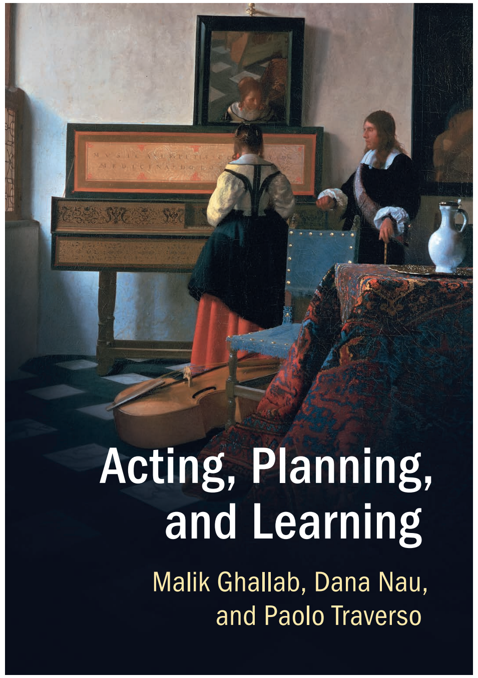
Last update: 11:09 PM, March 7, 2025



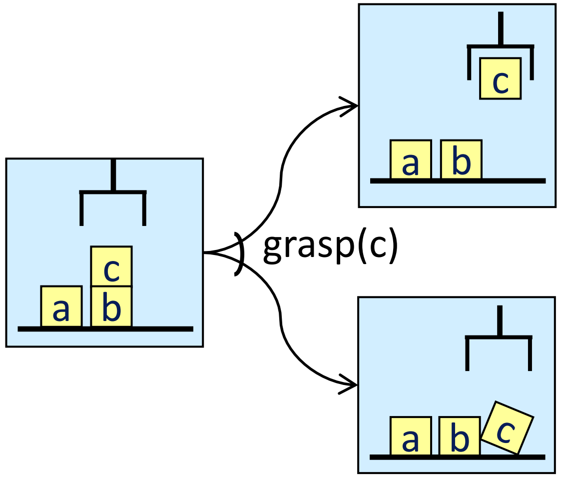
Chapter 11
Acting with Nondeterministic Models

Dana S. Nau
University of Maryland

Motivation

- In Chapters 8–10, we had probability distributions over the possible outcomes of actions

- Sometimes we want to reason about nondeterminism *without* the probability distributions
  - Probabilities might not be available
  - We might want policies that satisfy safety conditions:
    - Guaranteed to work for all possible action outcomes



Credit: Dennis Hill, CC BY 2.0

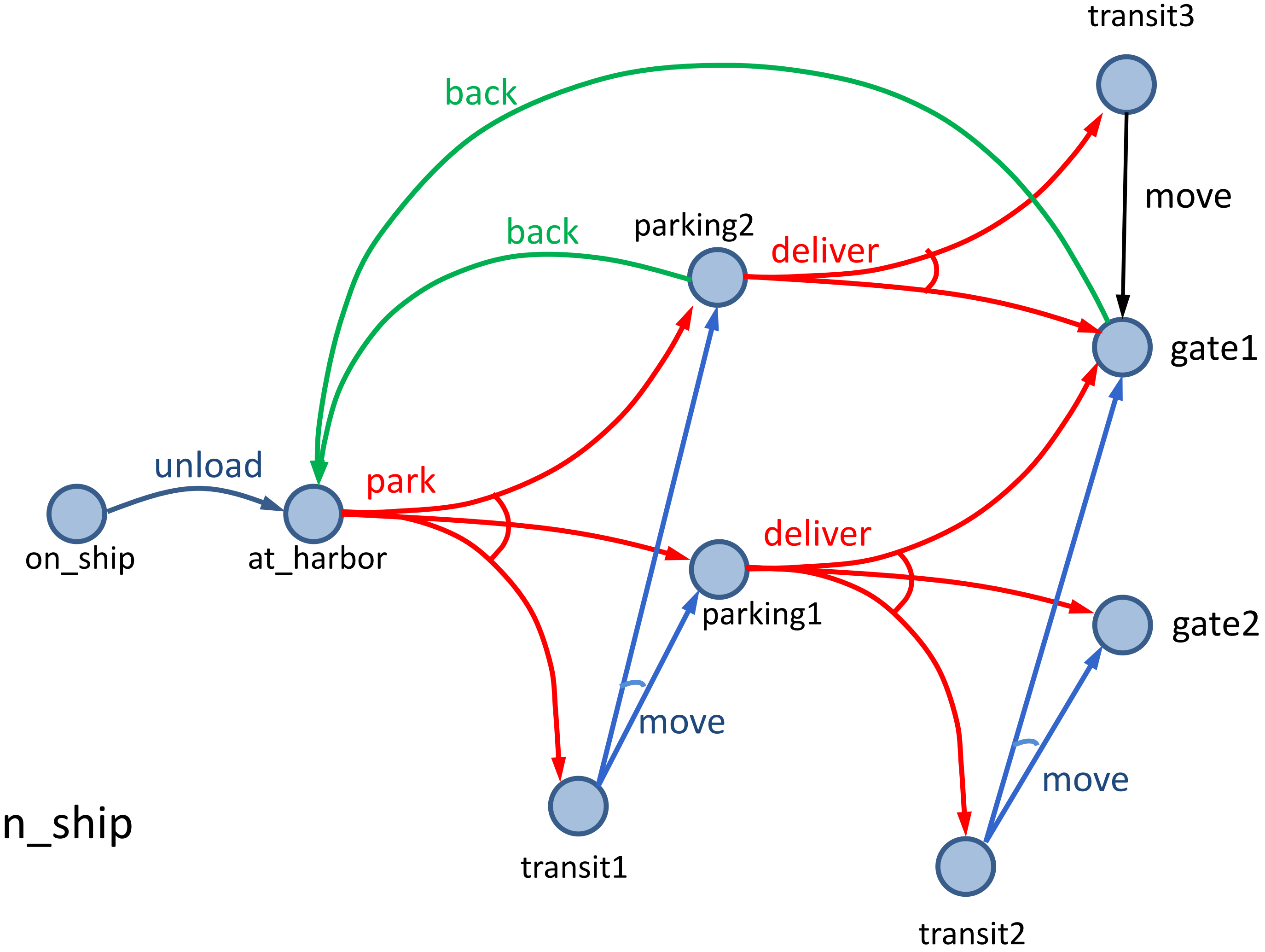
Credit: Airtuna08, CC BY-SA 3.0

Credit: David Wilson, CC BY 2.0

Credit: Slaunger, CC BY-SA 3.0

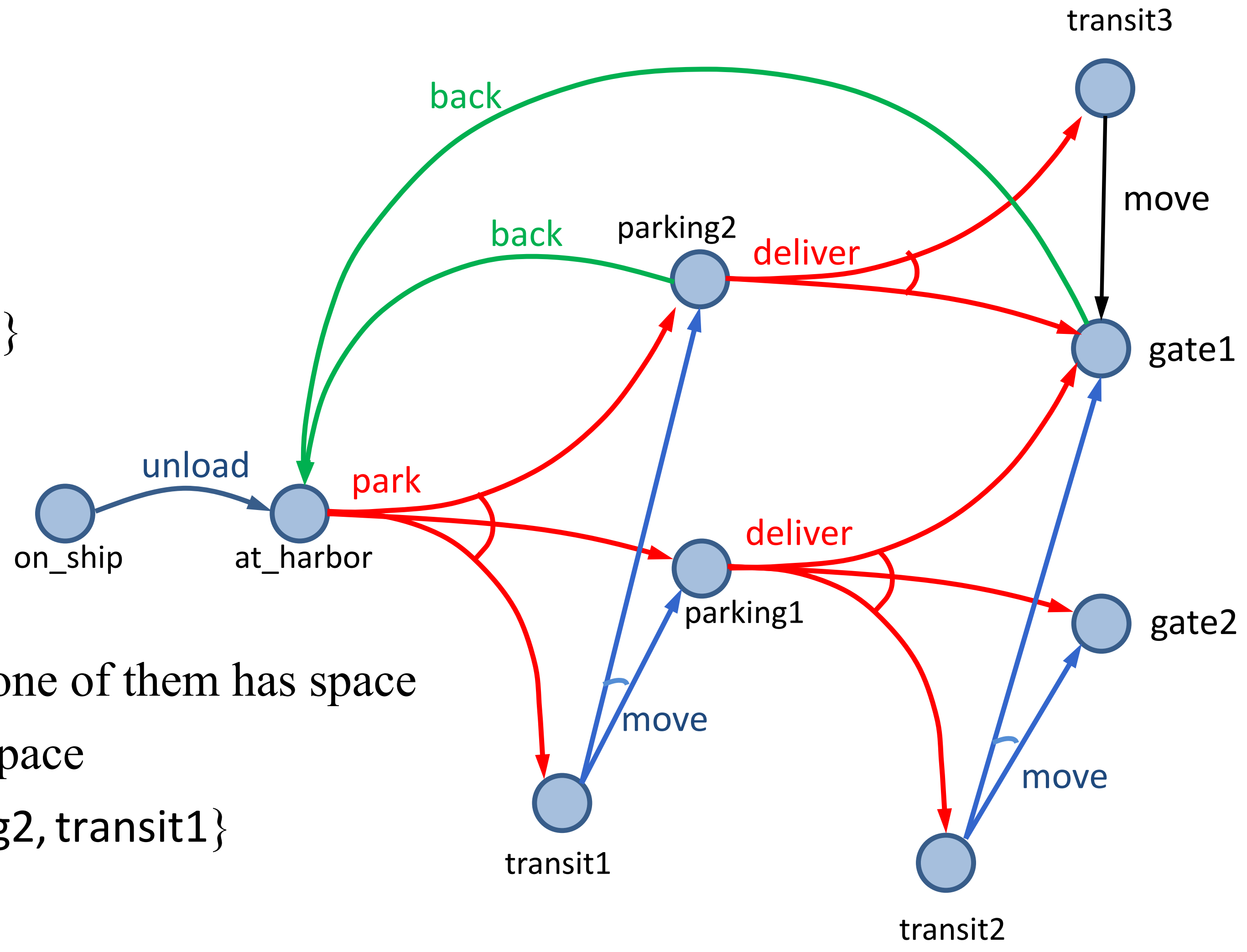
Example

- Very simple harbor management domain
  - Unload a single item from a ship
  - Move it around a harbor
- One state variable: pos(item)
  - Simplified names for states
  - For {pos(item)=on_ship}, just write on_ship



Nondeterministic Planning Domains

- 3-tuple (S, A, γ)
  - S and A – finite sets of states and actions
  - $\gamma: S \times A \to 2^S$
- $\gamma(s,a)$ = {all possible "next states" after applying action a in state s}
  - a is applicable in state s iff $\gamma(s,a) \neq \emptyset$
- Applicable(s) = {all actions applicable in s} = $\{a \in A \mid \gamma(s,a) \neq \emptyset\}$
- Example:
  - Applicable(at_harbor) = {park}
  - park has three possible outcomes
    - put item in parking1 or parking2 if one of them has space
    - or in transit1 if there's no parking space
  - γ(at_harbor, park) = {parking1, parking2, transit1}

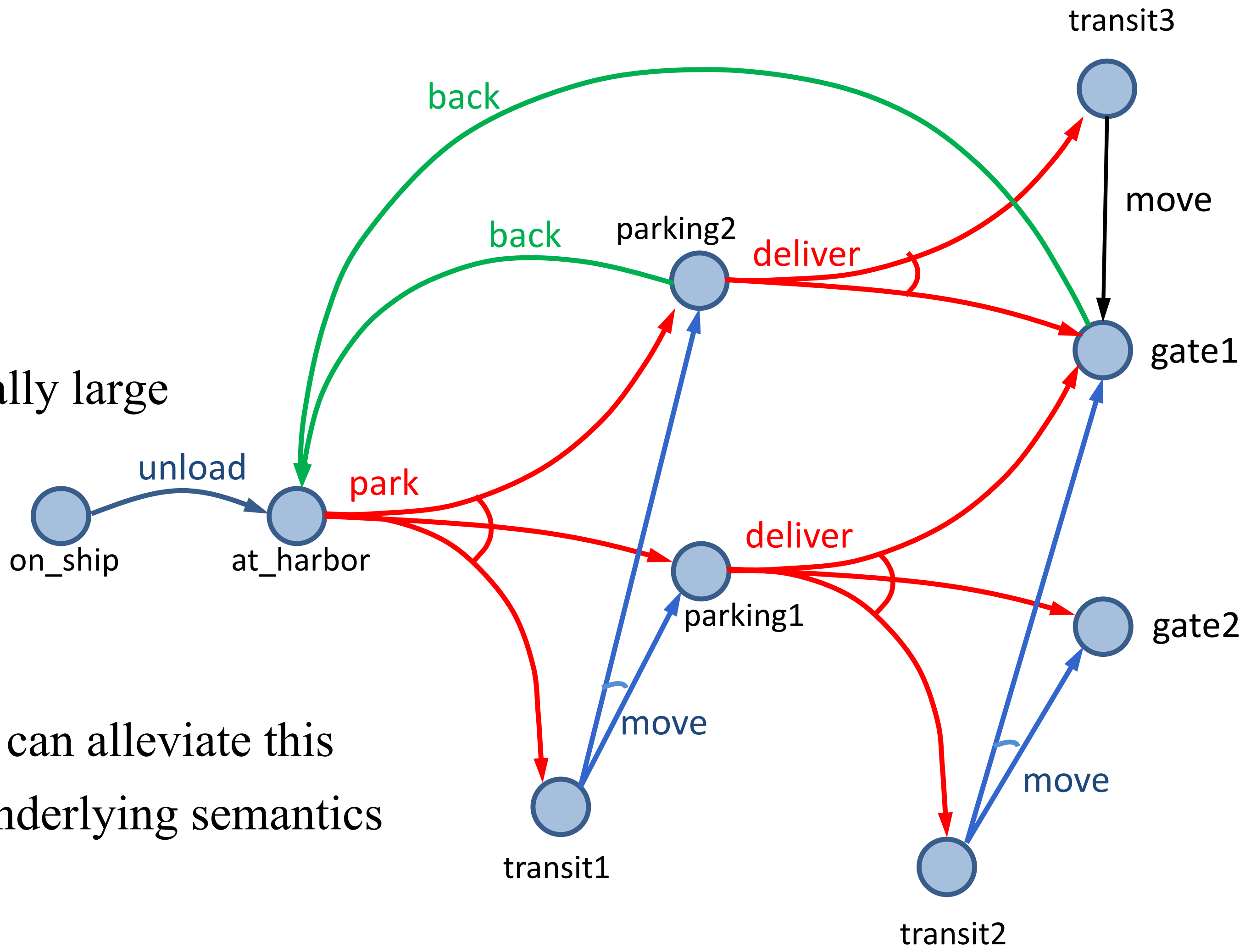


Nondeterministic Planning Domains

- One possible action representation:
  - like classical, but with n mutually exclusive "effects" lists
- e.g., park:

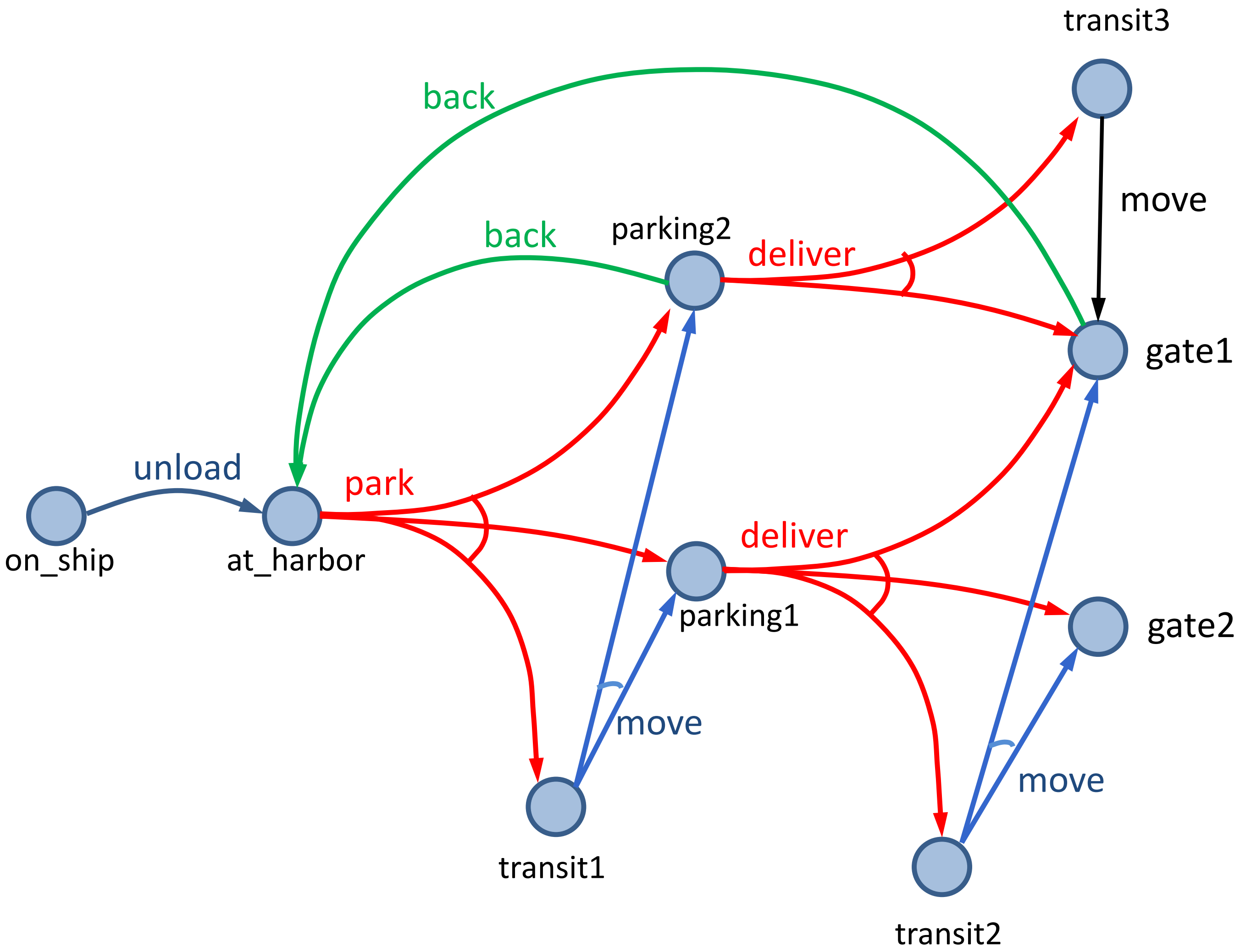
  pre: pos(item) = at_harbor
  eff_1: pos(item) ← parking1
  eff_2: pos(item) ← parking2
  eff_3: pos(item) ← transit1

- Problem:
  - number of effects lists may be combinatorially large
  - Suppose a can cause any possible combination of effects $e_1, e_2, \ldots, e_k$
  - Need $\text{eff}_1, \text{eff}_2, \ldots, \text{eff}_{2^k}$
    - One for for each combination
  - Section 12.3: a different representation that can alleviate this
- For now, ignore most of that, just look at the underlying semantics
  - states, actions ⇔ nodes, edges in a graph



Nondeterministic Planning Domains

- For deterministic planning problems, search space was a graph
- Now it's an AND/OR graph
  - *OR branch*:
    - several applicable actions, which one to choose?
  - *AND branch*:
    - multiple possible outcomes, must handle all of them
- Analogy to PSP in Chapter 2
  - *OR* branch ⇔ action selection
  - *AND* branch ⇔ flaw selection

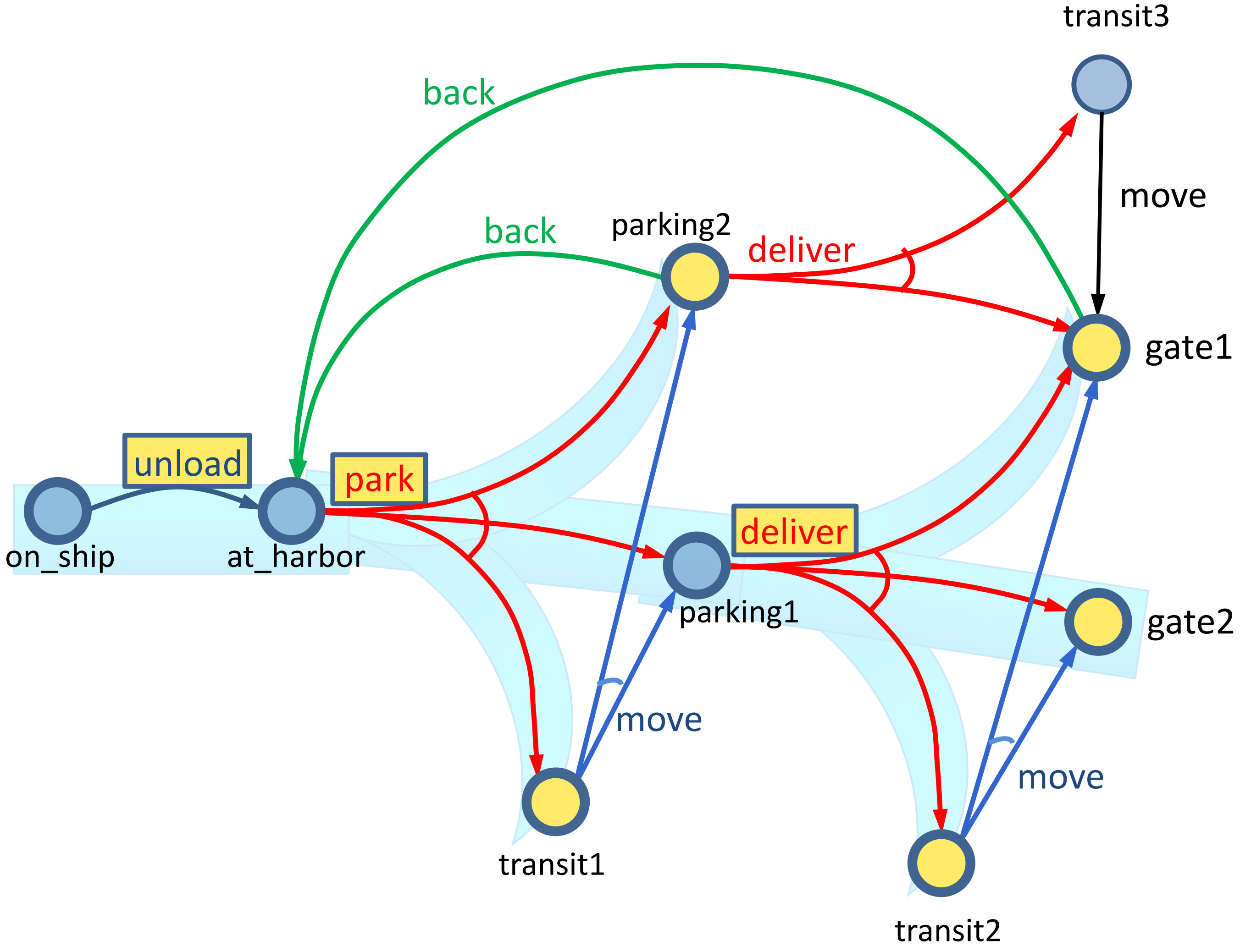


Policies

- *Policy*: a function $\pi : S' \to A$
  - $S' \subseteq S$
  - For every $s \in \text{Domain}(\pi)$, require $\pi(s) \in \text{Applicable}(s)$
- Two equivalent notations:

  π_1(on_ship) = unload,
  π_1 (at_harbor) = park,
  π_1 (parking1) = deliver

  $\pi_1 = \{$(on_ship, unload),
  (at_harbor, park),
  (parking1, deliver)$\}$
- That's just the notation
  - implementation could be quite different



Definitions Over Policies

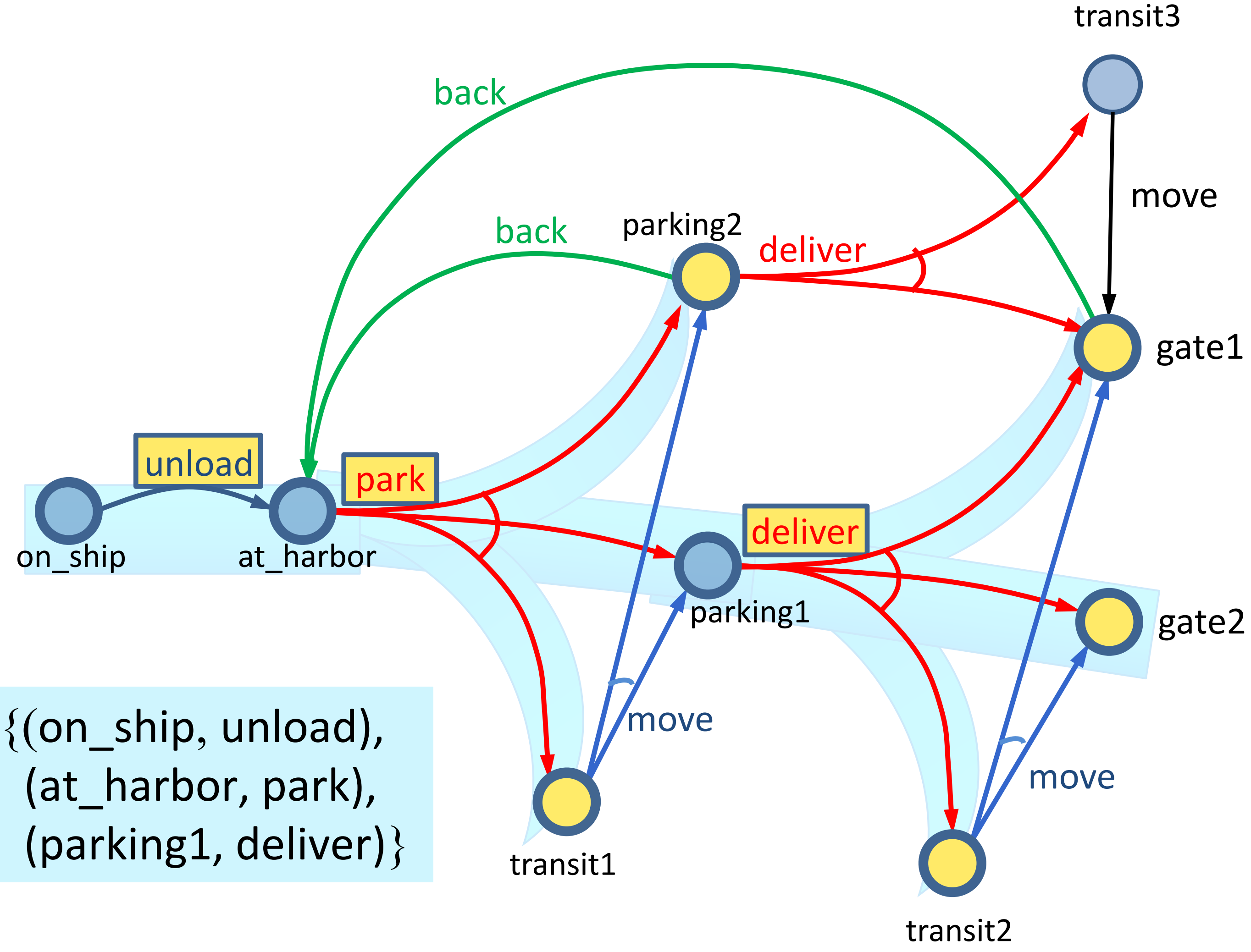
- *Transitive closure*:
  - $\hat{\gamma}(s,\pi)$ = {all states reachable from s using π}
  - $\hat{\gamma}(s,\pi) = S_0 \cup S_1 \cup S_2 \cup \ldots$

$$S_0 = \{s\}$$
$$S_1 = S_0 \cup \{\gamma(s_0,\pi(s_0)) \mid s_0 \in S_0\}$$
$$S_2 = S_1 \cup \{\gamma(s_1,\pi(s_1)) \mid s_1 \in S_1\}$$
$$\ldots$$

- *Reachability graph*: $\text{Graph}(s,\pi) = (V,E)$
  - $V = \hat{\gamma}(s,\pi)$
  - $E = \{(s_1,s_2) \mid s_1 \in V,\ s_2 \in \gamma(s_1,\pi(s_1))\}$

- *leaves*$(s,\pi) = \hat{\gamma}(s,\pi) \setminus \text{Dom}(\pi)$
  - may be empty

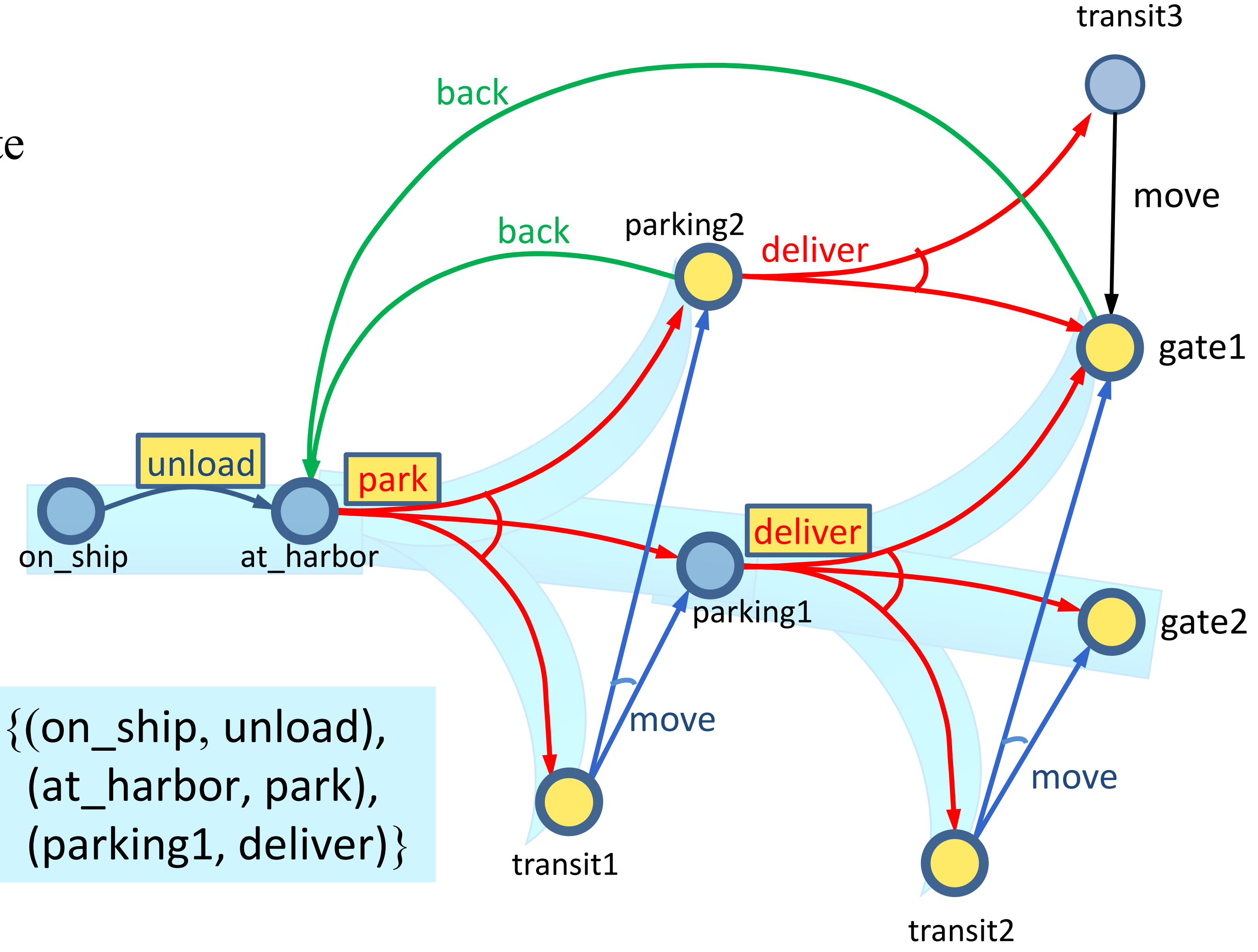
π_1 = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver)}



Acting with a Policy

- ActPolicy(π)

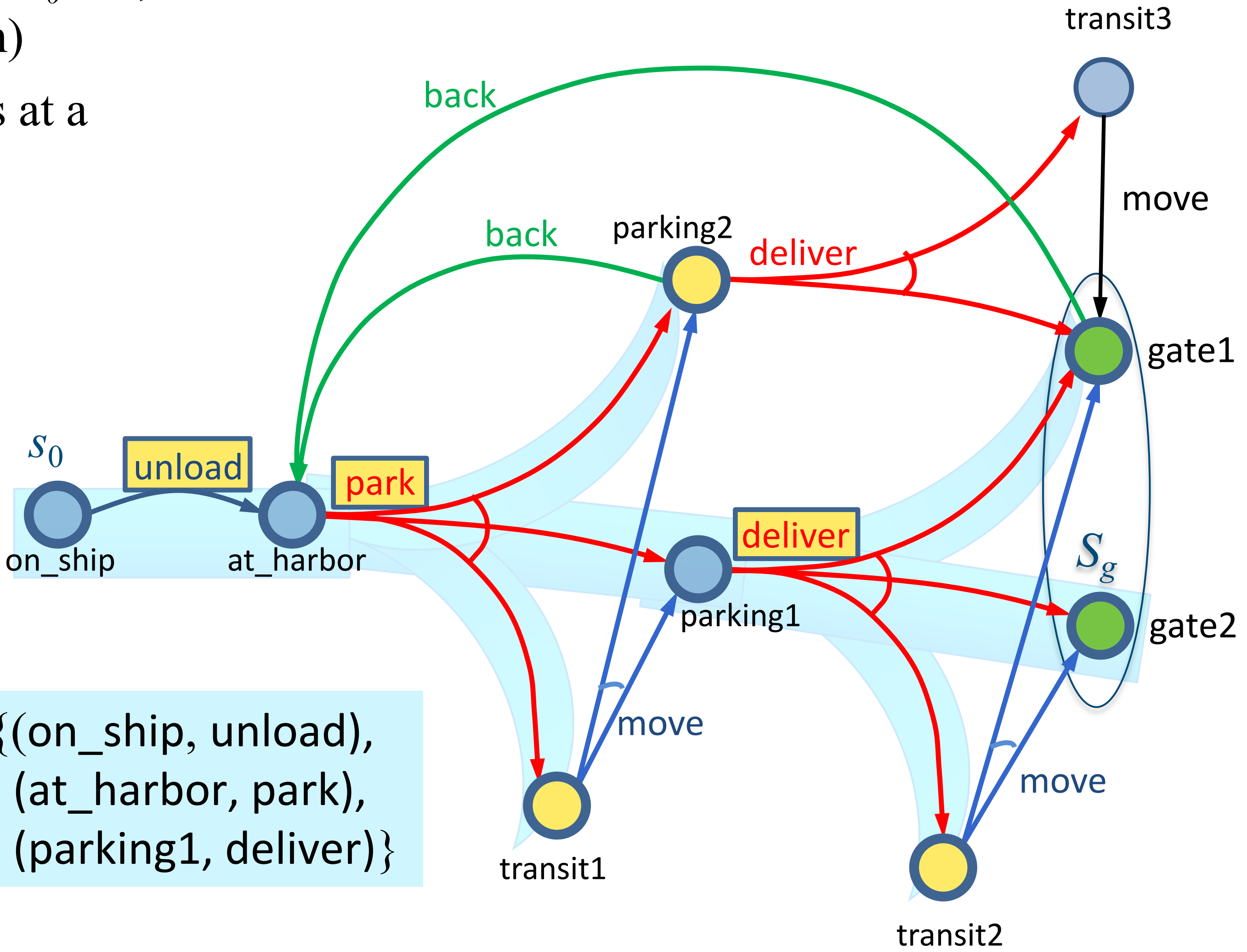
  $s \leftarrow$ observe current state
  while $s \in$ Domain(π) do
      perform action $\pi(s)$
      $s \leftarrow$ observe current state



$$\pi_1 = \{(\text{on\_ship, unload}), (\text{at\_harbor, park}), (\text{parking1, deliver})\}$$

Types of Policies

- Acting (or planning) problem $P = (\Sigma, s_0, S_g)$
  - planning domain $\Sigma = (S, A, \gamma)$, initial state $s_0 \in S$, set of goal states $S_g \subseteq S$ (shown in green)
- π is a *solution* if at least one execution ends at a goal
  - $leaves(s,\pi) \cap S_g \neq \emptyset$
- A policy π is *safe* if $\forall s \in \hat{\gamma}(s_0,\pi),\ leaves(s,\pi) \cap S_g \neq \emptyset$
  - at every state in $\hat{\gamma}(s_0,\pi)$, at least one of the execution paths from s using π stops at a goal state.
- Otherwise, *unsafe* policy



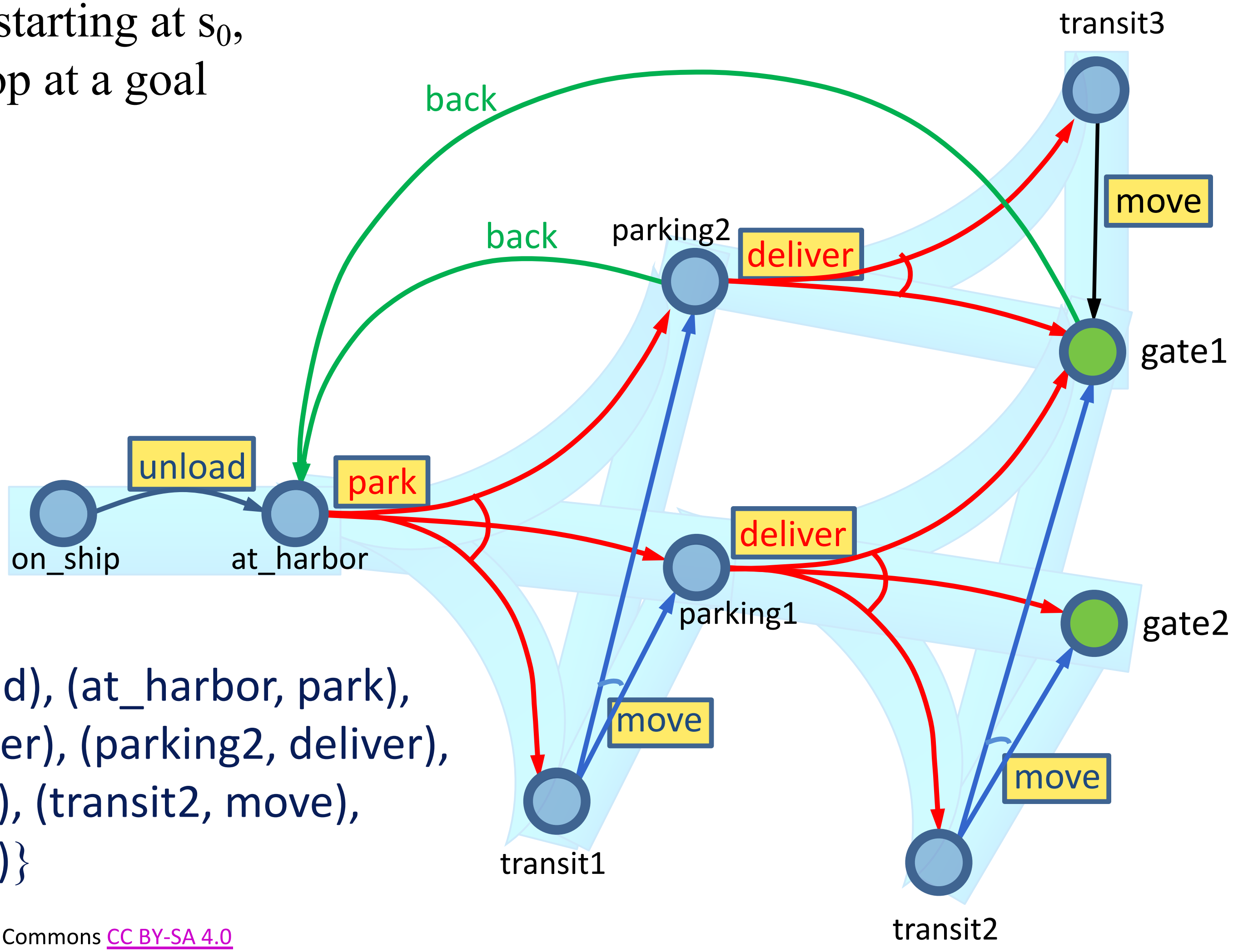
$\pi_1 = \{$(on_ship, unload), (at_harbor, park), (parking1, deliver)$\}$

Poll: Is π_1 safe or unsafe?

Safe Policies

- ActPolicy(π)
  - $s \leftarrow$ observe current state
  - while $s \in$ Domain(π) do
    - perform action $\pi(s)$
    - $s \leftarrow$ observe current state

- *Acyclic* safe policy
  - Graph(s_0,π) is acyclic, and *leaves*(s,π) $\subseteq S_g$
- If we run ActPolicy(π) starting at s_0, we're guaranteed to stop at a goal

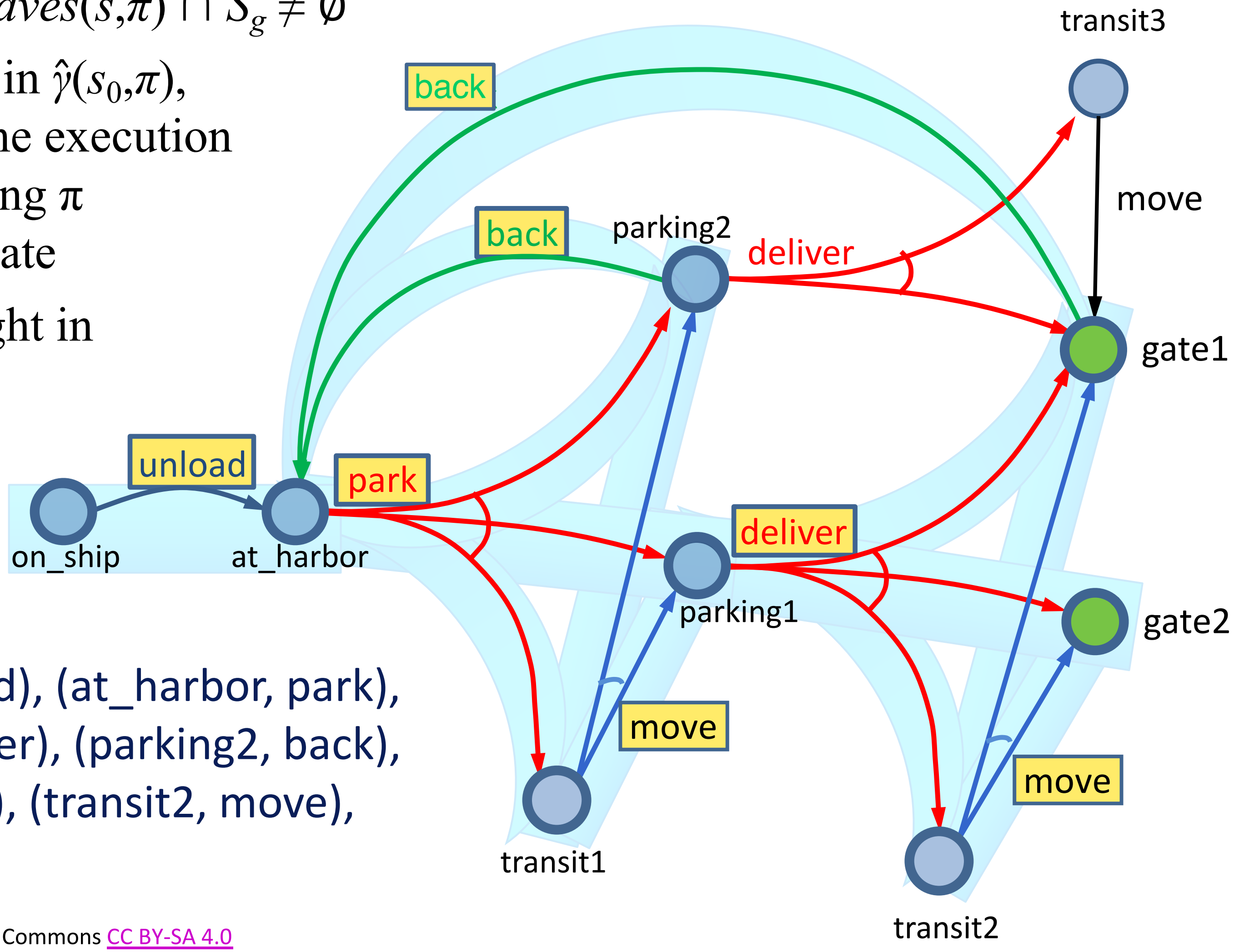


π_2 = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, deliver), (transit1, move), (transit2, move), (transit3, move)}

Safe Policies

- ActPolicy(π)
  - $s \leftarrow$ observe current state
  - while $s \in$ Domain(π) do
    - perform action $\pi(s)$
    - $s \leftarrow$ observe current state

- *Cyclic* safe policy
  - Graph(s_0, π) is cyclic, and $leaves(s,\pi) \subseteq S_g$, and $\forall s \in \hat{\gamma}(s_0,\pi)$, $leaves(s,\pi) \cap S_g \neq \emptyset$
    - At every state s in $\hat{\gamma}(s_0,\pi)$, at least one of the execution paths from s using π ends at a goal state
  - Will never get caught in a dead end



π_3 = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, back), (transit1, move), (transit2, move), (gate1, back)}

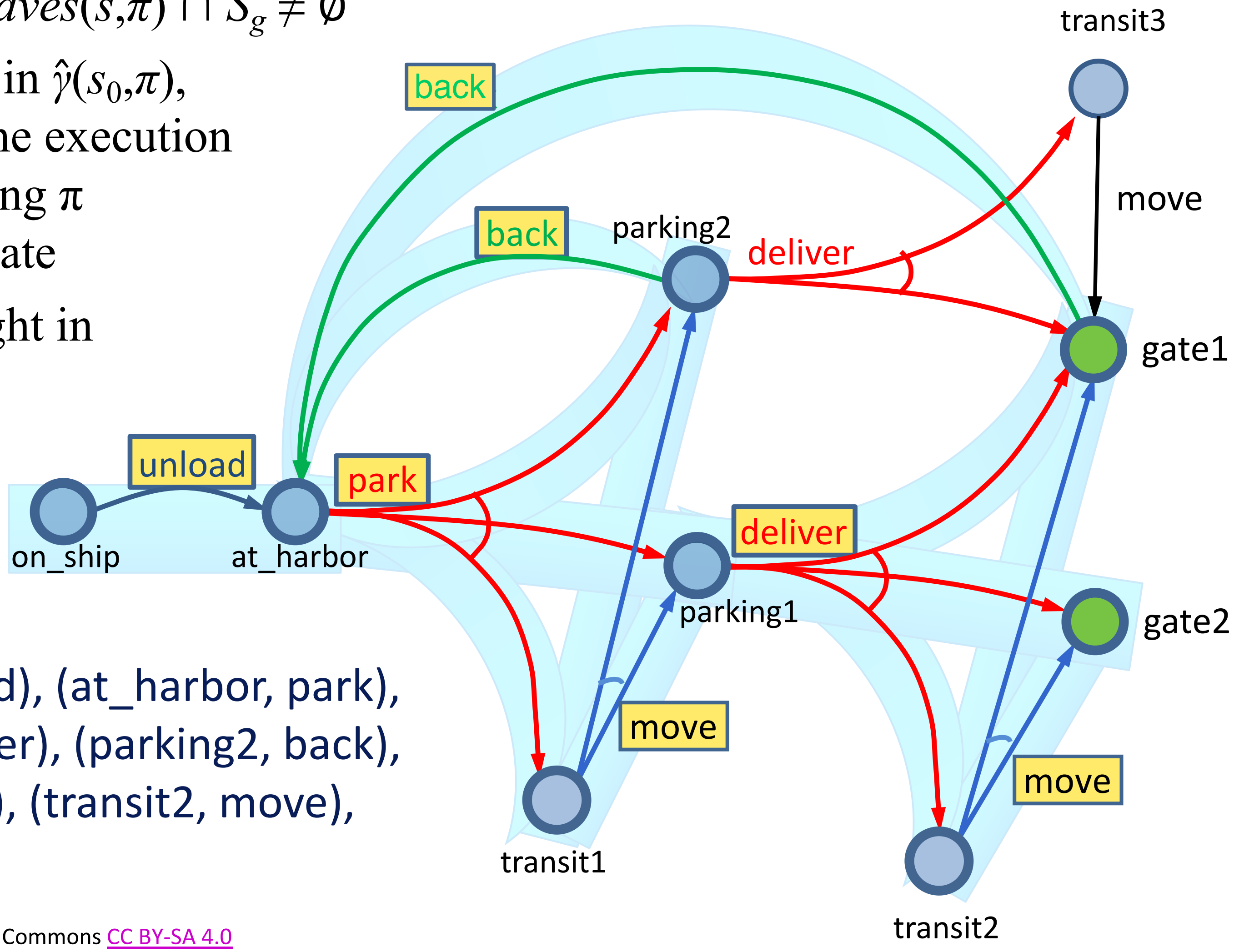
Poll: Let π be a cyclic safe solution. Suppose we run ActPolicy(π) starting at s_0.

1. Are there situations where we can be sure π will reach a goal?
2. Are there situations where we can't be sure π will reach a goal?

Safe Policies

- ActPolicy(π)
  $s \leftarrow$ observe current state
  while $s \in$ Domain(π) do
  perform action $\pi(s)$
  $s \leftarrow$ observe current state

- *Cyclic* safe policy
  - Graph(s_0, π) is cyclic, and $leaves(s,\pi) \subseteq S_g$, and $\forall s \in \hat{\gamma}(s_0,\pi)$, $leaves(s,\pi) \cap S_g \neq \emptyset$
    - At every state s in $\hat{\gamma}(s_0,\pi)$, at least one of the execution paths from s using π ends at a goal state
  - Will never get caught in a dead end
  - Every "fair" execution will reach a goal

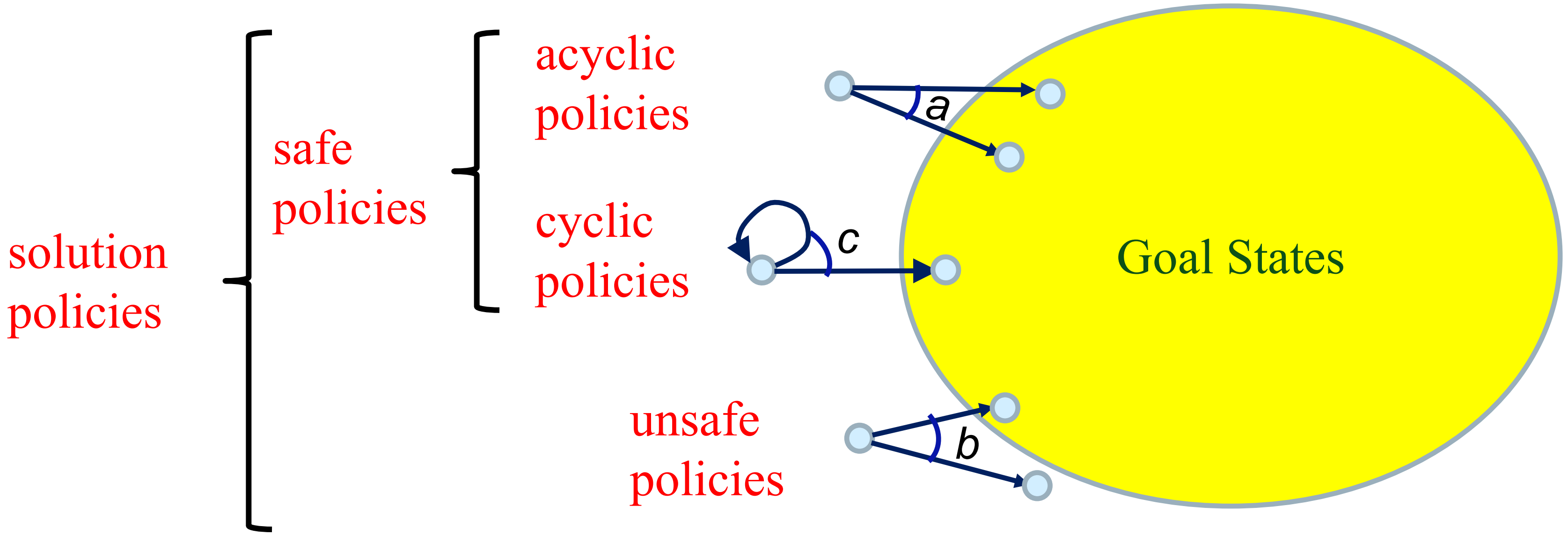


Poll:

1. Can you think of a real-world situation in which all executions are "fair"?
2. Can you think of a real-world situation in which there are "unfair" executions?

π_3 = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, back), (transit1, move), (transit2, move), (gate1, back)}

Kinds of Solution Policies



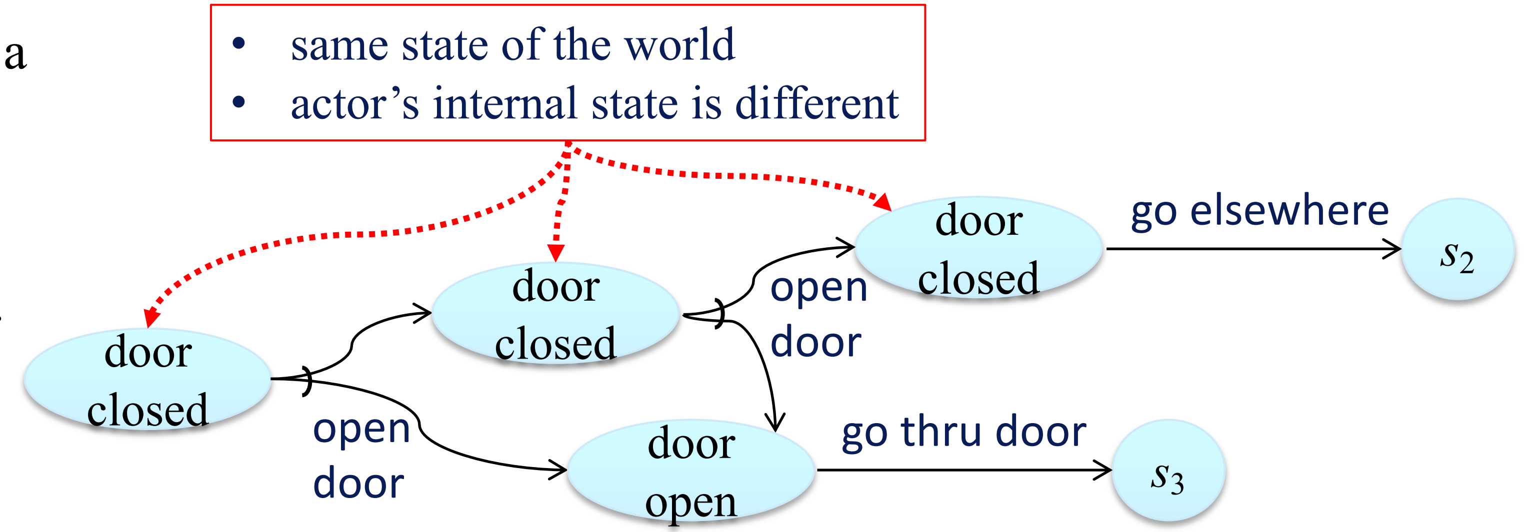
14

Beyond Policies

- Sometimes we want to give the actor instructions that can't be described as a policy, e.g.,
  - Try to open the door twice.
    If it opens, go through it.
    If it doesn't, go to another door

- The book describes two other ways to represent instructions to the actor
  - Input/Output Automata
  - Behavior Trees

- I'll discuss behavior trees in a separate set of slides



Summary

- Actions, plans, policies, planning problems
- Types of solution policies:
  - unsafe, safe (acyclic, cyclic)
- Motivation for instructions other than policies