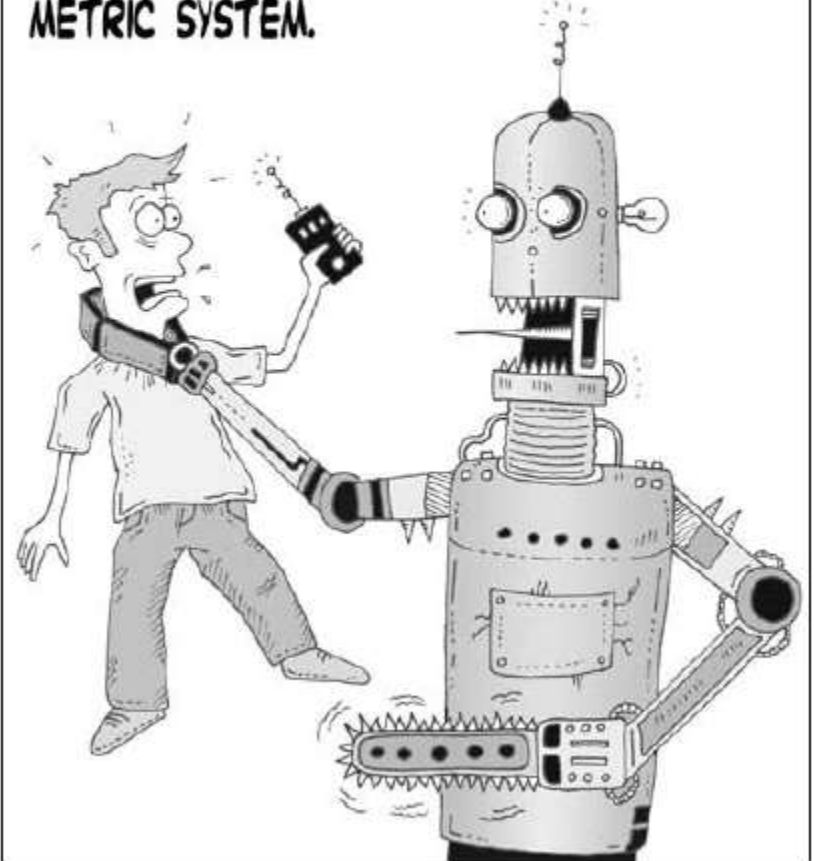


# Chapter 11

## Planning

Dr. Daisy Tang

IT WAS ALREADY TOO LATE WHEN JIMMY  
REALIZED HE HAD FORGOTTEN TO  
CONVERT HIS UNITS BACK INTO THE  
METRIC SYSTEM.



# Real-World Problems

---

- **Planning**: the task of coming up with a sequence of actions that will achieve a goal
  - Search-based problem-solving agent
  - Logical planning agent
  - Complex/large scale problems?
- For the discussion, we consider **classical planning environments** that are fully observable, deterministic, finite, static and discrete (in time, action, objects and effects)

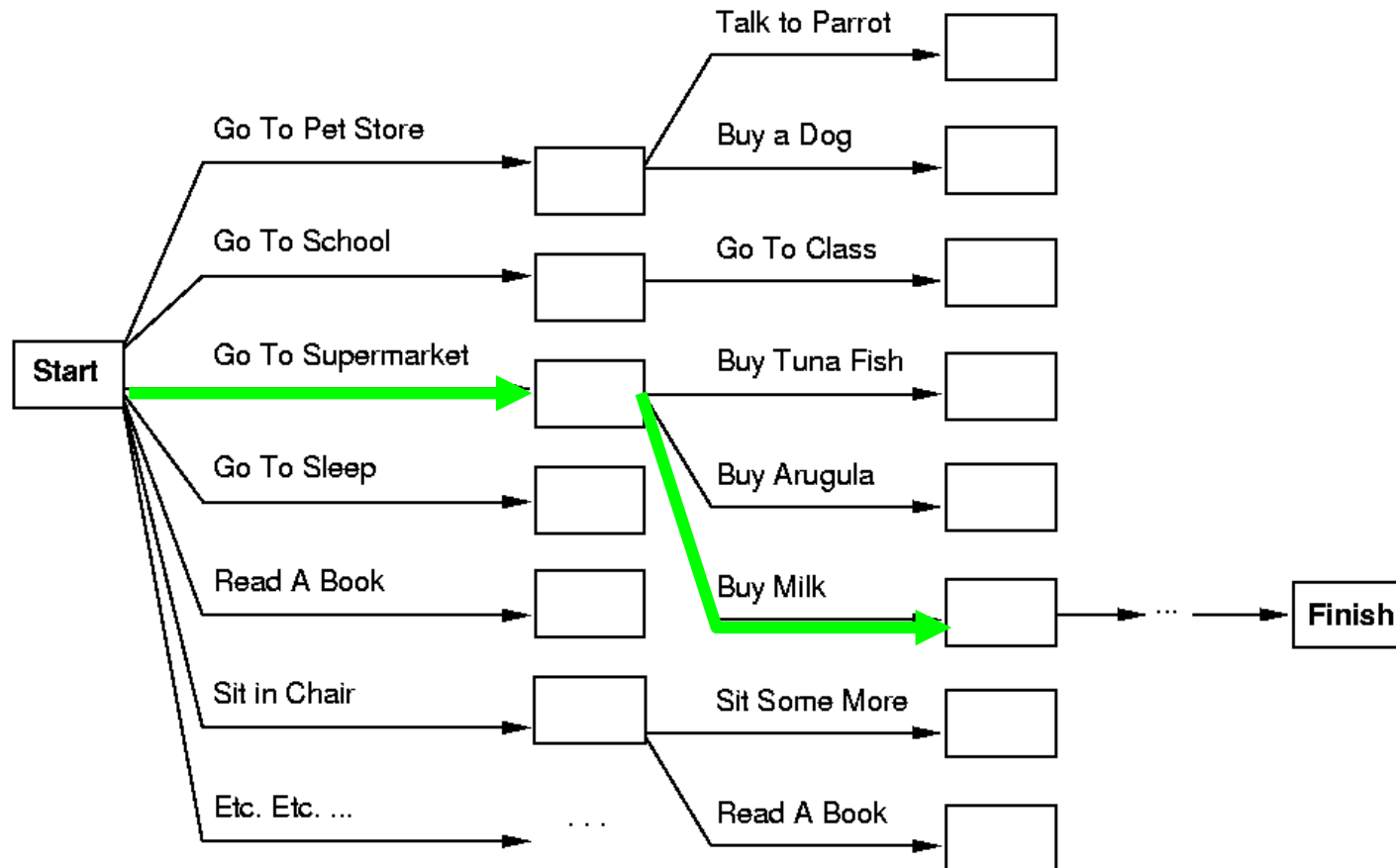
# Problems with Standard Search

---

- Overwhelmed by irrelevant actions
- Finding a good heuristic function is difficult
- Cannot take advantage of problem decomposition

# Example

- Consider the task: get milk, bananas, and a cordless drill
  - Standard search algorithms seem to fail miserably
  - Why? Huge branching factor & heuristics



# Problem Decomposition

---

- Perfectly decomposable problems are delicious but rare
  - **Partial-order planner** is based on the assumption that most real-world problems are **nearly decomposable**
  - Be careful, working on some subgoal may undo another subgoal

# Planning vs. Problem Solving

---

- Planning agent is very **similar** to problem solving agent
  - Constructs plans to achieve goals, then executes them
- Planning agent is **different** from problem solving agent in:
  - Representation of goals, states, actions
  - Use of explicit, logical representations
  - Way it searches for solutions

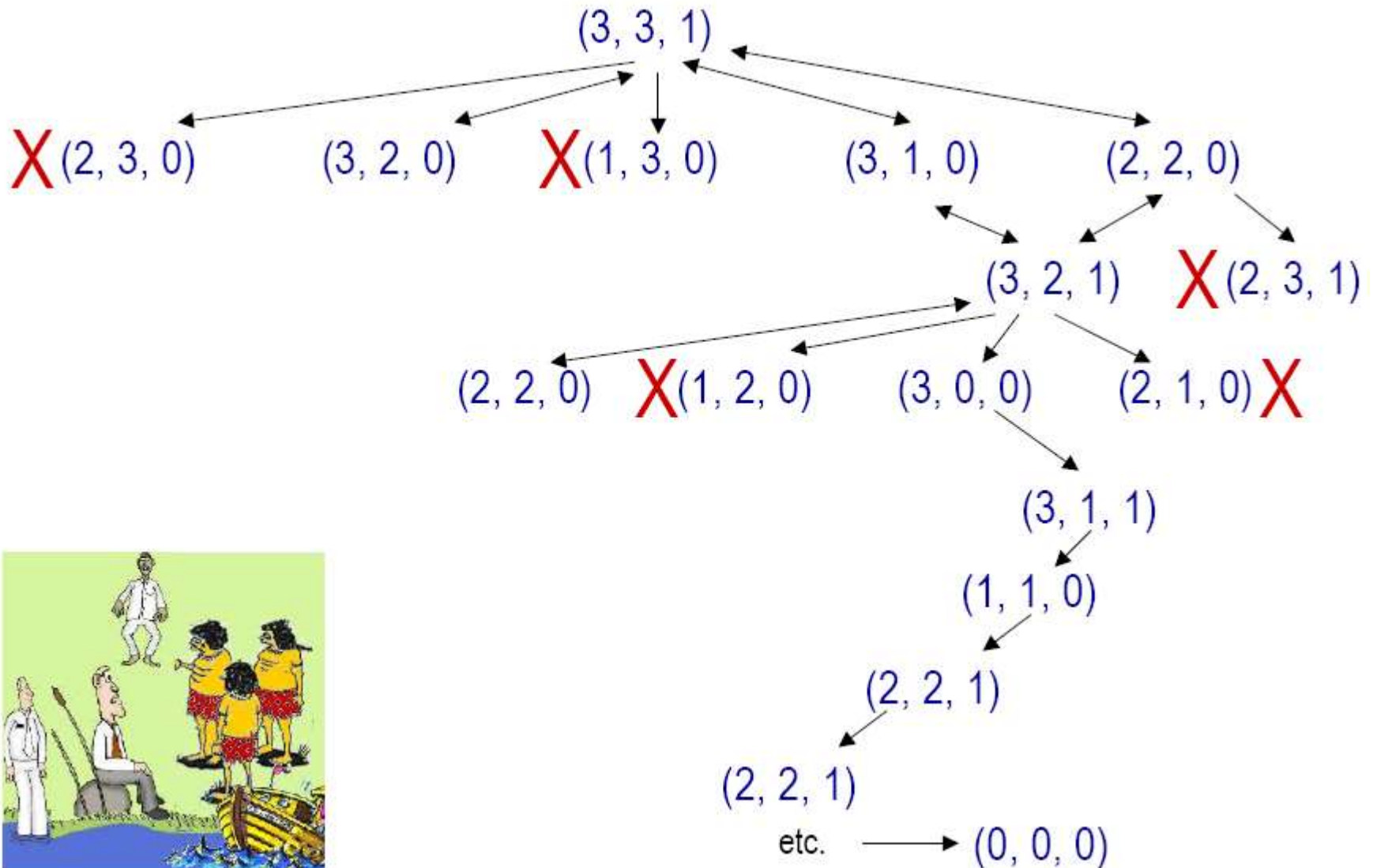
# Planning vs. Problem Solving

---

- Planning systems do the following:
  - divide-and-conquer
  - relax requirement for sequential construction of solutions

	<b>Problem Sol.</b>	<b>Planning</b>
States	data structures	logical sentences
Actions	code	preconditions/outcomes
Goal	code	logical sentences
Plan	sequence from $s_0$	constraints on actions

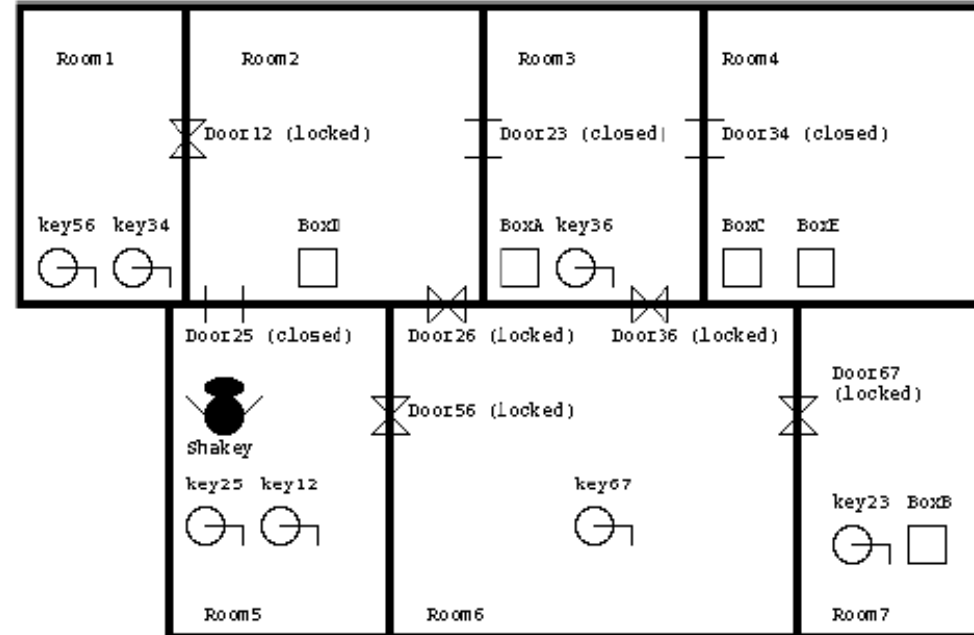
# Famous Problem Solver Task: "Missionaries and Cannibals"





# Planning-Based Approach to Robot Control

- **Job of planner:** generate a goal to achieve, and then construct a plan to achieve it from the current state
- Must define representations of:
  - **Actions:** generate successor state descriptions by defining preconditions and effects
  - **States:** data structure describing current situation
  - **Goals:** what is to be achieved
  - **Plans:** solution is a sequence of actions



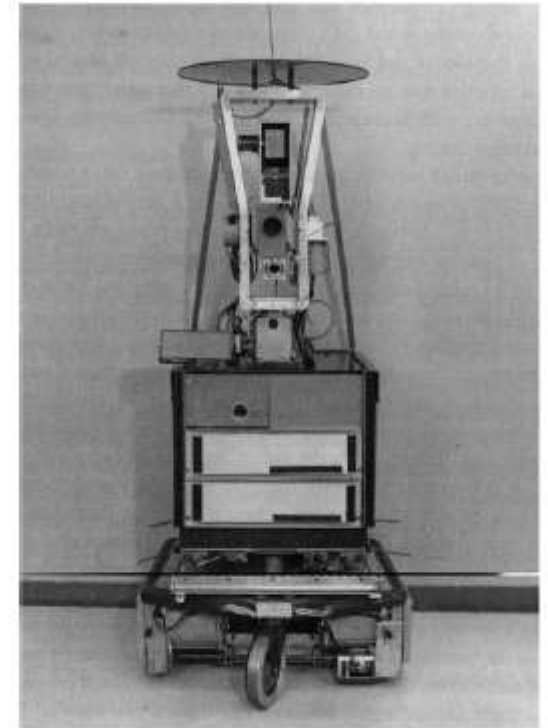
# Many AI Planners in History

---

- Well-known AI Planners:
  - STRIPS (Fikes and Nilsson, 1971): theorem-proving system
  - ABSTRIPS (Sacerdoti, 1974): added hierarchy of abstractions
  - HACKER (Sussman, 1975): use library of procedures to plan
  - NOAH (Sacerdoti, 1975): problem decomposition and plan reordering

# STRIPS-Based Approach to Robot Control

- Use **first-order logic** and theorem proving to plan strategies from start to goal
- STRIPS language:
  - “Classical” approach that most planners use
  - Lends itself to efficient planning algorithms
- **Environment**: office environment with specially colored and shaped objects
- **STRIPS planner**: developed for this system to determine the actions of the robot should take to achieve goals
- Cost of Shakey: \$100, 000



Shakey (SRI), 1960's

# STRIPS

---

- STRIPS (STanford Research Institute Problem Solver)
  - a restrictive way to express **states**, **actions** and **goals**, but leads to more efficiency
- **States**: conjunctions of ground, function-free, and positive literals, such as  $\text{At}(\text{Home}) \wedge \text{Have}(\text{Banana})$ 
  - Closed-world assumption is used
- **Goals**: conjunctions of literals, may contain variables (existential), hence goal may represent more than one state
  - E.g.  $\text{At}(\text{Home}) \wedge \neg \text{Have}(\text{Bananas})$
  - E.g.  $\text{At}(x) \wedge \text{Sells}(x, \text{Bananas})$
- **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. Any variables in it must also appear in parameter list
  - **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)

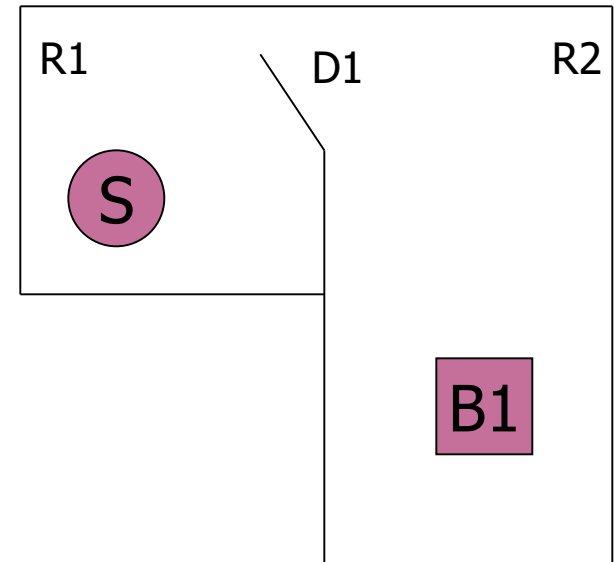
*At(p) Sells(p,x)*

<b>Buy(x)</b>
---------------

*Have(x)*

# An Example of STRIPS World Model

- A world model is a set of facts
- The robot's knowledge can be represented by the following predicates:
  - **INROOM**( $x, r$ ), where  $x$  is a movable object,  $r$  is a room
  - **NEXTTO**( $x, t$ )
  - **STATUS**( $d, s$ ), where  $d$  is a door,  $s$  means OPEN or CLOSED
  - **CONNECTS**( $d, rx, ry$ )
- The world model in this figure can be represented with the above predicates, with the initial state and goal state



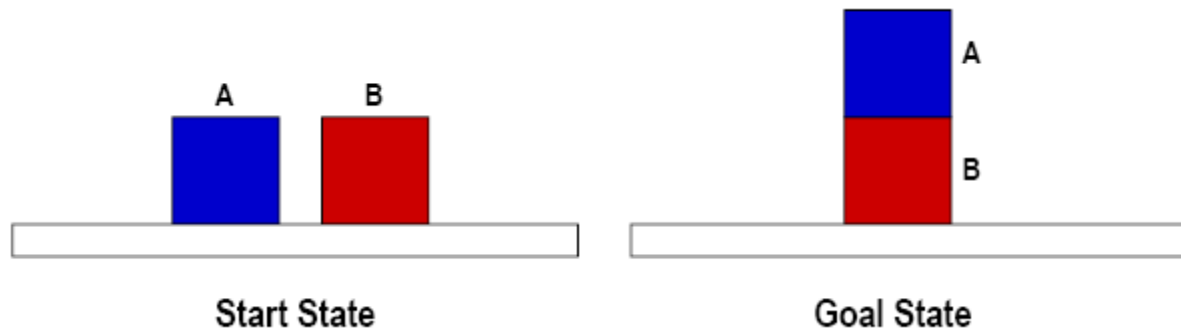
# Shakey's STRIPS World

---

- Types of actions Shakey can make
  - Move from place to place:
    - GOTODOOR(S, dx):
      - PRECOND:  $\text{INROOM}(S, rk) \wedge \text{CONNECT}(dx, rk, rm)$
      - Effect:
        - add-list:  $\text{NEXTTO}(S, dx)$
        - delete-list: null
    - GOTHRU DOOR(S, dx):
      - PRECOND:  $\text{CONNECT}(dx, rk, rm) \wedge \text{NEXTTO}(S, dx) \wedge \text{STATUS}(dx, \text{OPEN}) \wedge \text{INROOM}(S, rk)$
      - Effect:
        - add-list:  $\text{INROOM}(S, rm)$
        - delete-list:  $\text{INROOM}(S, rk)$

# Simple Example of STRIPS-Style Planning

- Goal state: ON(A, B)
- Start state: ON(A, Table); ON(B, Table); EMPTYTOP(A); EMPTYTOP(B)
- Operators:
  - Move(x, y)
    - Preconditions: ?
    - Add-list: ?
    - Delete-list: ?



- Preconditions: ON(x, Table); EMPTYTOP(y)
- Add-list: ON(x, y)
- Delete-list: EMPTYTOP(y); ON(x, Table)



# Challenges of AI and Planning

---

- **Closed world assumption**: assumes that world model contains everything the robot needs to know: there can be no surprise
- **Frame problem**: how to represent real-world situations in a manner that is computationally tractable

# Planning with State-Space Search

---

- Planning algorithms:
  - The most straightforward approach is to use state-space search
  - Forward state-space search (**Progression**)
  - Backward state-space search (**Regression**)

# Problem Formulation for Progression

---

- Initial state:
  - Initial state of the planning problem
- Actions:
  - Applicable to the current state (actions' preconditions are satisfied)
- Goal test:
  - Whether the state satisfies the goal of the planning
- Step cost:
  - Each action is 1

# Progression

---

- A plan is a sequence of STRIPS operators
- 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 state-space search using STRIPS operators

# Regression

---

- A plan is a sequence of STRIPS operators
- 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

# STRIPS Program To Control Shakey

---

- Work on board

# 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

# Relaxation Problem

---

- **Idea:** removing all preconditions from the actions
  - **Almost** implies that the number of steps required to solve a conjunction of goals is the number of unsatisfied goals
    - There may be two actions, each of which deletes the goal literal achieved by the other
    - Or, some action may achieve multiple goals
  - Combining relaxation with subgoal independence assumption → exact # of unsatisfied goals



# Removing Negative Effects

---

- Generate a relaxed problem by removing negative effects:
  - Empty-delete-list heuristic
  - Quite accurate, but computing it involves running a simple planning algorithm

# Extensions – ADL

---

- **State**: positive and negative literals
- **Open world assumptions**: unmentioned literals are unknown
- **Effect**:  $P \wedge \neg Q$  means add  $P$  and  $\neg Q$  and delete  $\neg P$  and  $Q$
- **Goal**: quantified variables; conjunction and disjunction
- **Effects**: conditional effects
  - when  $P$ :  $E$  means  $E$  is an effect only if  $P$  is satisfied
- **Equality predicate** ( $x = y$ )
- Variables can have **types** ( $p$ : Plane)

# Example: Air Cargo Transport

- Loading/unloading cargo onto/off planes and flying it from place to place.
- Actions: Load, Unload, and Fly
- States:
  - $In(c, p)$ : cargo  $c$  is inside plane  $p$
  - $At(x, a)$ : object  $x$  (plane/cargo) at airport  $a$

```
Init( $At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$   
     $\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$   
     $\wedge Airport(JFK) \wedge Airport(SFO)$ )  
Goal( $At(C_1, JFK) \wedge At(C_2, SFO)$ )  
Action( $Load(c, p, a)$ ,  
    PRECOND:  $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$   
    EFFECT:  $\neg At(c, a) \wedge In(c, p)$ )  
Action( $Unload(c, p, a)$ ,  
    PRECOND:  $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$   
    EFFECT:  $At(c, a) \wedge \neg In(c, p)$ )  
Action( $Fly(p, from, to)$ ,  
    PRECOND:  $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$   
    EFFECT:  $\neg At(p, from) \wedge At(p, to)$ )
```

# Example: Spare Tire Problem

- **Initial state:** a flat tire on the axle and a good spare tire in the trunk
- **Goal state:** have a good spare tire properly mounted onto the car's axle

```
Init(At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk),
  PRECOND: At(Spare, Trunk)
  EFFECT:  $\neg$  At(Spare, Trunk) ∧ At(Spare, Ground))
Action(Remove(Flat, Axle),
  PRECOND: At(Flat, Axle)
  EFFECT:  $\neg$  At(Flat, Axle) ∧ At(Flat, Ground))
Action(PutOn(Spare, Axle),
  PRECOND: At(Spare, Ground) ∧ ¬ At(Flat, Axle)
  EFFECT:  $\neg$  At(Spare, Ground) ∧ At(Spare, Axle))
Action(LeaveOvernight,
  PRECOND:
  EFFECT:  $\neg$  At(Spare, Ground) ∧ ¬ At(Spare, Axle) ∧ ¬ At(Spare, Trunk)
          $\wedge$   $\neg$  At(Flat, Ground) ∧ ¬ At(Flat, Axle))
```

# Total-Order Planning

---

- Forward/backward state-space searches are forms of totally ordered plan search
  - explore only strictly **linear sequences** of actions directly connected to the start or goal
  - **cannot** take advantages of **problem decomposition**

# Partial-Order Planning

---

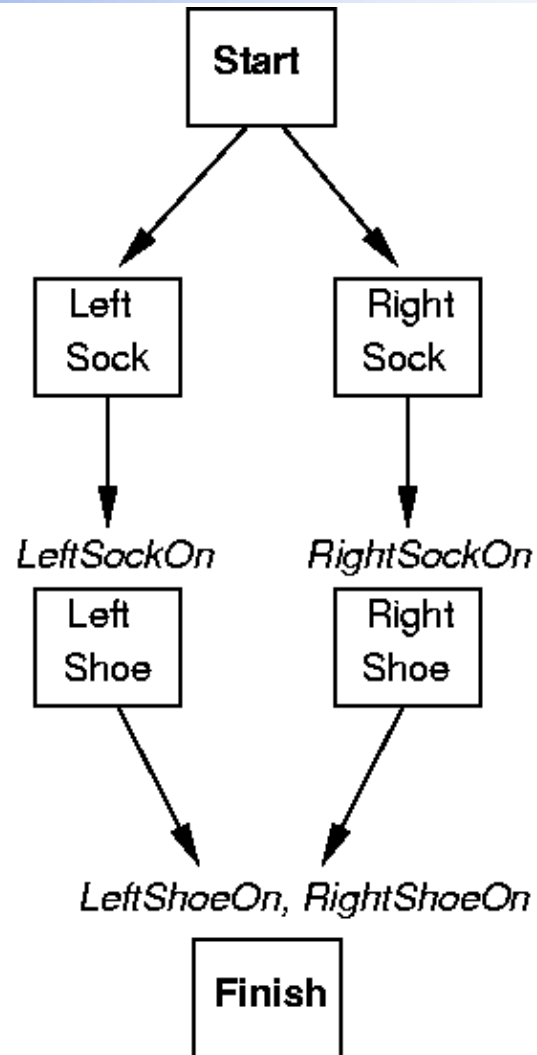
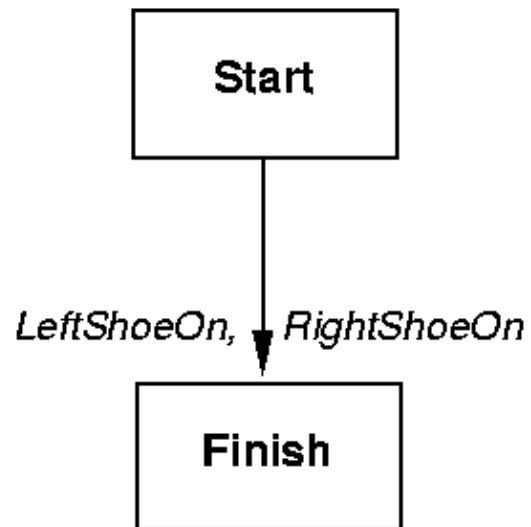
- Idea:
  - works on several subgoals independently
  - solves them with subplans
  - combines the subplans
  - flexibility in ordering the subplans
  - **least commitment strategy**:
    - delaying a choice during search
    - Example, leave actions unordered, unless they must be sequential

# POP Example

---

- Putting on a pair of shoes:
  - Goal(RightShoeOn  $\wedge$  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

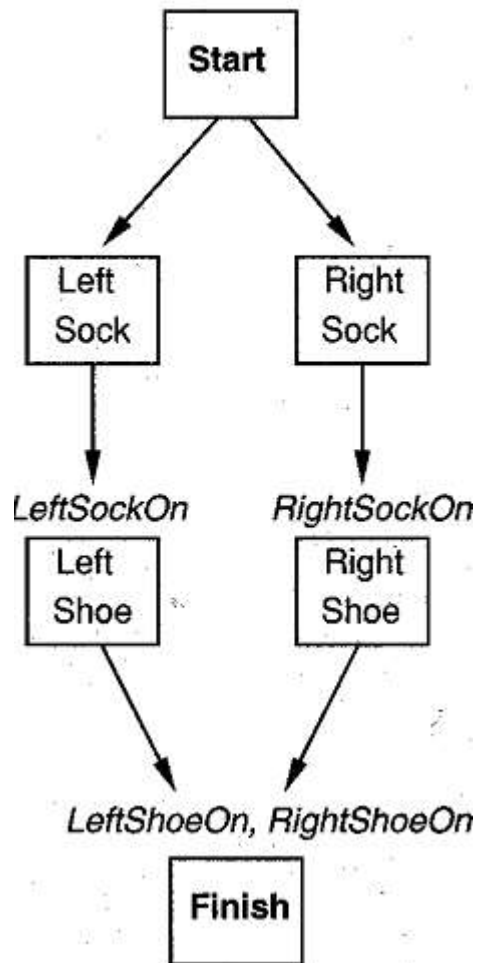
# POP Example



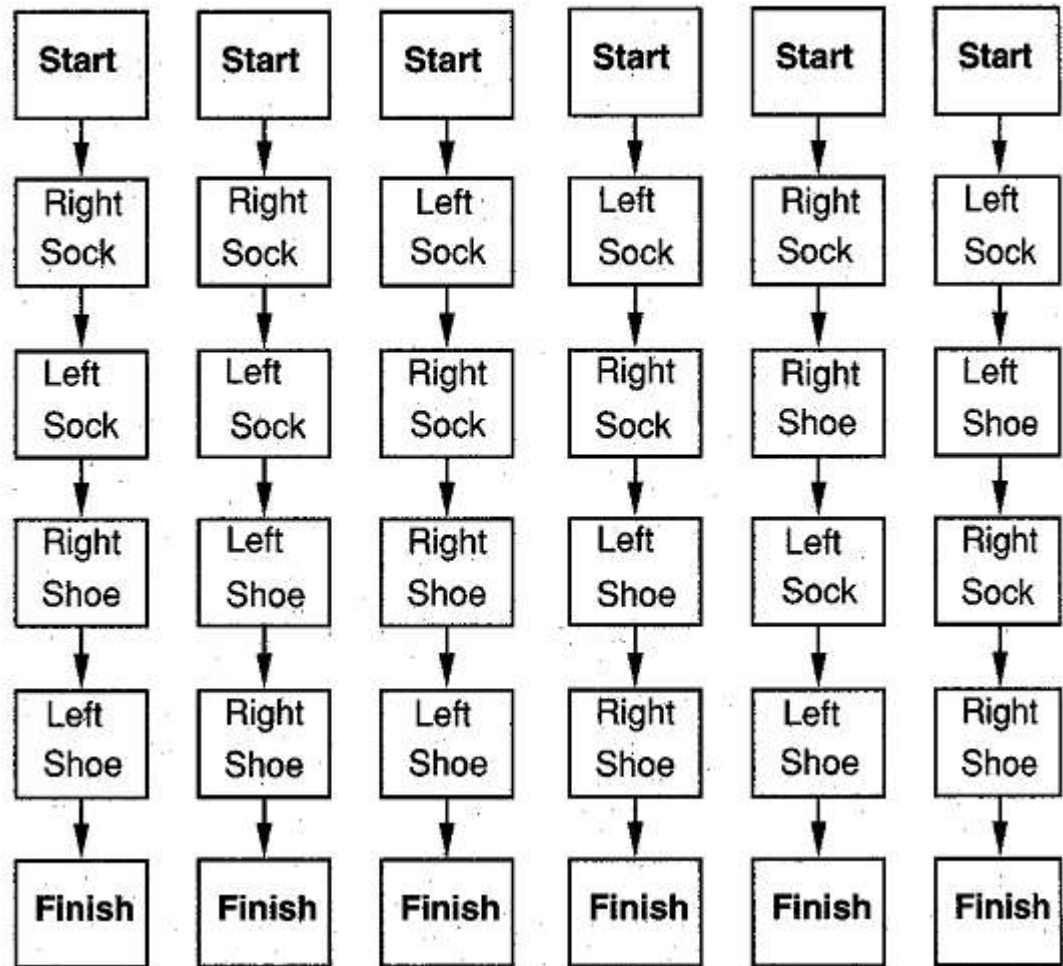


# Partial Order Plan to Total Order Plan

**Partial Order Plan:**



**Total Order Plans:**

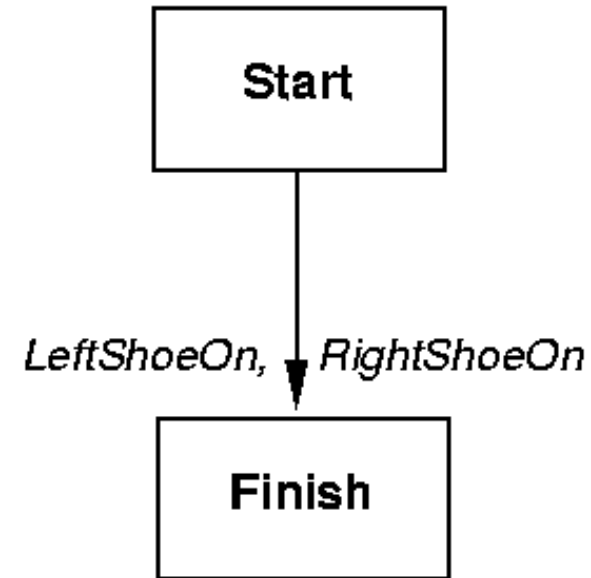


# How to Define Partial Order Plan?

- A set of **actions** that make up the steps of the plan
- A set of **ordering constraints**:  $A \prec B$ 
  - A before B
- A set of **causal links**:  $A \xrightarrow{p} B$ 
  - A achieves p for B       $RightSock \xrightarrow{RightSockOn} RightShoe$
  - May be conflicts if C has the effect of  $\neg p$  and if C comes after A and before B
- A set of **open preconditions**:
  - a precondition is open if it is not achieved by some action in the plan

# The Initial Plan

- Initial plan contains:
  - **Start:**
    - PRECOND: none
    - EFFECT: Add all propositions that are initially true
  - **Finish:**
    - PRECOND: Goal state
    - EFFECT: none
- Ordering constraints:  $Start \prec Finish$
- Causal links:  $\{\}$
- Open preconditions:  $\{\text{preconditions of Finish}\}$



# Next...

---

- Successor function
  - arbitrarily picks one open precondition  $p$  on an action  $B$  and generates a successor plan for every possible consistent way of choosing an action  $A$  that achieves  $p$
  - **Consistency:**
    - Causal link  $A \xrightarrow{p} B$  and the ordering constraint are added ( $A \prec B$   $Start \prec A$   $A \prec Finish$ )
    - Resolve conflict: add  $B \prec C$  or  $C \prec A$
- Goal test:
  - There are no open preconditions

# Example: Final Plan

---

- The final plan has the following components:
  - **Actions:** {RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish}
  - **Orderings:** {RightSock < RightShoe, LeftSock < LeftShoe}
  - **Open preconditions:** {}
  - **Links:**

$$RightSock \xrightarrow{RightSockOn} RightShoe$$
$$LeftSock \xrightarrow{LeftSockOn} LeftShoe$$
$$RightShoe \xrightarrow{RightShoeOn} Finish$$
$$LeftShoe \xrightarrow{LeftShoeOn} Finish$$

# Example Algorithm for POP

- POP: A sound, complete partial order planner using STRIPS representation

```
function POP(initial, goal, operators) returns plan
  plan ← MAKE-MINIMAL-PLAN(initial, goal)
  loop do
    if SOLUTION?(plan) then return plan
     $S_{need}, c$  ← SELECT-SUBGOAL(plan)
    CHOOSE-OPERATOR(plan, operators,  $S_{need}, c$ )
    RESOLVE-THREATS(plan)
  end
```

where: \*  $c$  is a precondition of a step  $S_{need}$   
\* RESOLVE-THREATS: orders steps as needed to ensure  
intermediate steps don't undo preconditions needed by other steps

# POP Example: Flat Tire

```
Init(At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk),
  PRECOND: At(Spare, Trunk)
  EFFECT:  $\neg \text{At}(\text{Spare}, \text{Trunk}) \wedge \text{At}(\text{Spare}, \text{Ground})$ )
Action(Remove(Flat, Axle),
  PRECOND: At(Flat, Axle)
  EFFECT:  $\neg \text{At}(\text{Flat}, \text{Axle}) \wedge \text{At}(\text{Flat}, \text{Ground})$ )
Action(PutOn(Spare, Axle),
  PRECOND: At(Spare, Ground) ∧ ¬ At(Flat, Axle)
  EFFECT:  $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \text{At}(\text{Spare}, \text{Axle})$ )
Action(LeaveOvernight,
  PRECOND:
  EFFECT:  $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \neg \text{At}(\text{Spare}, \text{Axle}) \wedge \neg \text{At}(\text{Spare}, \text{Trunk})$ 
          $\wedge \neg \text{At}(\text{Flat}, \text{Ground}) \wedge \neg \text{At}(\text{Flat}, \text{Axle})$ )
```

# Strengths of POP

---

- The causal links lead to **early pruning** of portions of the search space because of irresolvable conflicts
- The solution is a partial-order plan
  - linearizations produce **flexible** plans



# From Problem Solvers to Planners

---

- Key ideas:
  - “Open up” representation of states, goals, and actions
    - Use descriptions in a formal language – FOL
    - States/goals represented by sets of sentences
    - Actions represented by logical description of preconditions and effects
    - Enables planner to make direct connections between states and actions
  - Planner can add actions to plan whenever needed, rather than in strictly incremental fashion
    - No necessary connection between order of planning and order of execution
  - Most parts of the world are independent of most other parts of the world
    - Conjunctions can be separated and handled independently
    - Divide-and-conquer algorithms

# Summary of Planning Problem

---

- Planning agents use look-ahead to find actions to contribute to goal achievement
- Planning agents differ from problem solvers in their use of more flexible representation of states, actions, goals, and plans
- The STRIPS language describes actions in terms of preconditions and effects
- Principle of least commitment is preferred
- POP is a sound and complete algorithm for planning using STRIPS representation

# In-Class Exercise 11.1

---

- The monkey-and-bananas problem is faced by a monkey in a lab with some bananas hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the bananas if he climbs on it.
- Initially the monkey is at A, bananas at B, the box at C. The monkey and box have height Low, but if the monkey climbs onto the box he will have height High, the same as the bananas.

# Exercise 11.1 Cont'd

---

## ■ Actions:

- Go from one place to another
- Push an object from one place to another
- ClimbUp onto or ClimbDown from an object
- Grasp or Ungrasp an object
- The results of a Grasp is that the monkey holds the object if the monkey and object are in the same place at the same height.

## ■ Questions

- Write down the initial state description
- Write down the goal state (monkey has banana)
- Write the six action schemas
- Generate the plan

# In-Class Exercises 11.2

---

- Describe the differences and similarities between problem solving and planning.

# In-Class Exercises 11.3

---

- Explain why dropping negative effects from every action schema in a planning problem results in a relaxed problem.