

# Advanced Topics Data Science and AI

# Automated Planning and

# Acting

Nondeterministic Models

Tanya Braun



UNIVERSITÄT ZU LÜBECK  
INSTITUT FÜR INFORMATIONSSYSTEME

# Content

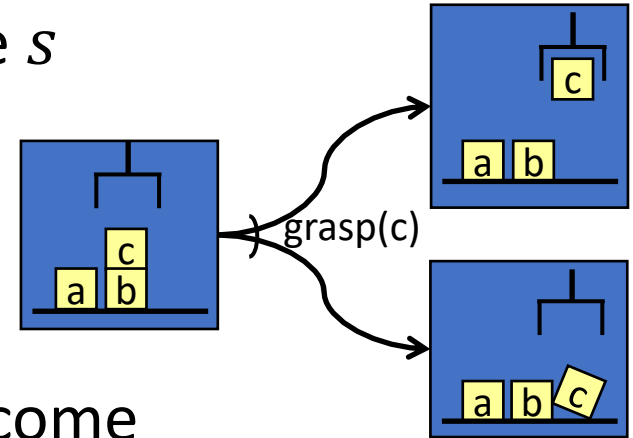
---

1. Planning and Acting with **Deterministic** Models
2. Planning and Acting with **Refinement** Methods
3. Planning and Acting with **Temporal** Models
4. Planning and Acting with **Nondeterministic** Models
- a. Planning Problem
- b. And/Or Graph Search
- c. Determinisation
- d. Online Approaches
5. **Standard** Decision Making
6. Planning and Acting with **Probabilistic** Models
7. **Advanced** Decision Making
8. **Human-aware** Planning

# Motivation

- We have assumed action  $a$  in state  $s$  has just one possible outcome

- $\gamma(s, a)$



- Often more than one possible outcome
  - Unintended outcomes
  - Exogenous events
  - Inherent uncertainty



# Outline per the Book

---

## ***5.2 Planning Problem***

- Planning domains
- Plans as policies
- Planning problems and solutions

## ***5.3 And/Or Graph Search***

- Planning by forward search

## ***5.5 Determinisation Techniques***

- Guided planning for safe solutions
- Planning for safe solutions by determinisation

## ***5.6 Online Approaches***

- Lookahead
- Lookahead by determinisation
- Lookahead with a bounded number of steps

# Nondeterministic Planning Domains

- Planning domain: 3-tuple  $(S, A, \gamma)$ 
  - $S$  and  $A$  – finite sets of states and actions
  - $\gamma : S \times A \rightarrow 2^S$
- $\gamma(s, a) = \{\text{all possible “next states” after applying action } a \text{ in state } s\}$ 
  - $a$  is **applicable** in state  $s$  iff  $\gamma(s, a) \neq \emptyset$
- $\text{Applicable}(s) = \{\text{all actions applicable in } s\} = \{a \in A \mid \gamma(s, a) \neq \emptyset\}$
- One action representation:
  - $n$  mutually exclusive “effects” lists
  - **Problem:**  $n$  may be combinatorially large
    - Suppose  $a$  can cause any possible combination of effects  $e_1, e_2, \dots, e_k$
    - Need  $\text{eff}_1, \text{eff}_2, \dots, \text{eff}_{2^k}$ 
      - One for for each combination
    - *Section 5.4: a way to alleviate this*
  - For now, ignore most of that
    - states, actions  $\Leftrightarrow$  nodes, edges in a graph

$a(z_1, \dots, z_k)$
pre: $p_1, \dots, p_m$
eff <sub>1</sub> : $e_{11}, e_{12}, \dots$
eff <sub>2</sub> : $e_{21}, e_{22}, \dots$
⋮
eff <sub><math>n</math></sub> : $e_{n1}, e_{n2}, \dots$

# Nondeterministic Planning Domains

- For deterministic planning problems, search space was a graph

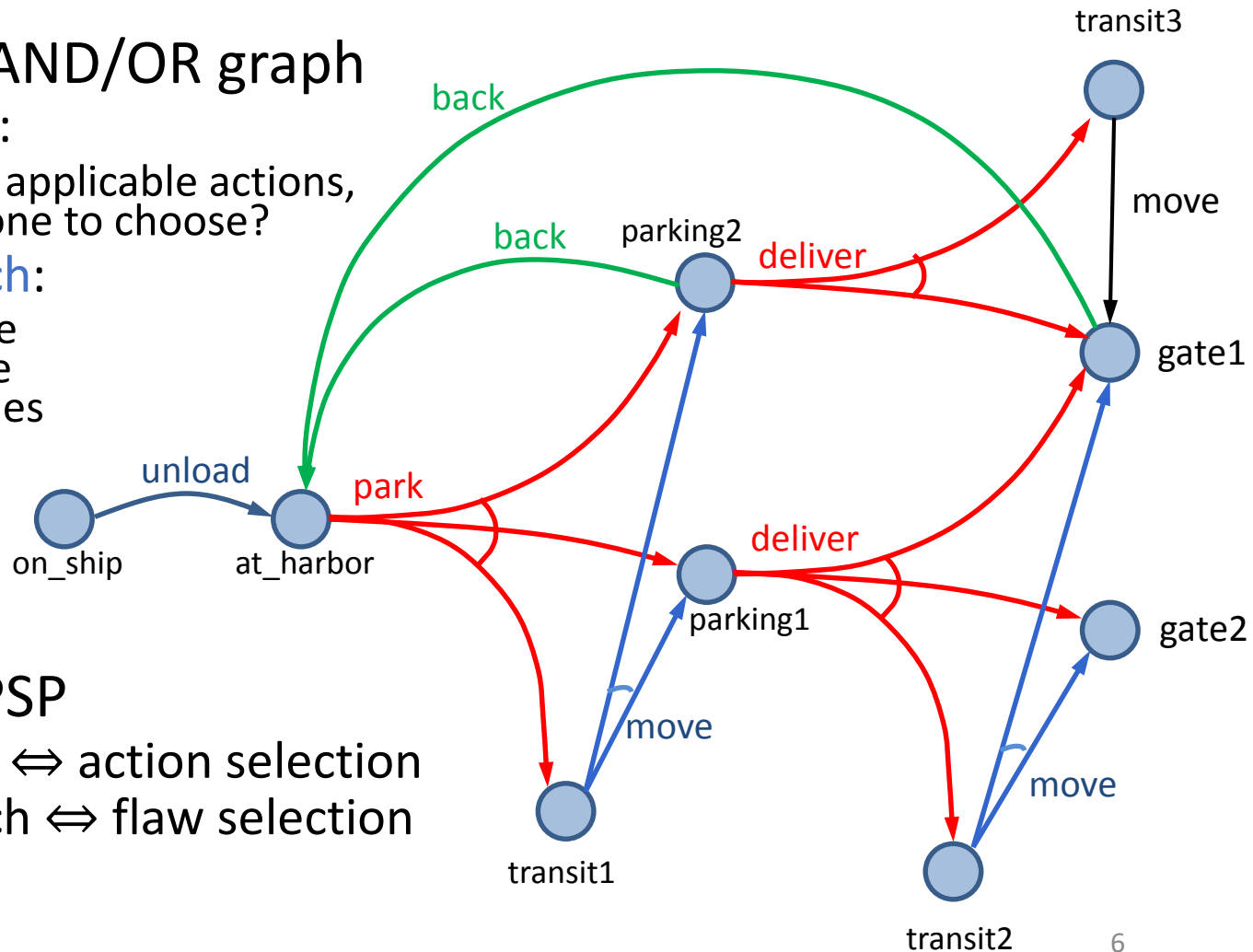
- Now it's an AND/OR graph

- **OR branch:**

- Several applicable actions, which one to choose?

- **AND branch:**

- Multiple possible outcomes
    - Must handle all of them

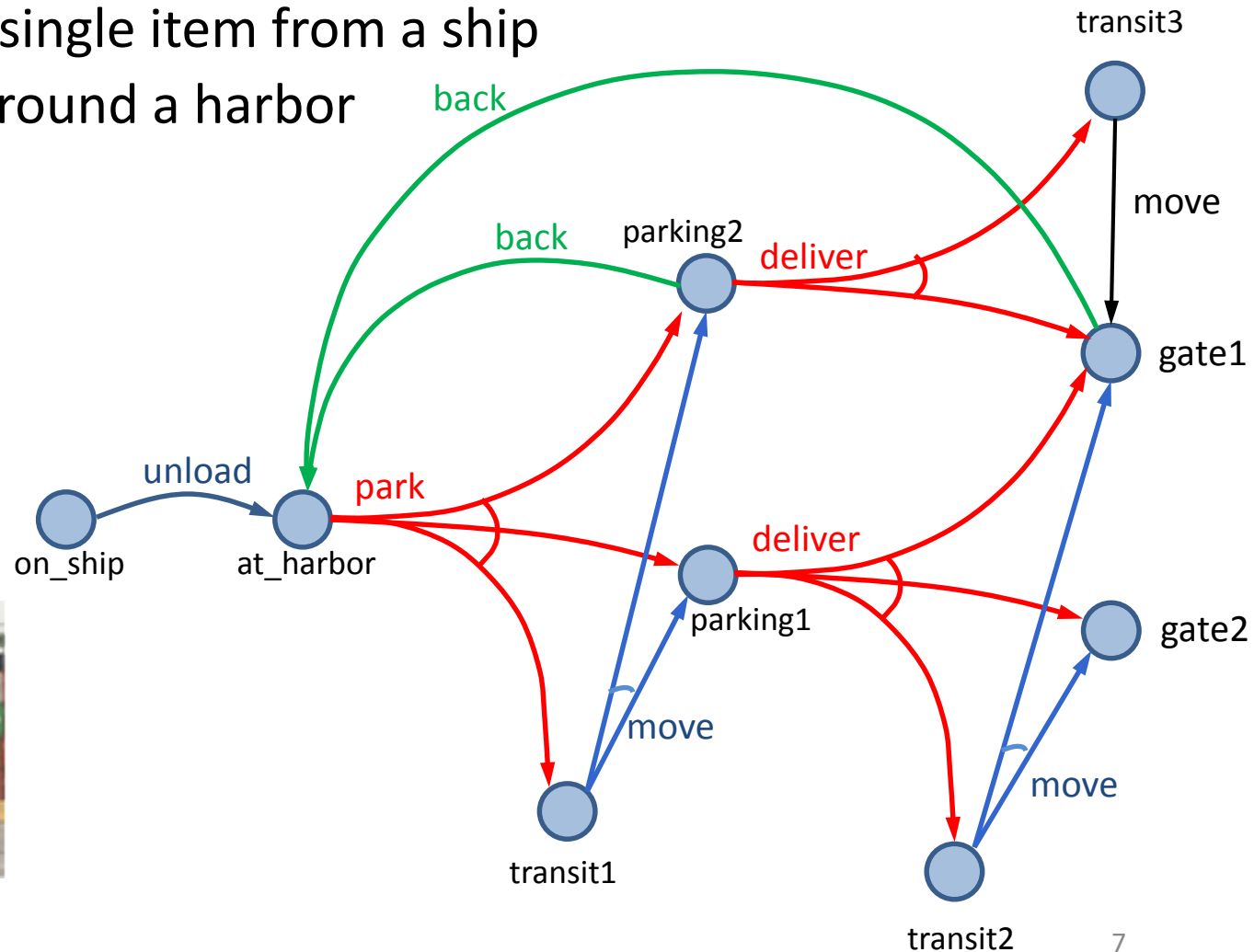


- Analogy to PSP

- *OR* branch  $\Leftrightarrow$  action selection
  - *AND* branch  $\Leftrightarrow$  flaw selection

# Example

- Very simple harbor management domain
  - Unload a single item from a ship
  - Move it around a harbor



# Example

- One state variable:  $pos(item)$

- Five actions

- Deterministic:

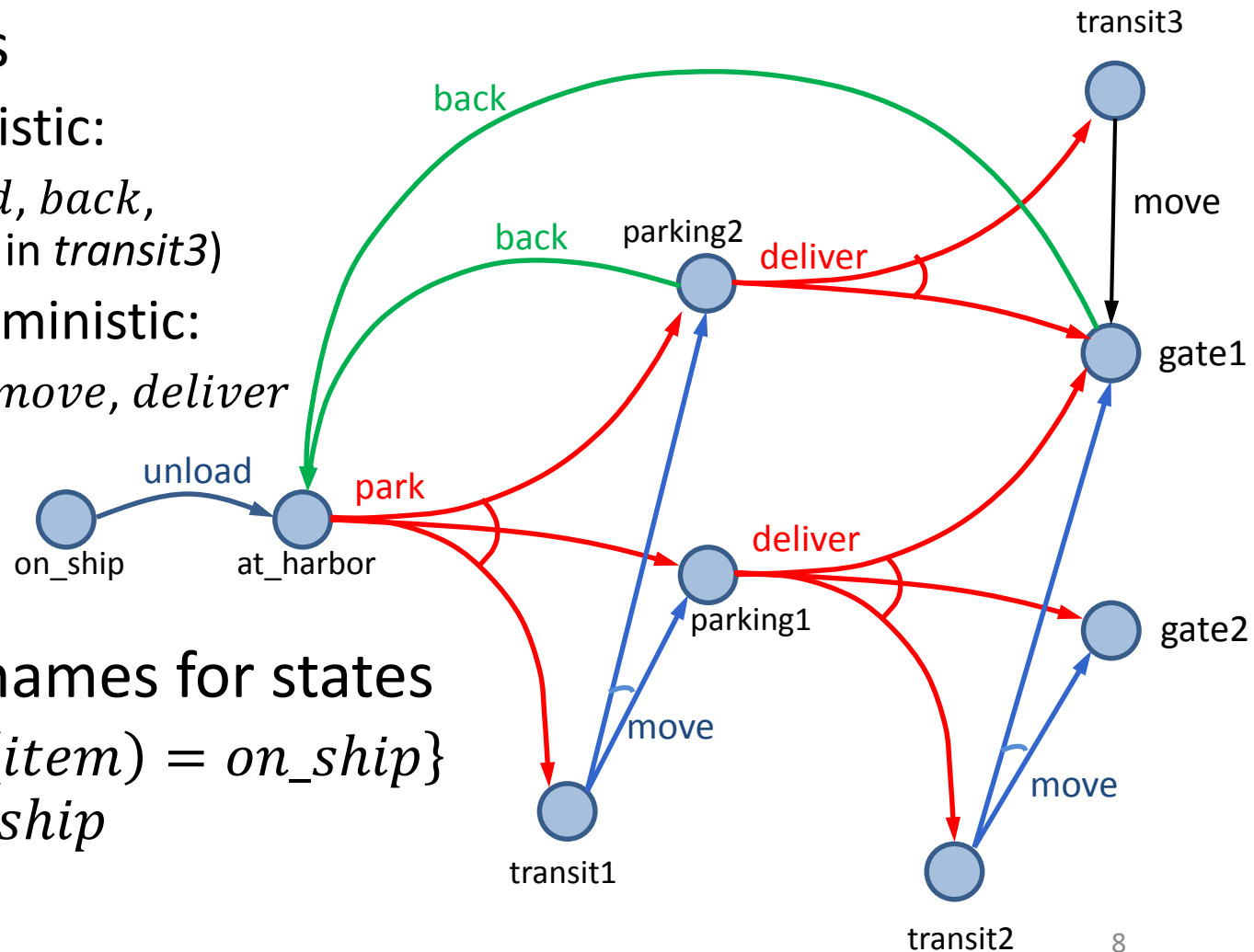
- $unload$ ,  $back$ ,  
( $move$  in  $transit3$ )

- Nondeterministic:

- $park$ ,  $move$ ,  $deliver$

- Simplified names for states

- For  $\{pos(item) = on\_ship\}$   
write  $on\_ship$





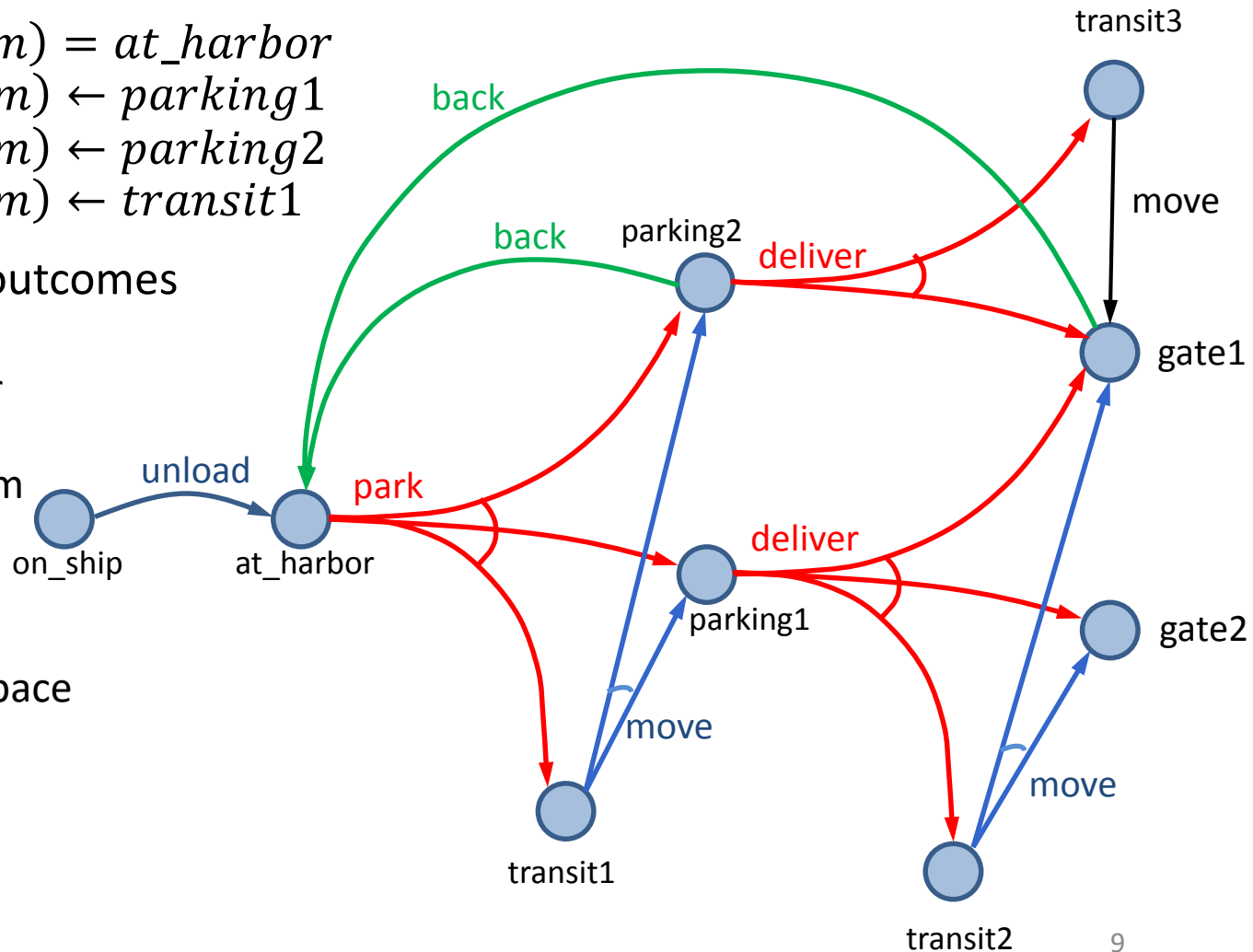
# Actions

- Action:  
*park*

pre:  $pos(item) = at\_harbor$   
 eff<sub>1</sub>:  $pos(item) \leftarrow parking1$   
 eff<sub>2</sub>:  $pos(item) \leftarrow parking2$   
 eff<sub>3</sub>:  $pos(item) \leftarrow transit1$

- Three possible outcomes

- Put item in *parking1* or *parking2* if one of them has space or
- in *transit1* if there is no parking space



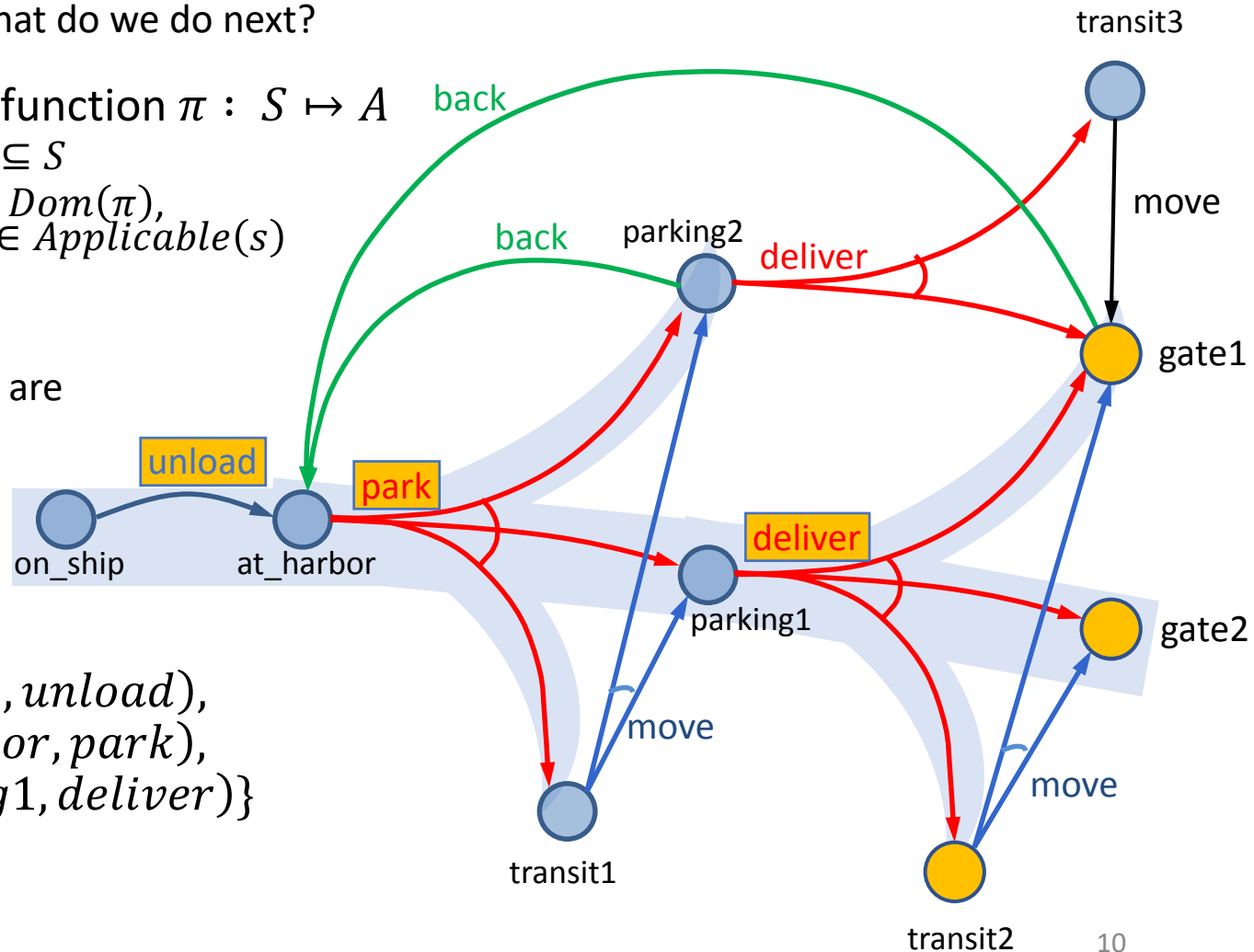
# Plans Policies

- Need something more general than a sequence of actions
  - After park, what do we do next?

- **Policy:** a *partial* function  $\pi : S \mapsto A$ 
  - i.e.,  $Dom(\pi) \subseteq S$
  - For every  $s \in Dom(\pi)$ , require  $\pi(s) \in Applicable(s)$

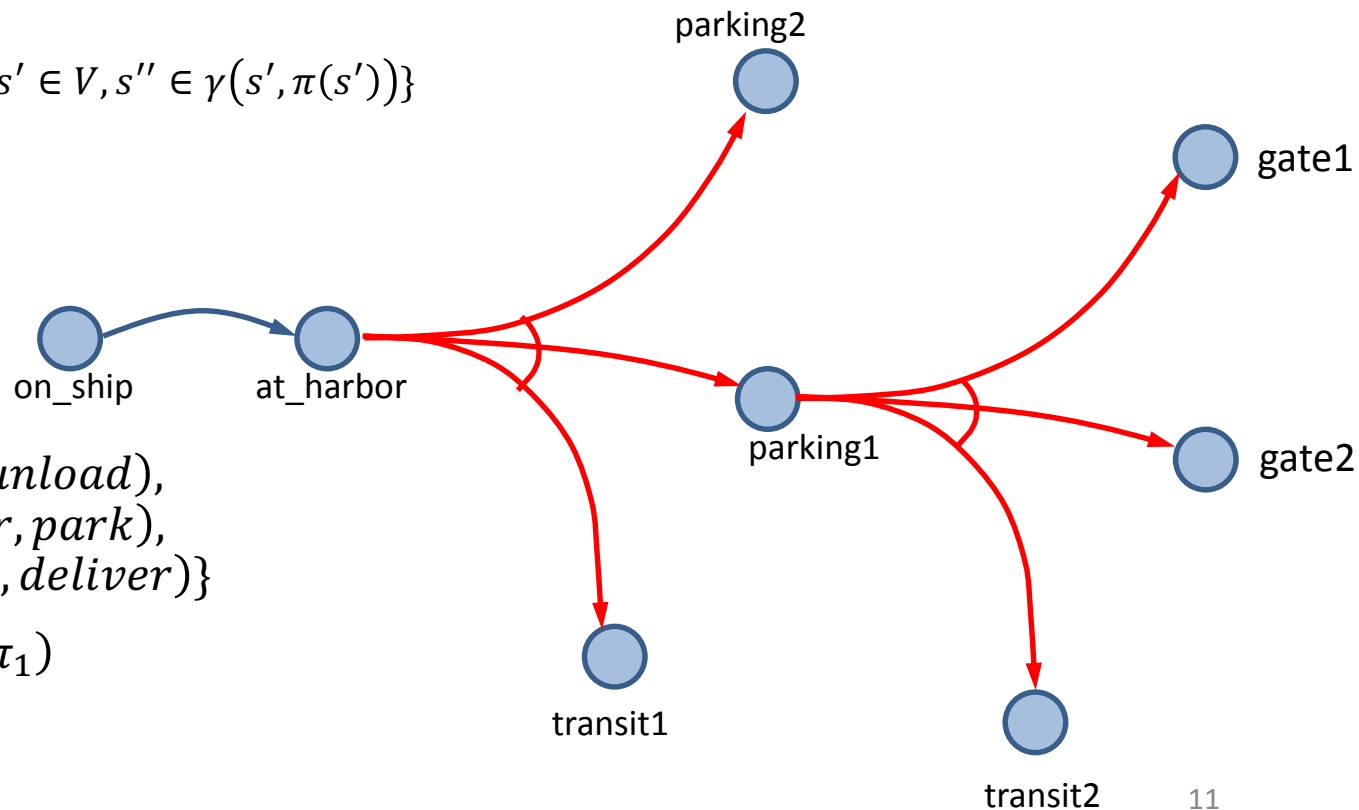
- Meaning:
  - Perform  $\pi(s)$  whenever we are in state  $s$

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



# Definitions Over Policies

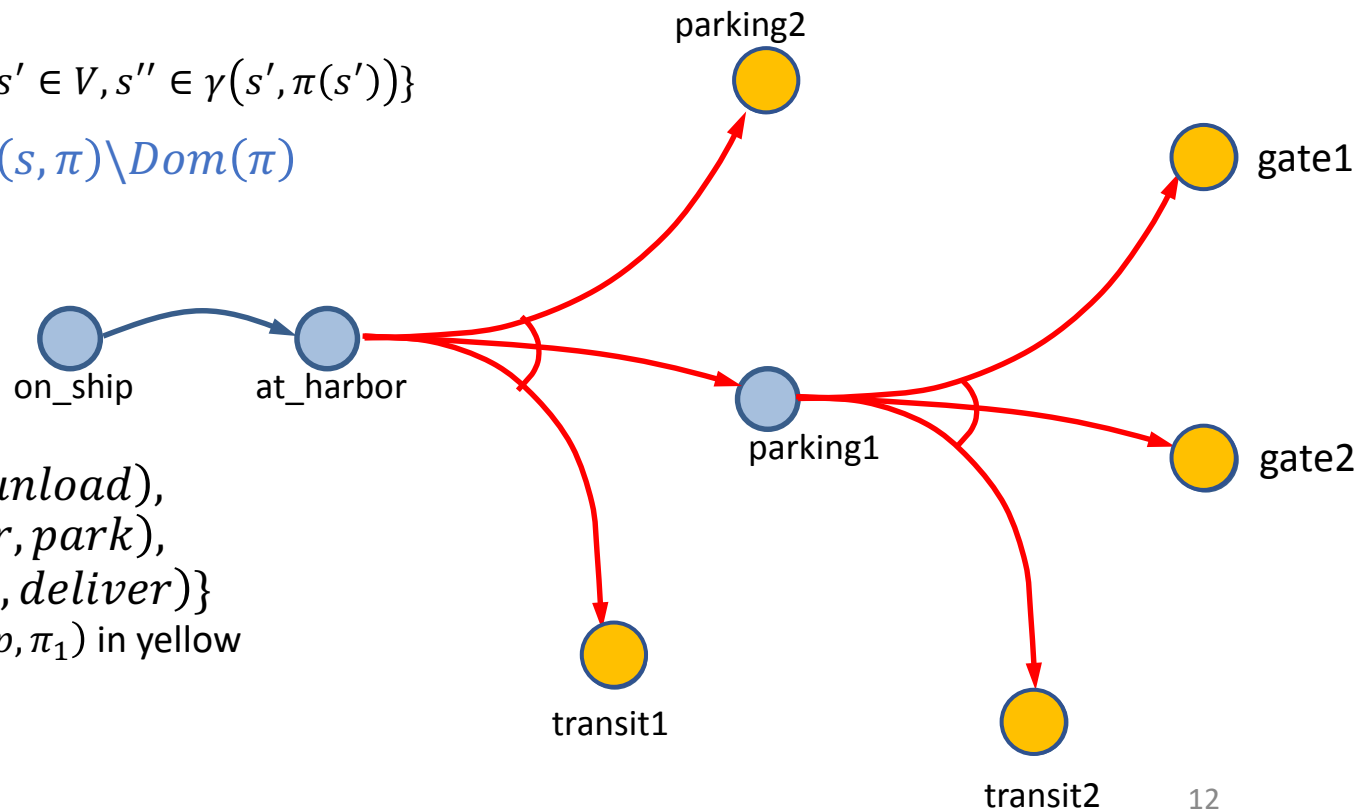
- **Transitive closure:**  
{all states reachable from  $s$  using  $\pi$ }
  - $\hat{\gamma}(s, \pi) = S_0 \cup S_1 \cup S_2 \cup \dots$ 
    - $S_0 = \{s\}$
    - $S_{i+1} = \cup\{\gamma(s, \pi(s)) \mid s \in S_i\}, i \geq 0$
- **Reachability graph**  $Graph(s, \pi) = (V, E)$ 
  - $V = \hat{\gamma}(s, \pi)$
  - $E = \{(s', s'') \mid s' \in V, s'' \in \gamma(s', \pi(s'))\}$



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

# Definitions Over Policies

- **Transitive closure:**  
{all states reachable from  $s$  using  $\pi$ }
  - $\hat{\gamma}(s, \pi) = S_0 \cup S_1 \cup S_2 \cup \dots$ 
    - $S_0 = \{s\}$
    - $S_{i+1} = \cup\{\gamma(s, \pi(s)) \mid s \in S_i\}, i \geq 0$
- **Reachability graph**  $Graph(s, \pi) = (V, E)$ 
  - $V = \hat{\gamma}(s, \pi)$
  - $E = \{(s', s'') \mid s' \in V, s'' \in \gamma(s', \pi(s'))\}$
- $leaves(s, \pi) = \hat{\gamma}(s, \pi) \setminus Dom(\pi)$ 
  - May be empty



- $\pi_1 = \{(on\_ship, unload), (at\_harbor, park), (parking1, deliver)\}$ 
  - $leaves(on\_ship, \pi_1)$  in yellow

# Performing a Policy

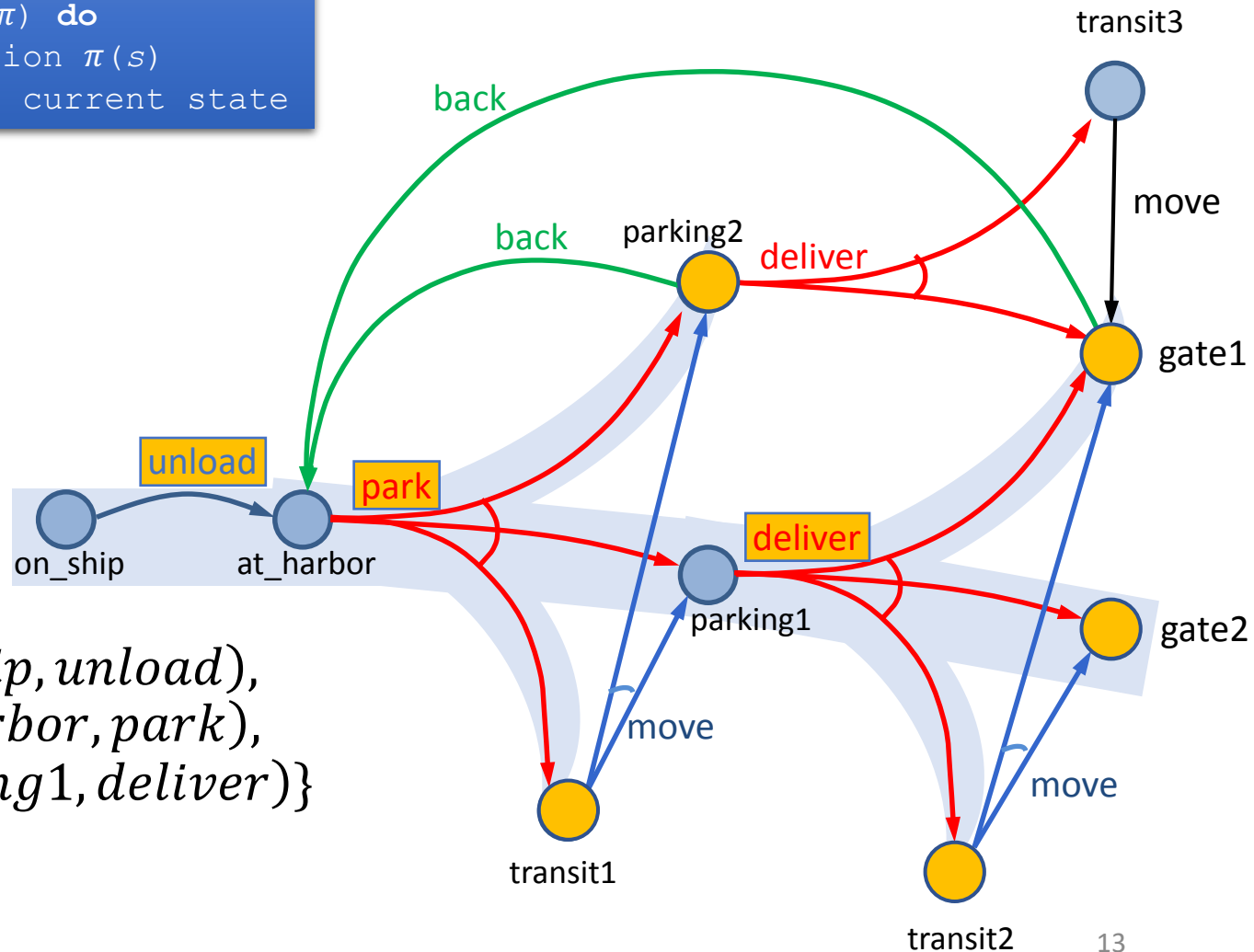
```
PerformPolicy( $\pi$ )
```

```
   $s \leftarrow$  observe current state
```

```
  while  $s \in \text{Dom}(\pi)$  do
```

```
    perform action  $\pi(s)$ 
```

```
     $s \leftarrow$  observe current state
```



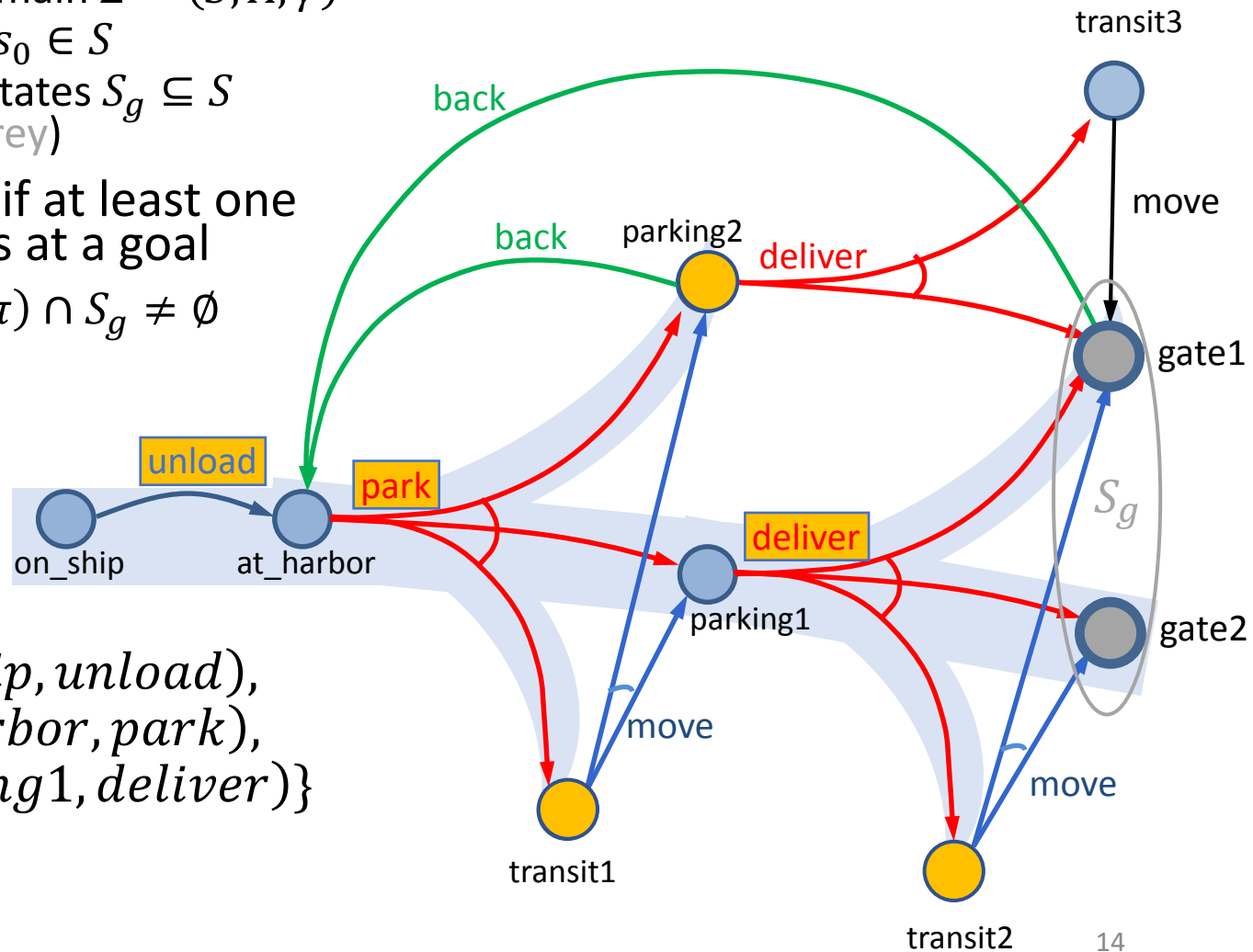
- $\pi_1 = \{(on\_ship, unload), (at\_harbor, park), (parking1, deliver)\}$

# Planning Problems and Solutions

- 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 grey)
- $\pi$  is a **solution** if at least one execution ends at a goal
  - $leaves(s_0, \pi) \cap S_g \neq \emptyset$

Is  $\pi_1$  a solution?

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

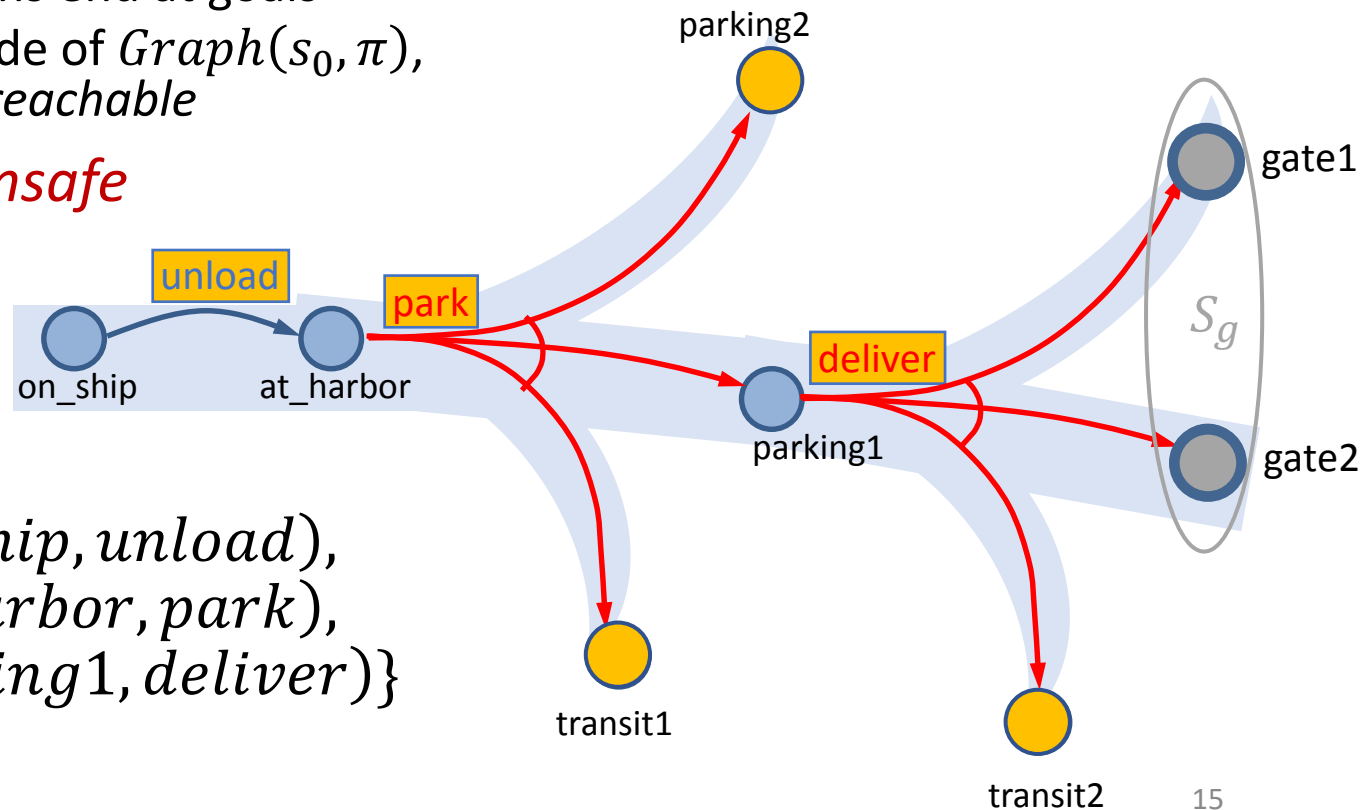


# Safe Solutions

- A solution  $\pi$  is **safe** if

$$\forall s \in \hat{\gamma}(s_0, \pi), \\ \text{leaves}(s, \pi) \cap S_g \neq \emptyset$$

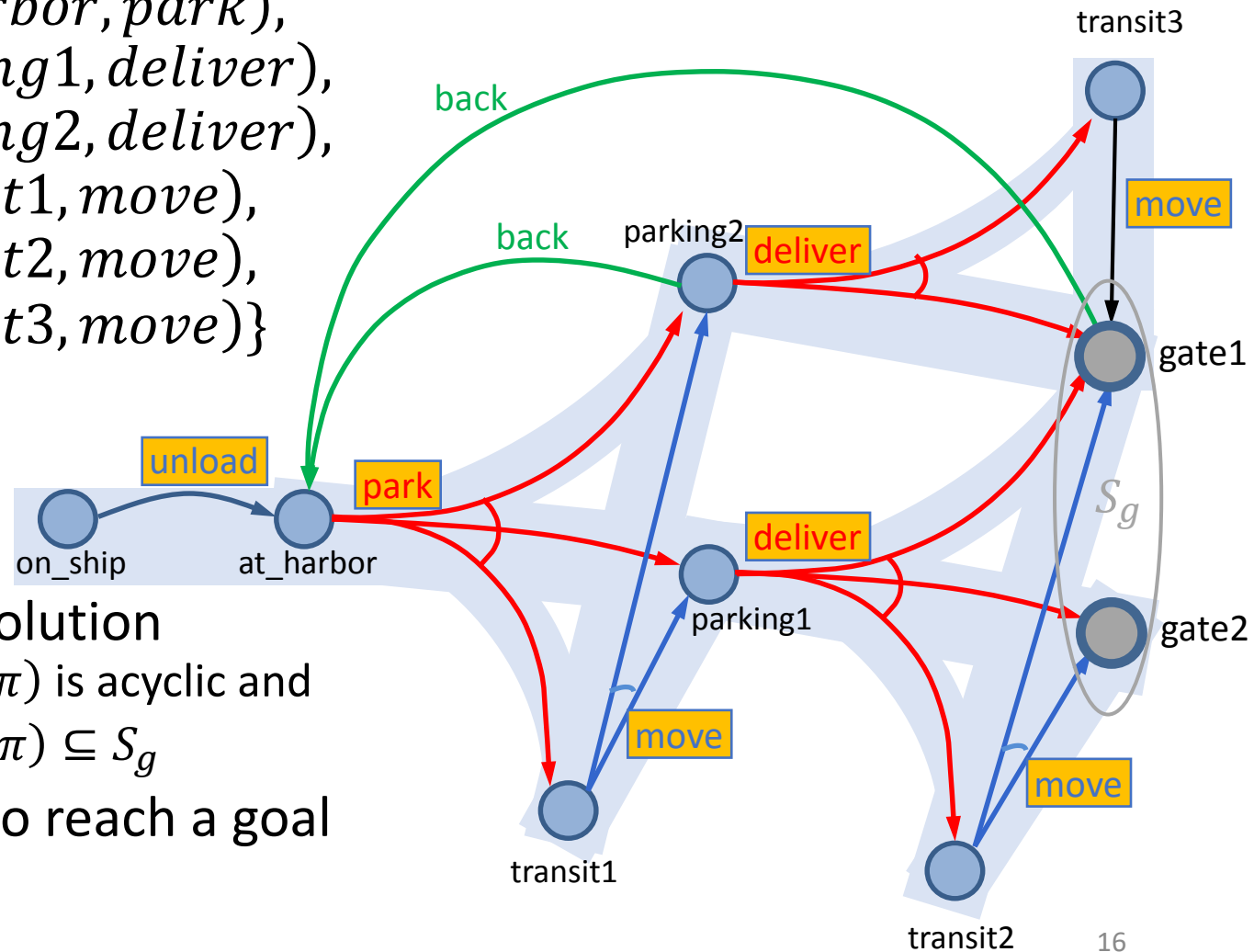
- ~~all executions end at goals~~
- at every node of  $\text{Graph}(s_0, \pi)$ , the goal is *reachable*
- Otherwise, **unsafe**



# Safe Solutions

- $\pi_2 = \{(on\_ship, unload), (at\_harbor, park), (parking1, deliver), (parking2, deliver), (transit1, move), (transit2, move), (transit3, move)\}$

- **Acyclic safe solution**
  - $Graph(s_0, \pi)$  is acyclic and
  - $leaves(s_0, \pi) \subseteq S_g$
- Guaranteed to reach a goal



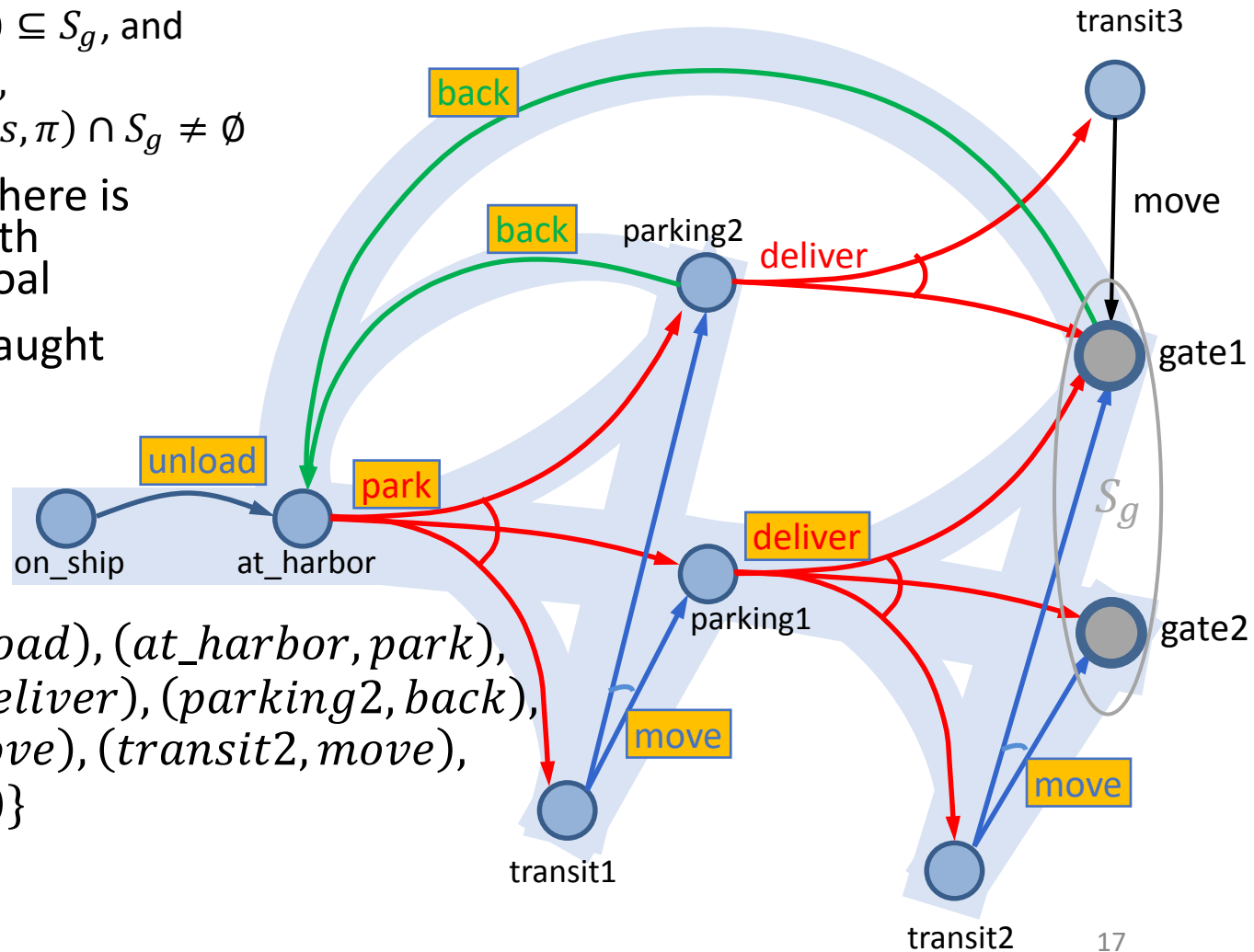


# Safe Solutions

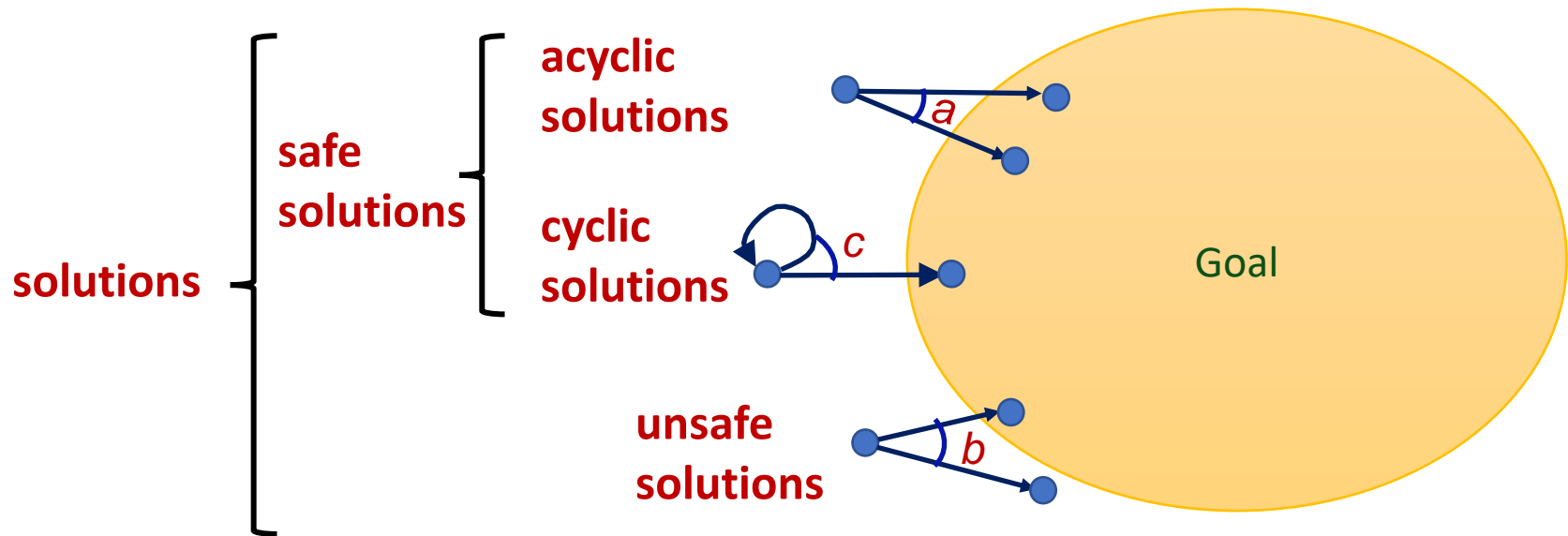
- **Cyclic** safe solution
  - $Graph(s_0, \pi)$  is cyclic,
  - $leaves(s_0, \pi) \subseteq S_g$ , and
  - $\forall s \in \hat{\gamma}(s_0, \pi), leaves(s, \pi) \cap S_g \neq \emptyset$

- At every state, there is an execution path that ends at a goal
- Will never get caught in a dead end

- $\pi_3 =$   
 $\{(on\_ship, unload), (at\_harbor, park), (parking1, deliver), (parking2, back), (transit1, move), (transit2, move), (gate1, back)\}$



# Kinds of Solutions



# Intermediate Summary

---

- Planning Problems
  - Planning domains
  - Plans as policies
  - Planning problems and solutions
    - Types of solutions: safe, unsafe, acyclic, cyclic

# Outline per the Book

---

## *5.2 Planning Problem*

- Planning domains
- Plans as policies
- Planning problems and solutions

## ***5.3 And/Or Graph Search***

- Planning by forward search

## *5.5 Determinisation Techniques*

- Guided planning for safe solutions
- Planning for safe solutions by determinisation

## *5.6 Online Approaches*

- Lookahead
- Lookahead by determinisation
- Lookahead with a bounded number of steps

# Finding (Unsafe) Solutions

For comparison:  
Forward-search with  
deterministic models

**Find-Solution** ( $\Sigma, s_0, S_g$ )

```
 $s \leftarrow s_0$   
 $\pi \leftarrow \emptyset$   
 $Visited \leftarrow \{s_0\}$   
loop  
  if  $s \in S_g$  then  
    return  $\pi$   
   $A' \leftarrow \text{Applicable}(s)$   
  if  $A' = \emptyset$  then  
    return failure  
  nondeterministically choose  $a \in A'$   
  nondeterministically choose  $s' \in \gamma(s, a)$   
  if  $s' \in Visited$  then  
    return failure  
   $\pi(s) \leftarrow a$   
   $Visited \leftarrow Visited \cup \{s'\}$   
   $s \leftarrow s'$ 
```

**Forward-search** ( $\Sigma, s_0, g$ )

```
 $s \leftarrow s_0$   
 $\pi \leftarrow \langle \rangle$   
loop  
  if  $s$  satisfies  $g$  then  
    return  $\pi$   
   $A' \leftarrow \{a \in A \mid a \text{ is applicable in } s\}$   
  if  $A' = \emptyset$  then  
    return failure  
  nondeterministically choose  $a \in A'$   
   $s \leftarrow \gamma(s, a)$   
   $\pi \leftarrow \pi.a$ 
```

Decide which state to plan for

Cycle-checking

Find-Solution( $\Sigma, s_0, S_g$ )

$s \leftarrow s_0$

$\pi \leftarrow \emptyset$

$Visited \leftarrow \{s_0\}$

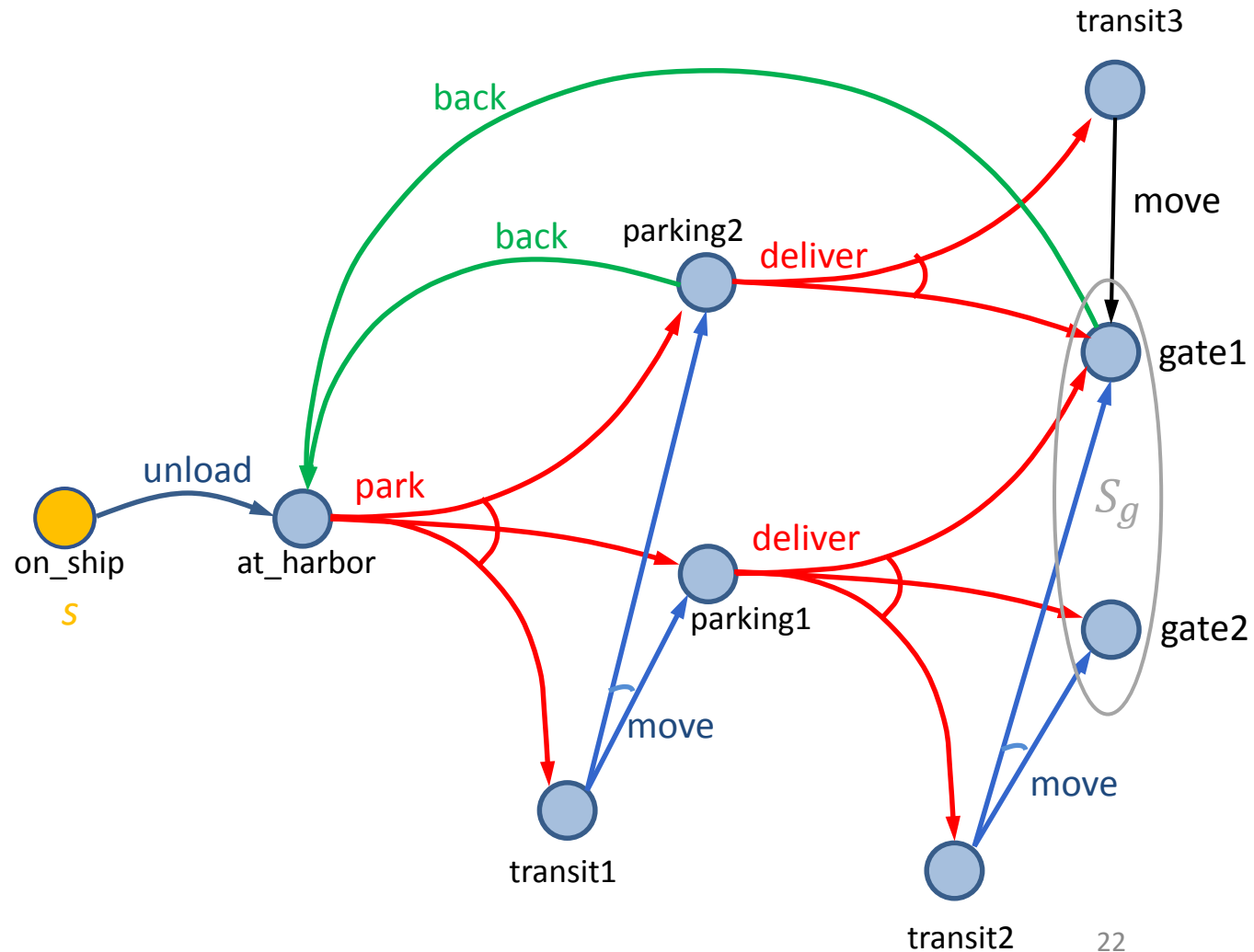
...

# Example

$s = \text{on\_ship}$

$\pi = \{\}$

$Visited = \{\text{on\_ship}\}$



Find-Solution( $\Sigma, s_0, S_g$ )

...

loop

if  $s \in S_g$  then

return  $\pi$

...

nondeterministically choose  $a \in \text{Applicable}(s)$

nondeterministically choose  $s' \in \gamma(s, a)$

...

$\pi(s) \leftarrow a$

$Visited \leftarrow Visited \cup \{s'\}$

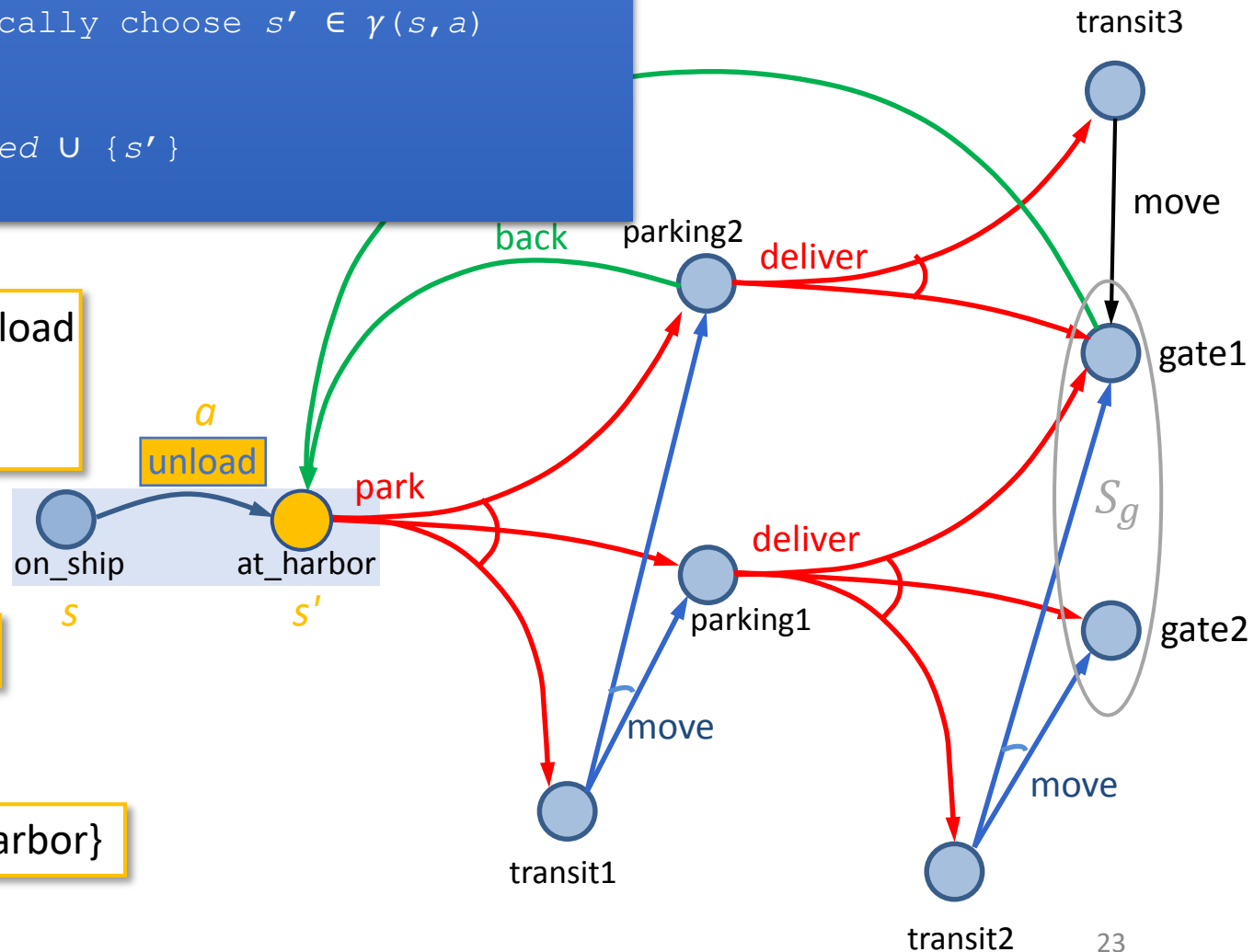
$s \leftarrow s'$

# Example

$s = \text{on\_ship}$ ,  $a = \text{unload}$   
 $\gamma(s, a) = \{\text{at\_harbor}\}$   
 $s' = \text{at\_harbor}$

$\pi = \{(\text{on\_ship}, \text{unload})\}$

$Visited = \{\text{on\_ship}, \text{at\_harbor}\}$



Find-Solution( $\Sigma, s_0, S_g$ )

...

loop

if  $s \in S_g$  then

return  $\pi$

...

nondeterministically choose  $a \in \text{Applicable}(s)$

nondeterministically choose  $s' \in \gamma(s, a)$

...

$\pi(s) \leftarrow a$

$\text{Visited} \leftarrow \text{Visited} \cup \{s'\}$

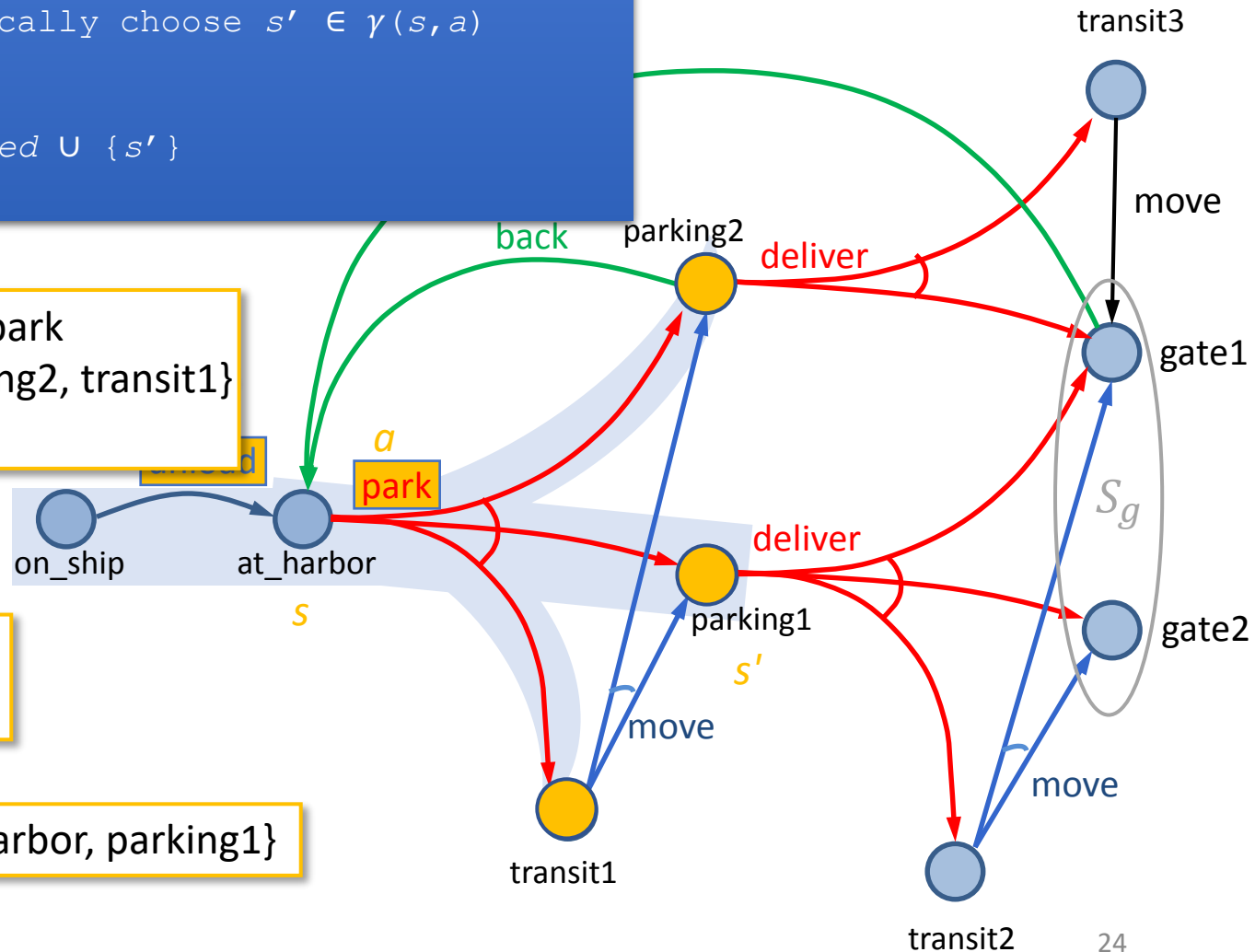
$s \leftarrow s'$

# Example

$s = \text{at\_harbor}$ ,  $a = \text{park}$   
 $\gamma(s, a) = \{\text{parking1}, \text{parking2}, \text{transit1}\}$   
 $s' = \text{parking1}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park})\}$

$\text{Visited} = \{\text{on\_ship}, \text{at\_harbor}, \text{parking1}\}$









# Finding Acyclic Safe Solutions

```
Find-Acyclic-Solution( $\Sigma, s_0, S_g$ )
```

```
 $\pi \leftarrow \emptyset$ 
```

```
Frontier  $\leftarrow \{s_0\}$ 
```

```
for every  $s \in \text{Frontier} \setminus S_g$  do
```

```
  Frontier  $\leftarrow \text{Frontier} \setminus \{s\}$ 
```

```
  if  $\text{Applicable}(s) = \emptyset$  then
```

```
    return failure
```

```
  nondeterministically choose  $a \in \text{Applicable}(s)$ 
```

```
   $\pi \leftarrow \pi \cup (s, a)$ 
```

```
  Frontier  $\leftarrow \text{Frontier} \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$ 
```

```
  if  $\text{has-loops}(\pi, s, \text{Frontier})$  then
```

```
    return failure
```

```
return  $\pi$ 
```

Keep track of unexpanded states, like in A\*

Add all outcomes that  $\pi$  does not already handle

Cycle-checking

- Check for cycles

- For each  $s' \in (\gamma(s, a) \cap \text{Dom}(\pi))$

- Is  $s' \in \hat{\gamma}(s', \pi)$ ?

- Formally,  $\text{has-loops}(\pi, s, \text{Frontier})$  iff

- $\exists s' \in (\gamma(s, a) \cap \text{Dom}(\pi)) : s' \in \hat{\gamma}(s', \pi)$

- I.e., a state  $s'$  is reachable from itself

# Find-Acyclic-Solution ( $\Sigma, s_0, S_g$ )

$\pi \leftarrow \emptyset$

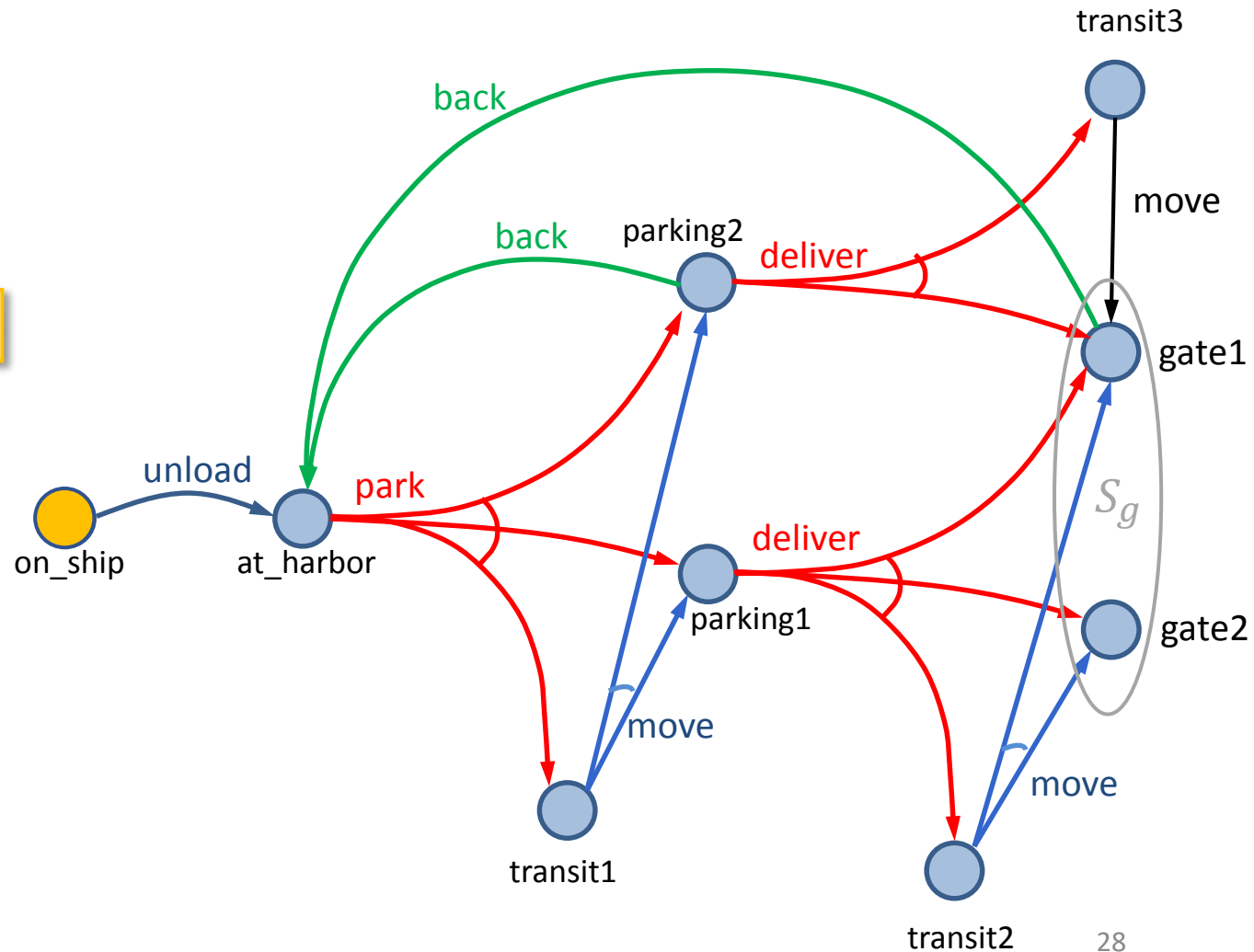
$Frontier \leftarrow \{s_0\}$

...

# Example

$Frontier \setminus S_g = \{on\_ship\}$

$\pi = \{\}$



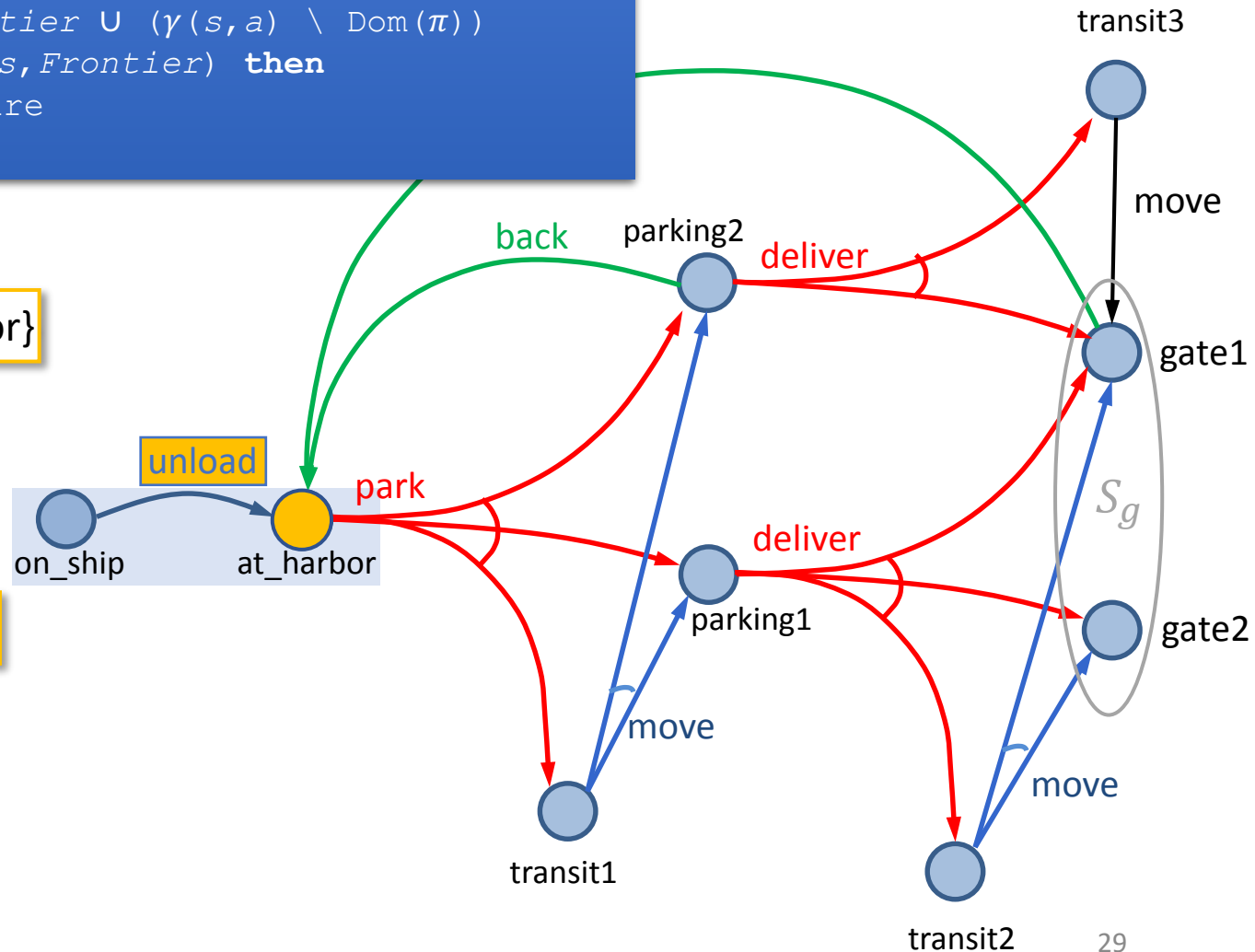
## Find-Acyclic-Solution( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{on\_ship}$

$\text{Frontier} \setminus S_g = \{\text{at\_harbor}\}$

$\pi = \{(\text{on\_ship}, \text{unload})\}$



# Example

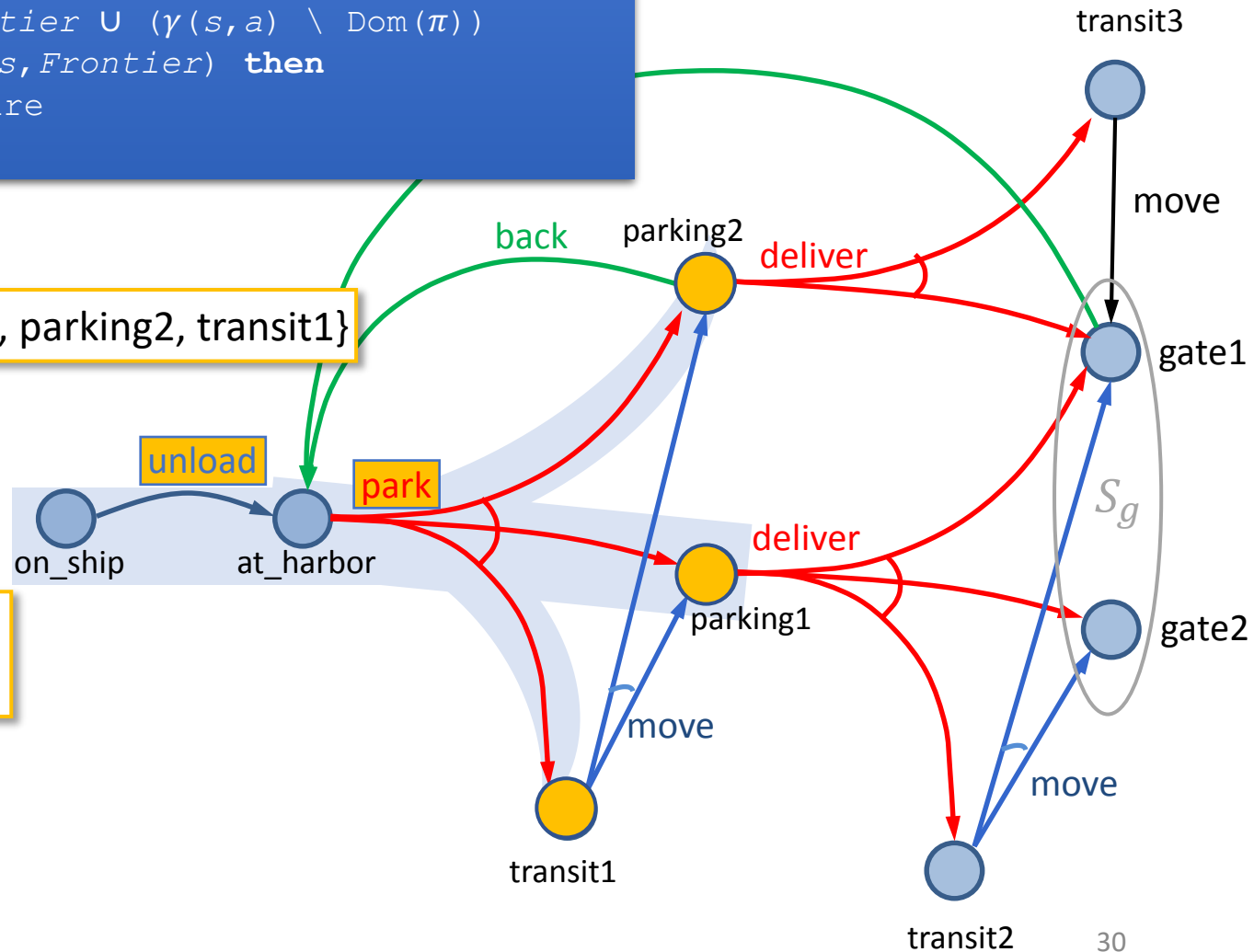
## Find-Acyclic-Solution( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{at\_harbor}$

$\text{Frontier} \setminus S_g = \{\text{parking1}, \text{parking2}, \text{transit1}\}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park})\}$



# Example

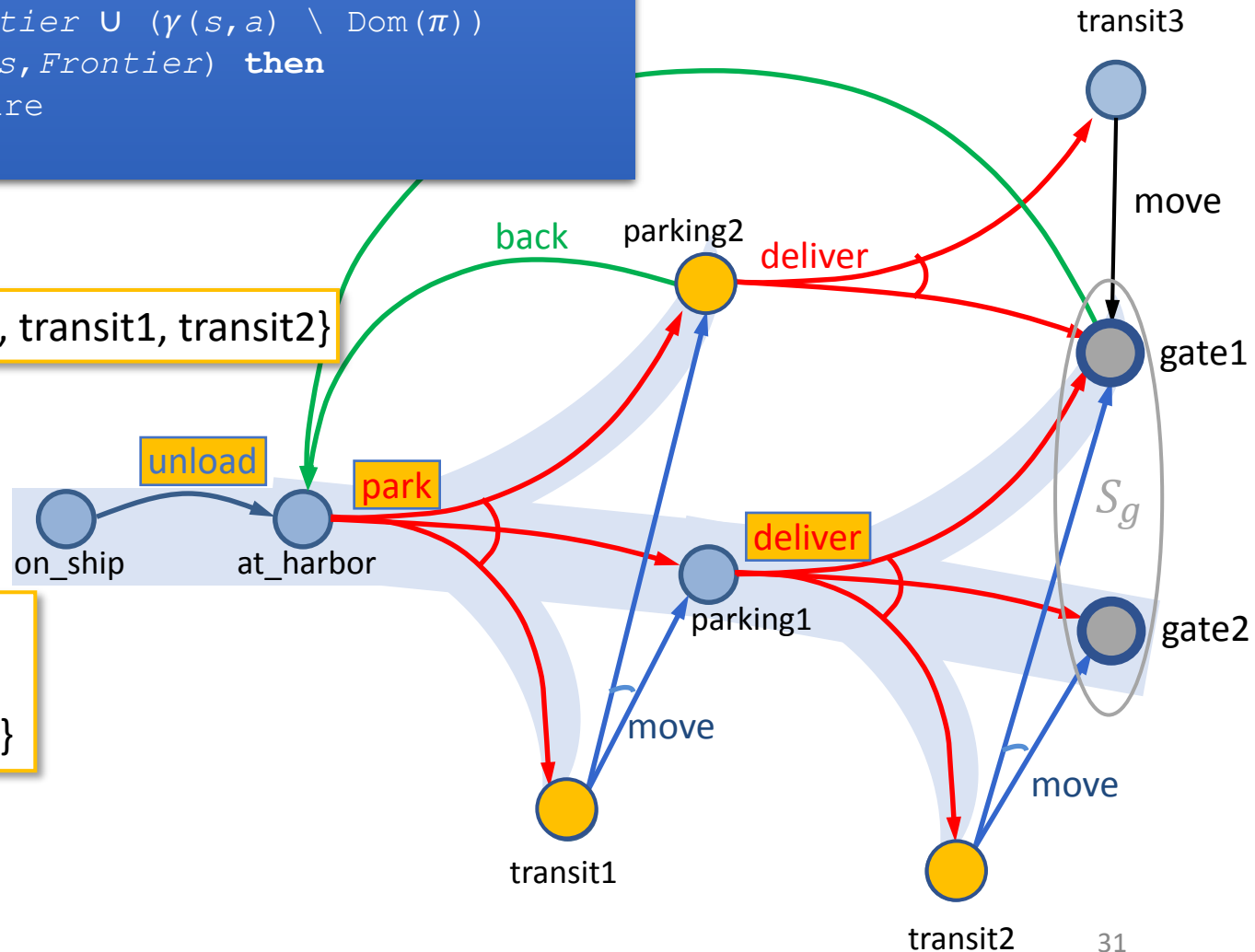
## Find-Acyclic-Solution( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{parking1}$

$\text{Frontier} \setminus S_g = \{\text{parking2}, \text{transit1}, \text{transit2}\}$

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



# Example







## Find-Acyclic-Solution( $\Sigma, s_0, S_g$ )

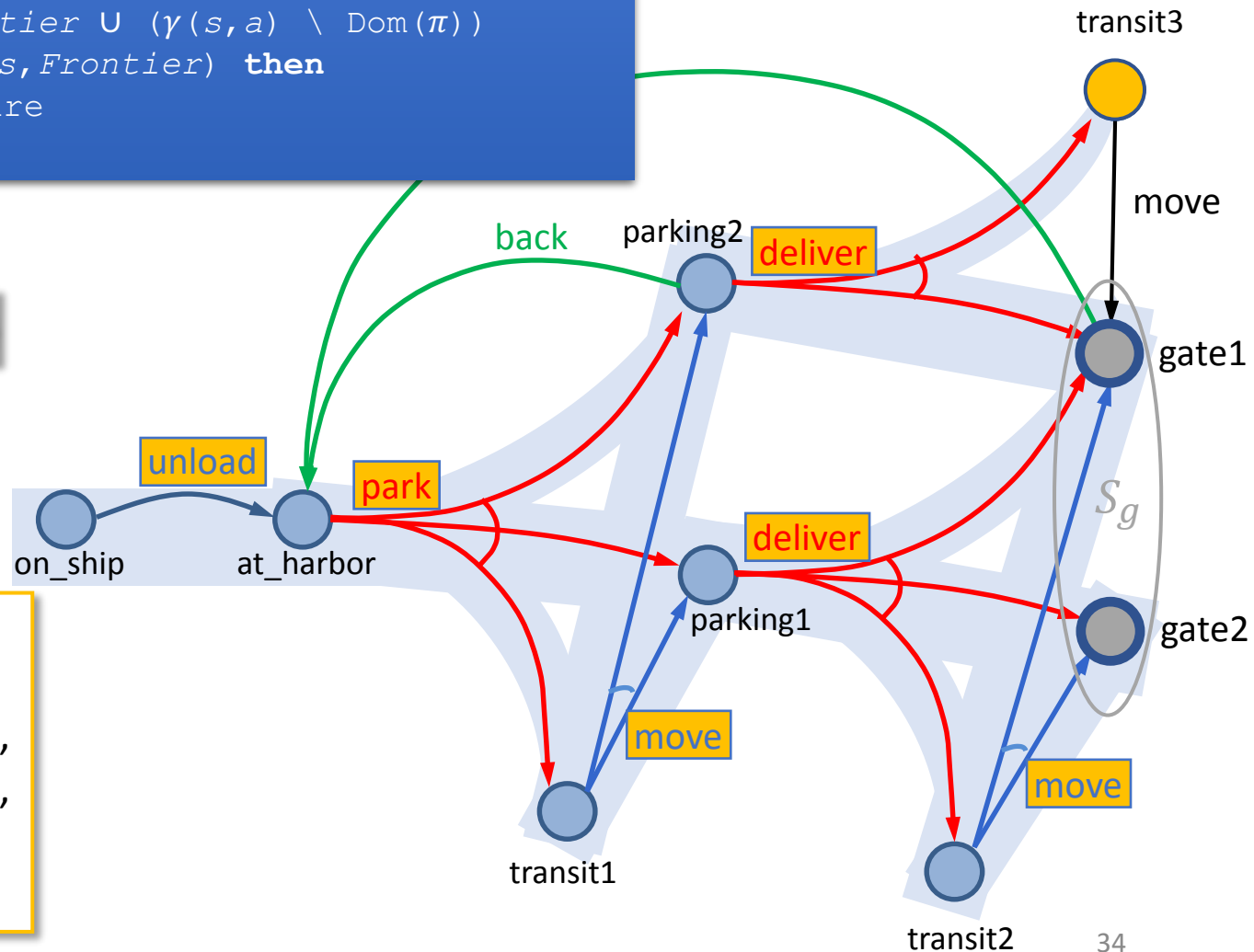
```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{transit2}$

$\text{Frontier} \setminus S_g = \{\text{transit3}\}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 $(\text{transit1}, \text{move}),$   
 $(\text{transit2}, \text{move})\}$

# Example





# Finding Safe Solutions

```
Find-Safe-Solution( $\Sigma, s_0, S_g$ )
```

```
 $\pi \leftarrow \emptyset$ 
```

```
Frontier  $\leftarrow \{s_0\}$ 
```

```
for every  $s \in \text{Frontier} \setminus S_g$  do
```

```
  Frontier  $\leftarrow \text{Frontier} \setminus \{s\}$ 
```

```
  if  $\text{Applicable}(s) = \emptyset$  then
```

```
    return failure
```

```
  nondeterministically choose  $a \in \text{Applicable}(s)$ 
```

```
   $\pi \leftarrow \pi \cup (s, a)$ 
```

```
  Frontier  $\leftarrow \text{Frontier} \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$ 
```

```
  if  $\text{has-unsafe-loops}(\pi, s, \text{Frontier})$  then
```

```
    return failure
```

```
return  $\pi$ 
```

Different cycle-checking

- Same as Find-Acyclic-Solution except for cycle-checking
- has-unsafe-loops instead of has-loops
- Check if  $\pi$  contains any cycles that cannot be escaped:
  - For each  $s' \in (\gamma(s, a) \cap \text{Dom}(\pi))$ 
    - Is  $\hat{\gamma}(s', \pi) \cap \text{Frontier} = \emptyset$ ?
  - Formally,  $\text{has-unsafe-loops}(\pi, s, \text{Frontier})$  iff
$$\exists s' \in (\gamma(s, a) \cap \text{Dom}(\pi)) : \hat{\gamma}(s', \pi) \cap \text{Frontier} = \emptyset$$

Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

$\pi \leftarrow \emptyset$

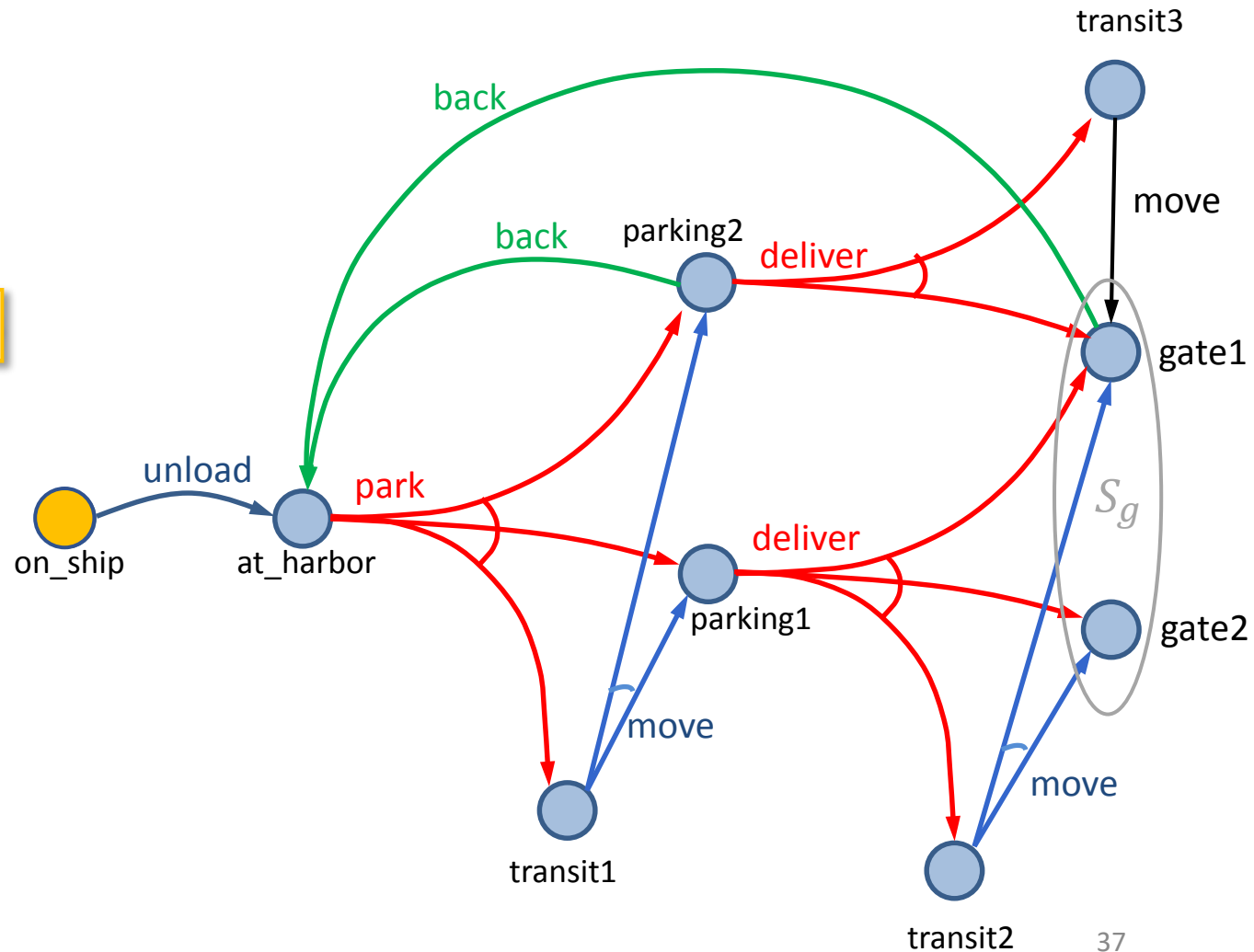
Frontier  $\leftarrow \{s_0\}$

...

# Example

Frontier  $\setminus S_g = \{\text{on\_ship}\}$

$\pi = \{\}$



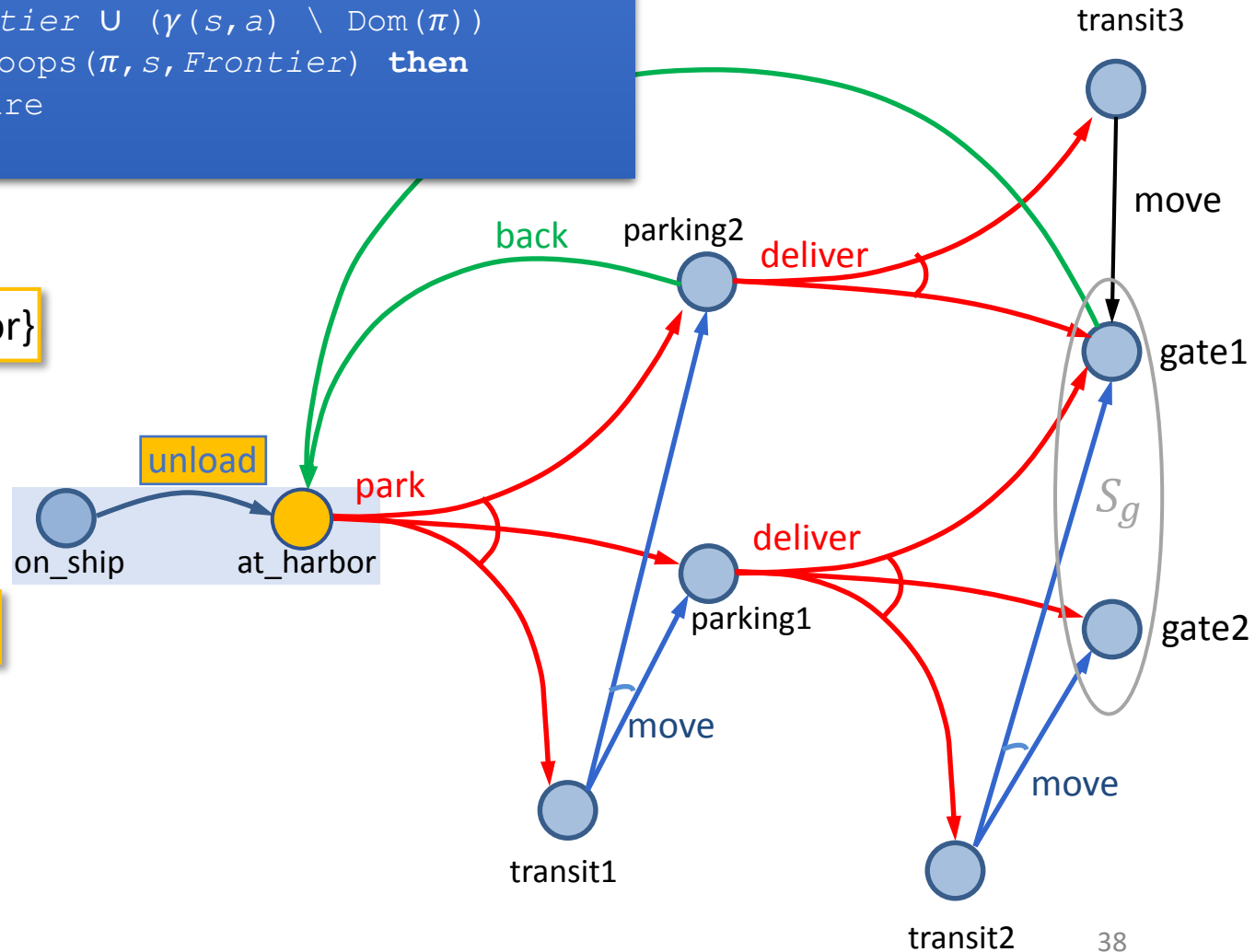
## Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-unsafe-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{on\_ship}$

$\text{Frontier} \setminus S_g = \{\text{at\_harbor}\}$

$\pi = \{(\text{on\_ship}, \text{unload})\}$



# Example

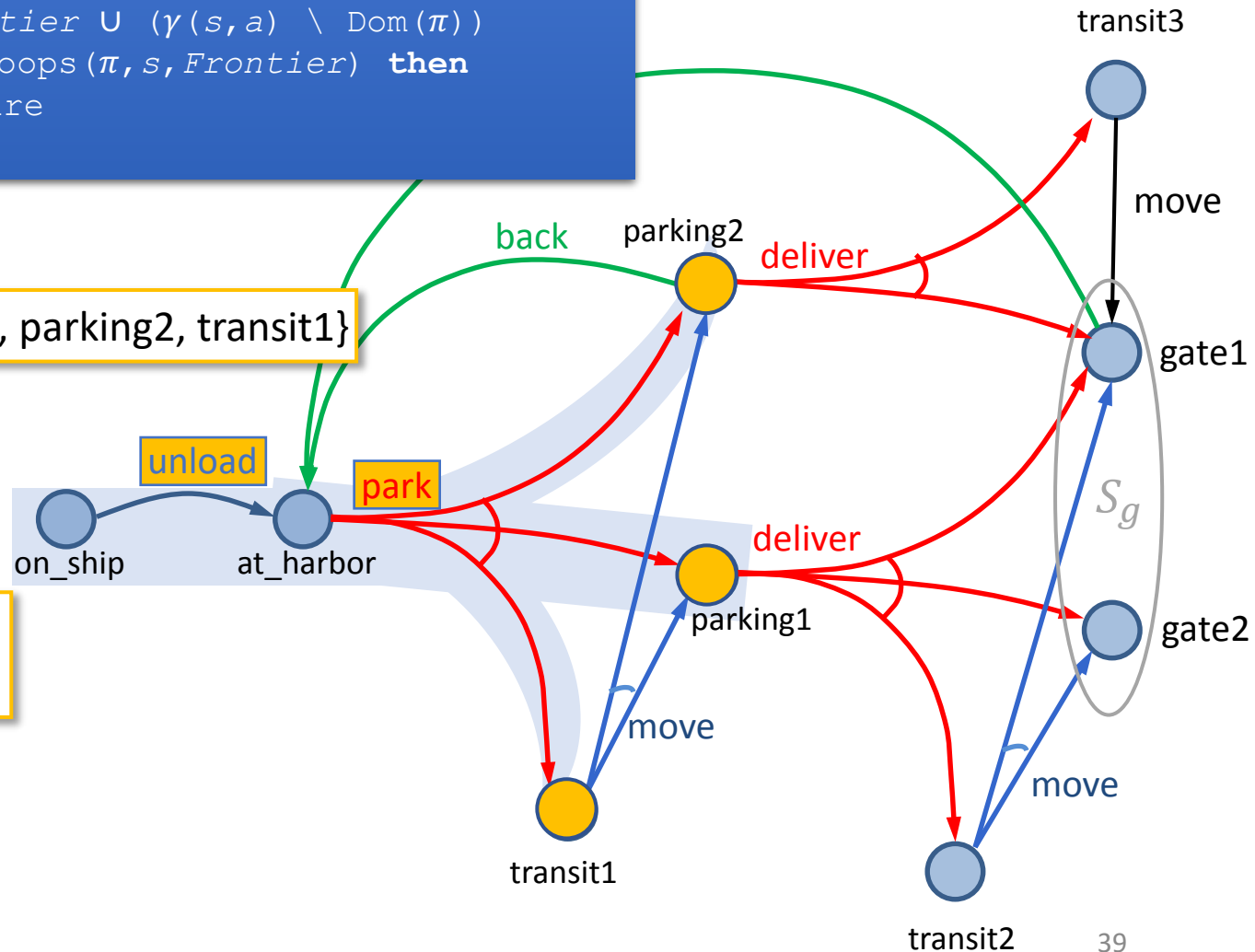
## Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-unsafe-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{at\_harbor}$

$\text{Frontier} \setminus S_g = \{\text{parking1}, \text{parking2}, \text{transit1}\}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park})\}$



# Example

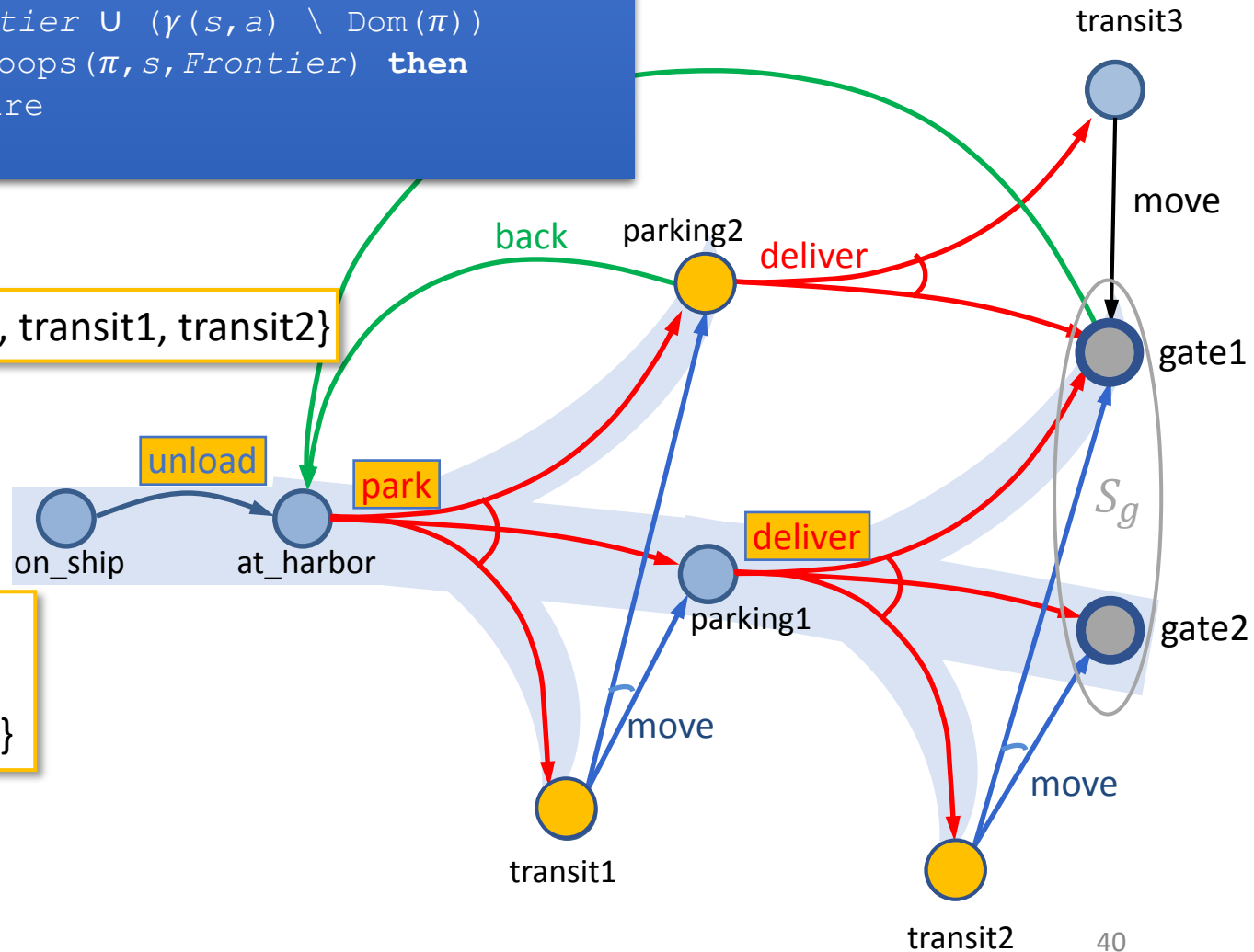
## Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-unsafe-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{parking1}$

$\text{Frontier} \setminus S_g = \{\text{parking2}, \text{transit1}, \text{transit2}\}$

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







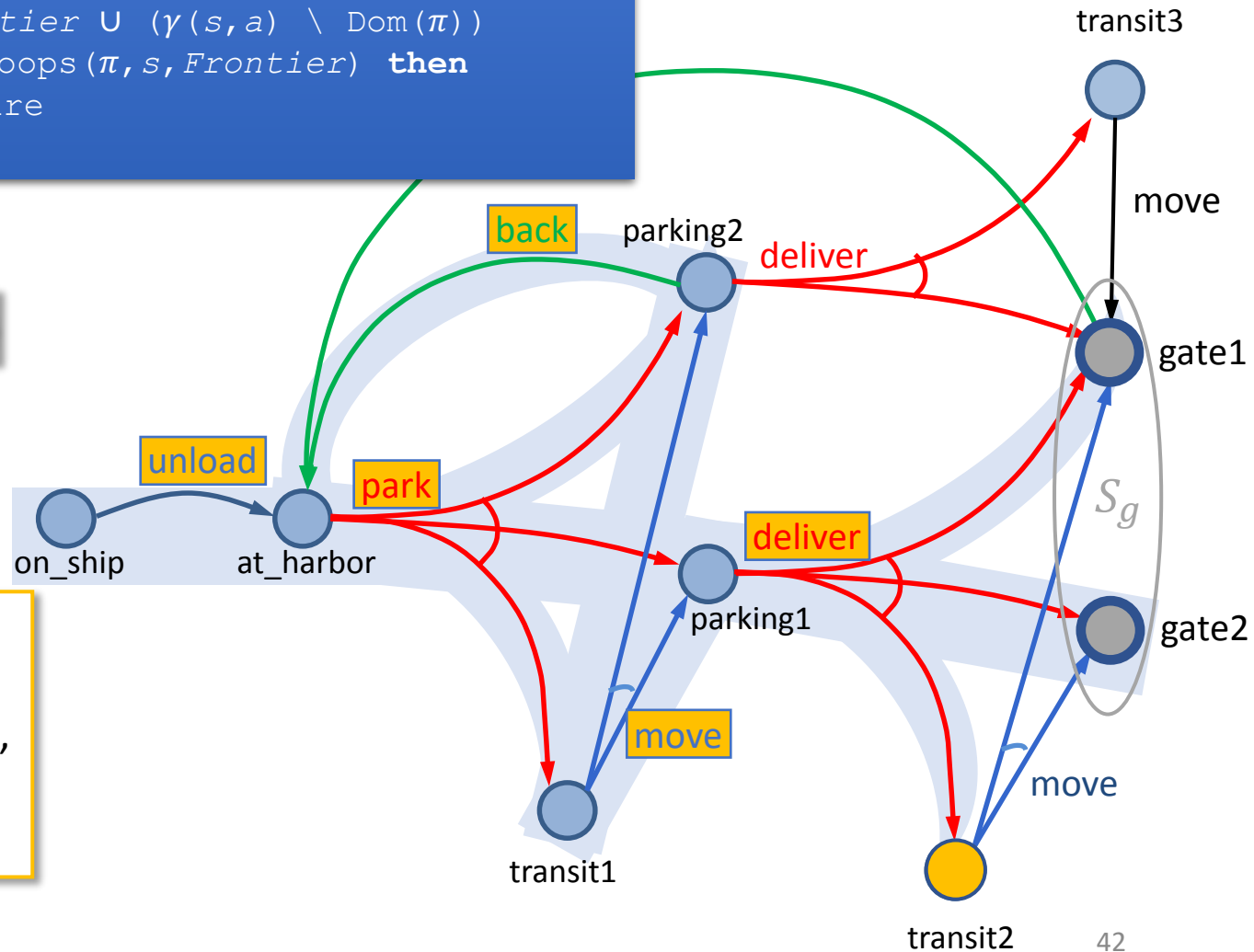
## Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

```
...  
for every  $s \in \text{Frontier} \setminus S_g$  do  
  Frontier  $\leftarrow$  Frontier  $\setminus$  { $s$ }  
  ...  
  nondeterministically choose  $a \in \text{Applicable}(s)$   
   $\pi \leftarrow \pi \cup (s, a)$   
  Frontier  $\leftarrow$  Frontier  $\cup$  ( $\gamma(s, a) \setminus \text{Dom}(\pi)$ )  
  if has-unsafe-loops( $\pi, s, \text{Frontier}$ ) then  
    return failure  
return  $\pi$ 
```

$s = \text{transit1}$

$\text{Frontier} \setminus S_g = \{\text{transit2}\}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{back}),$   
 $(\text{transit1}, \text{move})\}$



# Example



# Intermediate Summary

---

- And/Or Graph Search
  - Analogue to forward search in deterministic models
  - Algorithms for each type of solution
    - Unsafe
    - Cyclic safe
    - Acyclic safe

# Outline per the Book

---

## *5.2 Planning Problem*

- Planning domains
- Plans as policies
- Planning problems and solutions

## *5.3 And/Or Graph Search*

- Planning by forward search

## ***5.5 Determinisation Techniques***

- Guided planning for safe solutions
- Planning for safe solutions by determinisation

## *5.6 Online Approaches*

- Lookahead
- Lookahead by determinisation
- Lookahead with a bounded number of steps

# Guided-Find-Safe-Solution

---

- Motivation:
  - Much easier to find solutions if they don't have to be safe
  - Find-Safe-Solution needs plans for all possible outcomes of actions
  - Find-Solution only needs a plan for one of them
- Idea:
  - loop
    - Find a solution  $\pi$
    - Look at each leaf node of  $\pi$ 
      - If the leaf node is not a goal, find a solution and incorporate it into  $\pi$

# Guided-Find-Safe-Solution

**Guided-Find-Safe-Solution** ( $\Sigma, s_0, S_g$ )

**if**  $s_0 \in S_g$  **then**

**return**  $\emptyset$

**if**  $\text{Applicable}(s_0) = \emptyset$  **then**

**return** failure

$\pi \leftarrow \emptyset$

**loop**

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

**if**  $Q = \emptyset$  **then**

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$

**return**  $\pi$

    arbitrarily select  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

**if**  $\pi' \neq \text{failure}$  **then**

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

**else if**  $s = s_0$  **then**

**return** failure ← not in the book

**else**

**for** every  $s', a$  s.t.  $s \in \gamma(s', a)$  **do**

$\pi \leftarrow \pi \setminus \{(s', a)\}$

            make  $a$  not applicable in  $s'$

$\pi$  is a solution. Return the part that is reachable from  $s_0$ .

Choose any leaf  $s$  that is not a goal. Find a solution  $\pi'$  for  $s$ .

For each  $(s, a)$  in  $\pi'$ , add to  $\pi$  unless  $\pi$  already has an action at  $s$ .

$s$  is unsolvable. For each  $(s', a)$  that can produce  $s$ , modify  $\pi$  and  $\Sigma$  so we will never use  $a$  at  $s'$

# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{V}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

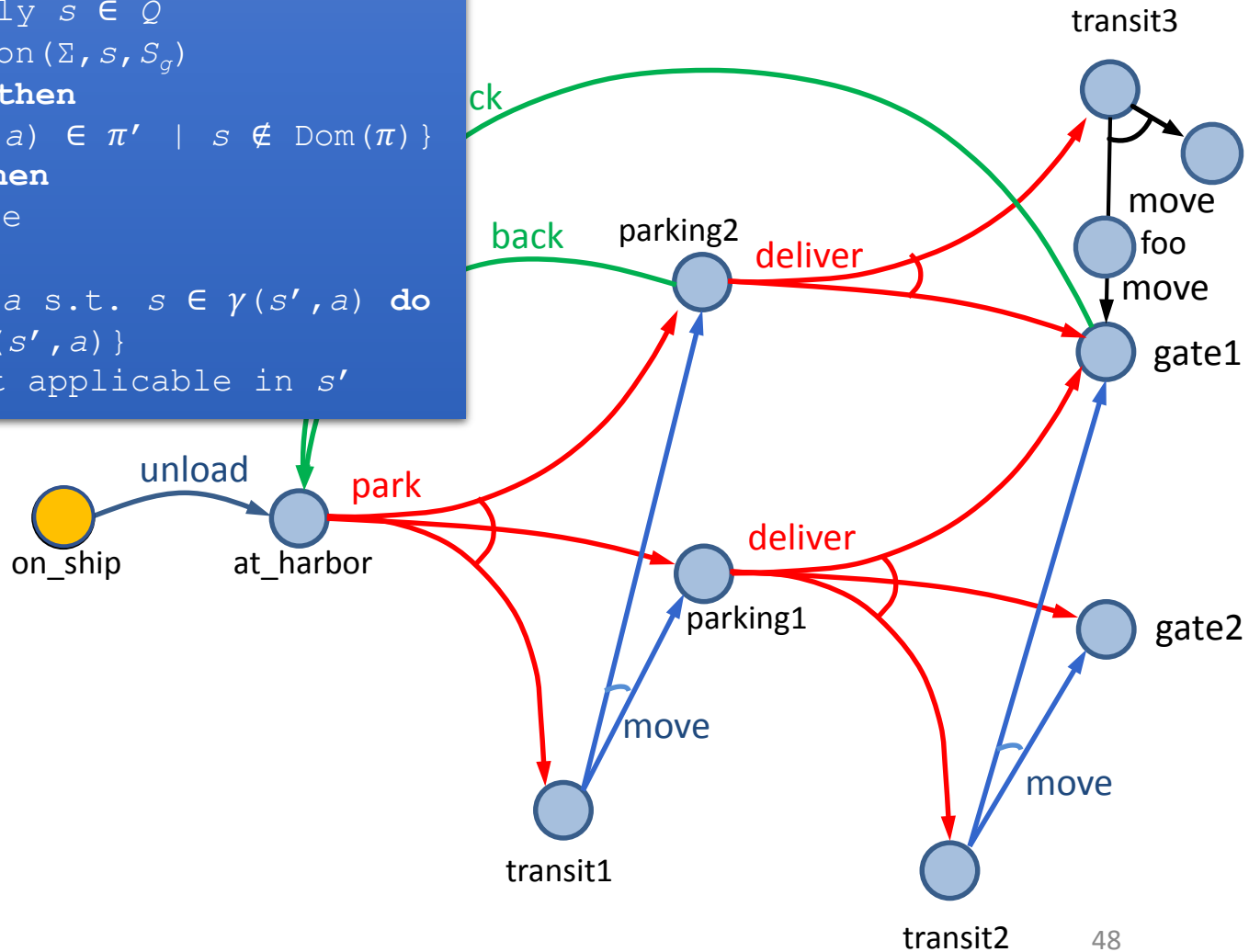
return failure

else

for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do  
 $\pi \leftarrow \pi \setminus \{(s', a)\}$   
make  $a$  not applicable in  $s'$

$s_0 = \text{on\_ship}$

$\pi = \{\}$



# Example



# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\nu}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

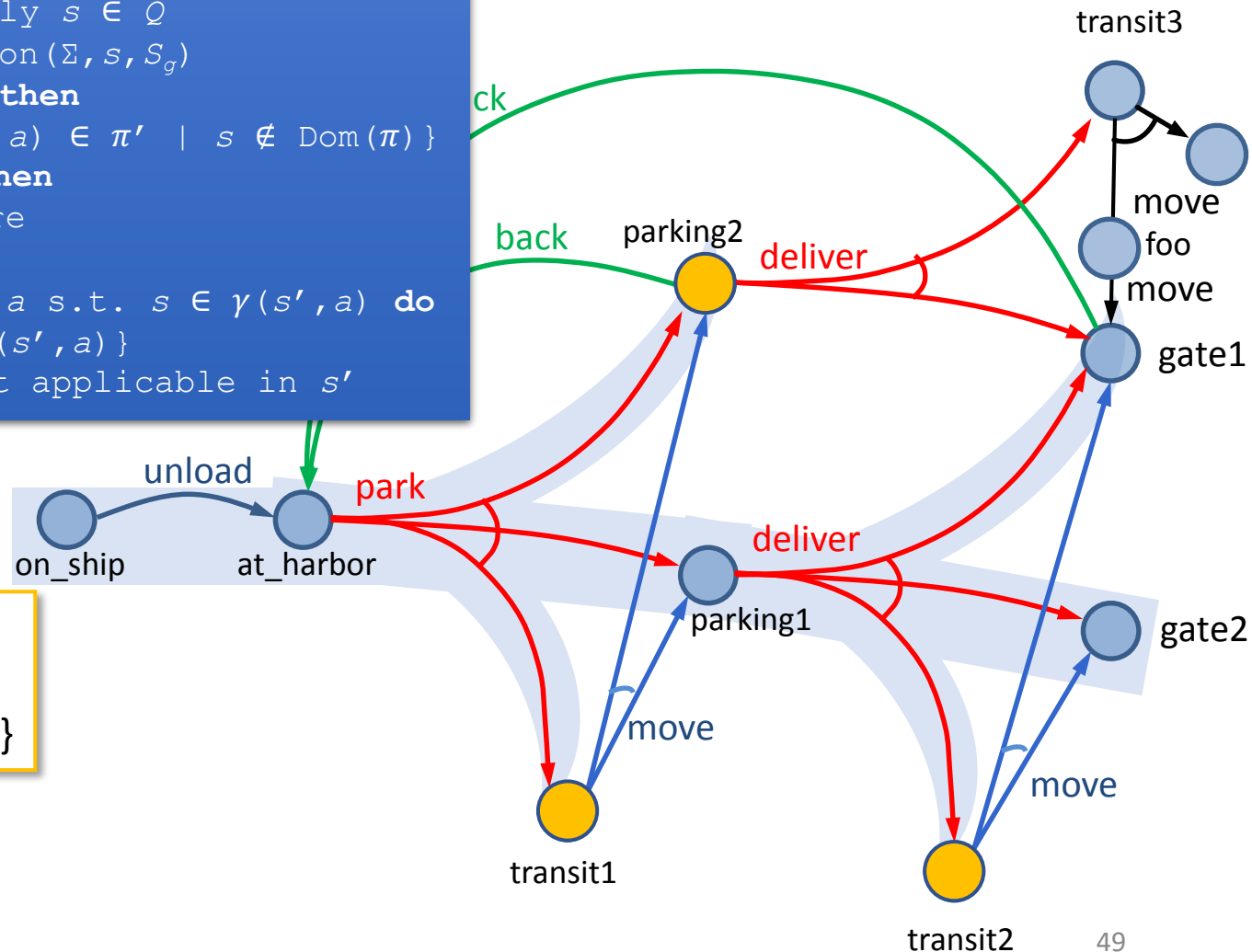
else

for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example



# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{V}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

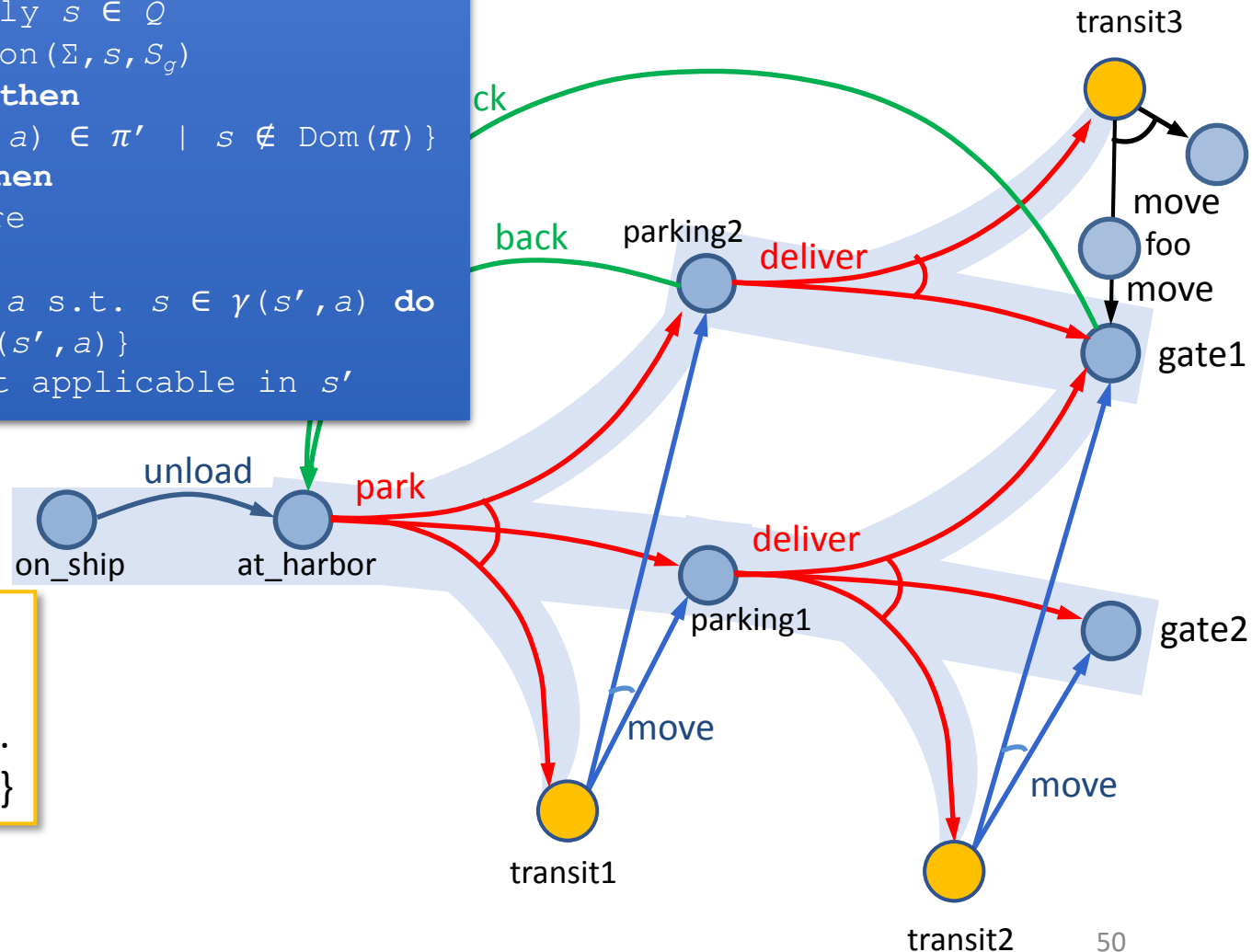
else

for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example



## Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

**if**  $Q = \emptyset$  **then**

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\nu}(s_0, \pi)\}$

**return**  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

**if**  $\pi' \neq \text{failure}$  **then**

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

**else if**  $s = s_0$  **then**

**return** failure

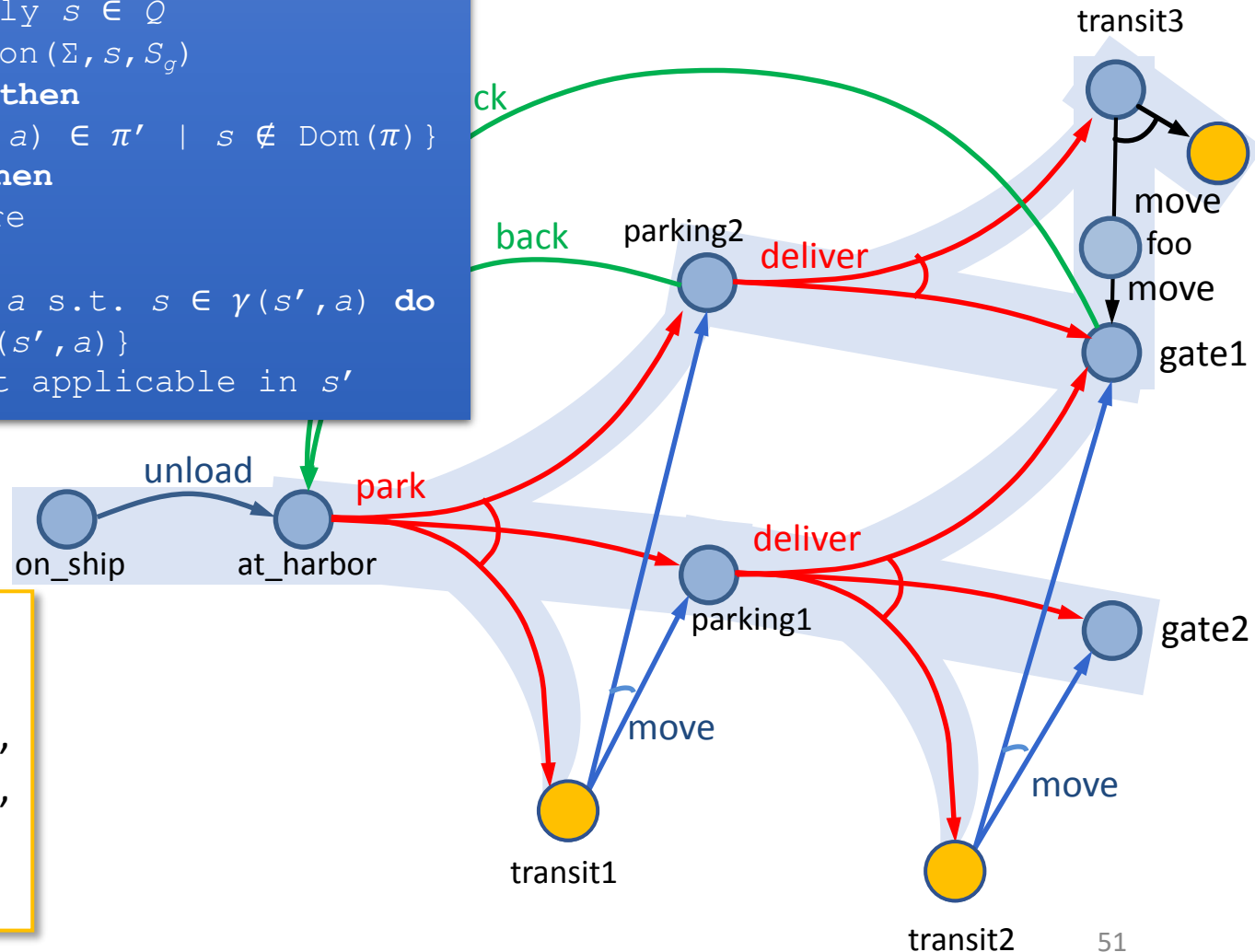
**else**

**for** every  $s', a$  s.t.  $s \in \gamma(s', a)$  **do**

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

## Example



# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\nu}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

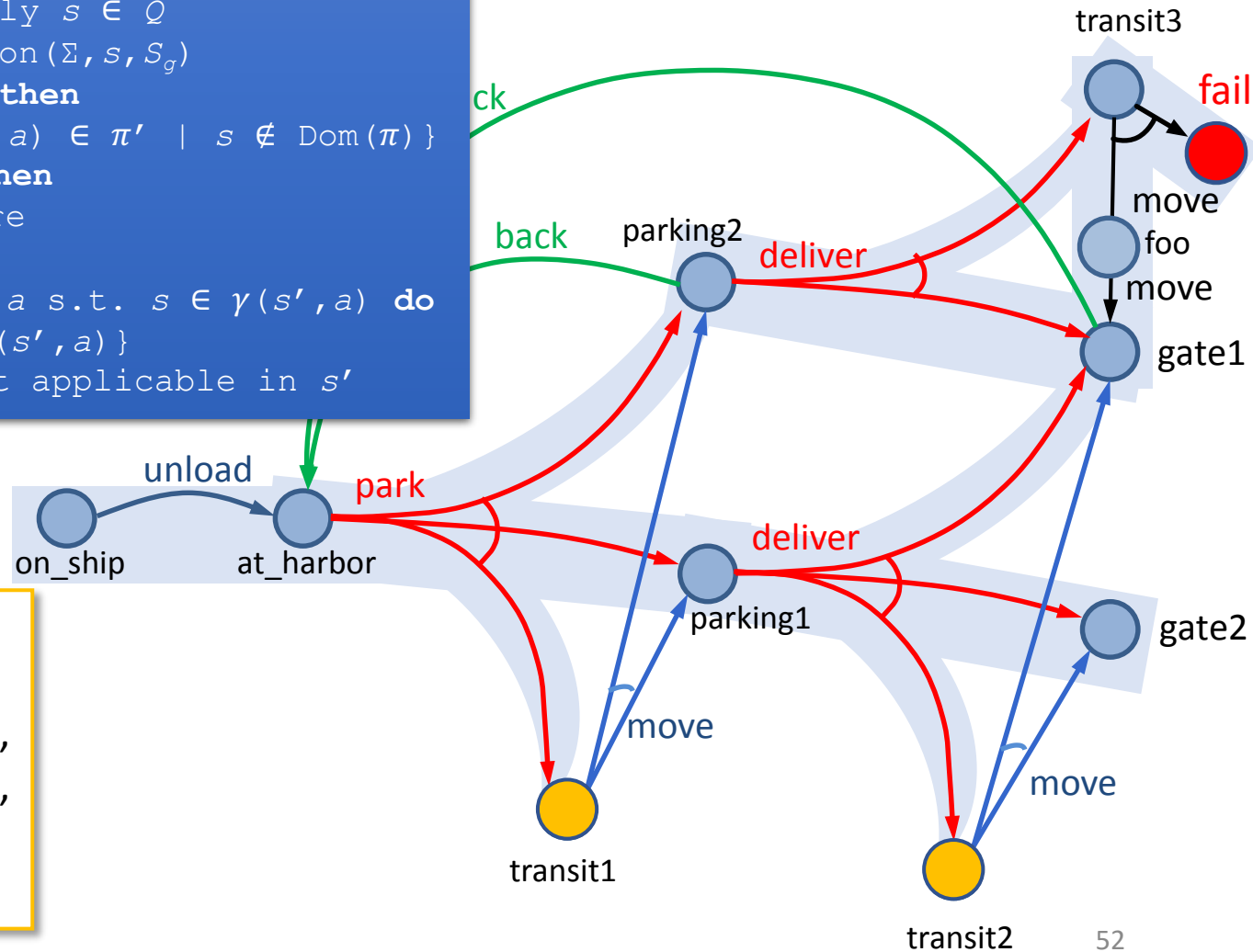
else

for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example



# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{V}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

else

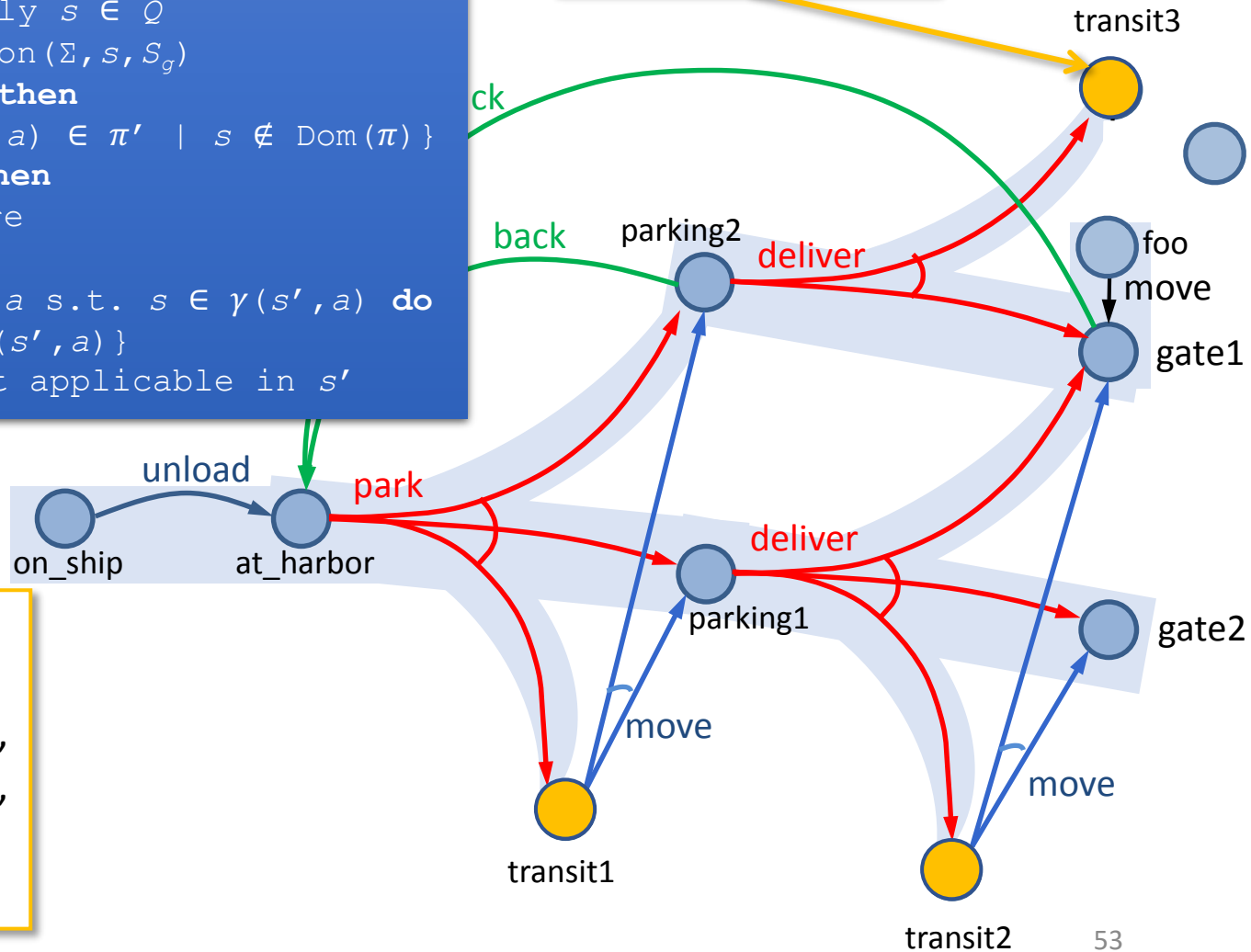
for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example

Modify  $\Sigma_d$  to make  
move inapplicable



$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 $(\text{transit3}, \text{move}),$   
 $(\text{foo}, \text{move})\}$

# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\nu}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

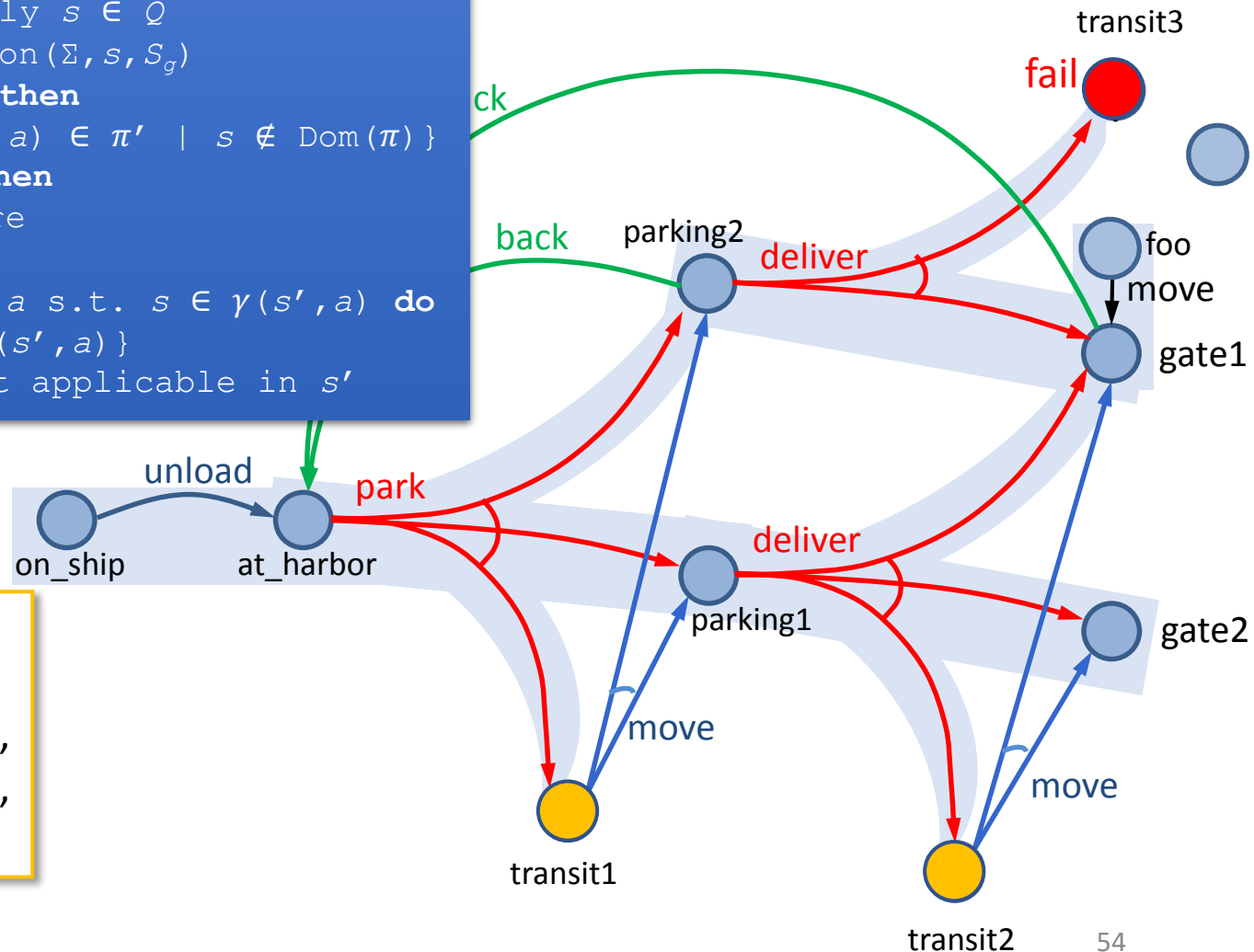
else

for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example



$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 $(\text{foo}, \text{move})\}$

# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{V}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

if  $\pi' \neq \text{failure}$  then

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

else

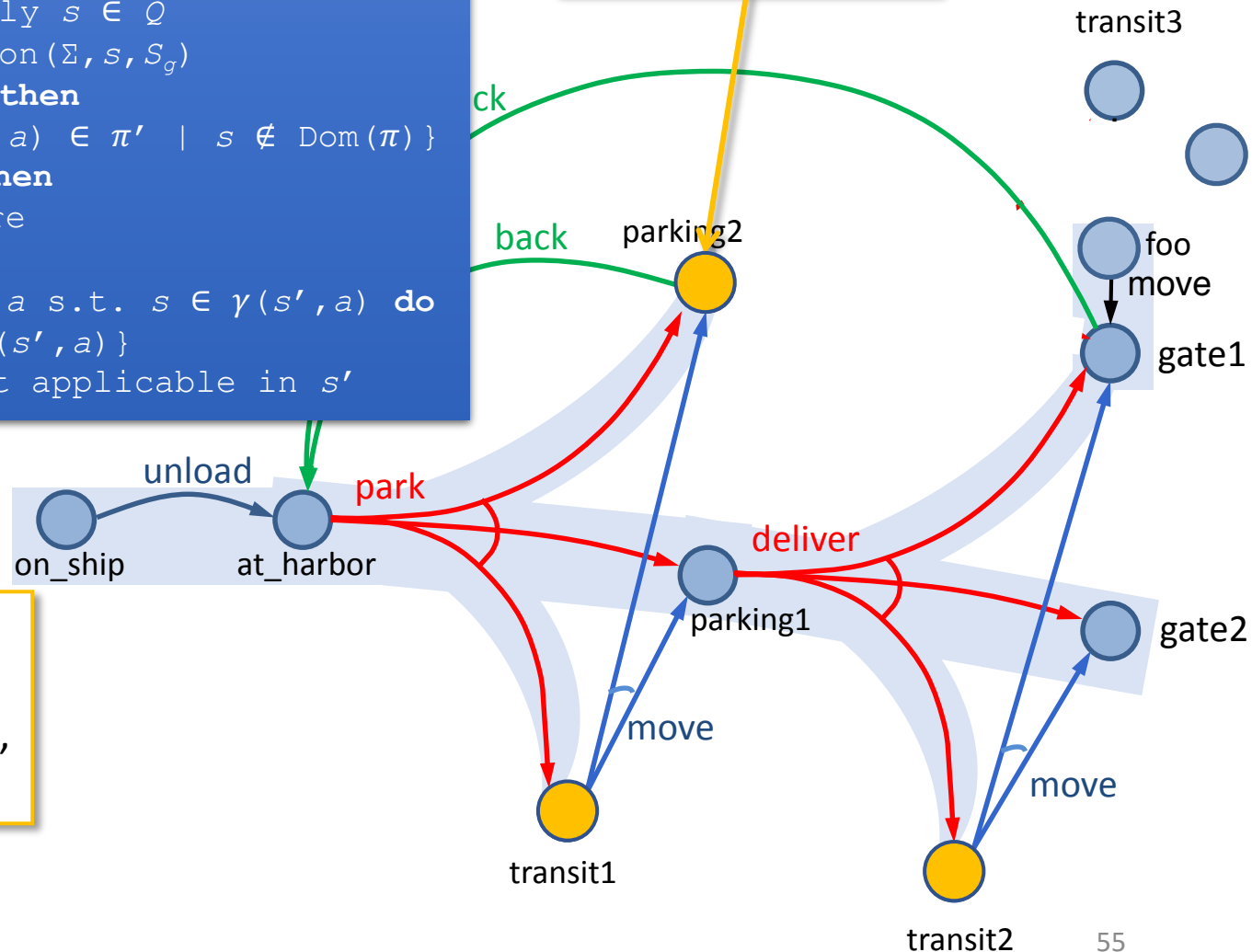
for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example

Modify  $\Sigma_d$  to make  
move inapplicable



## Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

**if**  $Q = \emptyset$  **then**

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\nu}(s_0, \pi)\}$

**return**  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

**if**  $\pi' \neq \text{failure}$  **then**

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

**else if**  $s = s_0$  **then**

**return** failure

**else**

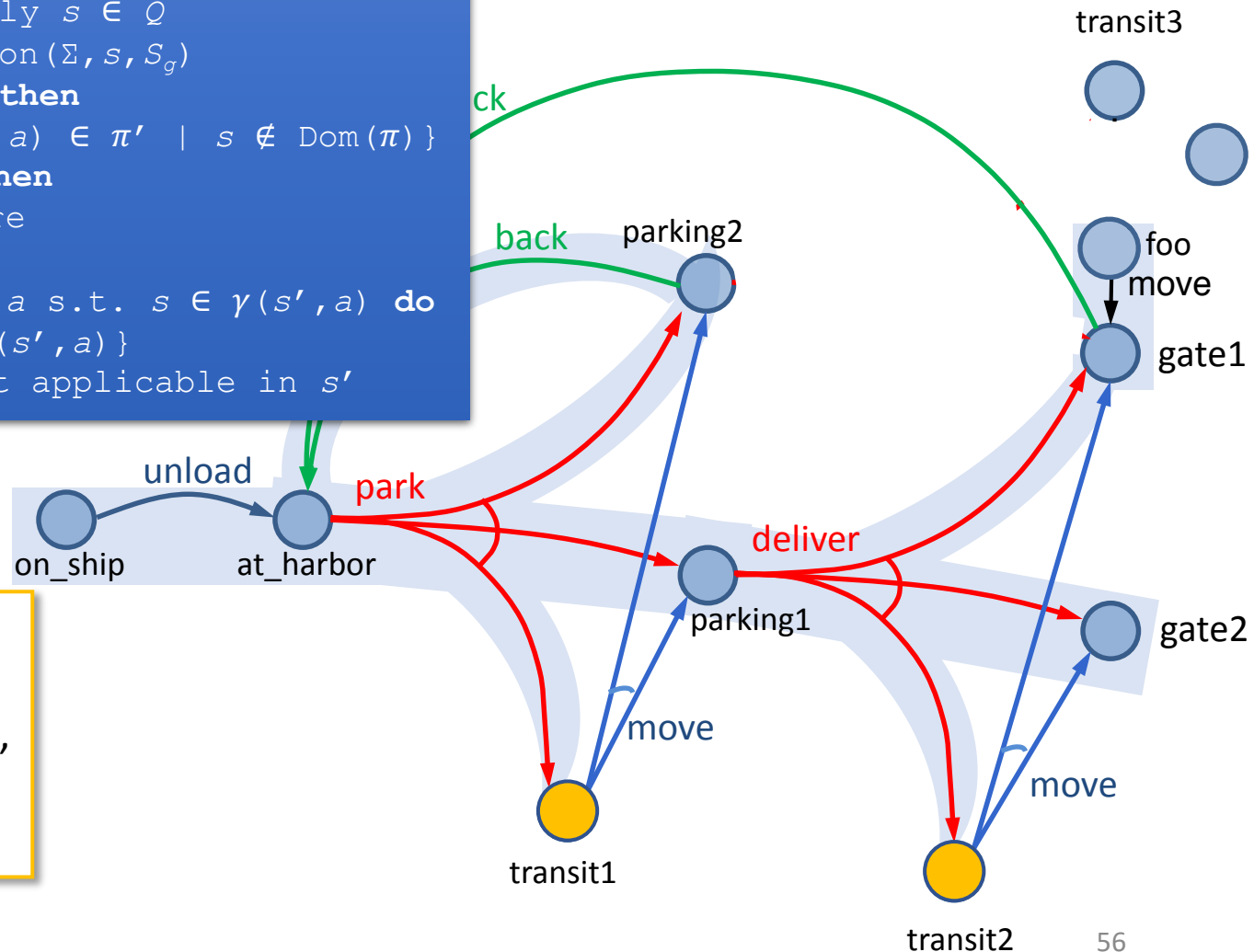
**for** every  $s', a$  s.t.  $s \in \gamma(s', a)$  **do**

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{foo}, \text{move}),$   
 $(\text{parking2}, \text{back})\}$

## Example







## Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

...

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

**if**  $Q = \emptyset$  **then**

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\nu}(s_0, \pi)\}$

**return**  $\pi$

select arbitrarily  $s \in Q$

$\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$

**if**  $\pi' \neq \text{failure}$  **then**

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

**else if**  $s = s_0$  **then**

**return** failure

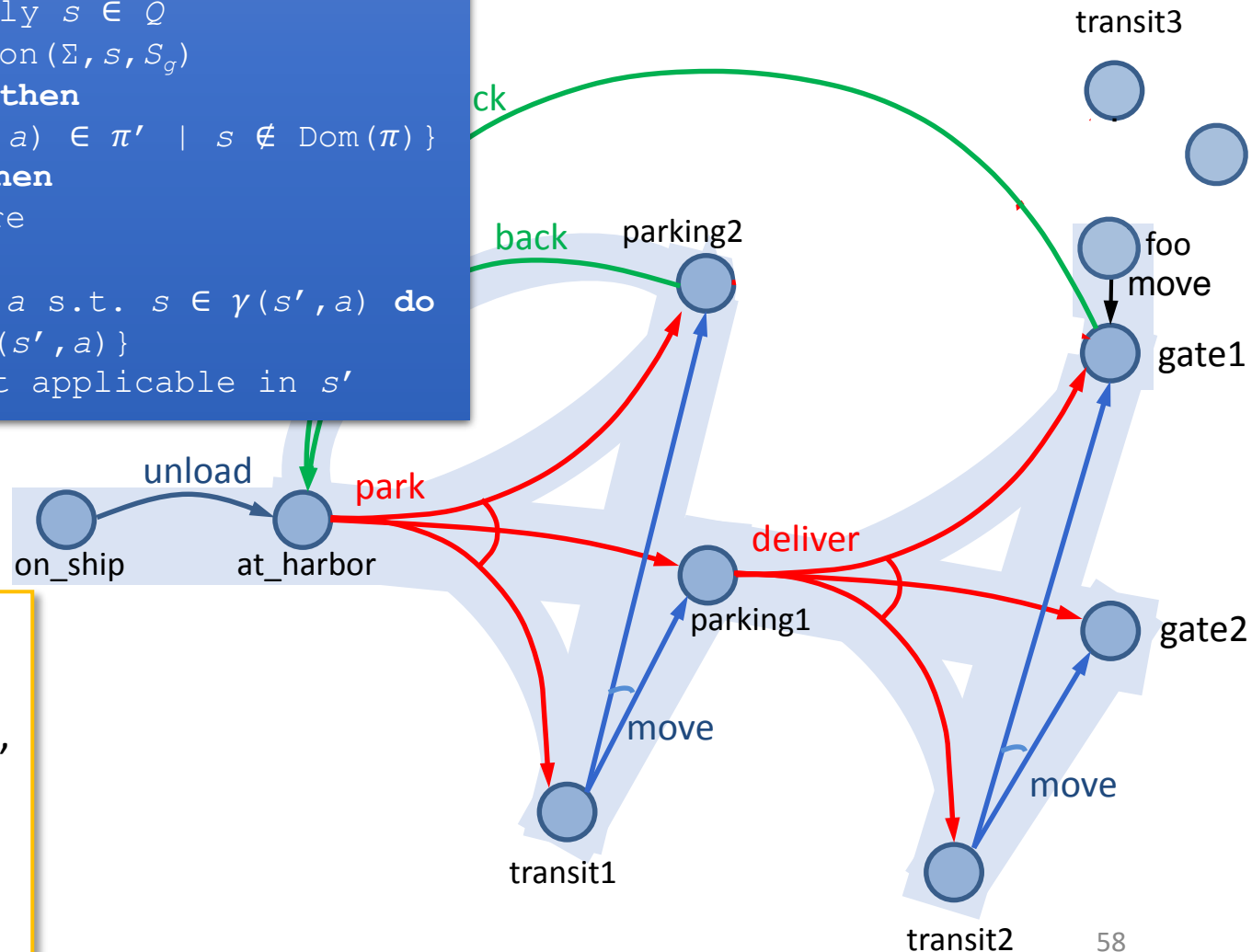
**else**

**for** every  $s', a$  s.t.  $s \in \gamma(s', a)$  **do**

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make  $a$  not applicable in  $s'$

# Example



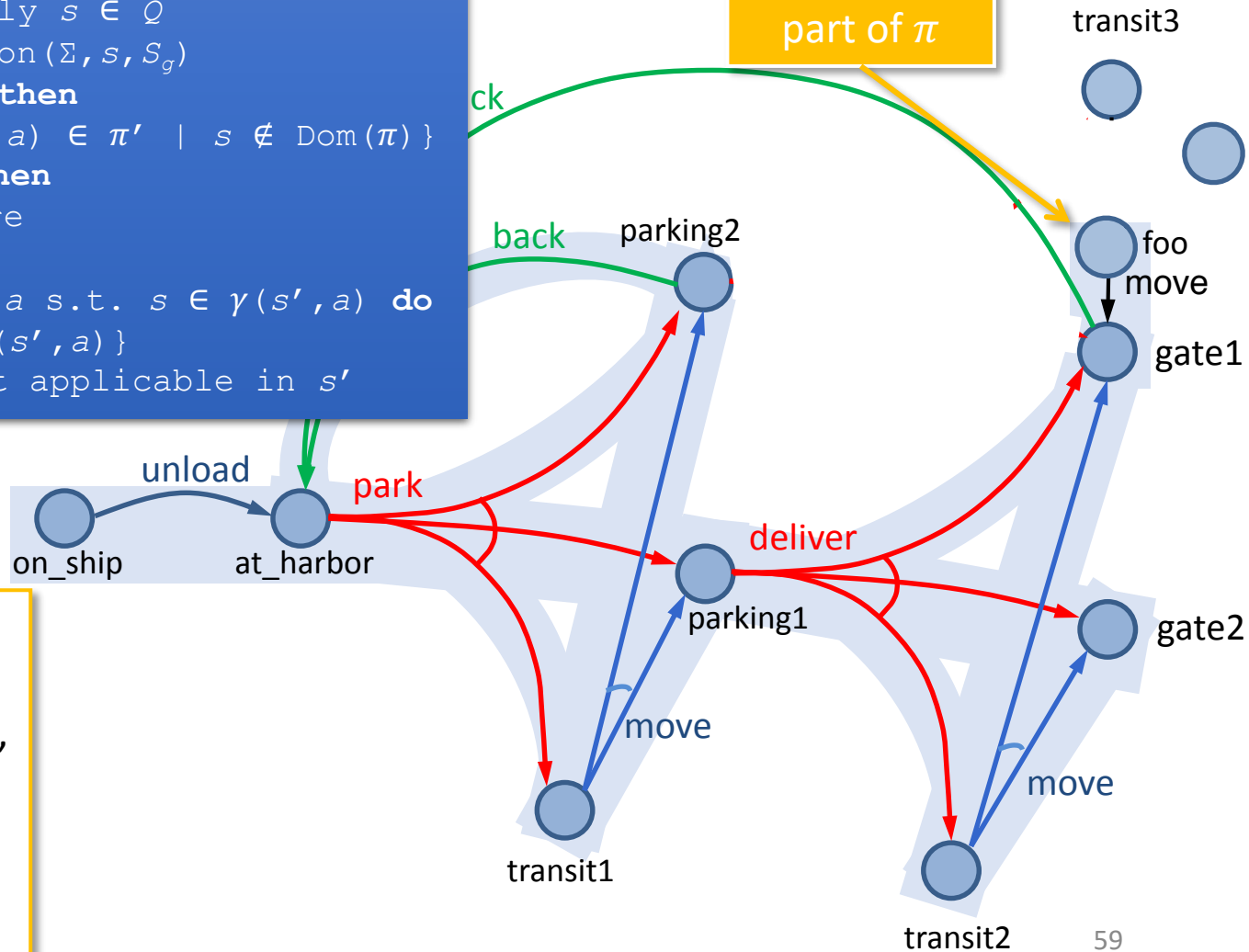
$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{foo}, \text{move}),$   
 $(\text{parking2}, \text{back}),$   
 $(\text{transit1}, \text{move}),$   
 $(\text{transit2}, \text{move})\}$

# Guided-Find-Safe-Solution ( $\Sigma, s_0, S_g$ )

```
...  
loop  
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$   
  if  $Q = \emptyset$  then  
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{V}(s_0, \pi)\}$   
    return  $\pi$   
  select arbitrarily  $s \in Q$   
   $\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$   
  if  $\pi' \neq \text{failure}$  then  
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$   
  else if  $s = s_0$  then  
    return failure  
  else  
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do  
       $\pi \leftarrow \pi \setminus \{(s', a)\}$   
      make  $a$  not applicable in  $s'$ 
```

# Example

Remove unreachable part of  $\pi$



$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 ~~$(\text{foo}, \text{move}),$~~   
 $(\text{parking2}, \text{back}),$   
 $(\text{transit1}, \text{move}),$   
 $(\text{transit2}, \text{move})\}$

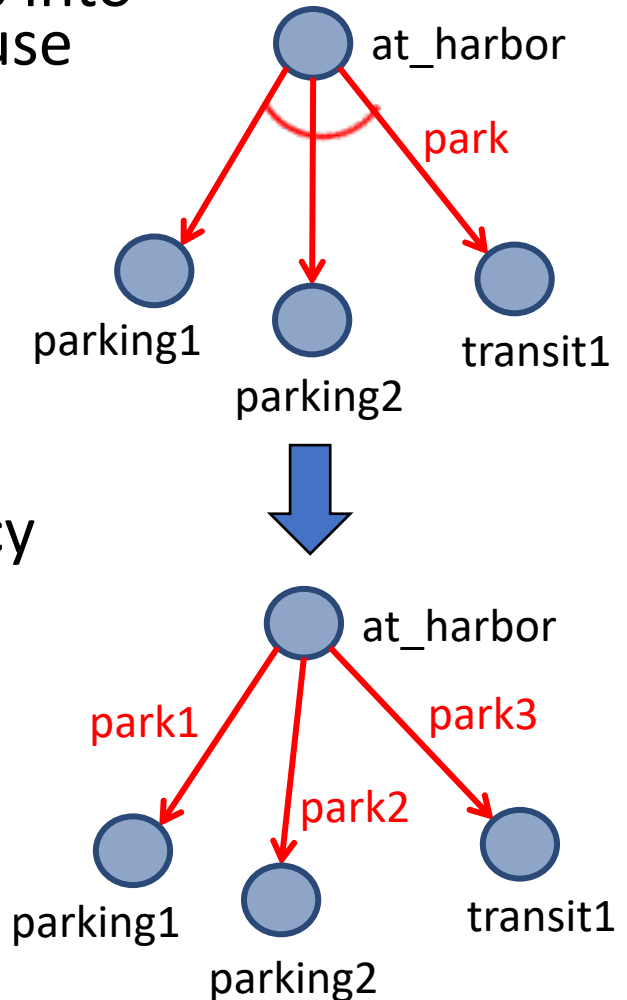
# Determinisation

```
Guided-Find-Safe-Solution( $\Sigma, s_0, S_g$ )
  if  $s_0 \in S_g$  then
    return  $\emptyset$ 
  if Applicable( $s_0$ ) =  $\emptyset$  then
    return failure
   $\pi \leftarrow \emptyset$ 
  loop
     $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
    if  $Q = \emptyset$  then
       $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
      return  $\pi$ 
    select arbitrarily  $s \in Q$ 
     $\pi' \leftarrow \text{Find-Solution}(\Sigma, s, S_g)$ 
    if  $\pi' \neq \text{failure}$  then
       $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
    else if  $s = s_0$  then
      return failure
    else
      for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
         $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
        make  $a$  not applicable in  $s'$ 
```

- How to implement it?
  - Need implementation of Find-Solution
  - Need it to be very efficient
    - Called many times
- Idea: instead, use a classical planner
  - Any algorithm from Chapter 2
  - Efficient algorithms, search heuristics
- For that, determinise actions

# Determinisation

- Convert the nondeterministic actions into something the classical planner can use
- **Determinise**
  - Suppose  $a_i$  has  $k$  possible outcomes
  - $k$  deterministic actions, one for each outcome
- Classical planner returns a plan  $p = \langle a_1, a_2, \dots, a_n \rangle$
- If  $p$  is acyclic, can convert it to a policy
  - (unsafe) solution for  $P$
  - $\{(s_0, \mathbf{a}_1), (s_1, \mathbf{a}_2), \dots, (s_{n-1}, \mathbf{a}_n)\}$  where
  - each  $\mathbf{a}_i$  is the nondeterministic action whose determinisation includes  $a_i$
  - $s_i \in \gamma(s_{i-1}, \mathbf{a}_i)$



# Determinisation

- Nondeterministic planning problem  $P = (\Sigma, s_0, S_g)$
- Determinisation  $P_d = (\Sigma_d, s_0, S_g)$
- Classical planner returns a solution for  $P_d$ 
  - a plan  $p = \langle a_1, a_2, \dots, a_n \rangle$
- If  $p$  is acyclic, can convert it to an (unsafe) solution for  $P$ 
  - $\{(s_0, \mathbf{a}_1), (s_1, \mathbf{a}_2), \dots, (s_{n-1}, \mathbf{a}_n)\}$where
  - each  $\mathbf{a}_i$  is the nondeterministic action whose determinisation includes  $a_i$
  - each  $s_i \in \gamma(s_{i-1}, \mathbf{a}_i)$

```
Plan2policy(p= $\langle a_1, \dots, a_n \rangle, s$ )  
   $\pi \leftarrow \emptyset$   
  for  $i$  from 1 to  $n$  do  
     $\pi \leftarrow \pi \cup \{s, \text{det2nondet}(a_i)\}$   
     $s \leftarrow \gamma_d(s, a_i)$   
  return  $\pi$ 
```

# Guided-Find-Safe-Solution

Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

if  $s_0 \in S_g$  then

return  $\emptyset$

if  $\text{Applicable}(s_0) = \emptyset$  then

return failure

$\pi \leftarrow \emptyset$

$\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$

loop

$Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$

if  $Q = \emptyset$  then

$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$

return  $\pi$

select arbitrarily  $s \in Q$

$p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$

if  $p' \neq \text{fail}$  then

$\pi \leftarrow \text{Plan2policy}(p', s)$

$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$

else if  $s = s_0$  then

return failure

else

for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do

$\pi \leftarrow \pi \setminus \{(s', a)\}$

make the actions in the  
determinisation not  
applicable in  $s'$

Same as Guided-Find-Safe-Solution.

Any classical planner that does not return cyclic plans.

Convert  $p'$  to a policy. Add each  $(s, a)$  to  $\pi$  unless  $\pi$  already has an action for  $s$ .

$s$  is unsolvable. For each  $(s', a)$  that can produce  $s$ , modify  $\pi$  and  $\Sigma_d$  such that we will never use  $a$  at  $s'$ .

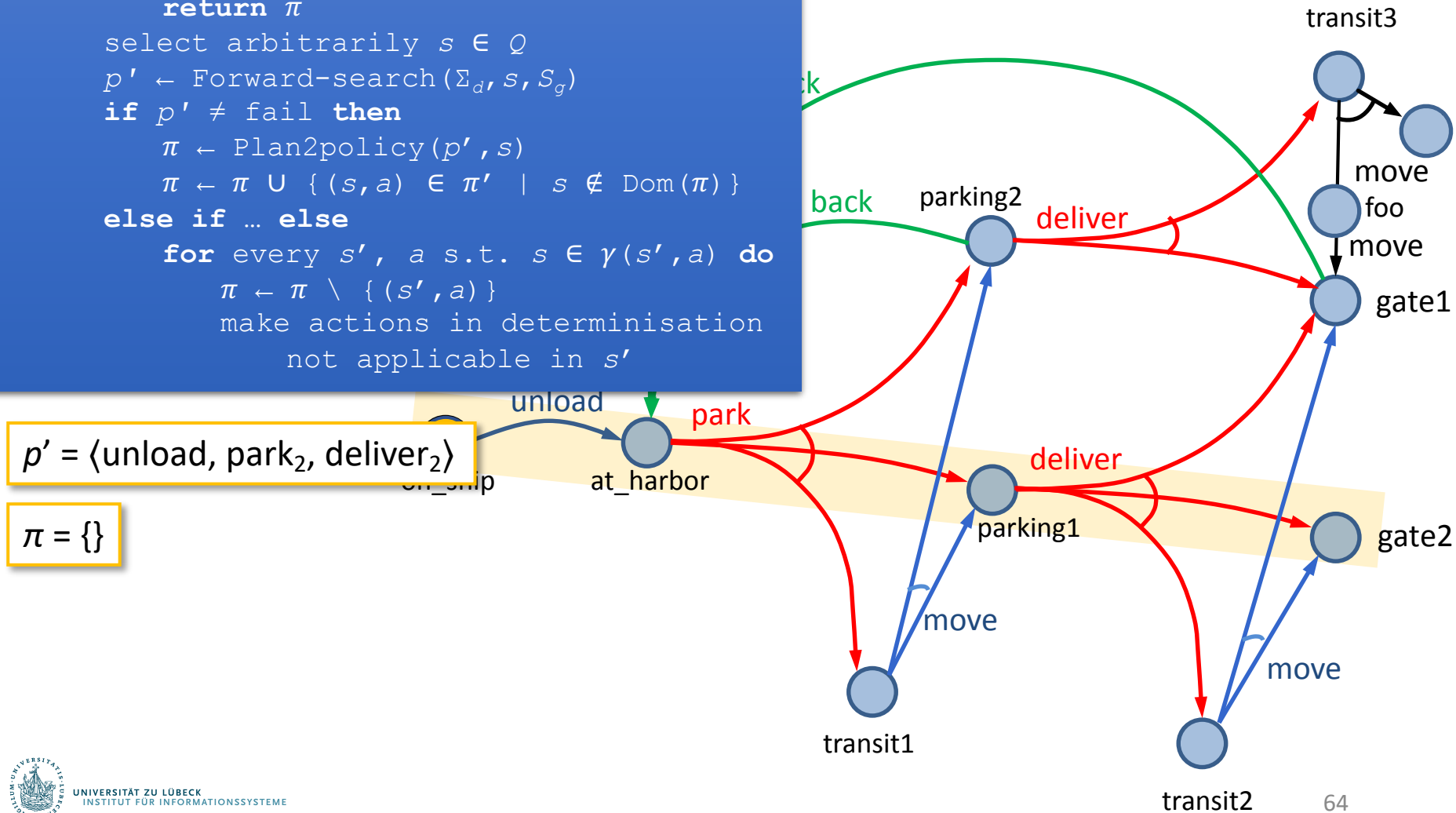
# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example



$p' = \langle \text{unload}, \text{park}_2, \text{deliver}_2 \rangle$

$\pi = \{\}$





# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

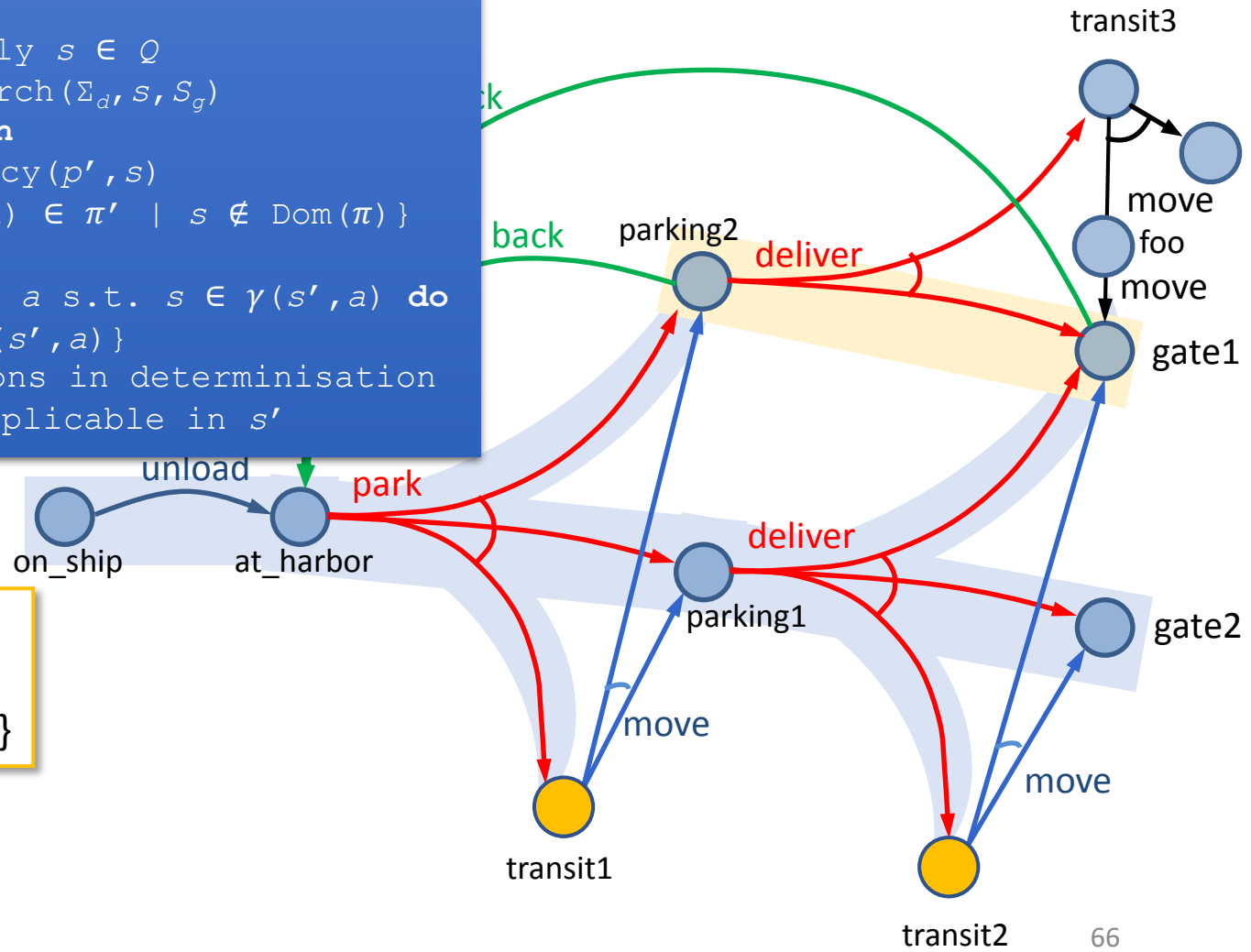
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example

$p' = \langle \text{deliver}_2 \rangle$

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



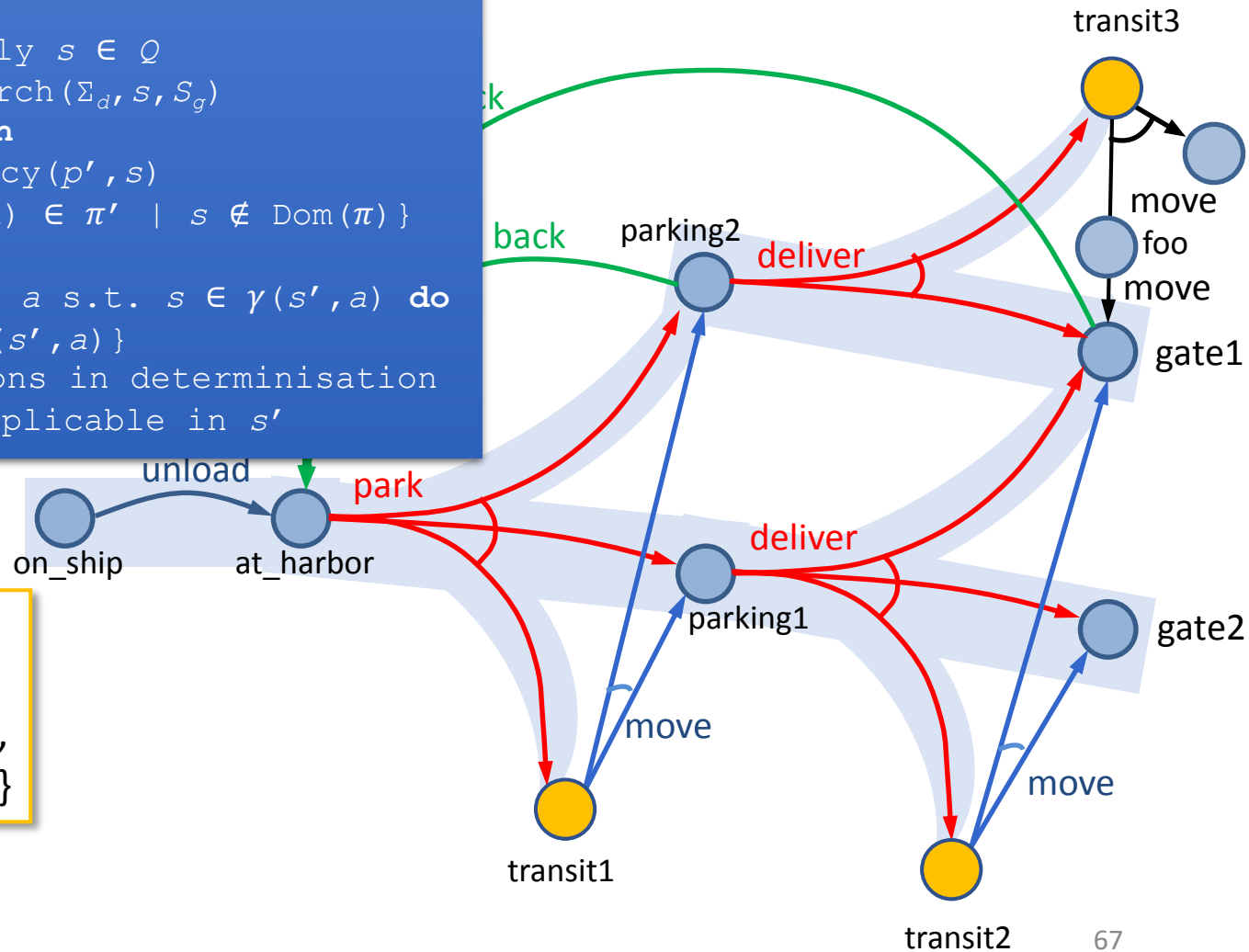
# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```
...  
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$   
loop  
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$   
  if  $Q = \emptyset$  then  
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$   
    return  $\pi$   
  select arbitrarily  $s \in Q$   
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$   
  if  $p' \neq \text{fail}$  then  
     $\pi \leftarrow \text{Plan2policy}(p', s)$   
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$   
  else if ... else  
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do  
       $\pi \leftarrow \pi \setminus \{(s', a)\}$   
      make actions in determinisation  
      not applicable in  $s'$ 
```

$p' = \langle \text{deliver}_2 \rangle$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver})\}$

# Example



# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

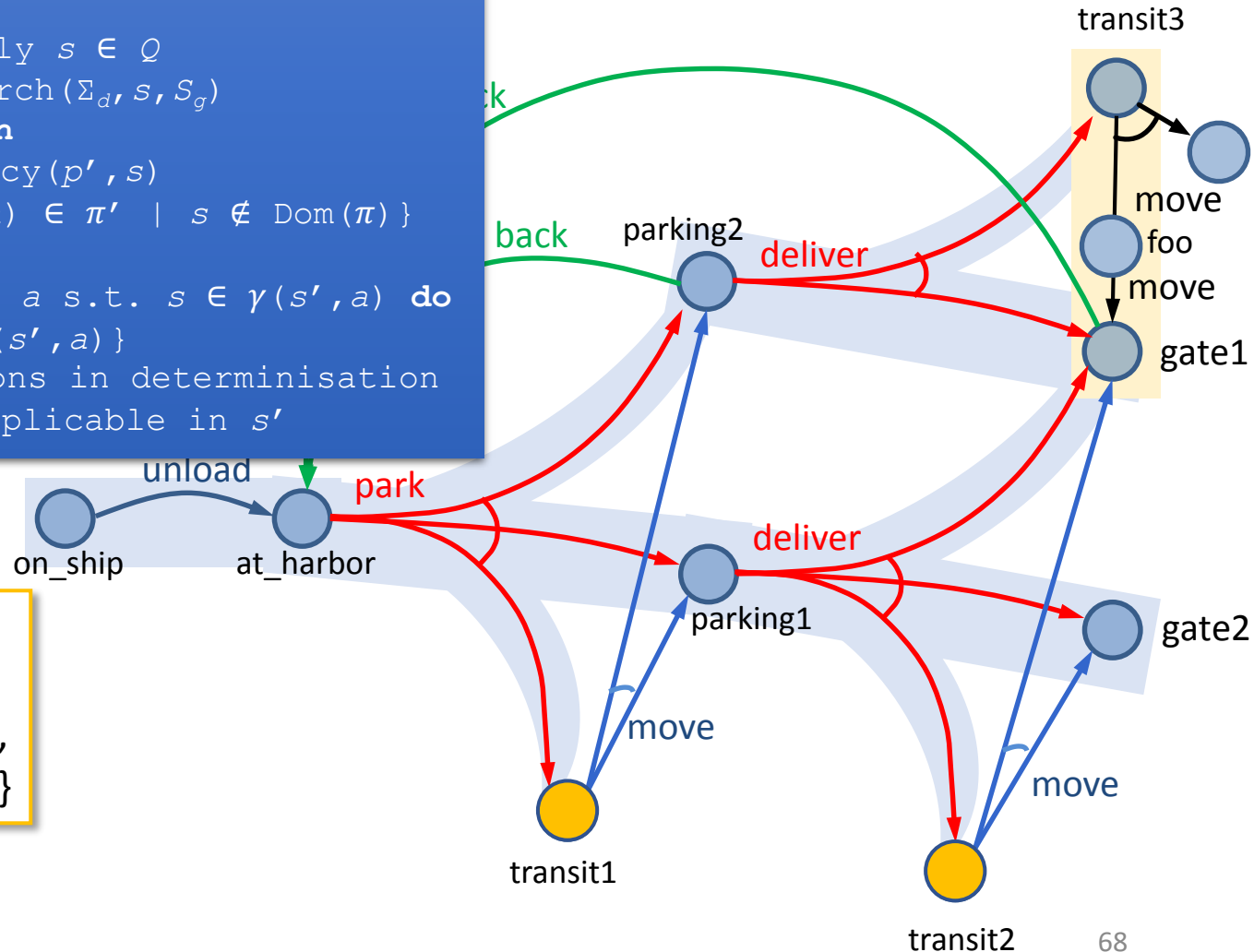
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example

$p' = \langle \text{move}_2, \text{move} \rangle$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver})\}$



# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

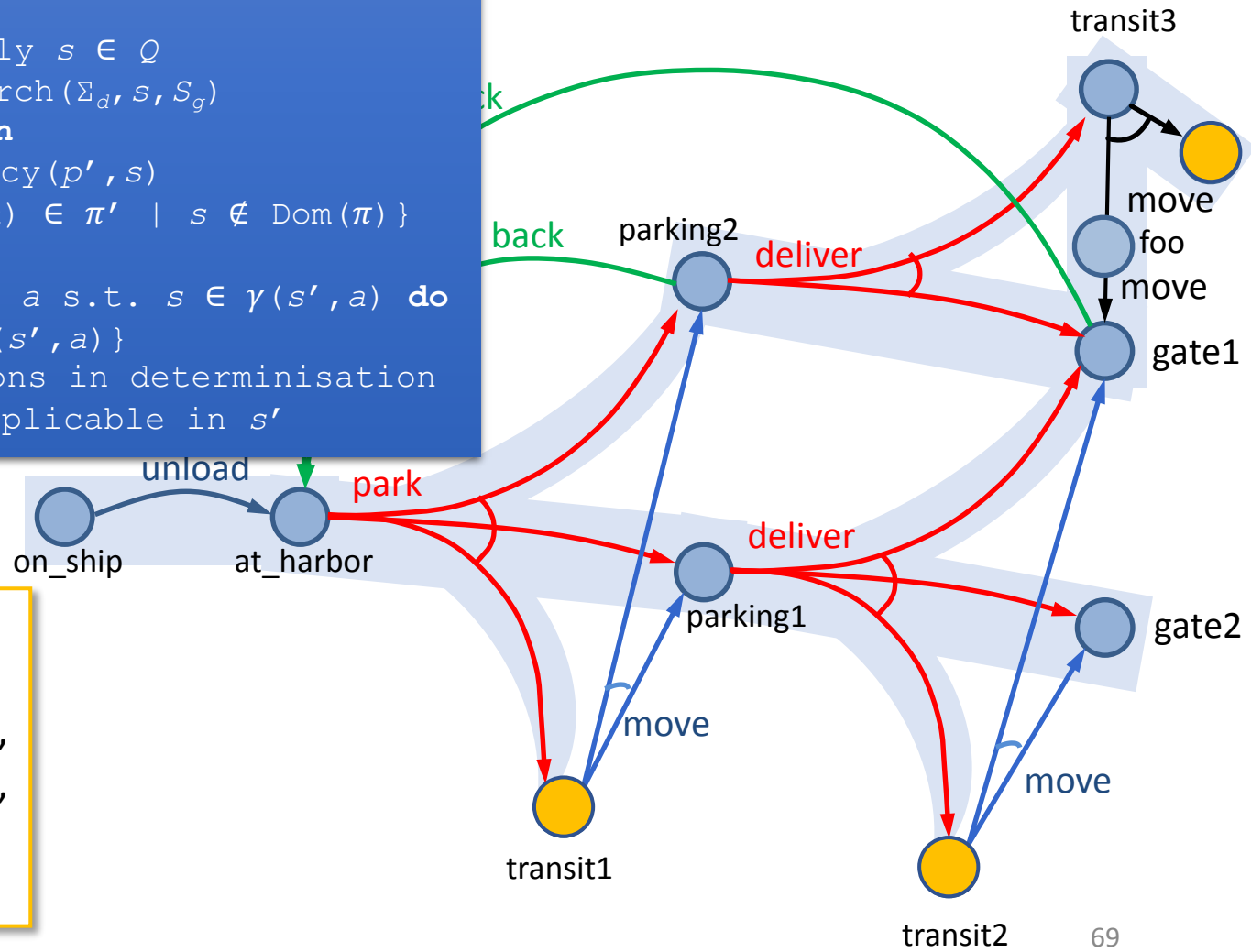
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example

$p' = \langle \text{move}_2, \text{move} \rangle$

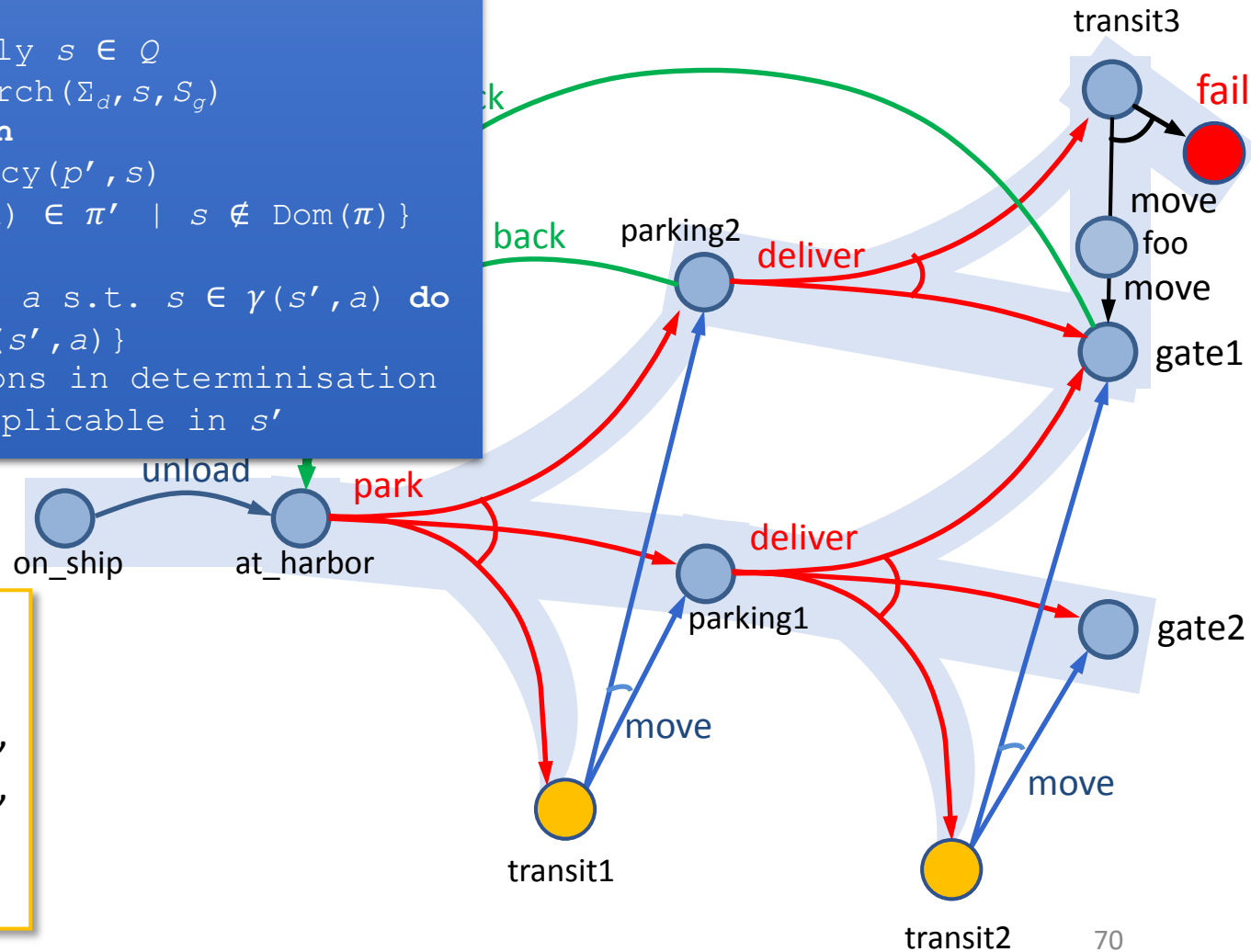
$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 $(\text{transit3}, \text{move}),$   
 $(\text{foo}, \text{move})\}$



# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 
```

# Example



$p' = \text{fail}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 $(\text{transit3}, \text{move}),$   
 $(\text{foo}, \text{move})\}$

# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

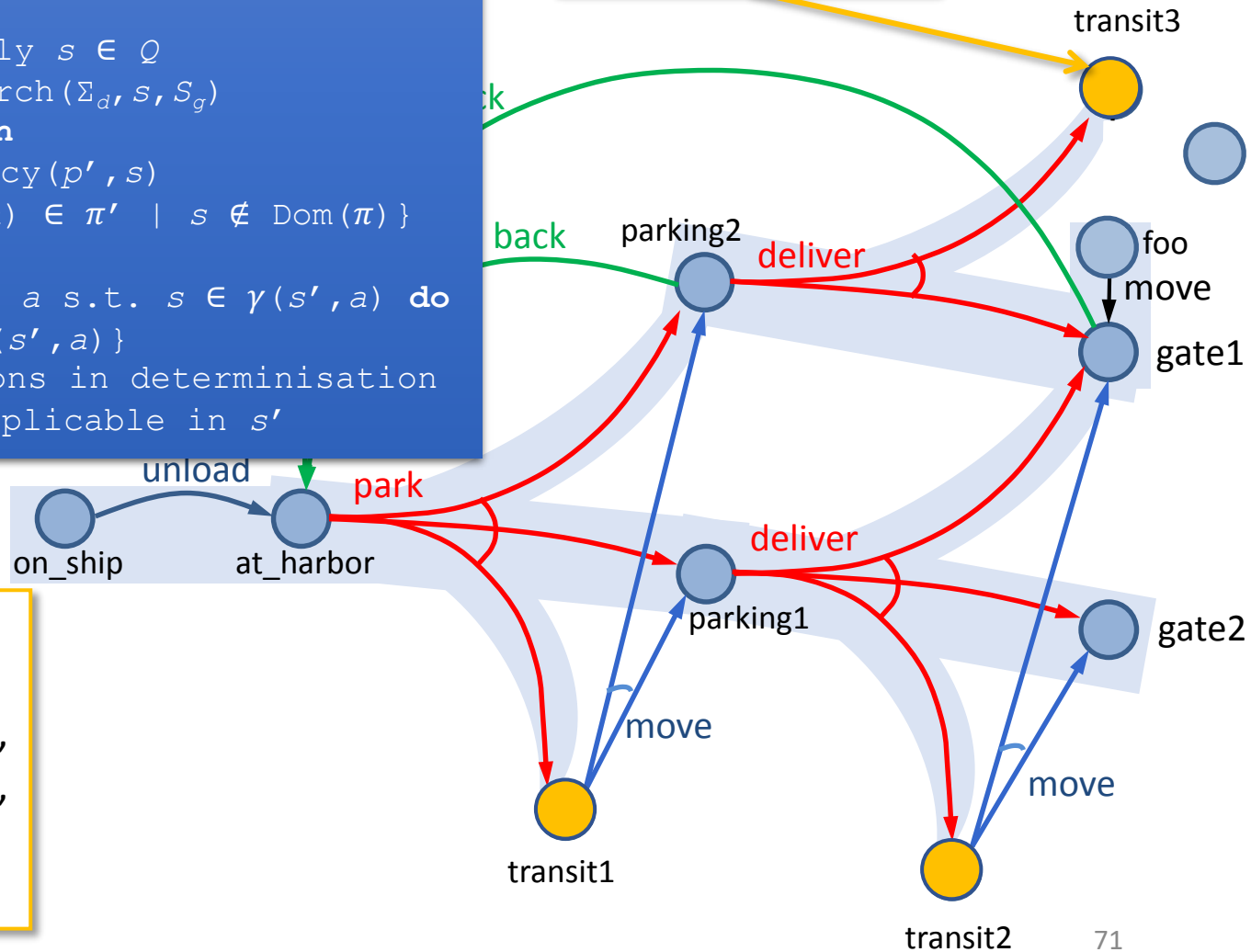
```

...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example

Modify  $\Sigma_d$  to make move inapplicable



$p' = \text{fail}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 ~~$(\text{transit3}, \text{move}),$~~   
 $(\text{foo}, \text{move})\}$

# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

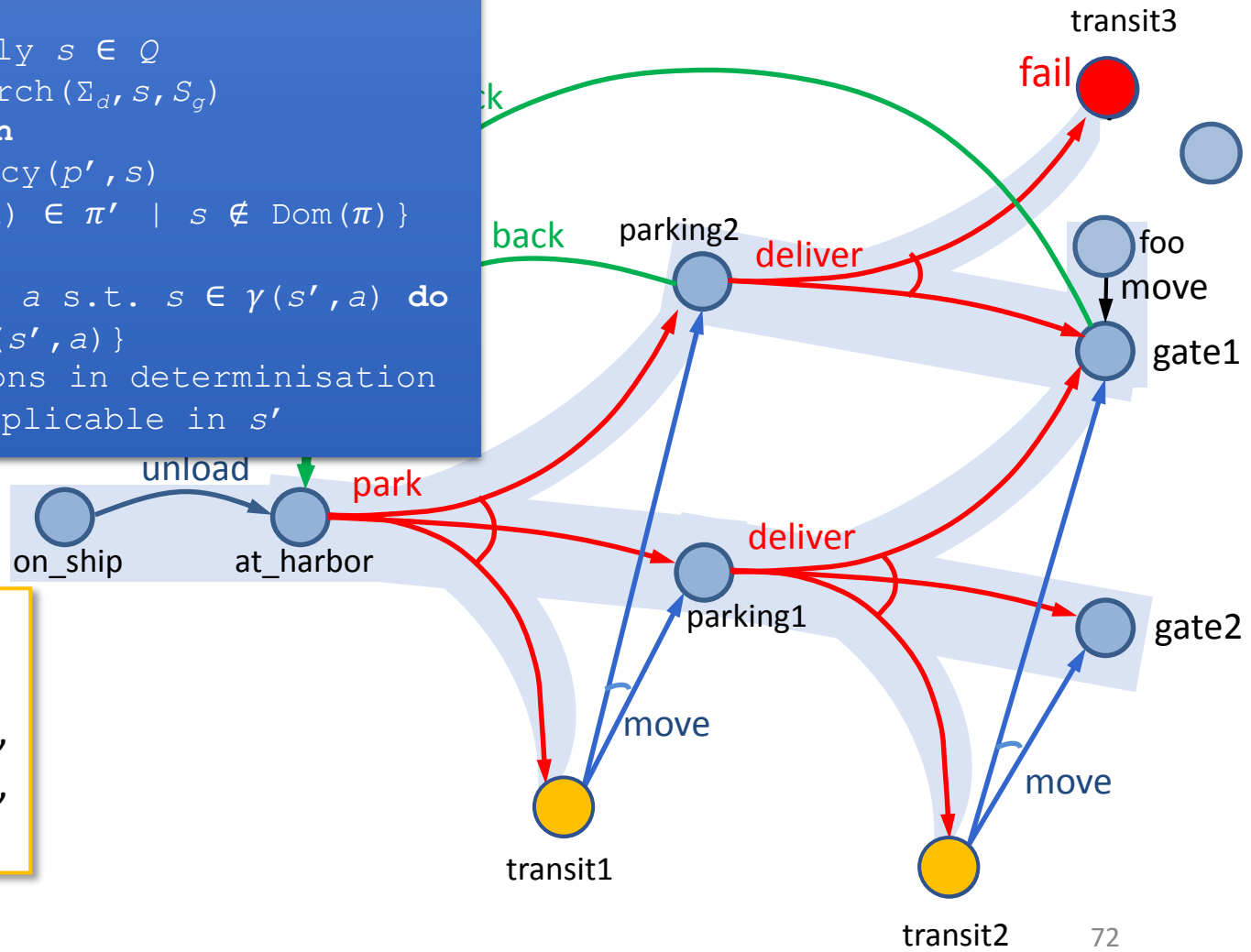
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example

$p' = \text{fail}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{parking2}, \text{deliver}),$   
 $(\text{foo}, \text{move})\}$





# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

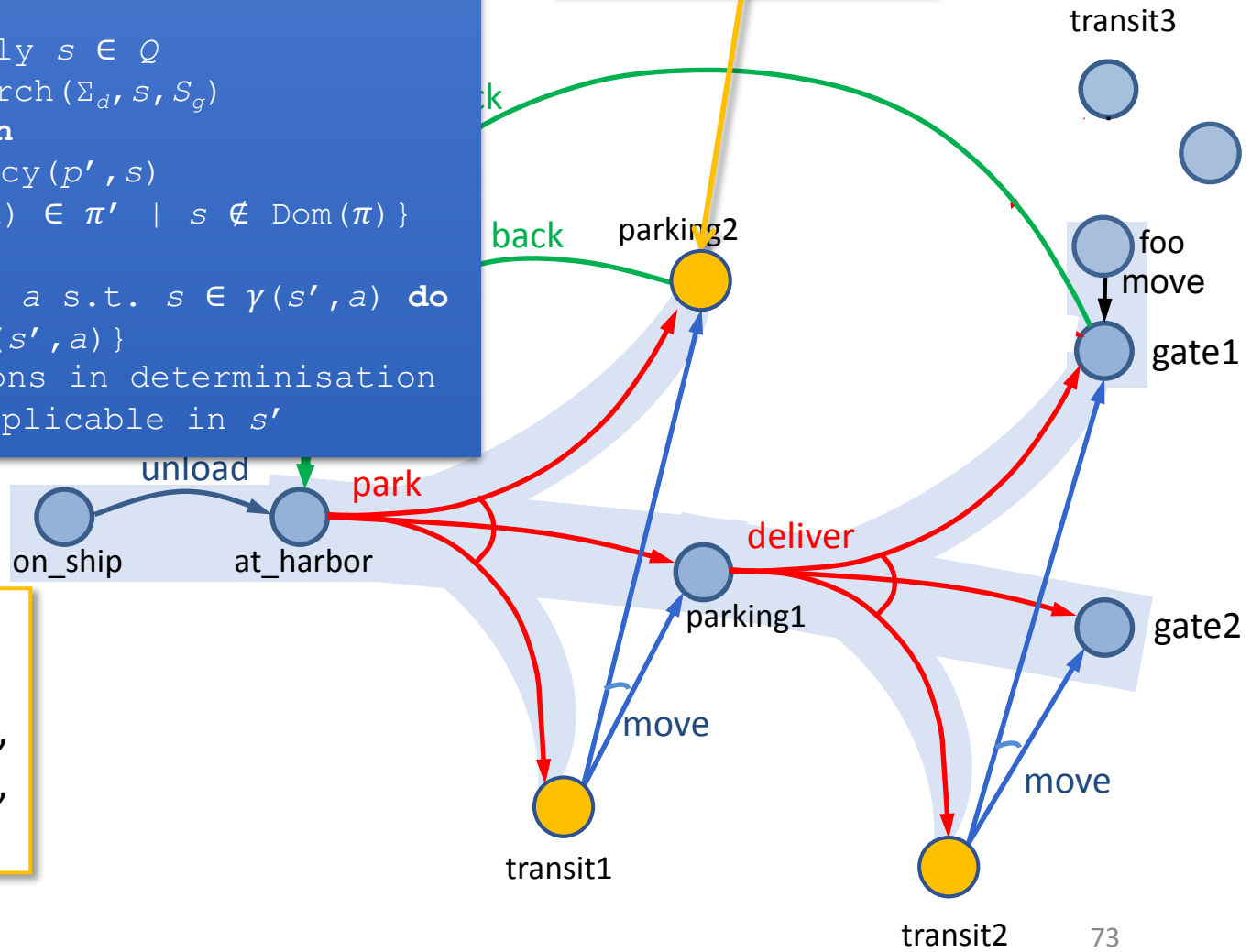
```

# Example

Modify  $\Sigma_d$  to make move inapplicable

$p' = \text{fail}$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 ~~$(\text{parking2}, \text{deliver}),$~~   
 $(\text{foo}, \text{move})\}$







# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```

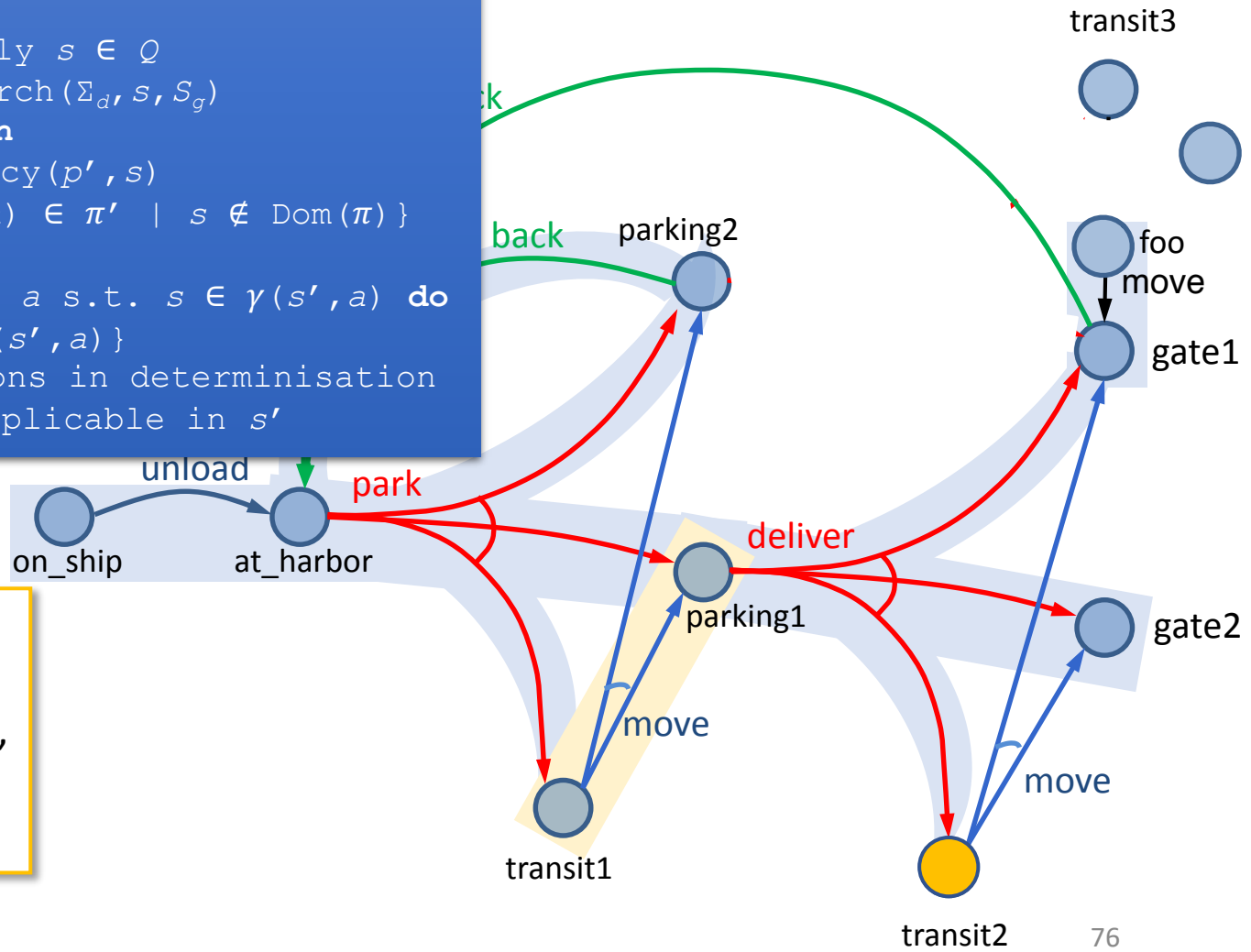
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 

```

# Example

$p' = \langle \text{move}_2, \text{deliver}_1 \rangle$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking}_1, \text{deliver}),$   
 $(\text{foo}, \text{move}),$   
 $(\text{parking}_2, \text{back})\}$



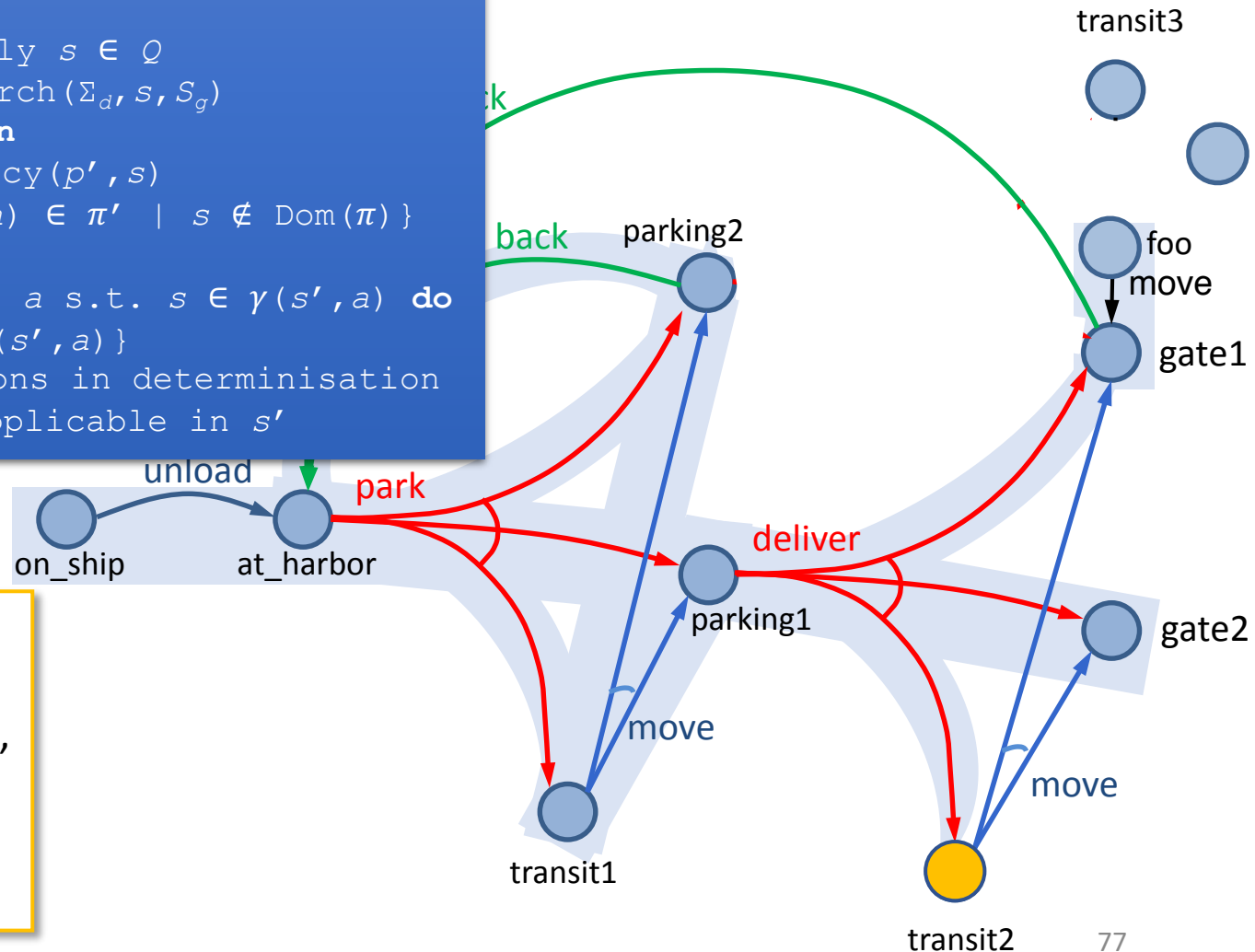
# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```
...  
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$   
loop  
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$   
  if  $Q = \emptyset$  then  
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$   
    return  $\pi$   
  select arbitrarily  $s \in Q$   
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$   
  if  $p' \neq \text{fail}$  then  
     $\pi \leftarrow \text{Plan2policy}(p', s)$   
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$   
  else if ... else  
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do  
       $\pi \leftarrow \pi \setminus \{(s', a)\}$   
      make actions in determinisation  
      not applicable in  $s'$ 
```

$p' = \langle \text{move}_2, \text{deliver}_1 \rangle$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking}_1, \text{deliver}),$   
 $(\text{foo}, \text{move}),$   
 $(\text{parking}_2, \text{back}),$   
 $(\text{transit}_1, \text{move})\}$

# Example



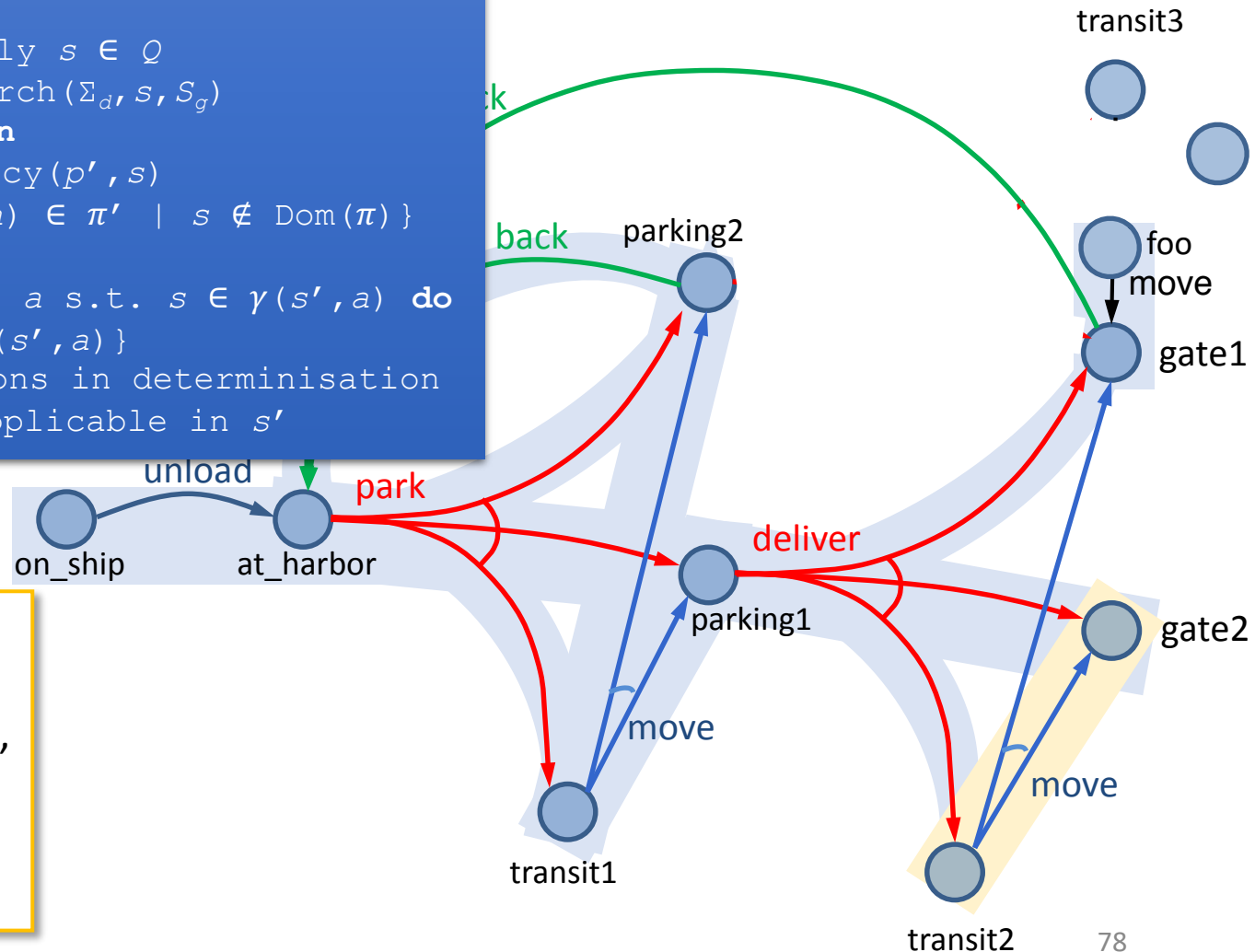
# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```
...
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
loop
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
  if  $Q = \emptyset$  then
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
    return  $\pi$ 
  select arbitrarily  $s \in Q$ 
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
  if  $p' \neq \text{fail}$  then
     $\pi \leftarrow \text{Plan2policy}(p', s)$ 
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
  else if ... else
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
       $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
      make actions in determinisation
      not applicable in  $s'$ 
```

# Example

$p' = \langle \text{move}_2 \rangle$

$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 $(\text{foo}, \text{move}),$   
 $(\text{parking2}, \text{back}),$   
 $(\text{transit1}, \text{move})\}$



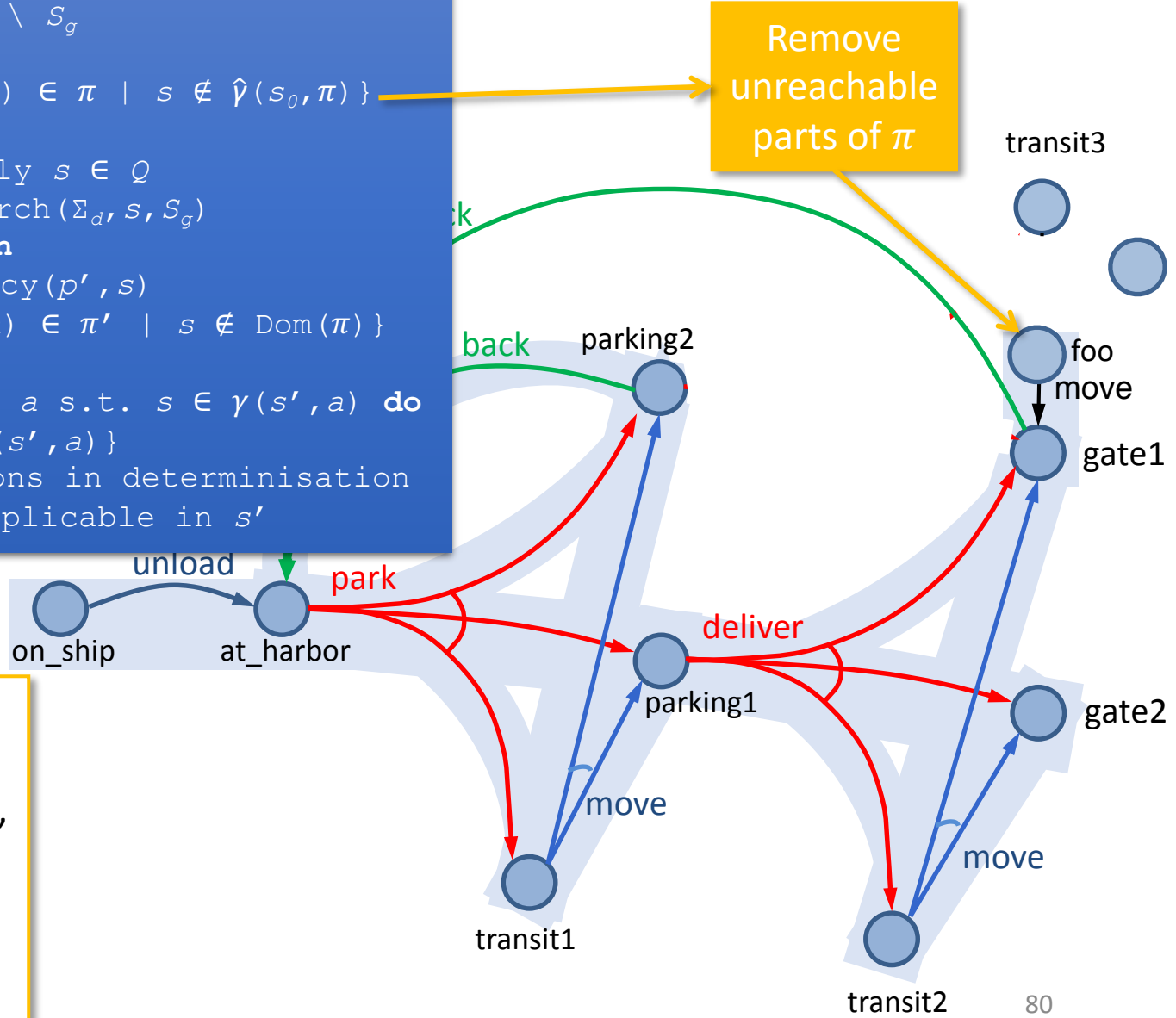


# Find-Safe-Solution-by-Determinisation ( $\Sigma, s_0, S_g$ )

```
...  
 $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$   
loop  
   $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$   
  if  $Q = \emptyset$  then  
     $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$   
    return  $\pi$   
  select arbitrarily  $s \in Q$   
   $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$   
  if  $p' \neq \text{fail}$  then  
     $\pi \leftarrow \text{Plan2policy}(p', s)$   
     $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$   
  else if ... else  
    for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do  
       $\pi \leftarrow \pi \setminus \{(s', a)\}$   
      make actions in determinisation  
      not applicable in  $s'$ 
```

# Example

Remove unreachable parts of  $\pi$



$\pi = \{(\text{on\_ship}, \text{unload}),$   
 $(\text{at\_harbor}, \text{park}),$   
 $(\text{parking1}, \text{deliver}),$   
 ~~$(\text{foo}, \text{move}),$~~   
 $(\text{parking2}, \text{back}),$   
 $(\text{transit1}, \text{move}),$   
 $(\text{transit2}, \text{move})\}$



# Making Actions Inapplicable

```
Find-Safe-Solution-by-Determinisation( $\Sigma, s_0, S_g$ )
  if  $s_0 \in S_g$  then
    return  $\emptyset$ 
  if  $\text{Applicable}(s_0) = \emptyset$  then
    return failure
   $\pi \leftarrow \emptyset$ 
   $\Sigma_d \leftarrow \text{mk-deterministic}(\Sigma)$ 
  loop
     $Q \leftarrow \text{leaves}(s_0, \pi) \setminus S_g$ 
    if  $Q = \emptyset$  then
       $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \hat{\gamma}(s_0, \pi)\}$ 
      return  $\pi$ 
    select arbitrarily  $s \in Q$ 
     $p' \leftarrow \text{Forward-search}(\Sigma_d, s, S_g)$ 
    if  $p' \neq \text{fail}$  then
       $\pi \leftarrow \text{Plan2policy}(p', s)$ 
       $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$ 
    else if  $s = s_0$  then
      return failure
    else
      for every  $s', a$  s.t.  $s \in \gamma(s', a)$  do
         $\pi \leftarrow \pi \setminus \{(s', a)\}$ 
        make the actions in the
          determinisation not
          applicable in  $s'$ 
```

- Modify  $\Sigma_d$  to make actions inapplicable: Exponential time in worst-case
- Better: table of bad state-action pairs
  - For every  $(s', a)$  s.t.  $s \in \gamma(s', a)$ ,  
$$\text{Bad}[s'] \leftarrow \text{Bad}[s'] \cup \text{determinization}(a)$$
- Modify classical planner to take the table as an argument
  - If  $s$  is current state, only choose actions in  $\text{Applicable}(s) \setminus \text{Bad}(s)$

# Intermediate Summary

---

- Determinisation Techniques
  - Guided-find-safe-solution
    - Call find-solution to get an unsafe solution
    - Call find-solution additional times on the leaves
  - Find-safe-solution-by-determinization
    - Use determinized actions
    - Call classical planner rather than find-solution
    - If dead-ends are encountered, modify actions that lead to them

# Outline per the Book

---

## *5.2 Planning Problem*

- Planning domains
- Plans as policies
- Planning problems and solutions

## *5.3 And/Or Graph Search*

- Planning by forward search

## *5.5 Determinisation Techniques*

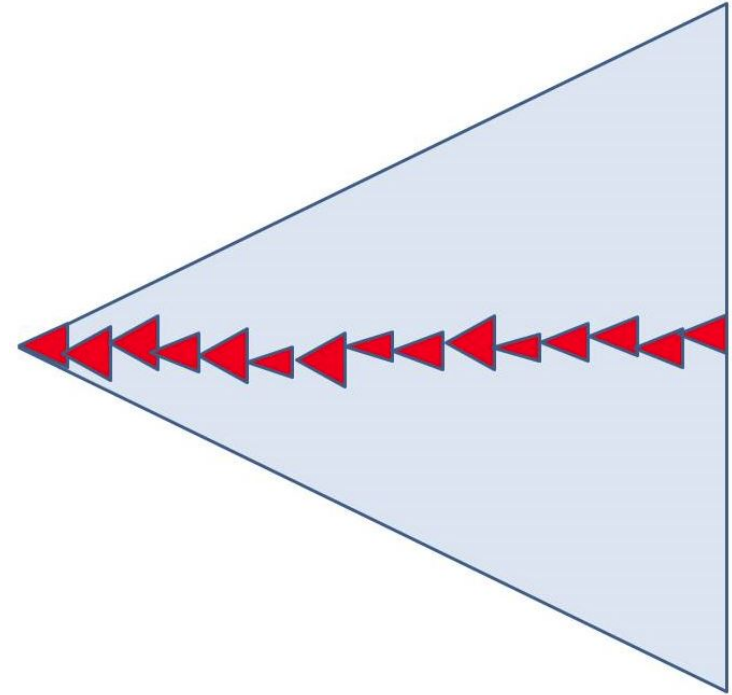
- Guided planning for safe solutions
- Planning for safe solutions by determinisation

## ***5.6 Online Approaches***

- Lookahead
- Lookahead by determinisation
- Lookahead with a bounded number of steps

# Online Approaches

- Motivation
  - Planning models are approximate – execution seldom works out as planned
  - Large problems may require too much planning time
- 2<sup>nd</sup> motivation even more stronger in nondeterministic domains
  - Nondeterminism makes planning exponentially harder
    - Exponentially more time, exponentially larger policies



Offline vs. Runtime  
Search Spaces

# Online Approaches

---

- Need to identify **good** actions without exploring entire search space
  - Can be done using heuristic estimates
- Some domains are **safely explorable**
  - Safe to create partial plans, because goal states are reachable from all situations
- Other domains contain dead-ends, partial planning will not guarantee success
  - Can get trapped in dead ends that we would have detected if we had planned fully
    - No applicable actions
      - Robot goes down a steep incline and can't come back up
    - Applicable actions, but caught in a loop
      - Robot goes into a collection of rooms from which there's no exit
  - However, partial planning can still make success more likely

# Lookahead-Partial-Plan

- Adaptation of Run-Lazy-Lookahead (Ch. 2)
- Lookahead is any planning algorithm that returns a policy  $\pi$ 
  - $\pi$  may be partial solution, or unsafe solution
  - Lookahead-Partial-Plan executes  $\pi$  as far as it will go, then calls Lookahead again
  - $\theta$  context-dependent vector of parameters to restrict in some way the search for a solution

```
Lookahead-Partial-Plan( $\Sigma, s_0, S_g$ )
```

```
 $s \leftarrow s_0$   
while  $s \notin S_g$  and  $\text{Applicable}(s) \neq \emptyset$  do  
   $\pi \leftarrow \text{Lookahead}(s, \theta)$   
  if  $\pi = \emptyset$  then  
    return failure  
  else  
    perform partial plan  $\pi$   
     $s \leftarrow$  observe current state
```

# FS-Replan

- Adaptation of Run-Lookahead (Ch. 2)
- Calls Forward-Search (Ch. 2) on determinized domain, converts to a policy
  - Unsafe solution
- Generalization:
  - Lookahead can be any planning algorithm that returns a policy  $\pi$

```
FS-Replan( $\Sigma, s, S_g$ )
```

```
 $\pi_d \leftarrow \emptyset$ 
```

```
while  $s \notin S_g$  and Applicable( $s$ )  $\neq \emptyset$  do
```

```
  if  $\pi_d$  undefined for  $s$  then
```

```
     $\pi_d \leftarrow$  Plan2policy(Forward-search( $\Sigma_d, s, S_g$ ),  $s$ )
```

```
    if  $\pi_d =$  failure then
```

```
      return failure
```

```
  perform action  $\pi_d(s)$ 
```

```
   $s \leftarrow$  observe resulting state
```

```
Generalised-FS-Replan( $\Sigma, s, S_g$ )
```

```
 $\pi_d \leftarrow \emptyset$ 
```

```
while  $s \notin S_g$  and Applicable( $s$ )  $\neq \emptyset$  do
```

```
  if  $\pi_d$  undefined for  $s$  then
```

```
     $\pi_d \leftarrow$  Lookahead( $s, \theta$ )
```

```
    if  $\pi_d =$  failure then
```

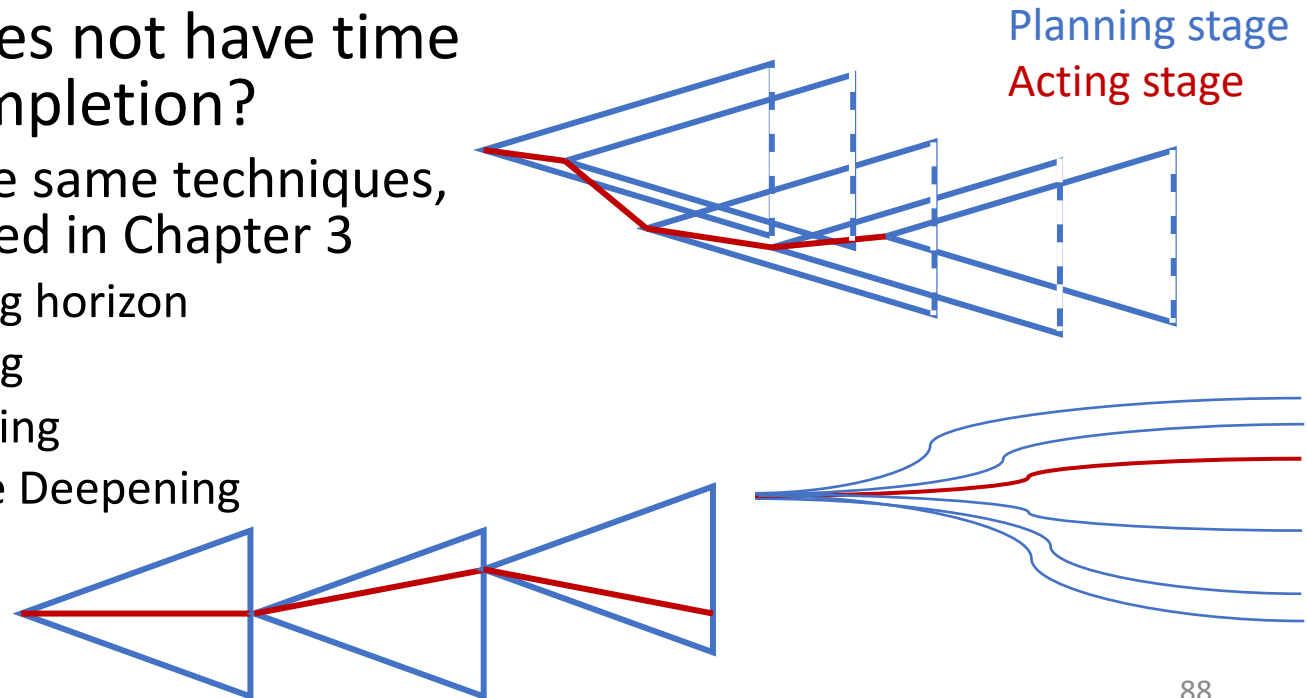
```
      return failure
```

```
  perform action  $\pi_d(s)$ 
```

```
   $s \leftarrow$  observe resulting state
```

# Possibilities for Lookahead

- Lookahead could be one of the algorithms we discussed earlier
  - Find-Safe-Solution
  - Find-Acyclic-Solution
  - Guided-Find-Safe-Solution
  - Find-Safe-Solution-by-Determinization
- What if it does not have time to run to completion?
  - Can use the same techniques, we discussed in Chapter 3
    - Receding horizon
    - Sampling
    - Subgoaling
    - Iterative Deepening





# Possibilities for Lookahead (cont'd)

- Full horizon, limited breadth:
  - Look for solution that works for *some* of the outcomes
  - E.g., modify Find-Acyclic-Solution to examine  $i$  outcomes of every action
- Iterative broadening:
  - For  $i = 1$ , increase  $i$  by 1 until time runs out
    - Look for a solution that handles  $i$  outcomes per action

$T \leftarrow i$  elements  
of  $\gamma(s, a) \setminus \text{Dom}(\pi)$   
 $\text{Frontier} \leftarrow \text{Frontier} \cup T$

```
Find-Acyclic-Solution( $\Sigma, s_0, S_g$ )
   $\pi \leftarrow \emptyset$ 
   $\text{Frontier} \leftarrow \{s_0\}$ 
  for every  $s \in \text{Frontier} \setminus S_g$  do
     $\text{Frontier} \leftarrow \text{Frontier} \setminus \{s\}$ 
    if Applicable( $s$ ) =  $\emptyset$  then
      return failure
    nondeterministically choose  $a \in \text{Applicable}(s)$ 
     $\pi \leftarrow \pi \cup (s, a)$ 
     $\text{Frontier} \leftarrow \text{Frontier} \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$ 
    if has-loops( $\pi, s, \text{Frontier}$ ) then
      return failure
  return  $\pi$ 
```

# MinMax Learning Real Time A\*

Looks ahead 1 step; can be modified to look ahead  $k$  steps

Assumes each action has cost 1; can easily be modified to use cost  $\neq 1$  by replacing 1 with  $c(s)$

Min-Max-LRTA\* ( $\Sigma, s_0, S_g$ )

$s \leftarrow s_0$

**while**  $s \notin S_g$  and  $\text{Applicable}(s) \neq \emptyset$  **do**

$a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$

$h(s) \leftarrow \max\{h(s), 1 + \max_{s' \in \gamma(s,a)} h(s')\}$

perform action  $a$

$s \leftarrow$  the current state

- MinMax LRTA\* for short
- Lookahead with a bounded number of steps
- Loop
  - choose an action  $a$  that (according to  $h$ ) has optimal worst-case cost
    - Update  $h(s)$  to use  $a$ 's worst-case cost
    - Perform  $a$

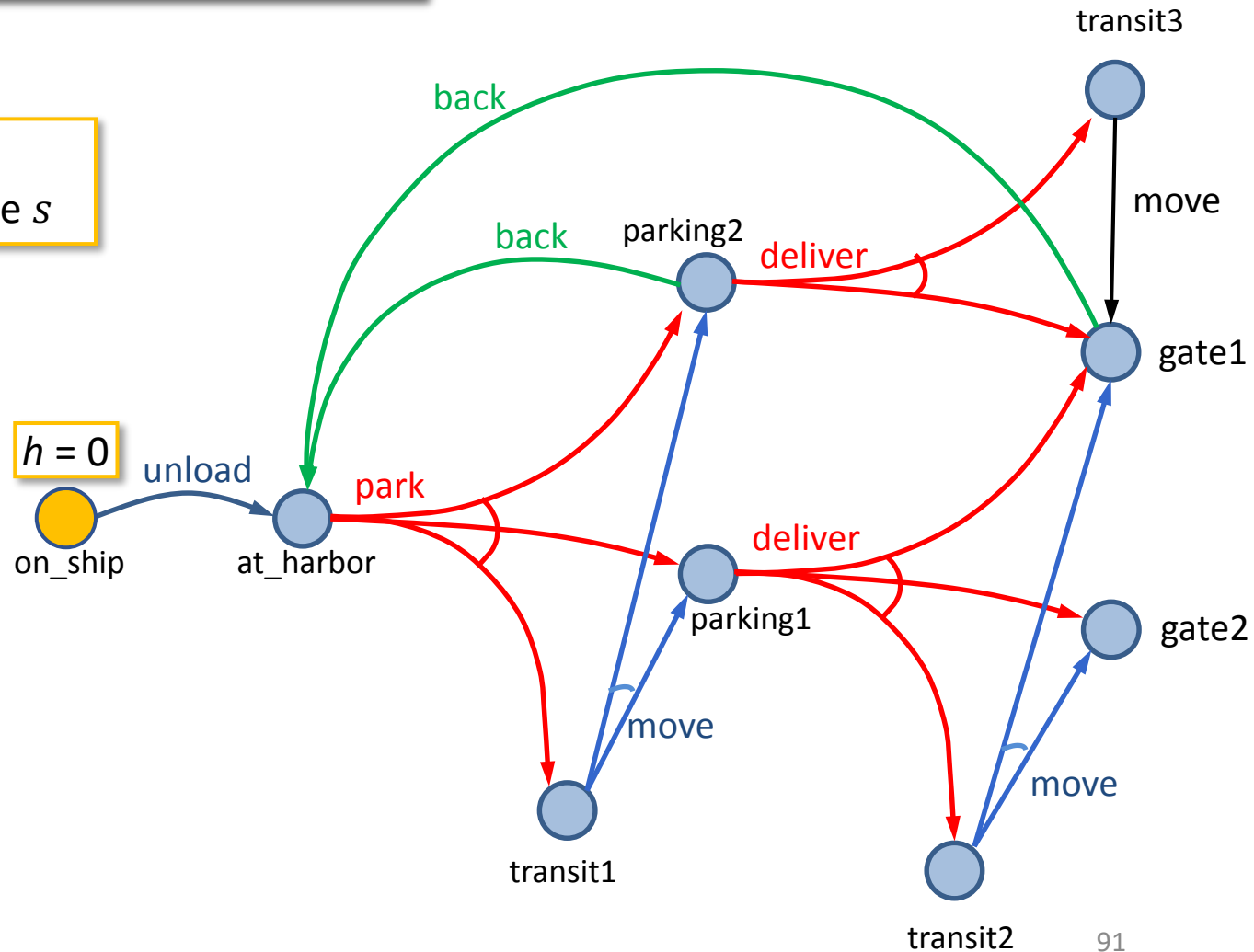
# Min-Max-LRTA\* ( $\Sigma, s_0, S_g$ )

$s \leftarrow s_0$

```
while  $s \notin S_g$  and  $\text{Applicable}(s) \neq \emptyset$  do  
   $a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$   
   $h(s) \leftarrow \max\{h(s), 1 + \max_{s' \in \gamma(s,a)} h(s')\}$   
  perform action  $a$   
   $s \leftarrow$  the current state
```

# Example

Suppose that initially,  
 $h(s) = 0$  for every state  $s$



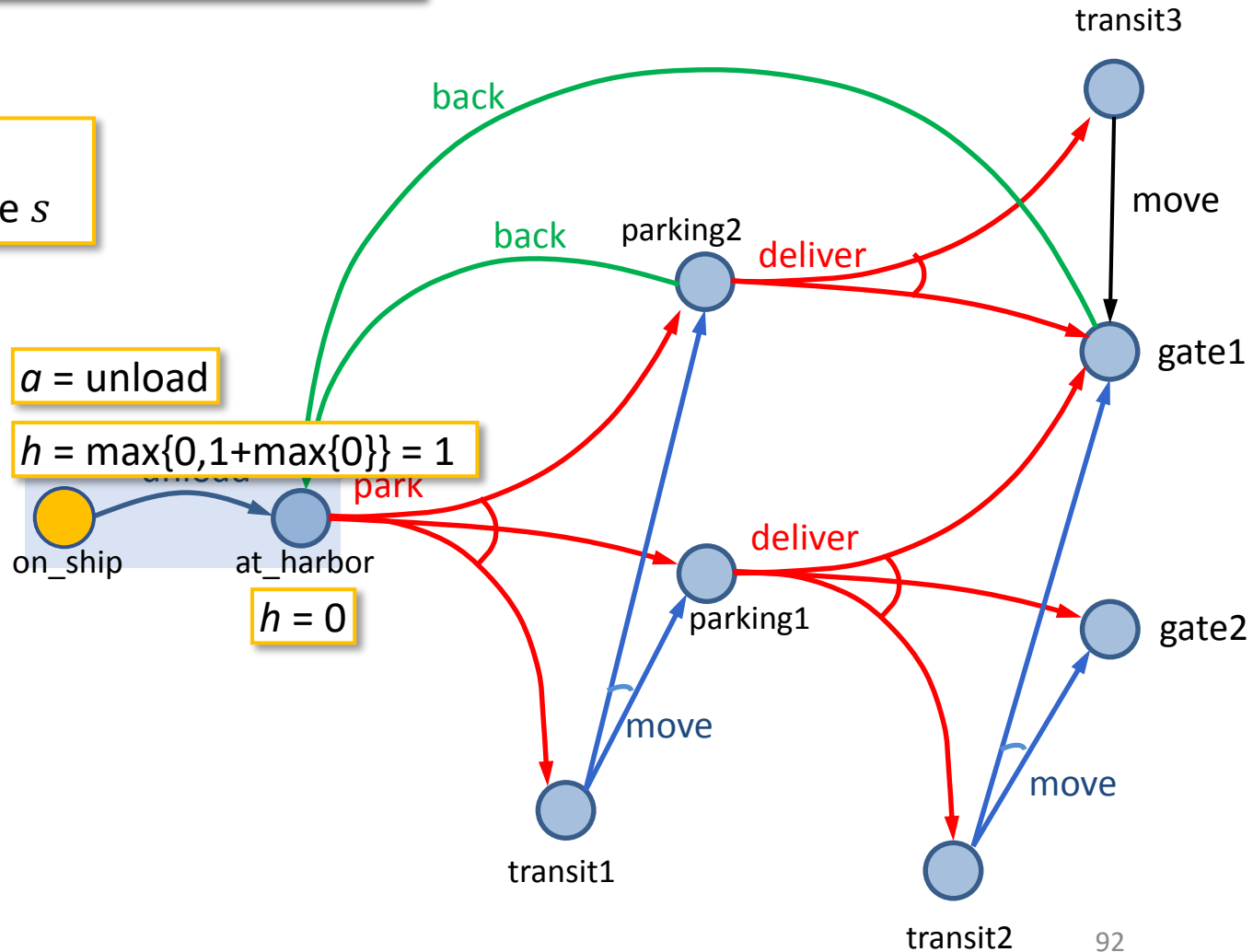
# Min-Max-LRTA\* ( $\Sigma, s_0, S_g$ )

$s \leftarrow s_0$

```
while  $s \notin S_g$  and  $\text{Applicable}(s) \neq \emptyset$  do  
   $a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$   
   $h(s) \leftarrow \max\{h(s), 1 + \max_{s' \in \gamma(s,a)} h(s')\}$   
  perform action  $a$   
   $s \leftarrow$  the current state
```

# Example

Suppose that initially,  
 $h(s) = 0$  for every state  $s$



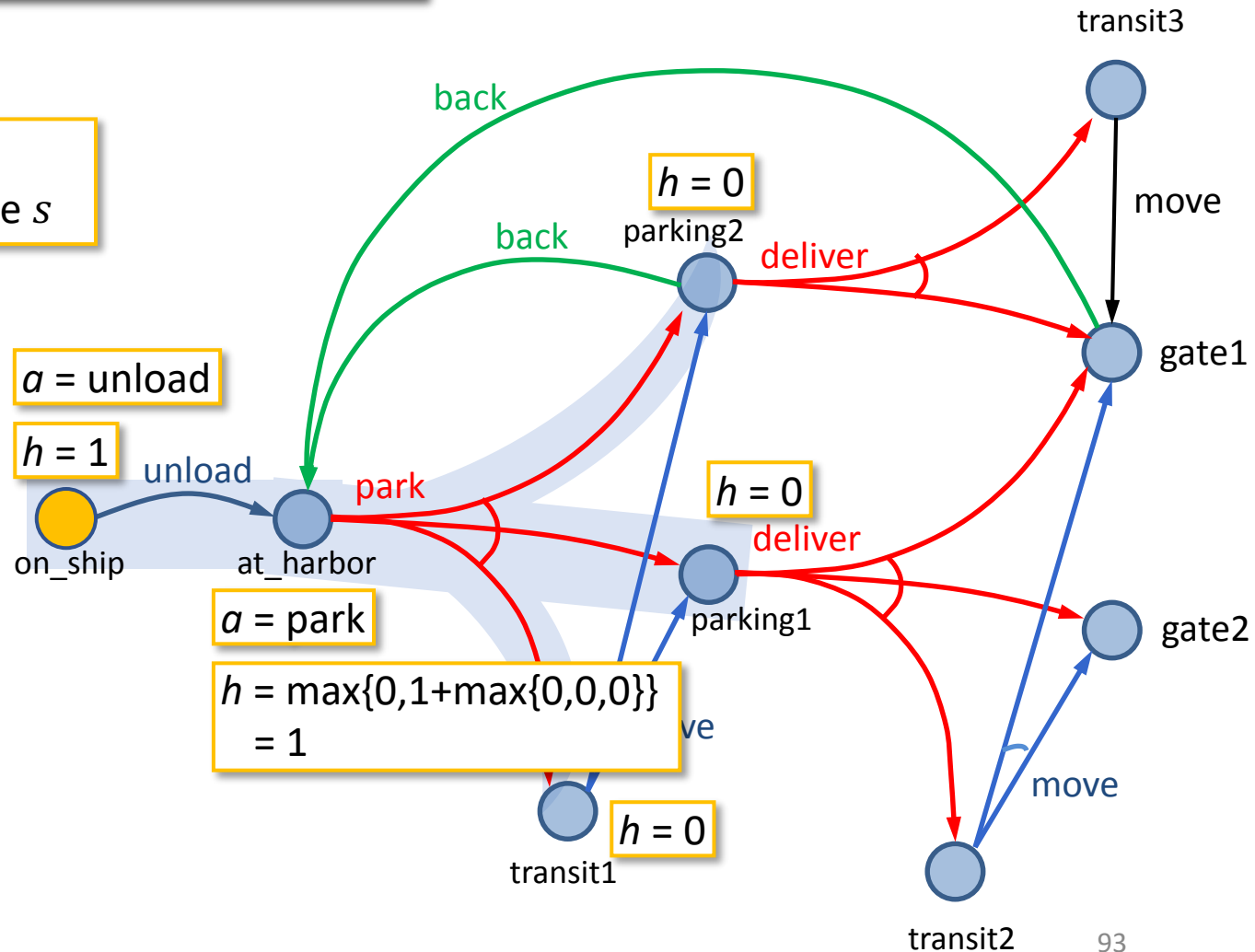
# Min-Max-LRTA\* ( $\Sigma, s_0, S_g$ )

$s \leftarrow s_0$

```
while  $s \notin S_g$  and  $\text{Applicable}(s) \neq \emptyset$  do  
   $a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$   
   $h(s) \leftarrow \max\{h(s), 1 + \max_{s' \in \gamma(s,a)} h(s')\}$   
  perform action  $a$   
   $s \leftarrow$  the current state
```

# Example

Suppose that initially,  
 $h(s) = 0$  for every state  $s$

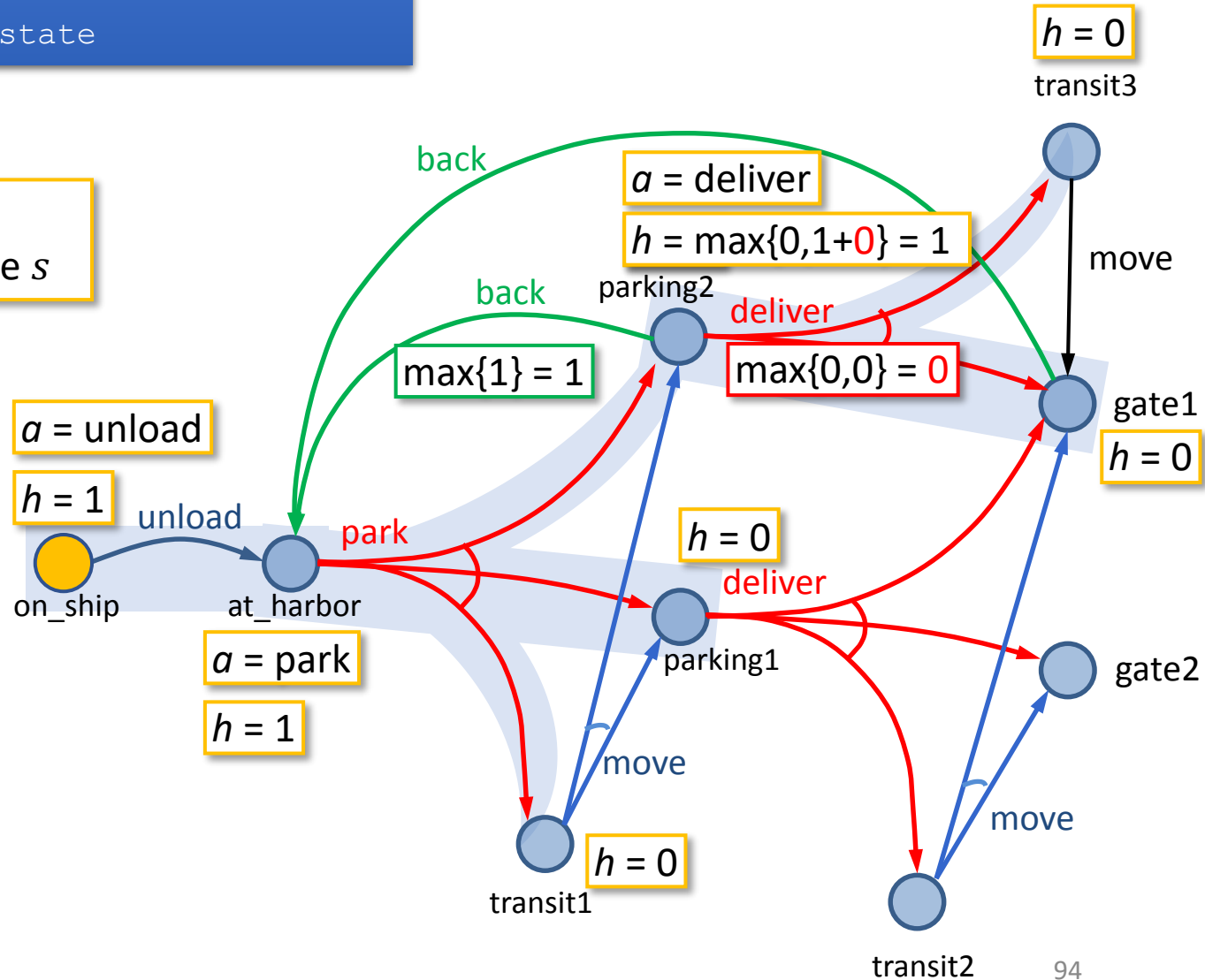


# Min-Max-LRTA\* ( $\Sigma, s_0, S_g$ )

$s \leftarrow s_0$

```
while  $s \notin S_g$  and  $\text{Applicable}(s) \neq \emptyset$  do  
   $a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$   
   $h(s) \leftarrow \max\{h(s), 1 + \max_{s' \in \gamma(s,a)} h(s')\}$   
  perform action  $a$   
   $s \leftarrow$  the current state
```

Suppose that initially,  
 $h(s) = 0$  for every state  $s$



# Safely Explorable Domains

---

- Safely explorable domain

- For every state  $s$ , at least one goal state is reachable from  $s$ 
  - No dead ends



What about picking a random action?

- In a safely explorable domain,

- Using Lookahead-Partial-Plan or FS-Replan
  - Lookahead never returns failure
  - Then we will eventually reach a goal

- Using MinMax LRTA\*

- Algorithm is guaranteed to terminate and generate a solution

# Intermediate Summary

---

- Online approaches
  - Lookahead-partial-plan
    - Adaptation of Run-Lazy-Lookahead
  - FS-replan
    - Adaptation of Run-Lookahead
- Ways to do the lookahead
  - Full breadth with limited depth
    - Iterative deepening
  - Full depth with limited breadth
    - Iterative broadening
- Min-Max-LRTA\*
- Convergence in safely explorable domains

Can also adapt  
Run-Concurrent-Lookahead

Can put bounds on both  
depth and breadth



# Outline per the Book

---

## *5.2 Planning Problem*

- Planning domains
- Plans as policies
- Planning problems and solutions

## *5.3 And/Or Graph Search*

- Planning by forward search

## *5.5 Determinisation Techniques*

- Guided planning for safe solutions
- Planning for safe solutions by determinisation

## *5.6 Online Approaches*

- Lookahead
- Lookahead by determinisation
- Lookahead with a bounded number of steps

⇒ Next: Making Simple Decisions