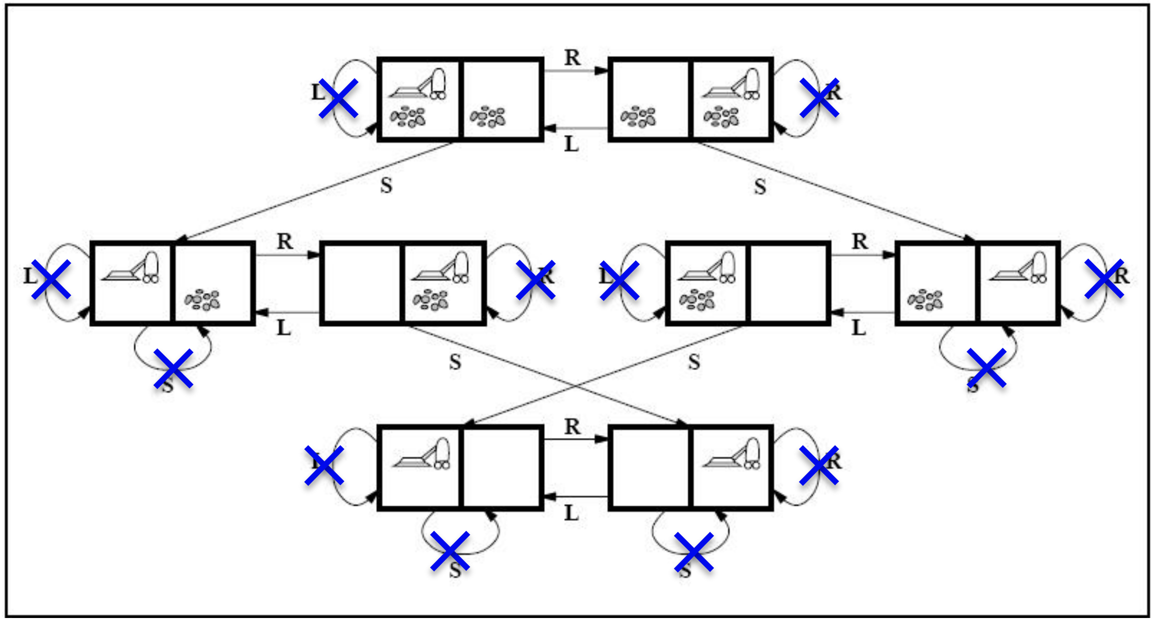
Update State with Action

- Delete list DEL(a): remove the fluents that appear as negative literals in the action's effects
- Add list ADD(a): add the fluents that are positive literals in the action's effects
- $s' = \gamma(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$

- Example in the vacuum cleaner's world
  - $s_1 = At(Left)$, $a_1 = MoveRight()$
    - EFFECT$(a_1) = At(Right) \wedge \neg At(Left)$
    - $s_1 -$ DEL$(a_1) = \emptyset$
    - $\gamma(s_1, a_1) = \emptyset \cup$ ADD$(a_1) = At(Right)$
  - $s_2 = At(Right)$, $a_1 = Suck(Right)$
    - EFFECT$(a_2) = Clean(Right)$
    - $s_2 -$ DEL$(a_2) = At(Right)$
    - $\gamma(s_2, a_2) = At(Right) \cup$ ADD$(a_2) = At(Right) \wedge Clean(Right)$

A Better PDDL

- $Init(At(Left))$
- $Goal(Clean(Left) \land Clean(Right))$
- $Action(MoveLeft(\,),$
- PRECOND: $At(Right)$
- EFFECT: $At(Left) \land \neg At(Right))$
- $Action(MoveRight(\,),$
- PRECOND: $At(Left)$
- EFFECT: $At(Right) \land \neg At(Left))$
- $Action(Suck(x),$
- PRECOND: $At(x) \land \neg Clean(x)$
- EFFECT: $Clean(x))$



Generalised PDDL

- Assuming there are four rooms {Left, Right, Top Bottom}
  - Can move from any room to any room
  - Otherwise, we need more information, e.g. $Adjacent(Left, Top)$, …

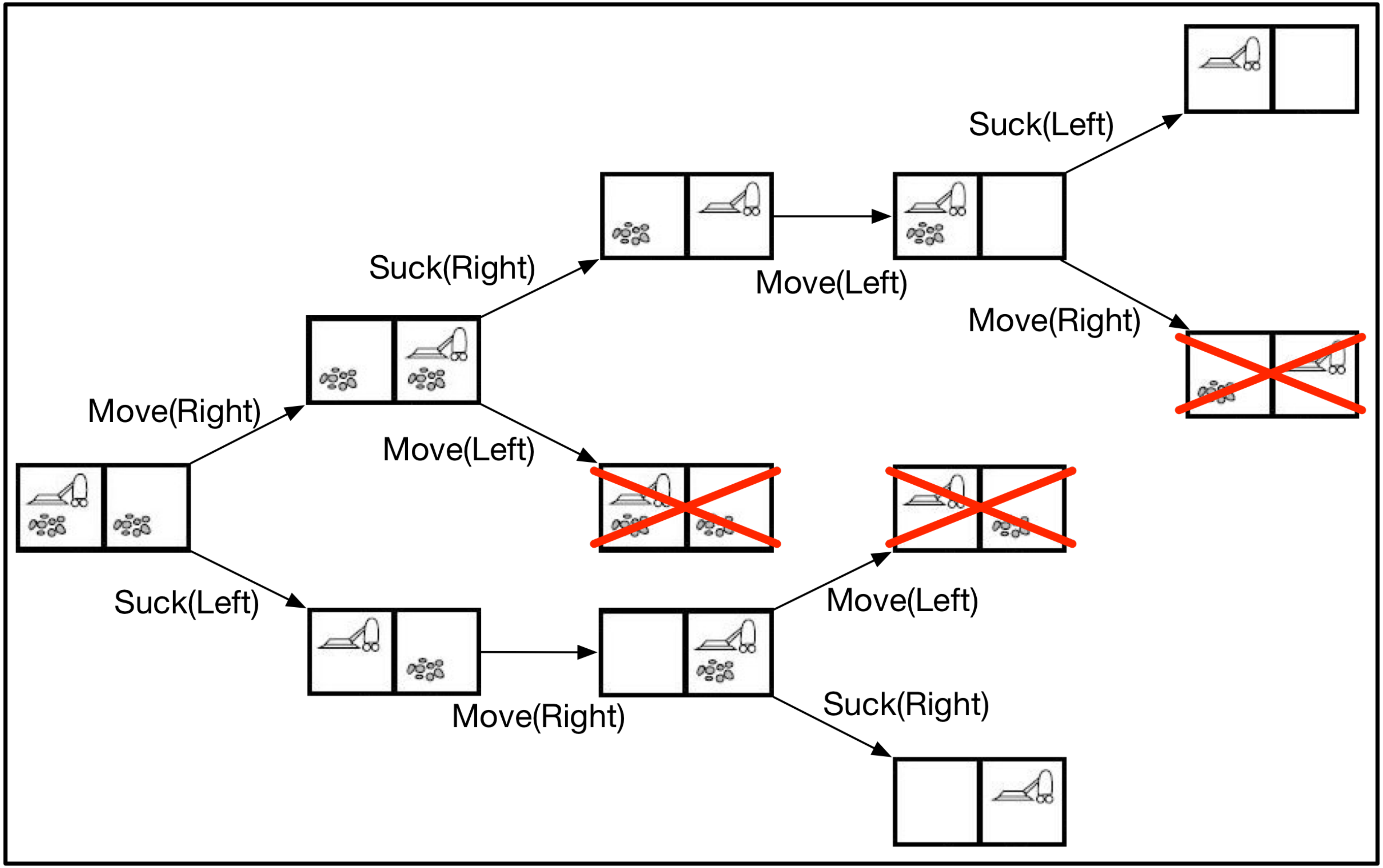
- $Init(At(Top), Adjacent(Left, Top), \ldots)$
- $Goal(Clean(Left) \wedge Clean(Right) \wedge Clean(Top) \wedge Clean(Bottom))$
- $Action(Move(x, y),$
- PRECOND: $At(x) \wedge \neg At(y) \wedge Adjacent(x, y)$
- EFFECT: $At(y) \wedge \neg At(x))$
- $Action(Suck(x),$
- PRECOND: $At(x) \wedge \neg Clean(x)$
- EFFECT: $Clean(x))$

Planning Algorithms as State-Space Search

- Forward (progression) state-space search
  - Start with the initial state
  - Examine all the applicable actions for the current state
  - Avoid loop – never go back to previous states
  - Until reach a goal state

- There can be multiple different goal states
  - All the goal state fluents are present
  - Other fluents can be present as well
  - E.g.
    - Both rooms are clean, the cleaner can be in either room
    - $Clean(Left) \land Clean(Right) \land At(Left)$
    - $Clean(Left) \land Clean(Right) \land At(Right)$

Planning Algorithms as State-Space Search



Planning Algorithms as State-Space Search

- A plan is a path from the root node to a non-loop leaf node

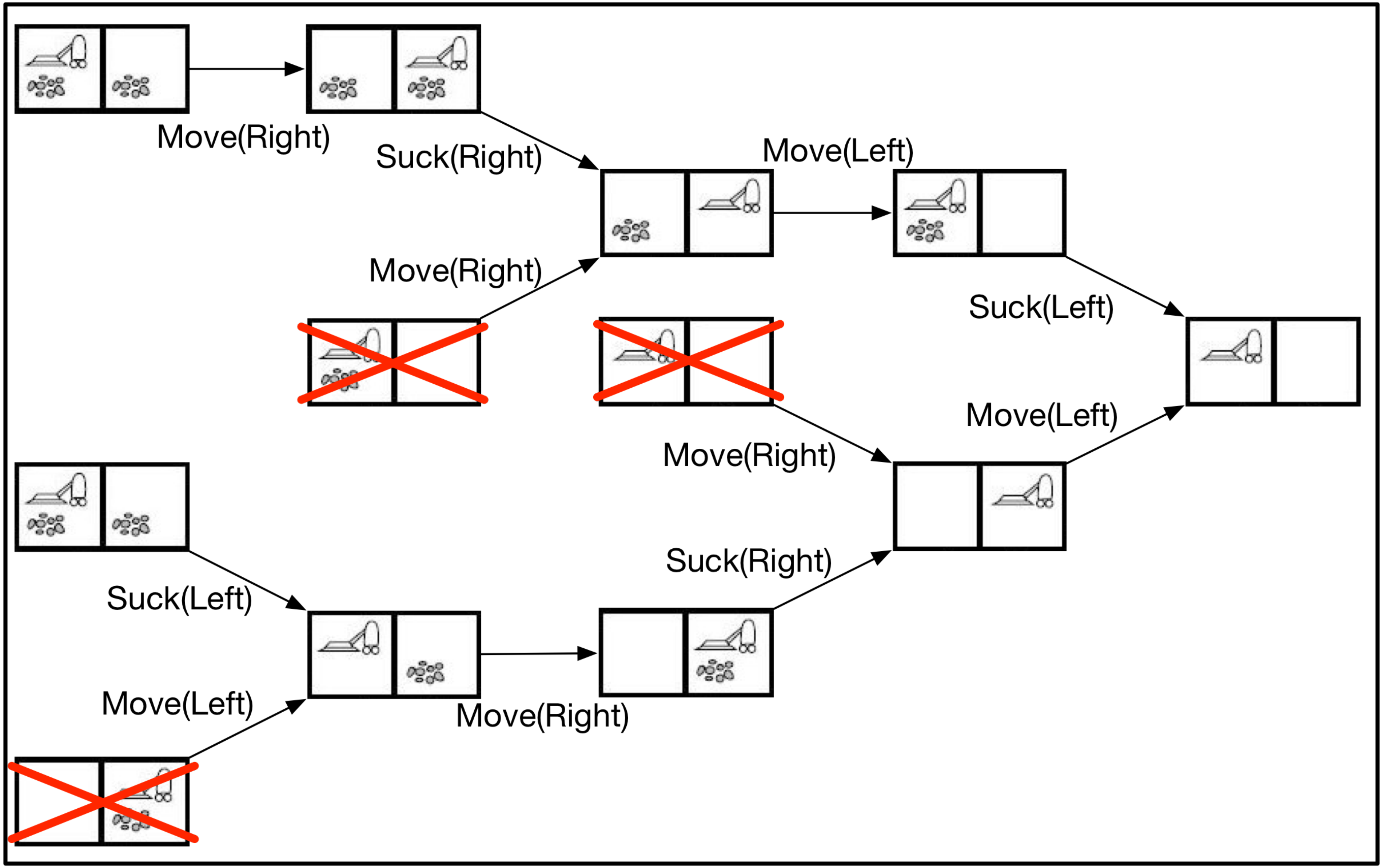
- Initial state: $At(Left)$
- Action 1: $Suck(Left)$
- State 1: $At(Left) \wedge Clean(Left)$
- Action 2: $Move(Right)$
- State 2: $At(Right) \wedge Clean(Left)$
- Action 3: $Suck(Right)$
- State 3 (Goal): $At(Right) \wedge Clean(Left) \wedge Clean(Right)$

Planning Algorithms as State-Space Search

- Backward (regression) relevant state-space search
  - Start with a goal state (random if there are more than one)
  - Examine all the relevant actions
    - Could be the *last* step leading to the current state
      - At least one effect is an element (positive fluent) of the current state
      - Has no effect that negates an element of the current state
      - Precondition of the action does not contain any conflicting fluent with the current state
  - Avoid loop
  - Until reach the initial state

$$s' = \gamma^{-1}(s, a) = \left(s - effects^{+}(a)\right) + precond(a)$$

Planning Algorithms as State-Space Search



Planning Algorithms as State-Space Search

- A plan is a path from a non-loop leaf node to the root node or the earliest goal state in the middle

- Initial state: $At(Left)$
- Action 1: $Suck(Left)$
- State 1: $At(Left) \wedge Clean(Left)$
- Action 2: $Move(Right)$
- State 2: $At(Right) \wedge Clean(Left)$
- Action 3: $Suck(Right)$
- State 3 (Goal): $At(Right) \wedge Clean(Left) \wedge Clean(Right)$

Summary

- What is planning? – Find a sequence of actions to achieve the goal state from the initial state
- Planning Domain Definition Language (PDDL) – a standard language to represent planning problems
- Planning algorithms as state-space search
  - Forward search
  - Backward search

- Suggested reading: Text book, chapter 10: Classical Planning