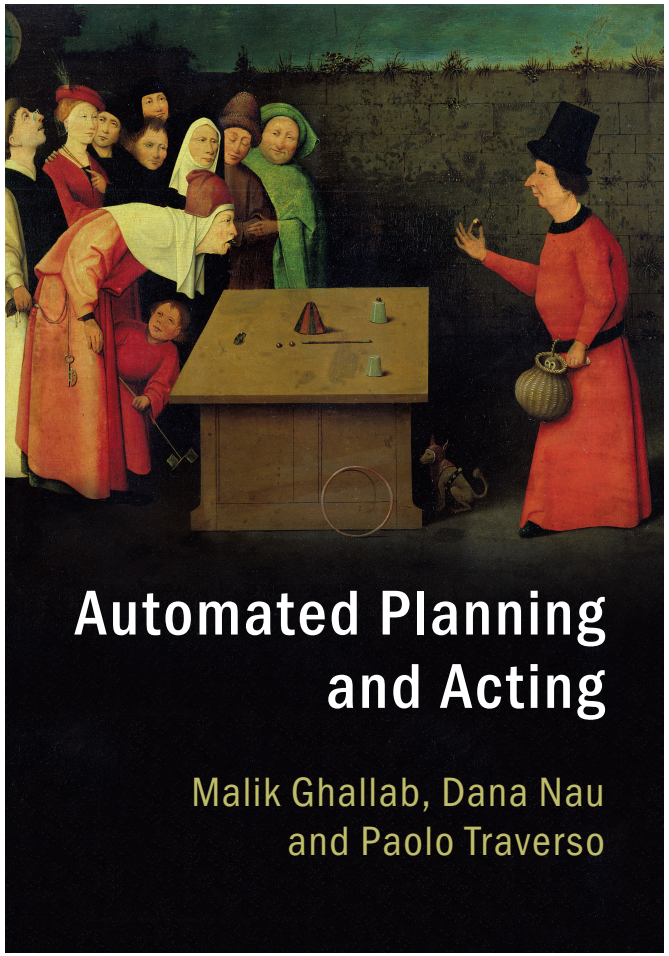
Last update: May 3, 2022

Chapter 5

Deliberation with Nondeterministic Domain Models

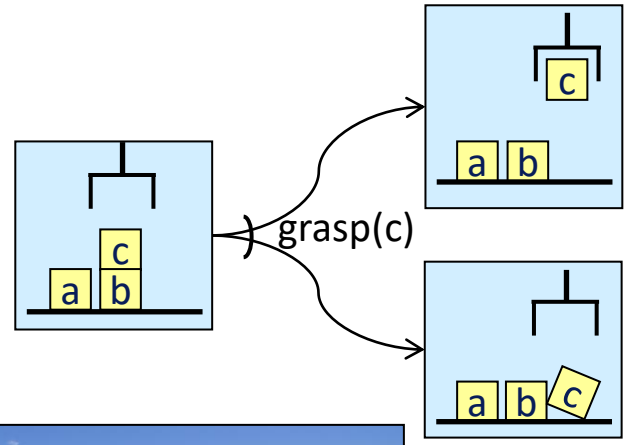
Dana S. Nau
University of Maryland



http://www.laas.fr/planning

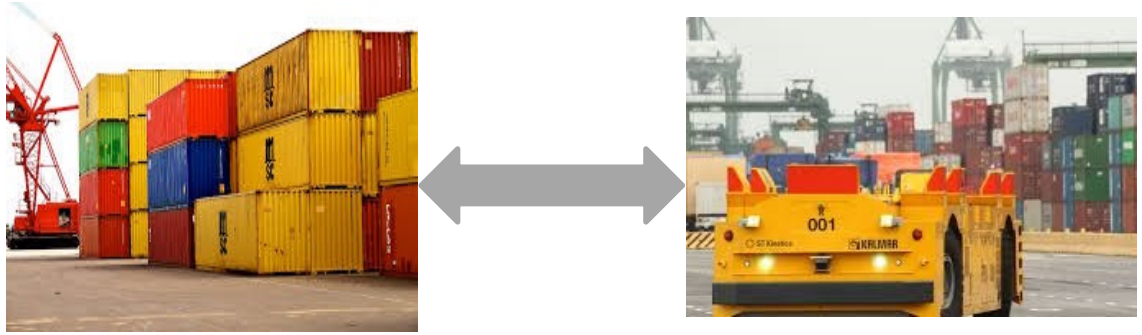
Motivation

- We've assumed action a in state s has just one possible outcome
  - $\gamma(s,a)$

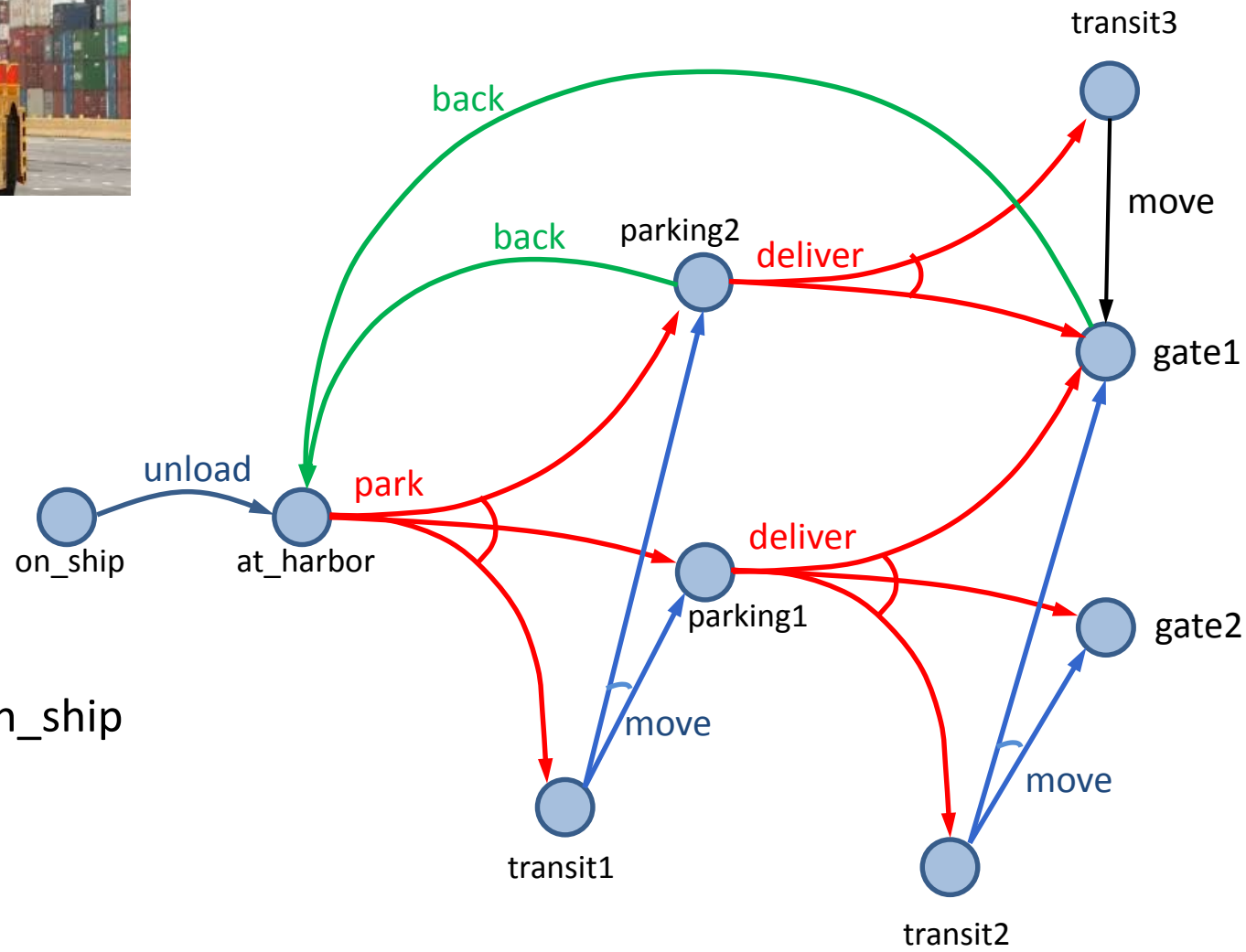


- Often more than one possible outcome
  - Unintended outcomes
  - Exogenous events
  - Inherent uncertainty

Example

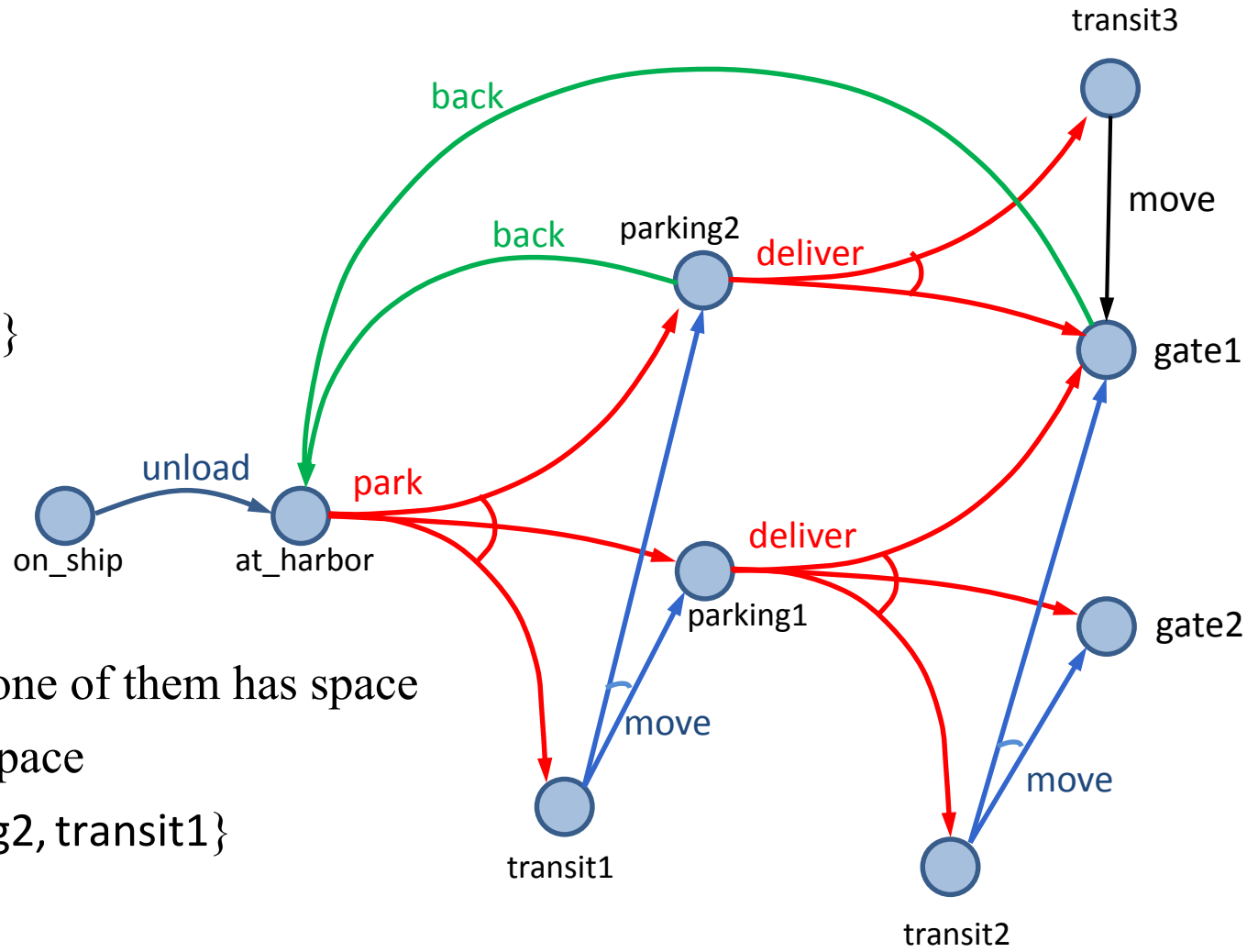


- Very simple harbor management domain
  - Unload a single item from a ship
  - Move it around a harbor
- One state variable: pos(item)
  - Simplified names for states
  - For {pos(item)=on_ship}, just write on_ship



Nondeterministic Planning Domains

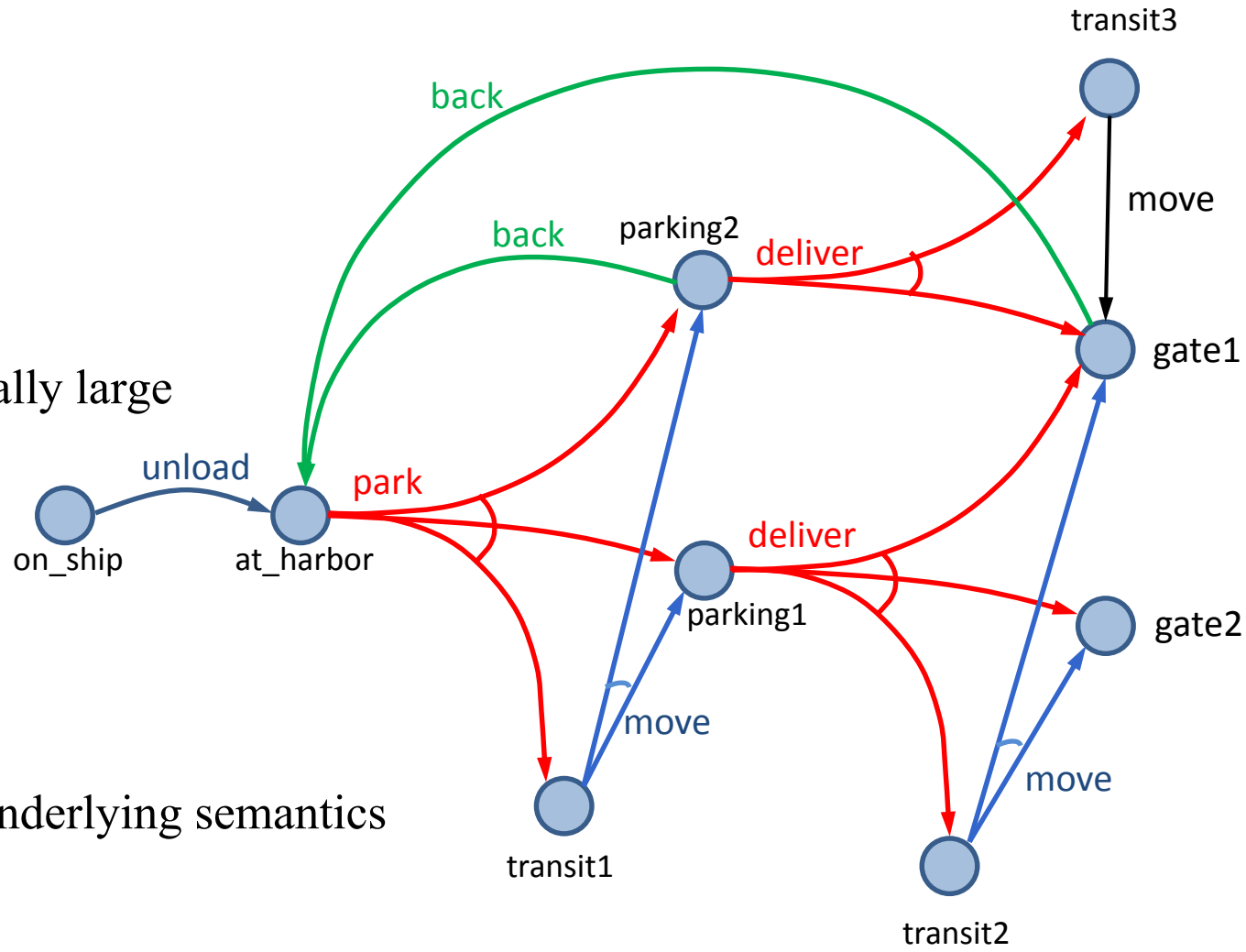
- 3-tuple (S, A, γ)
  - S and A – finite sets of states and actions
  - $\gamma: S \times A \rightarrow 2^S$
- $\gamma(s,a)$ = {all possible "next states" after applying action a in state s}
  - a is applicable in state s iff $\gamma(s,a) \neq \emptyset$
- Applicable(s) = {all actions applicable in s} = $\{a \in A \mid \gamma(s,a) \neq \emptyset\}$
- 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

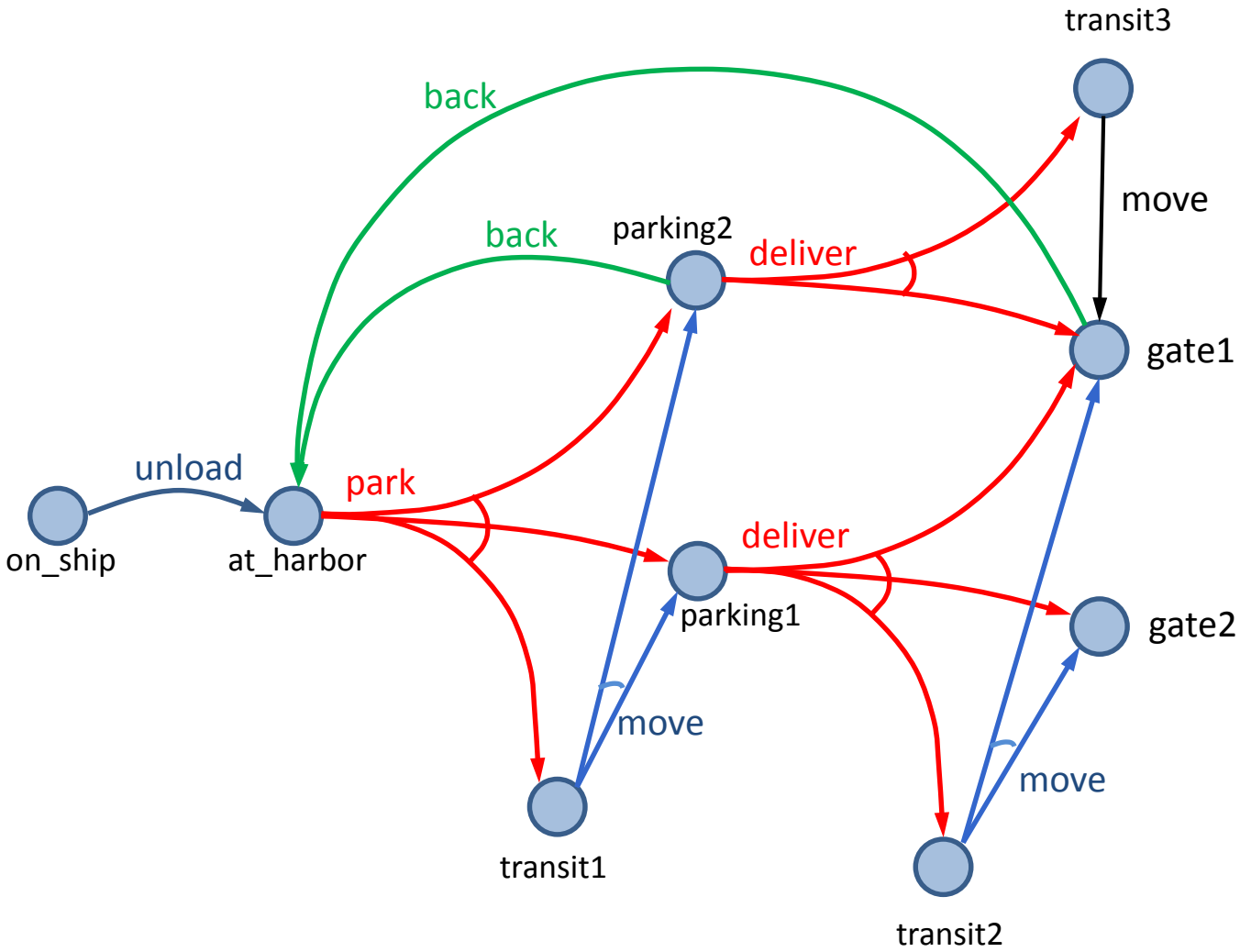
- One possible action representation:
  - like classical, but with n mutually exclusive "effects" lists
- e.g., park:

  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 $eff_1, eff_2, \ldots, eff_{2^k}$
    - One for for each combination
  - Section 5.4: a way to 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



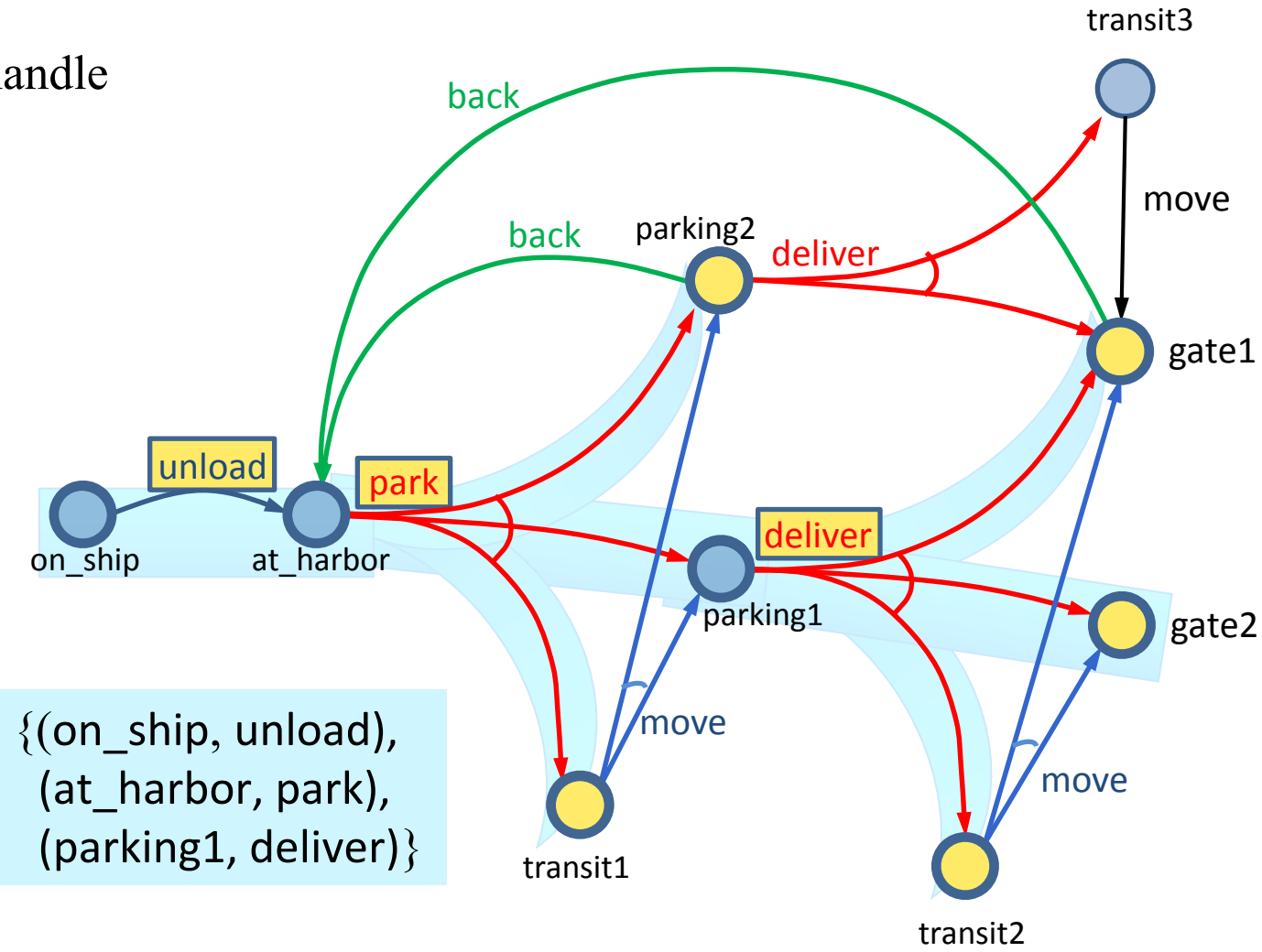
~~Plans~~ Policies

- Need something more general than a sequence of actions ⟨unload, park, deliver⟩
  - After park, what do we do next?
    - multiple possible outcomes, must handle all of them
- *Policy*: a *partial* function $\pi : S \nrightarrow A$
  - i.e., $\mathrm{Dom}(\pi) \subseteq S$
  - For every $s \in \mathrm{Dom}(\pi)$, require $\pi(s) \in \mathrm{Applicable}(s)$
- Meaning: perform $\pi(s)$ whenever we're in state 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}
  - $\hat{\gamma}(s,\pi) = S_0 \cup S_1 \cup S_2 \cup \ldots$

    $S_0 = \{s\}$

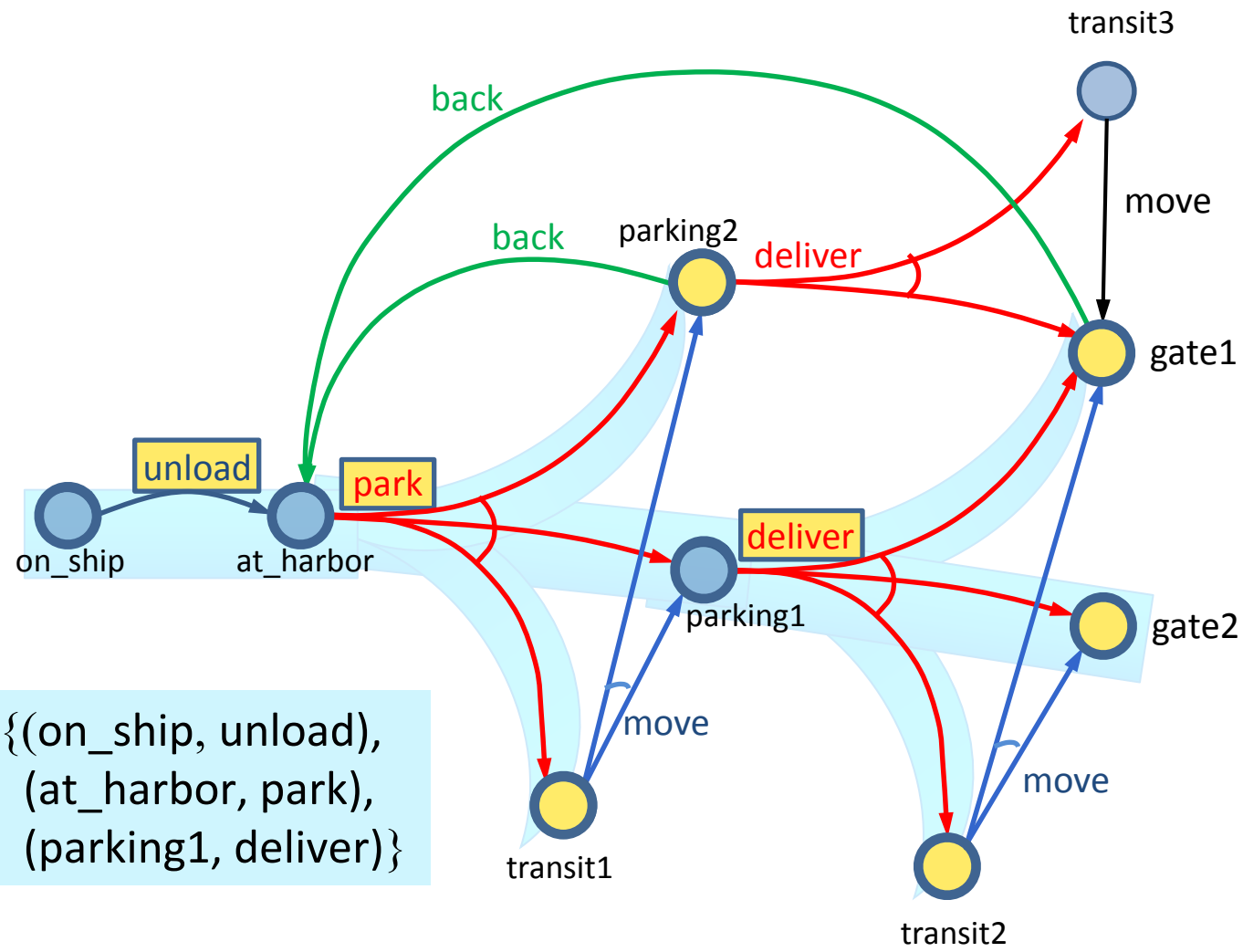
    $S_1 = S_0 \cup \{\gamma(s_0,\pi(s_0)) \mid s_0 \in S_0\}$

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

    ...

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

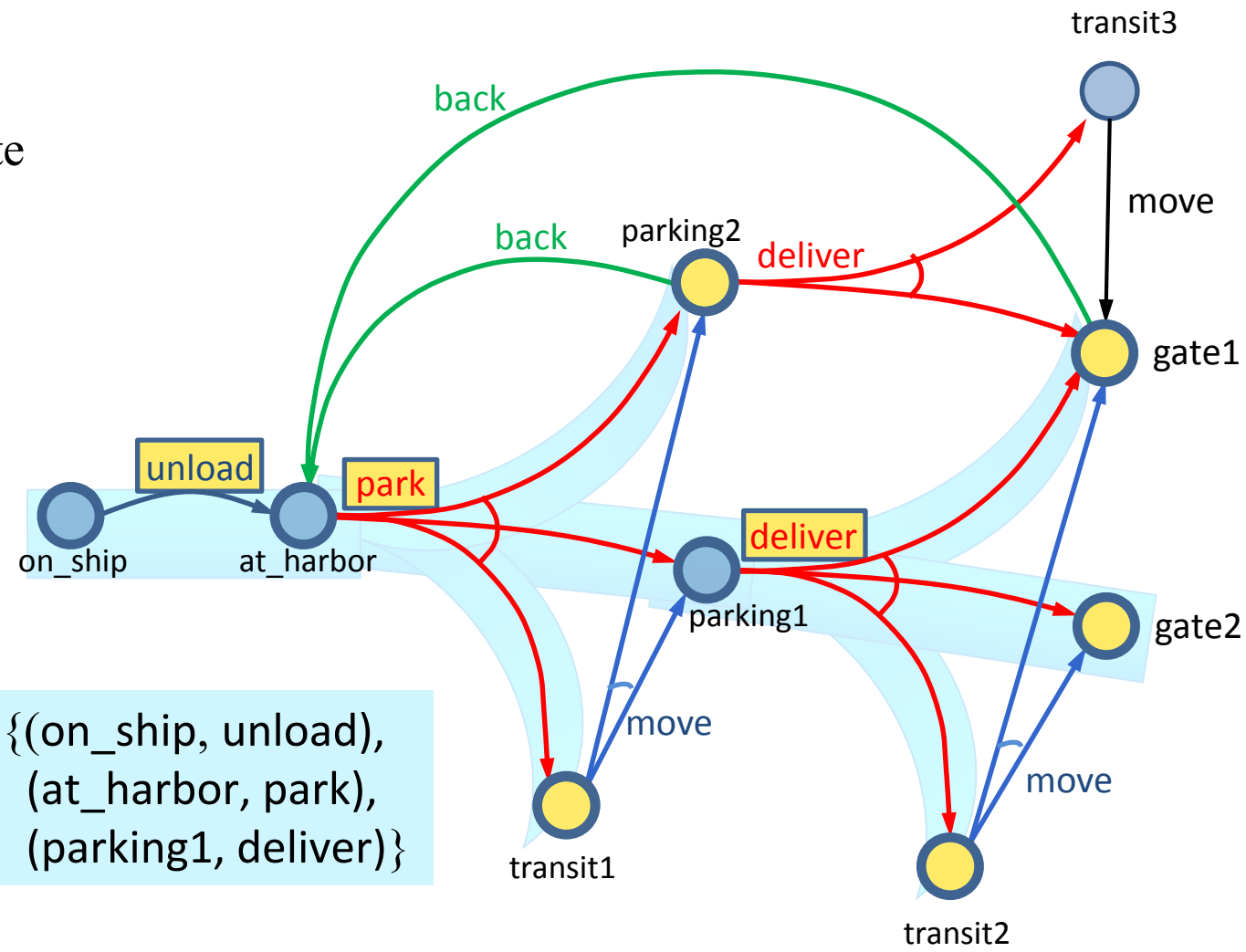
- *leaves*$(s,\pi) = \hat{\gamma}(s,\pi) \setminus \mathrm{Dom}(\pi)$
  - may be empty



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

Performing a Policy

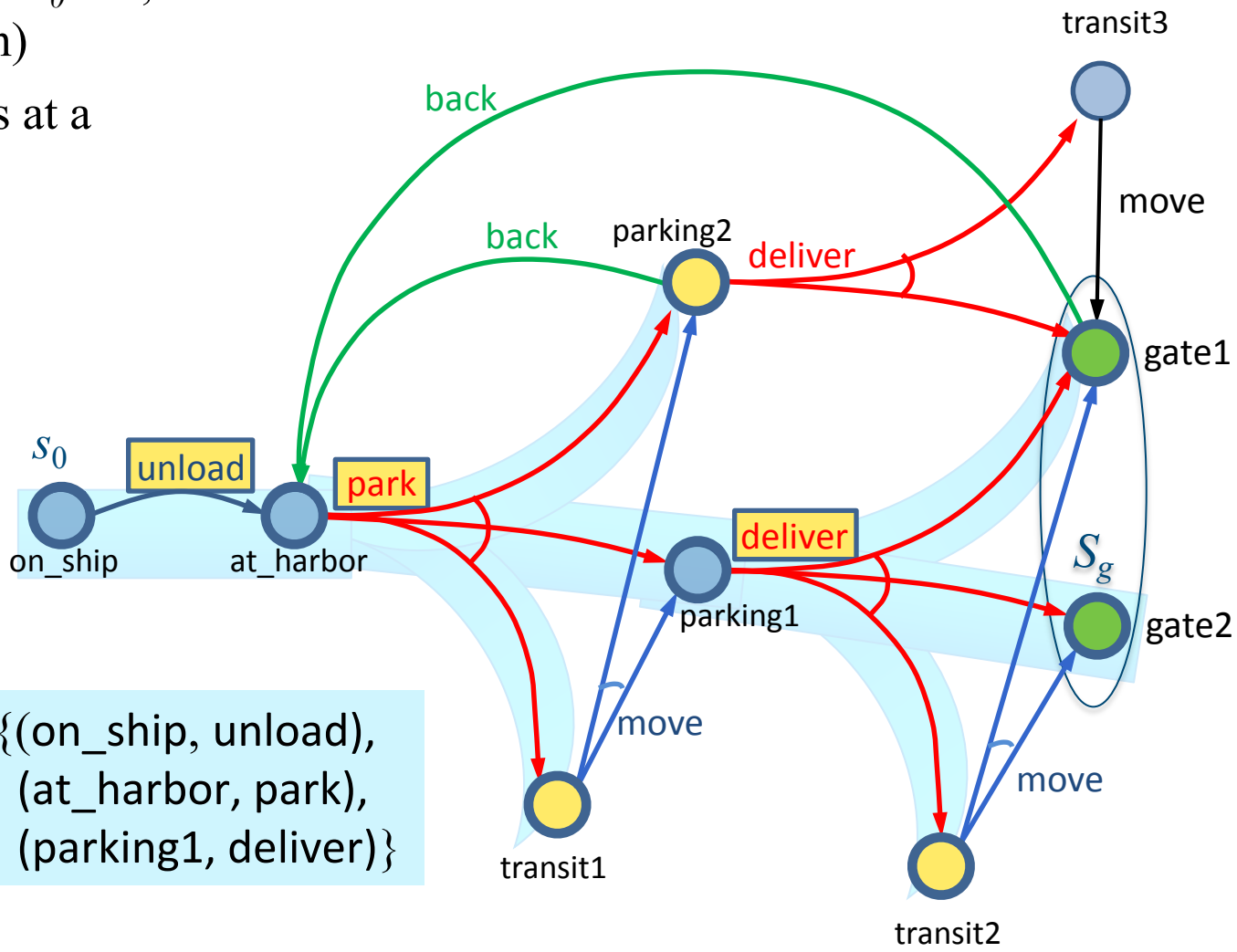
- PerformPolicy(π)
  $s \leftarrow$ observe current state
  while $s \in \text{Dom}(\pi)$ do
    perform action $\pi(s)$
    $s \leftarrow$ observe current state



$\pi_1 = \{$(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 green)
- π is a *solution* if at least one execution ends at a goal
  - $leaves(s,\pi) \cap S_g \neq \emptyset$
- A solution 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* solution
  - Is π_1 safe or unsafe?

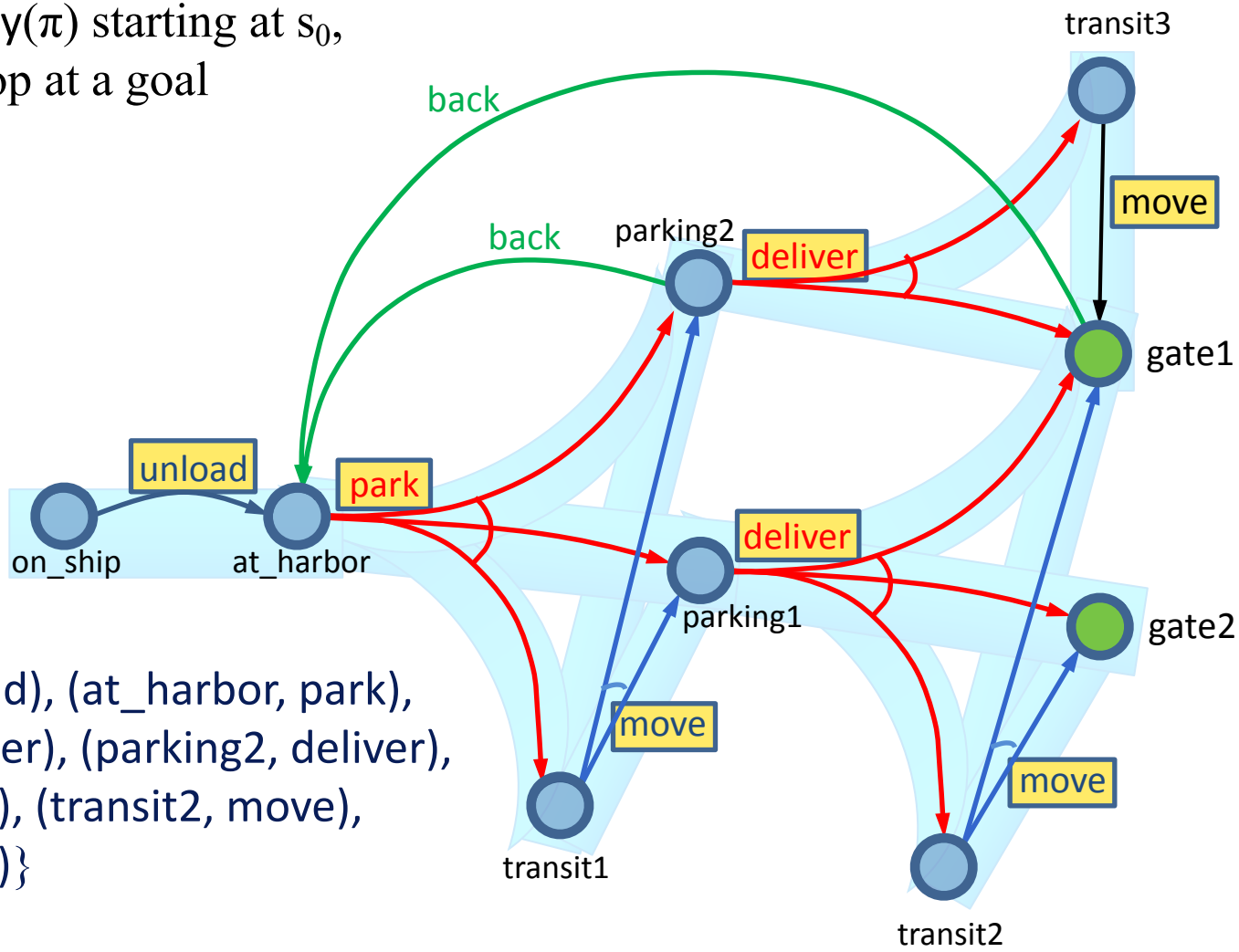


$\pi_1 = \{$(on_ship, unload), (at_harbor, park), (parking1, deliver)$\}$

Safe Solutions

- PerformPolicy(π)
  $s \leftarrow$ observe current state
  while $s \in \text{Dom}(\pi)$ do
    perform action $\pi(s)$
    $s \leftarrow$ observe current state

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

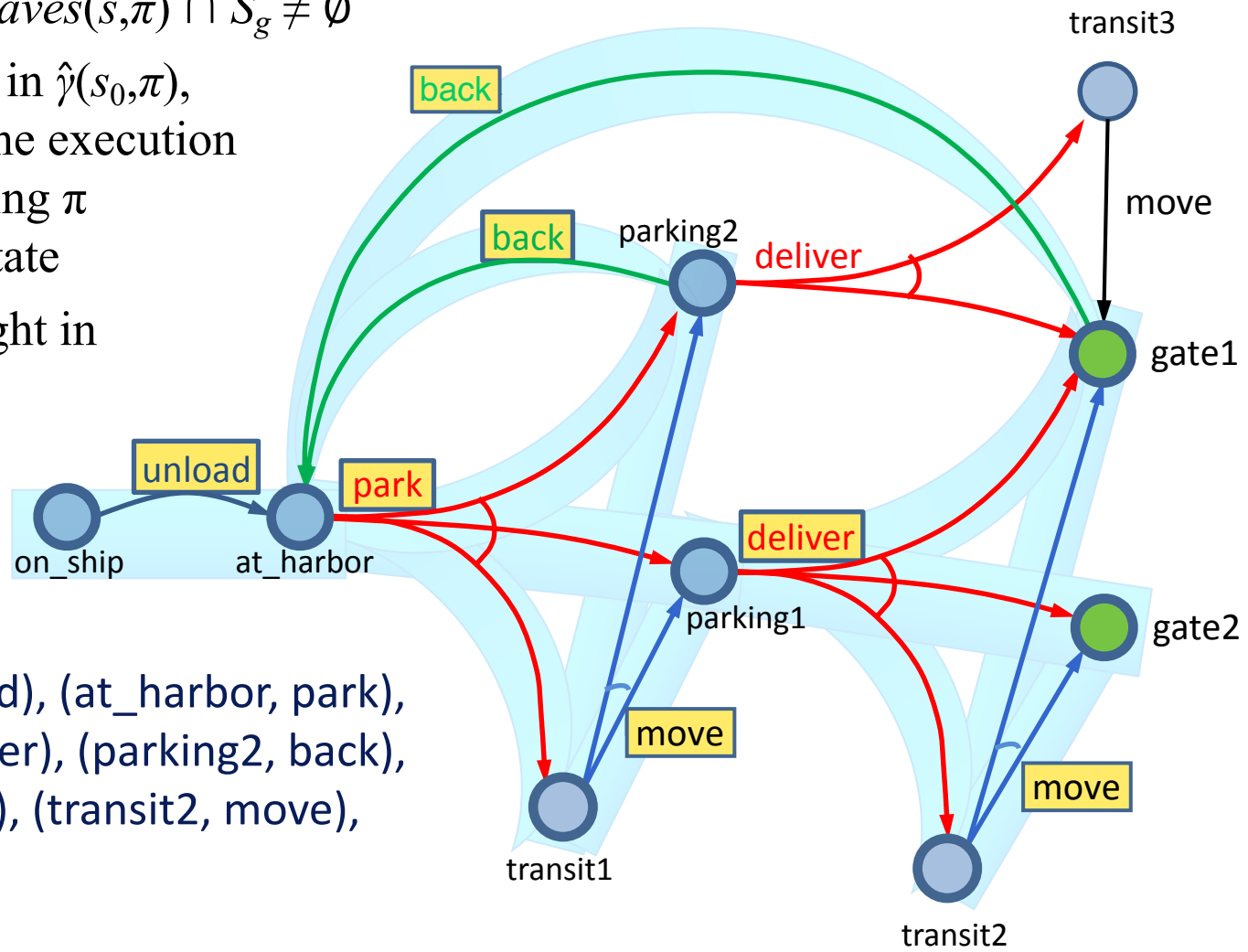


π_2 = {(on_ship, unload), (at_harbor, park),
(parking1, deliver), (parking2, deliver),
(transit1, move), (transit2, move),
(transit3, move)}

Safe Solutions

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

- *Cyclic* safe solution
  - 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)}

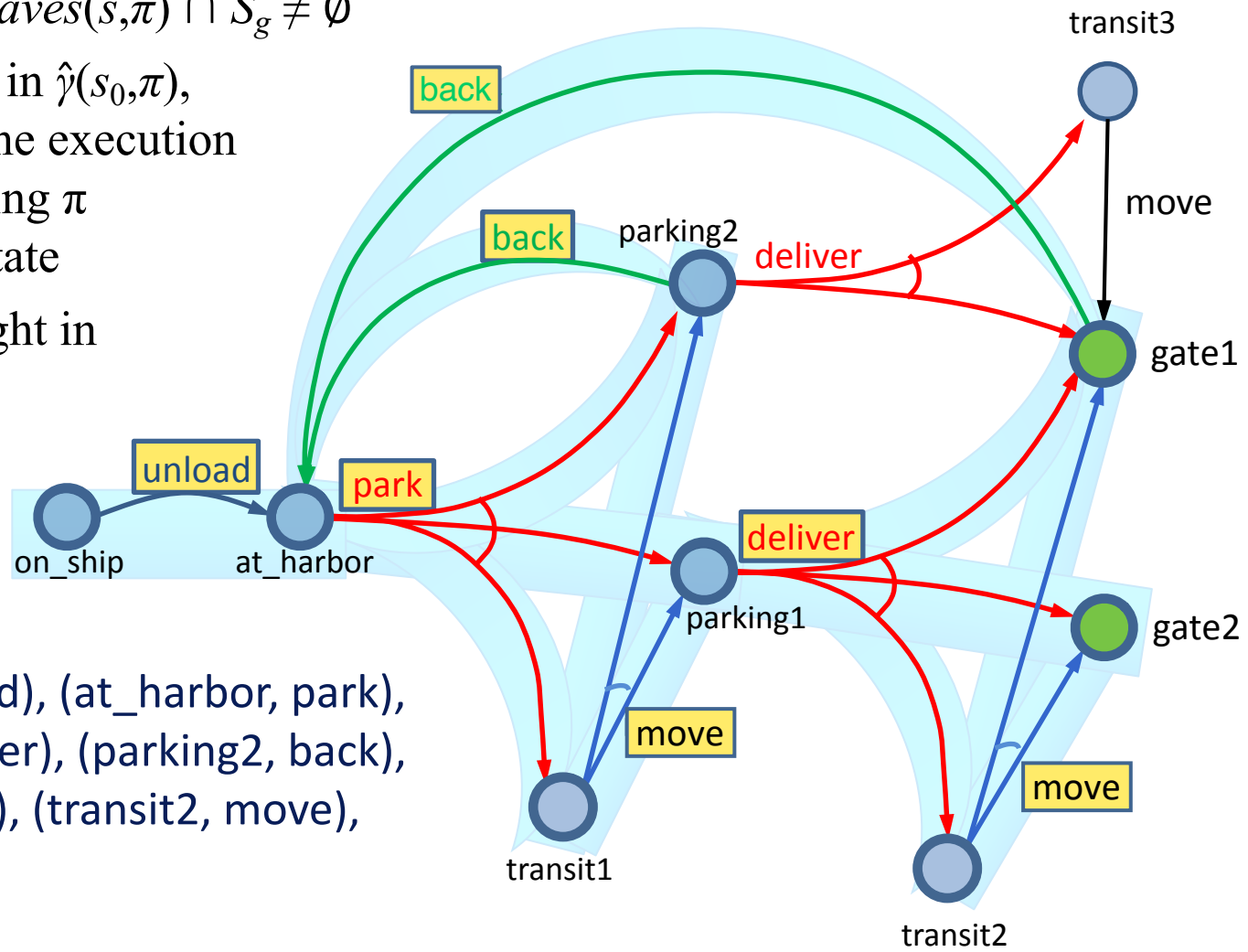
Poll: Let π be a cyclic safe solution. Suppose we run PerformPolicy(π) starting at s_0.

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 Solutions

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

- *Cyclic* safe solution
  - 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

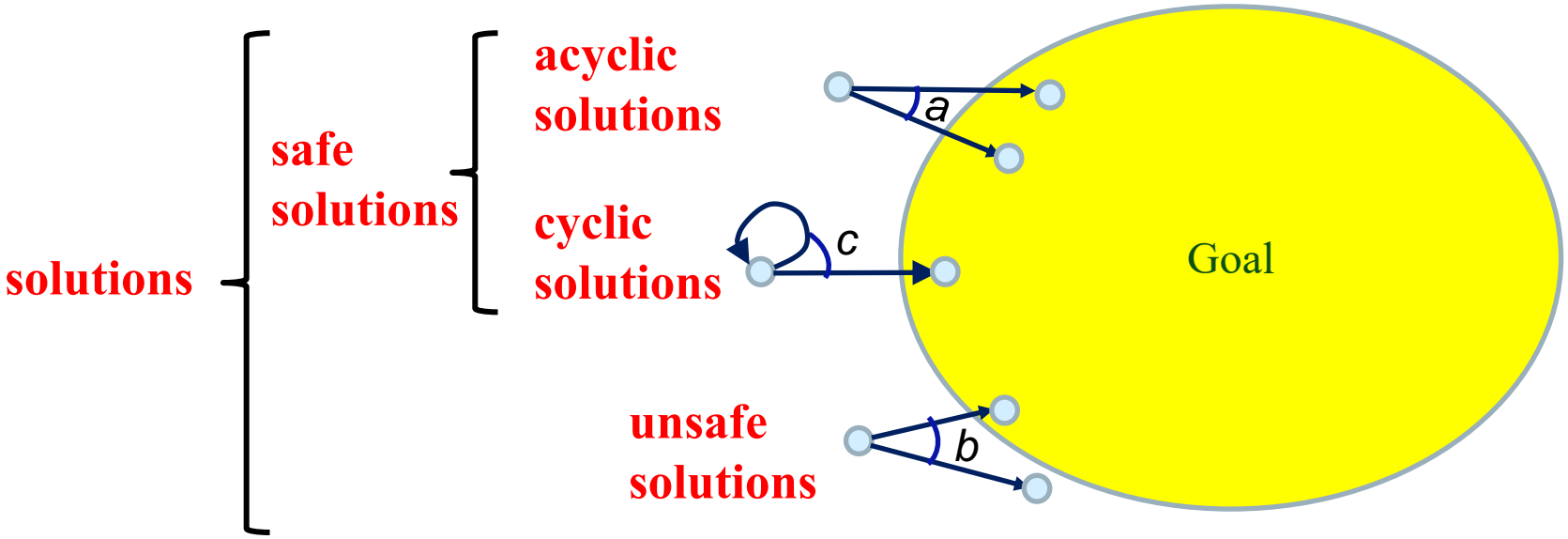


Poll:

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 Solutions



Finding (Unsafe) Solutions

For comparison:

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

Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $Visited \leftarrow \{s_0\}$
  loop
    if $s \in S_g$ then return π
    $A' \leftarrow$ Applicable(s)
    if $A' = \varnothing$ then return failure
    nondeterministically choose $a \in A'$
(*) nondeterministically choose $s' \in \gamma(s,a)$
    if $s' \in Visited$ then return failure
    $\pi(s) \leftarrow a$; $Visited \leftarrow Visited \cup \{s'\}$; $s \leftarrow s'$

Decide which state to plan for

Cycle-checking

Poll: which should (*) be?
A. nondeterministically choose
B. arbitrarily choose

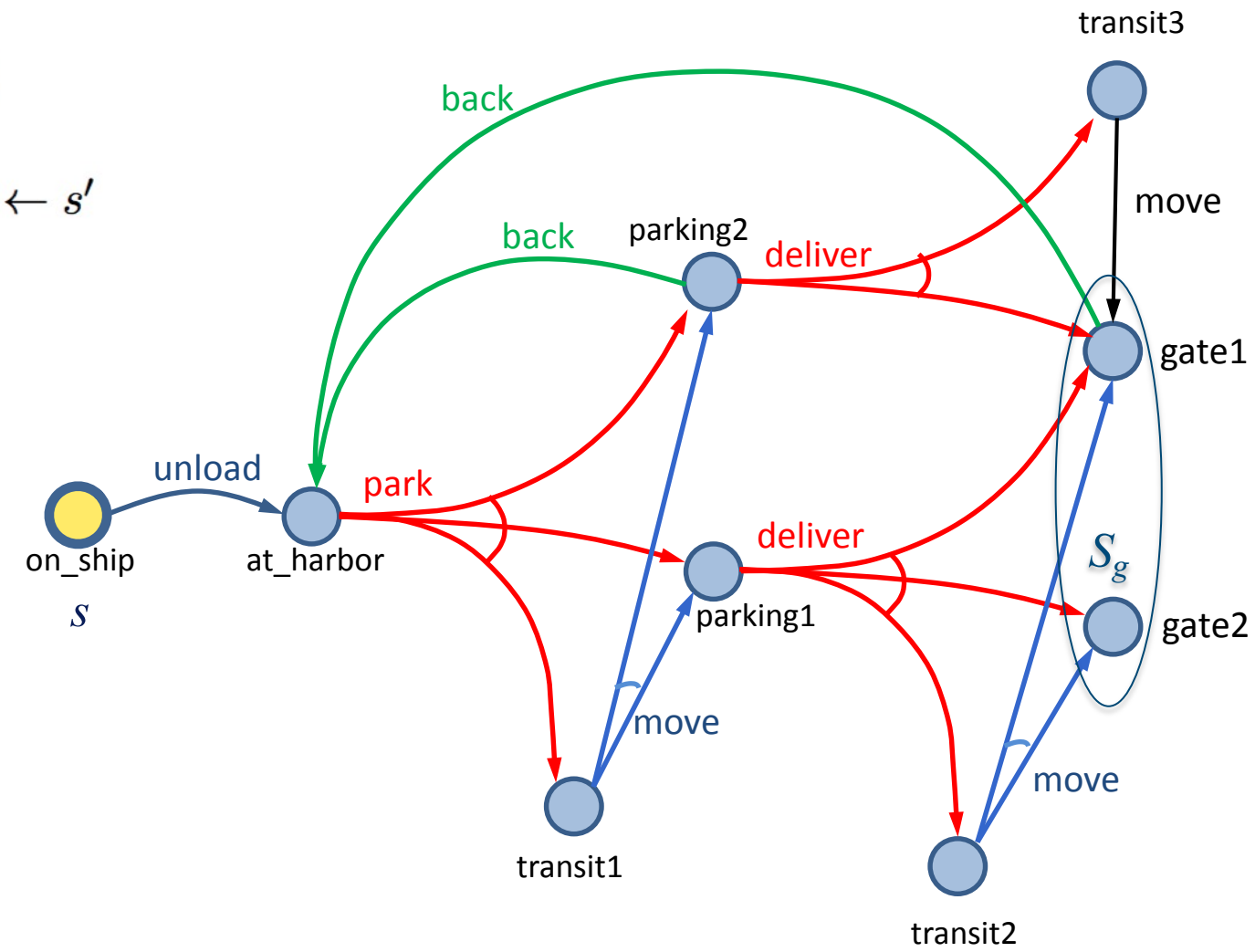
Example

Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $Visited \leftarrow \{s_0\}$
  loop
    if $s \in S_g$ then return π
    $A' \leftarrow \text{Applicable}(s)$
    if $A' = \varnothing$ then return failure
    nondeterministically choose $a \in A'$
    nondeterministically choose $s' \in \gamma(s, a)$
    if $s' \in Visited$ then return failure
    $\pi(s) \leftarrow a$; $Visited \leftarrow Visited \cup \{s'\}$; $s \leftarrow s'$

s = on_ship

$\pi = \{\}$

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



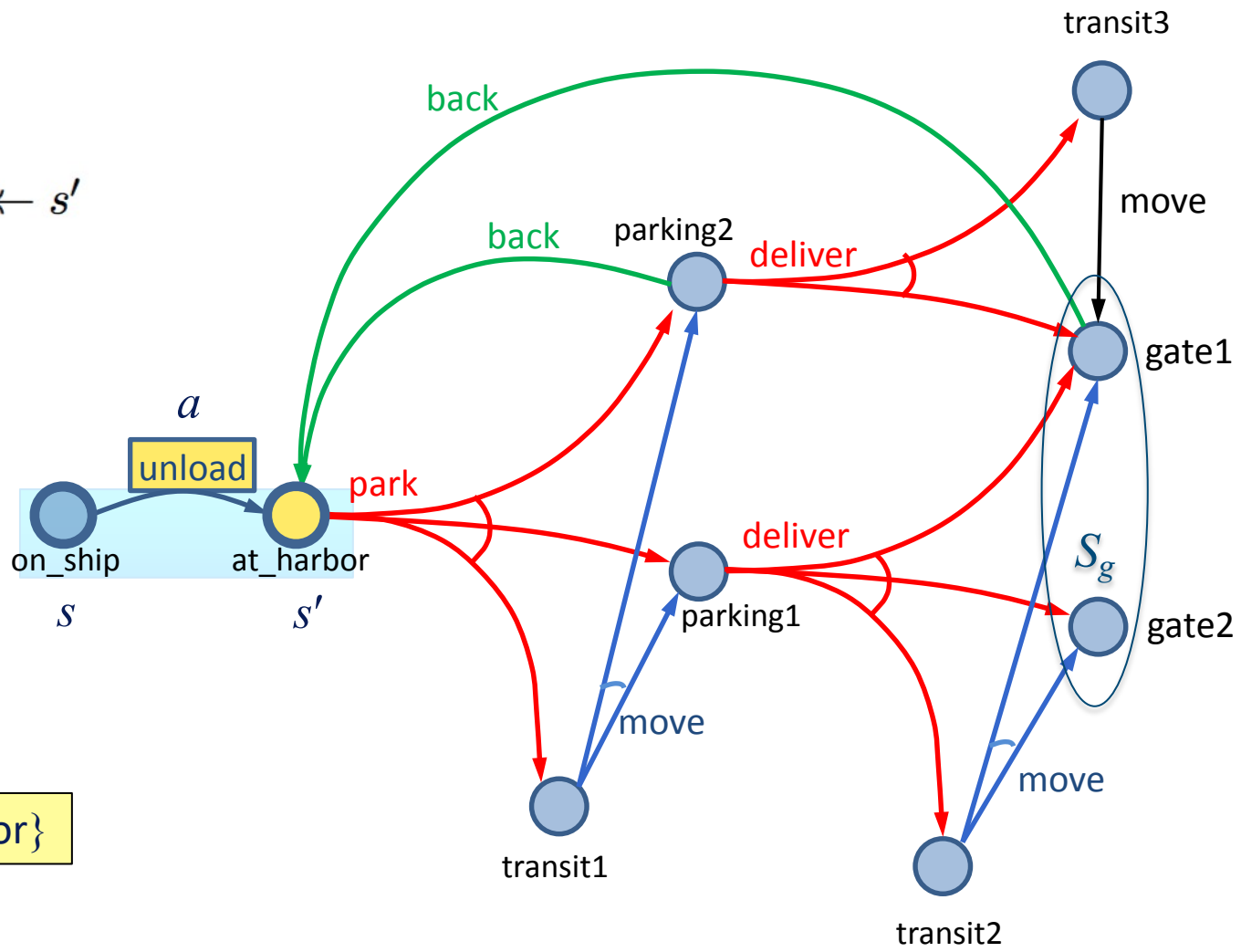
Example

Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $Visited \leftarrow \{s_0\}$
  loop
    if $s \in S_g$ then return π
    $A' \leftarrow \text{Applicable}(s)$
    if $A' = \varnothing$ then return failure
    nondeterministically choose $a \in A'$
    nondeterministically choose $s' \in \gamma(s, a)$
    if $s' \in Visited$ then return failure
    $\pi(s) \leftarrow a$; $Visited \leftarrow Visited \cup \{s'\}$; $s \leftarrow s'$

s = on_ship
a = unload
$\gamma(s,a)$ = {at_harbor}
$s \leftarrow s'$ = at_harbor

π = {(on_ship, unload)}

$Visited$ = {on_ship, at_harbor}



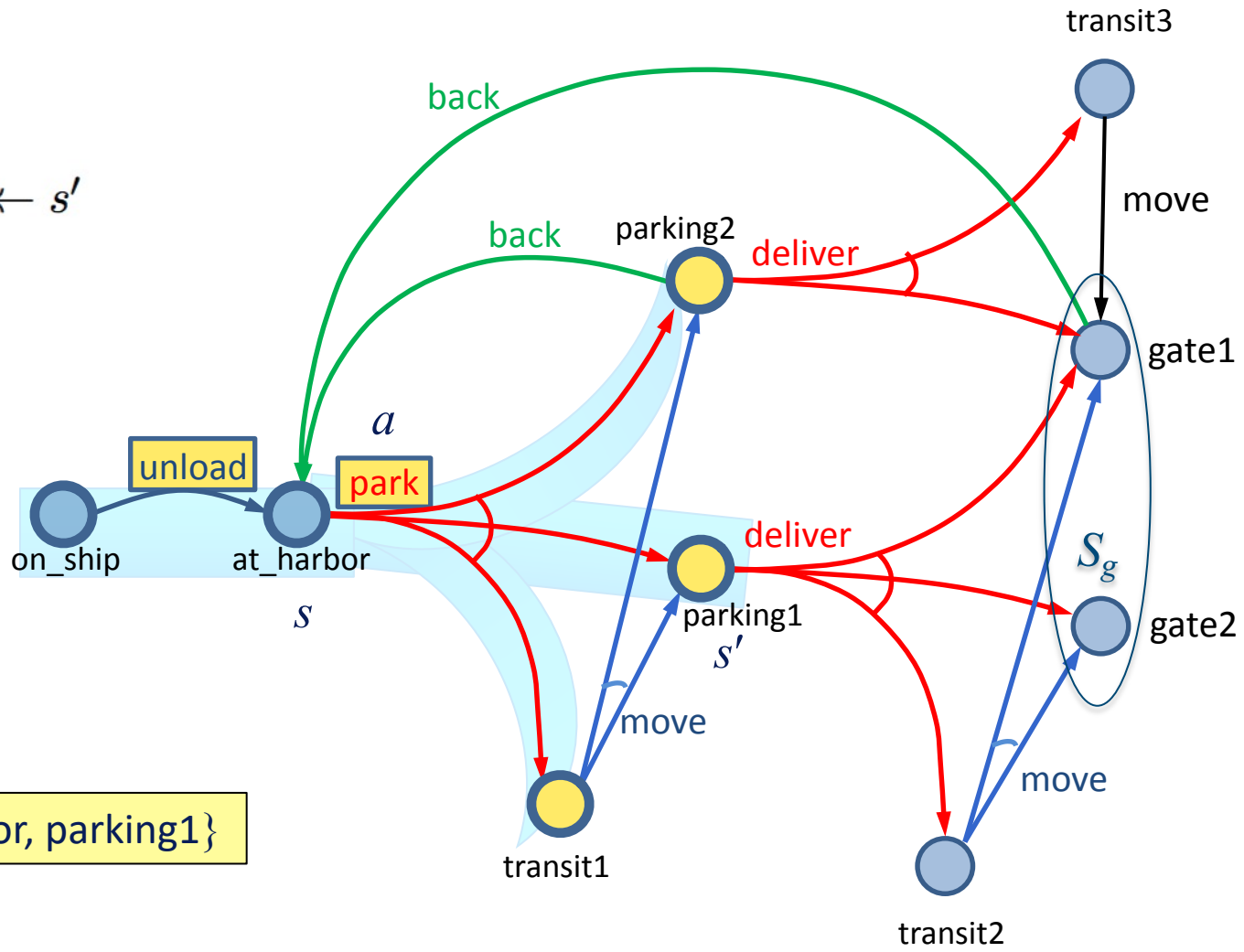
Example

Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $Visited \leftarrow \{s_0\}$
  loop
    if $s \in S_g$ then return π
    $A' \leftarrow$ Applicable(s)
    if $A' = \varnothing$ then return failure
    nondeterministically choose $a \in A'$
    nondeterministically choose $s' \in \gamma(s, a)$
    if $s' \in Visited$ then return failure
    $\pi(s) \leftarrow a$; $Visited \leftarrow Visited \cup \{s'\}$; $s \leftarrow s'$

s = at_harbor
a = park
$\gamma(s,a)$ = {parking1, parking2, transit1}
$s \leftarrow s'$ = parking1

π = {(on_ship, unload), (at_harbor, park)}

$Visited$ = {on_ship, at_harbor, parking1}



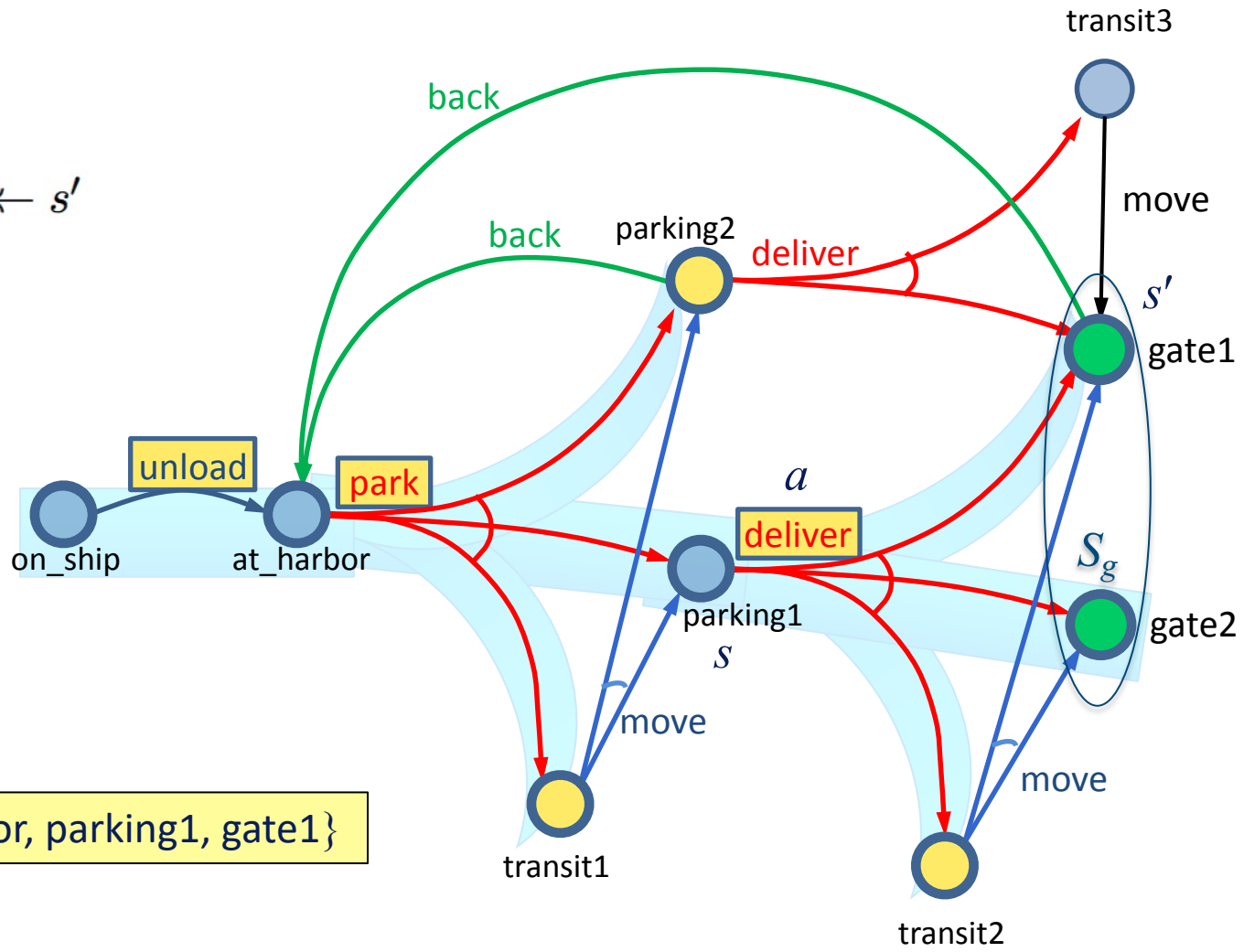
Example

Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $Visited \leftarrow \{s_0\}$
  loop
    if $s \in S_g$ then return π
    $A' \leftarrow \text{Applicable}(s)$
    if $A' = \varnothing$ then return failure
    nondeterministically choose $a \in A'$
    nondeterministically choose $s' \in \gamma(s, a)$
    if $s' \in Visited$ then return failure
    $\pi(s) \leftarrow a$; $Visited \leftarrow Visited \cup \{s'\}$; $s \leftarrow s'$

s = parking1
a = deliver
$\gamma(s,a)$ = {gate1, gate2, transit2}
$s \leftarrow s'$ = gate1

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

$Visited$ = {on_ship, at_harbor, parking1, gate1}



Example

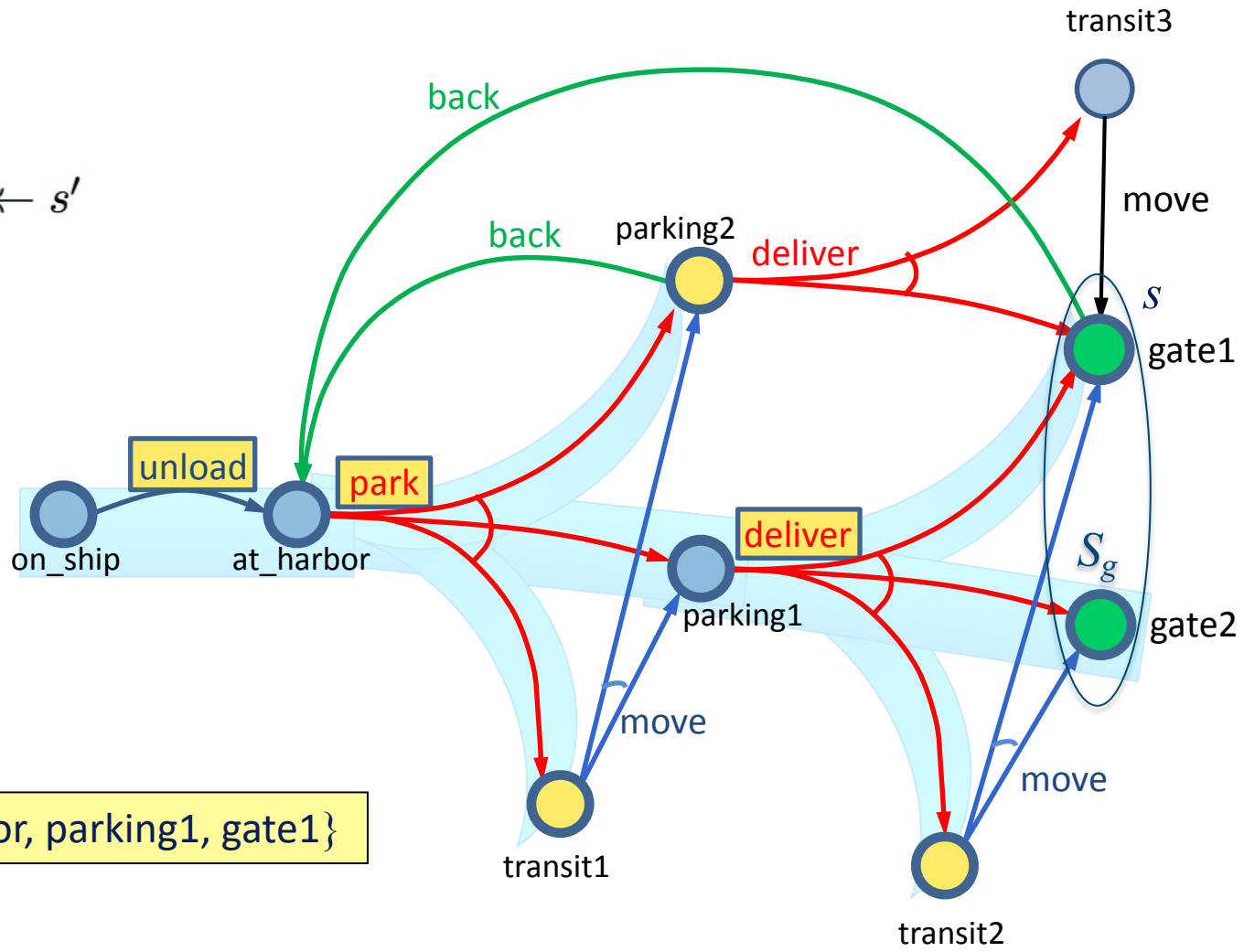
Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $Visited \leftarrow \{s_0\}$
  loop
    if $s \in S_g$ then return π
    $A' \leftarrow$ Applicable(s)
    if $A' = \varnothing$ then return failure
    nondeterministically choose $a \in A'$
    nondeterministically choose $s' \in \gamma(s, a)$
    if $s' \in Visited$ then return failure
    $\pi(s) \leftarrow a$; $Visited \leftarrow Visited \cup \{s'\}$; $s \leftarrow s'$

s = gate1

gate1 is a goal,
so return π

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

Visited = {on_ship, at_harbor, parking1, gate1}



Find-Acyclic-Solution

```
Find-Acyclic-Solution (Σ, s_0, S_g)
    π ← ∅
    Frontier ← {s_0}
    for every s ∈ Frontier \ S_g do
        Frontier ← Frontier \ {s}
        if Applicable(s) = ∅ then return failure
        nondeterministically choose a ∈ Applicable(s)
        π ← π ∪ (s, a)
        Frontier ← Frontier ∪ (γ(s, a) \ Dom(π))
        if has-loops(π, a, Frontier) then return failure
    return π
```

Keep track of unexpanded states, as in A*

Add all states s such that $\pi(s)$ isn't already defined

Check for cycles:

- Does $\gamma(s,a)$ include a state s' that is a π-ancestor of s?
  - for each $s' \in \gamma(s,a) \cap \mathrm{Dom}(\pi)$, is $s \in \hat{\gamma}(s',\pi)$?

Example

Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$Frontier \leftarrow \{s_0\}$

for every $s \in Frontier \setminus S_g$ do

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

  if Applicable$(s) = \varnothing$ then return failure

  nondeterministically choose $a \in$ Applicable(s)

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

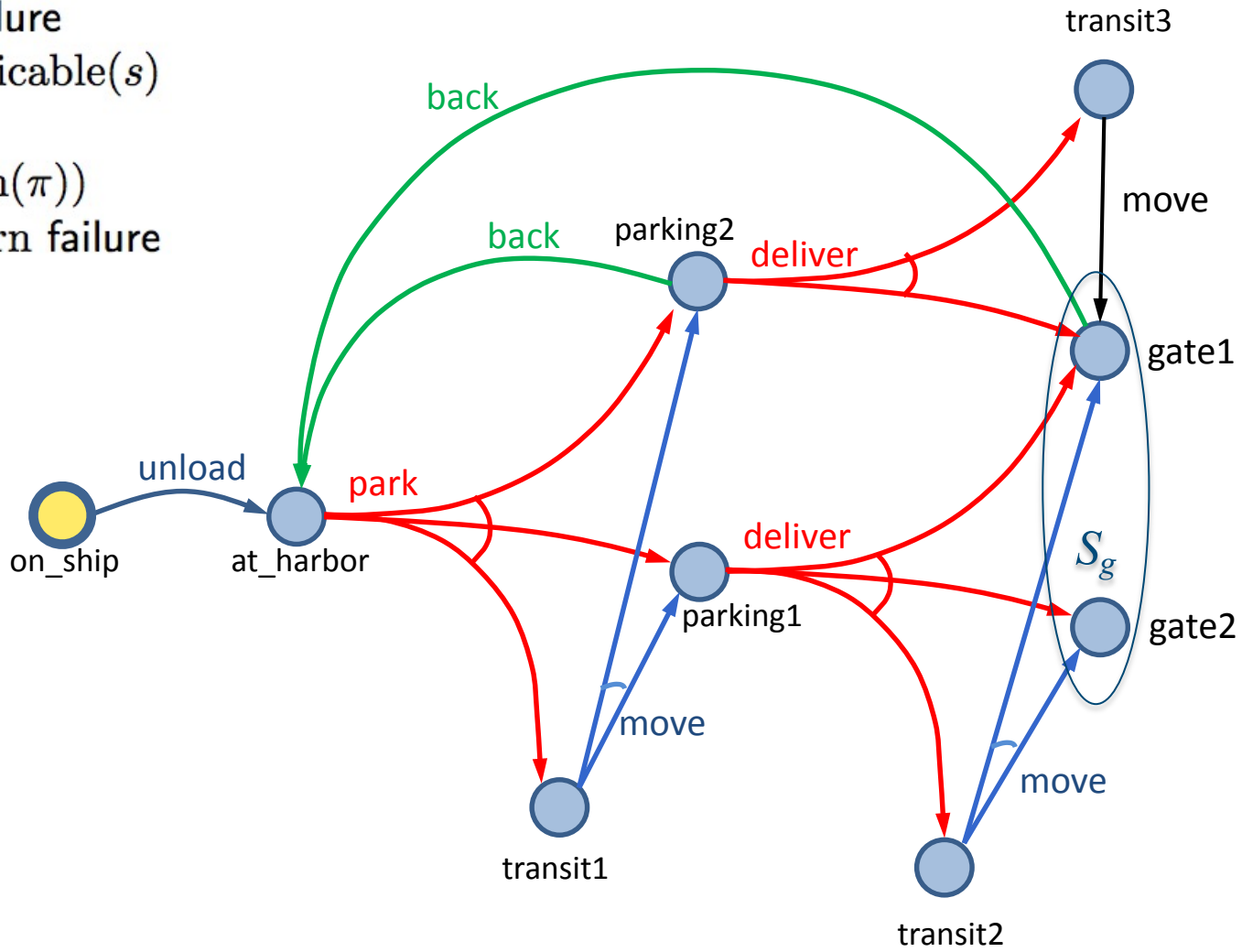
  $Frontier \leftarrow Frontier \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$

  if has-loops$(\pi, a, Frontier)$ then return failure

return π

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

$\pi = \{\}$



Example

Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$Frontier \leftarrow \{s_0\}$

for every $s \in Frontier \setminus S_g$ do

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

  if Applicable$(s) = \varnothing$ then return failure

  nondeterministically choose $a \in$ Applicable(s)

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

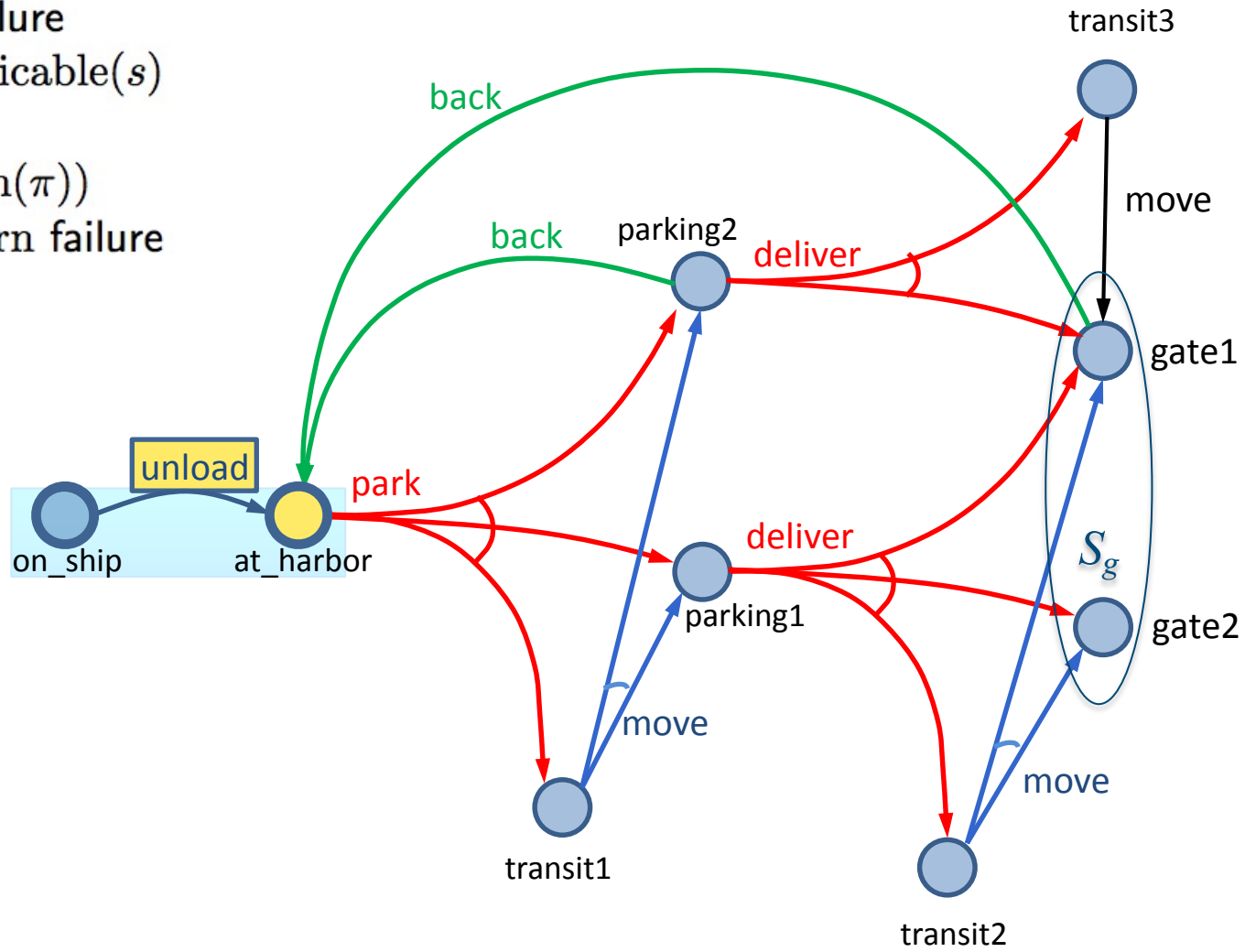
  $Frontier \leftarrow Frontier \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$

  if has-loops$(\pi, a, Frontier)$ then return failure

return π

$Frontier = \{$at_harbor$\}$

$\pi = \{$(on_ship, unload)$\}$



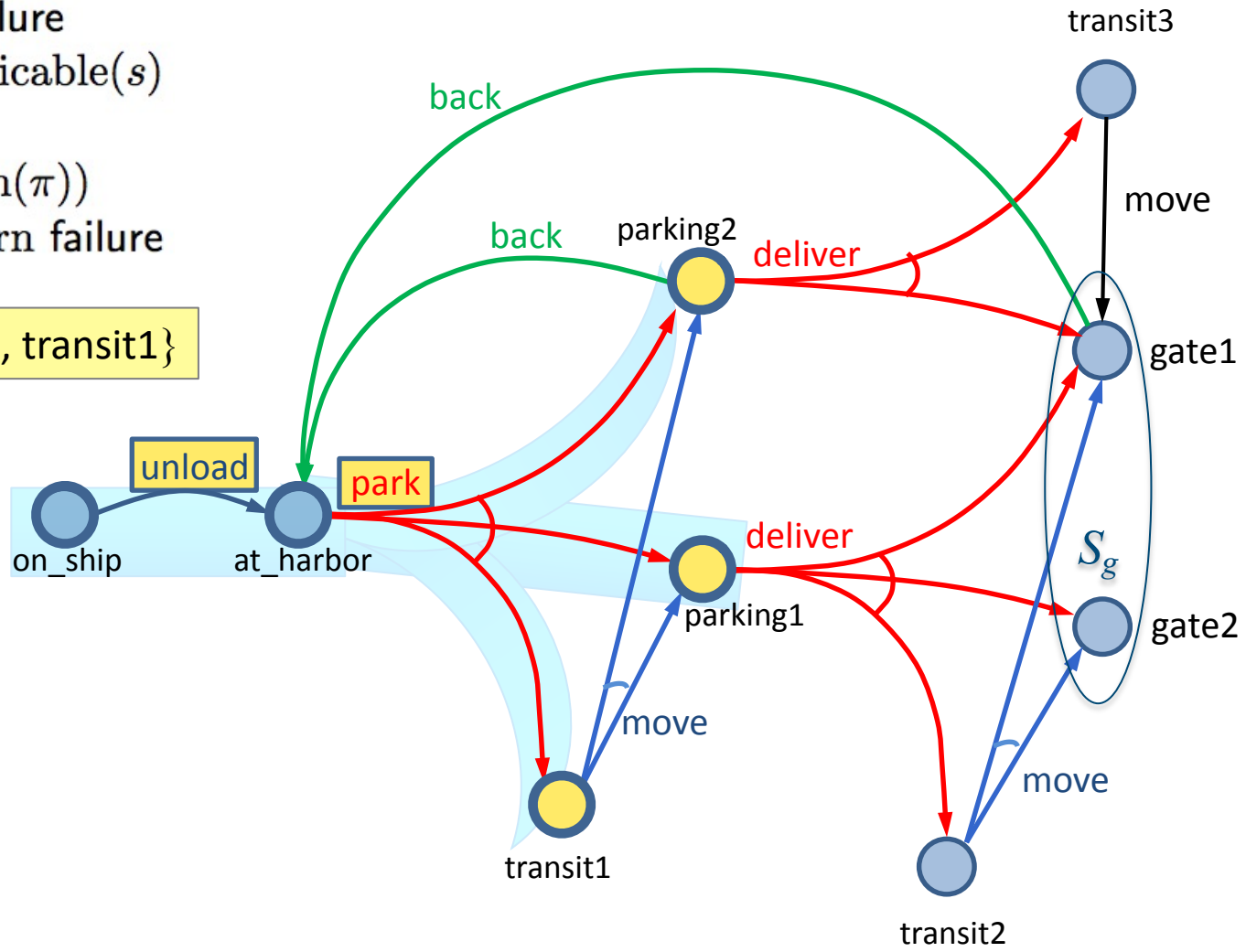
Example

Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$
$Frontier \leftarrow \{s_0\}$
for every $s \in Frontier \setminus S_g$ do
  $Frontier \leftarrow Frontier \setminus \{s\}$
  if $\text{Applicable}(s) = \varnothing$ then return failure
  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$(\pi, a, Frontier)$ then return failure
return π

$Frontier = \{$parking1, parking2, transit1$\}$

$\pi = \{$(on_ship, unload), (at_harbor, park)$\}$



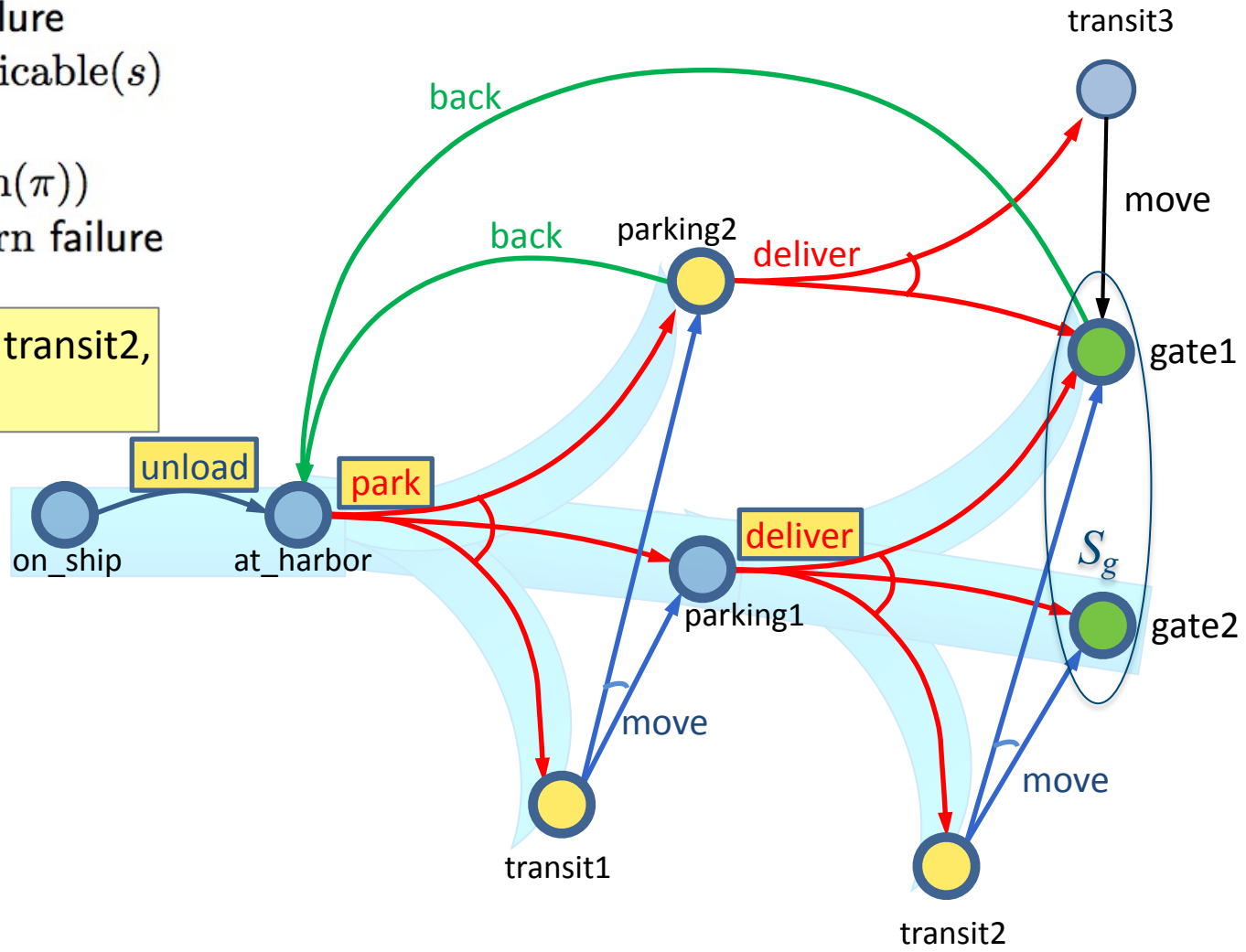
Example

Find-Acyclic-Solution (Σ, s_0, S_g)

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

$Frontier$ = {parking2, transit1, transit2, gate1, gate2}

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



Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$Frontier \leftarrow \{s_0\}$

for every $s \in Frontier \setminus S_g$ do

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

  if Applicable$(s) = \varnothing$ then return failure

  nondeterministically choose $a \in$ Applicable(s)

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

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

  if has-loops$(\pi, a, Frontier)$ then return failure

return π

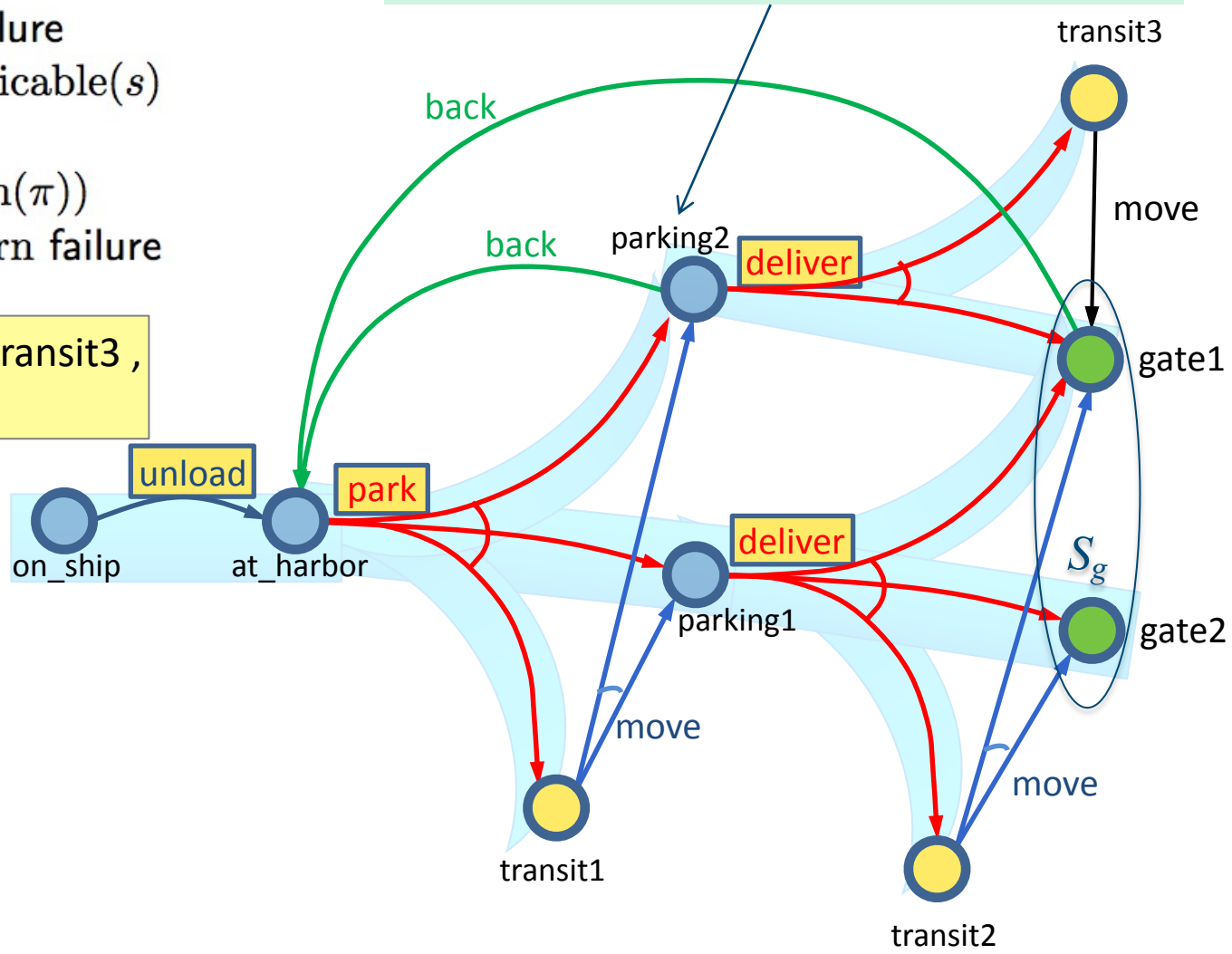
$Frontier$ = {transit1, transit2, transit3 , gate1, gate2}

π = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, deliver)}

Example

nondeterministically choose back or deliver

- back ⇒ cycle, so return failure
- deliver ⇒ no cycle, so continue



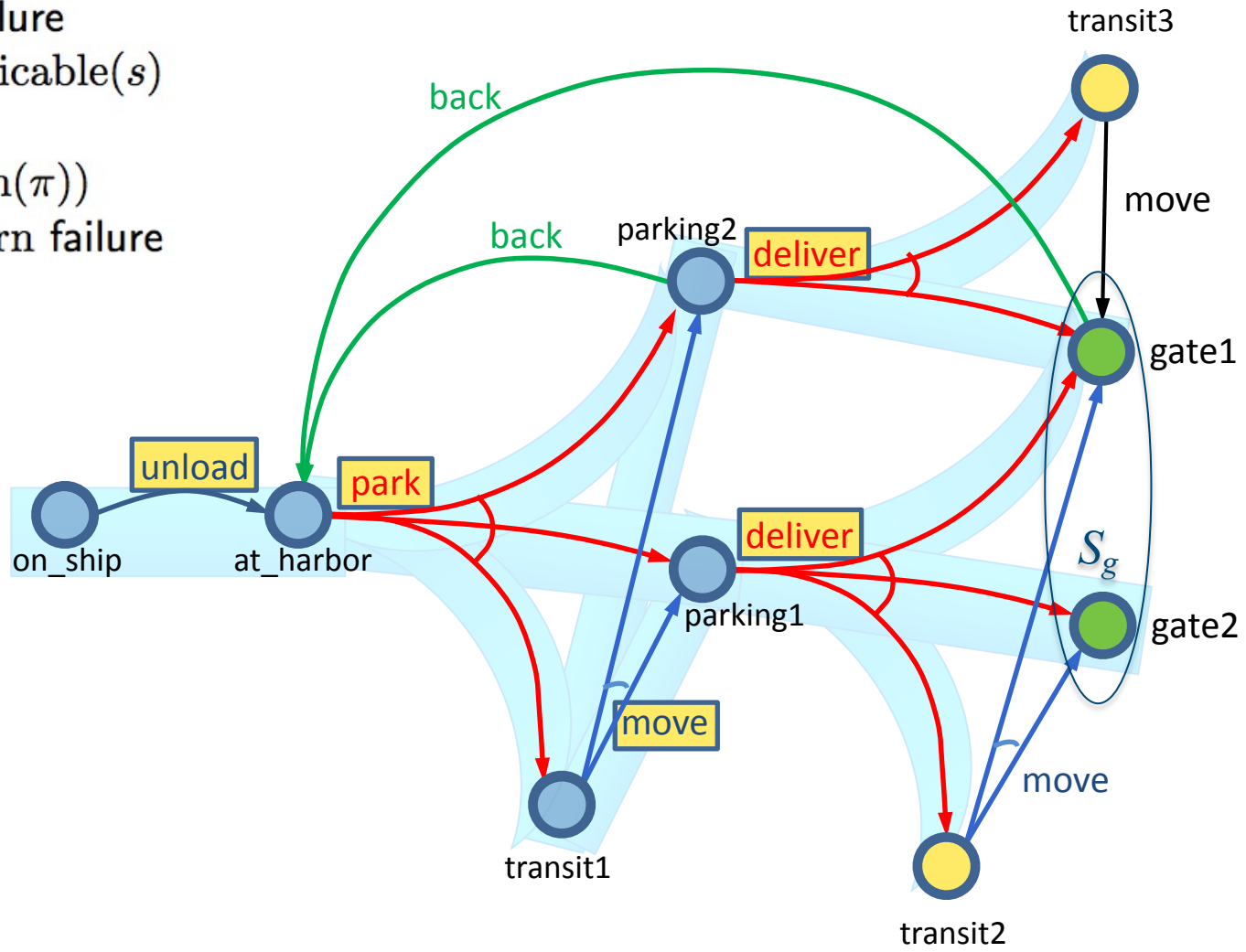
Example

Find-Acyclic-Solution (Σ, s_0, S_g)

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

$Frontier$ = {transit2, transit3, gate1, gate2}

π = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, deliver), (transit1, move)}



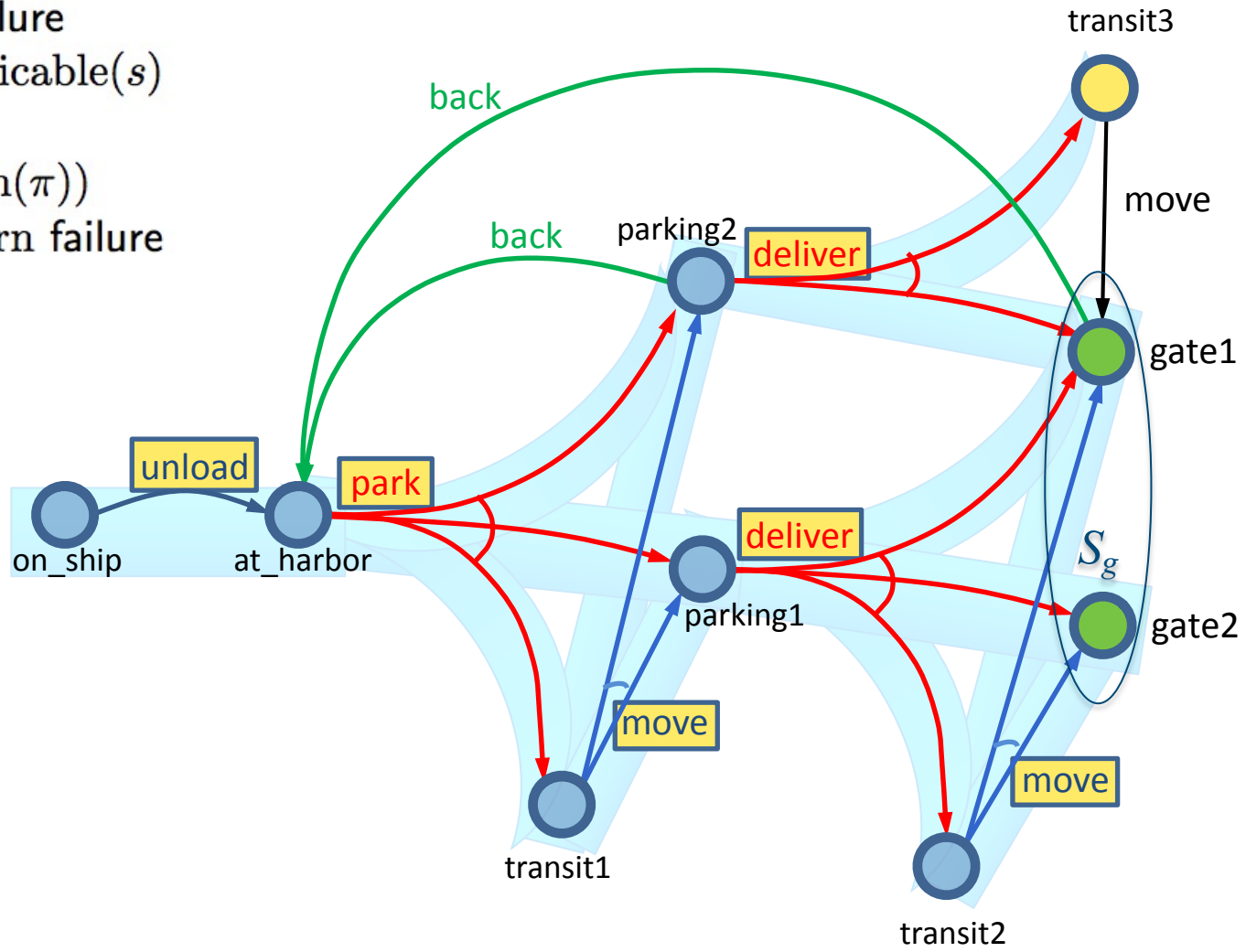
Example

Find-Acyclic-Solution (Σ, s_0, S_g)

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

$Frontier$ = {transit3, gate1, gate2}

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



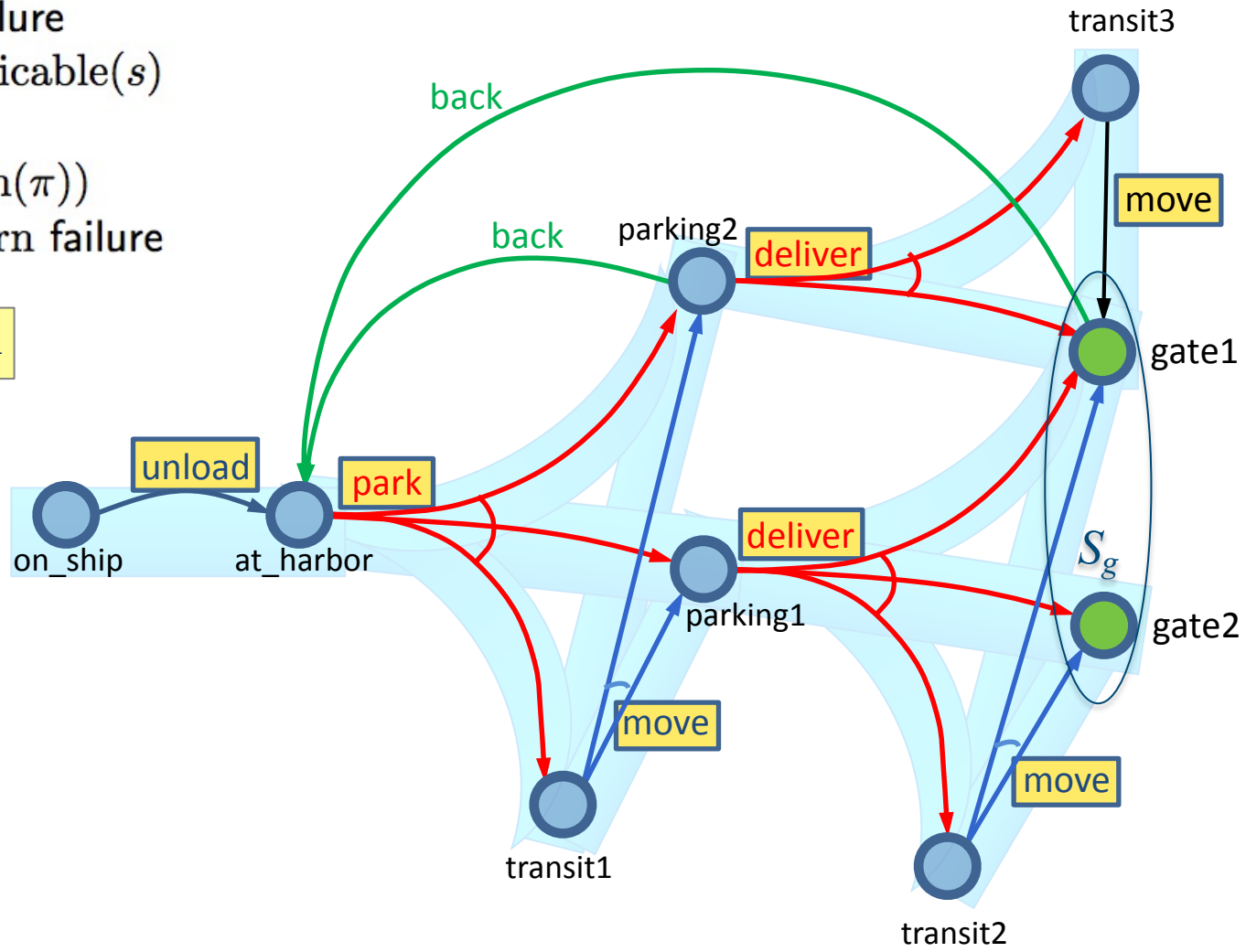
Example

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

$Frontier = \{\text{gate1, gate2}\} \subseteq S_g$

Found a solution

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



Find-Safe-Solution

Find-Safe-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$
  $Frontier \leftarrow \{s_0\}$
  for every $s \in Frontier \setminus S_g$ do
    $Frontier \leftarrow Frontier \setminus \{s\}$
    if $\text{Applicable}(s) = \varnothing$ then return failure
    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$(\pi, a, Frontier)$ then return failure
  return π

Like Find-Acyclic-Solution except here:

Check for *unsafe* cycles:

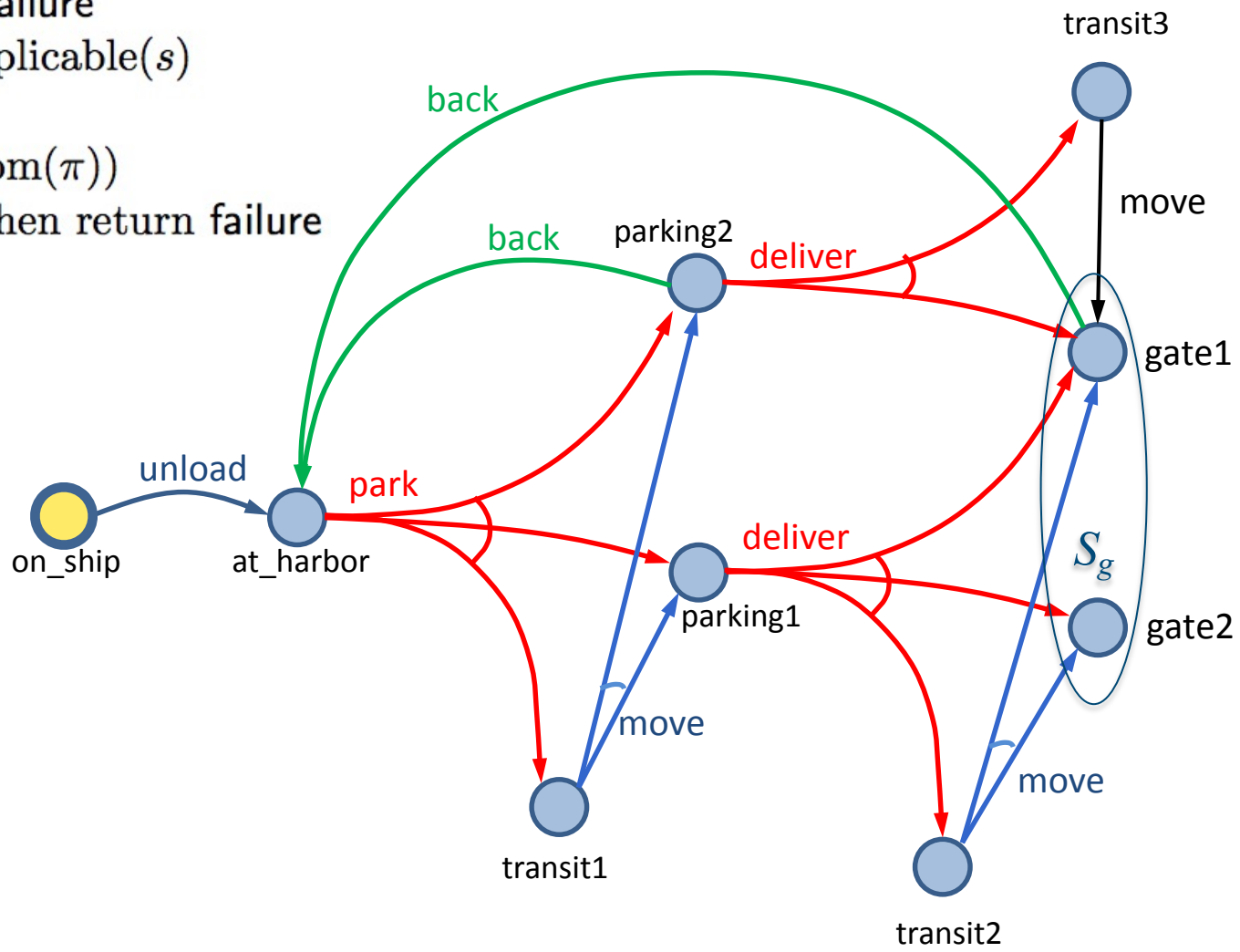
- Does $\gamma(s,a)$ include a state s' from which π can't take us to the frontier?
  - For each $s' \in \gamma(s,a) \cap \text{Dom}(\pi)$, is $\hat{\gamma}(s',\pi) \cap Frontier = \emptyset$?
- If so, then π contains a cycle that can't be escaped

Example

Find-Safe-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$
  $Frontier \leftarrow \{s_0\}$
  for every $s \in Frontier \setminus S_g$ do
    $Frontier \leftarrow Frontier \setminus \{s\}$
    if $\text{Applicable}(s) = \varnothing$ then return failure
    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$(\pi, a, Frontier)$ then return failure
  return π

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

$\pi = \{\}$

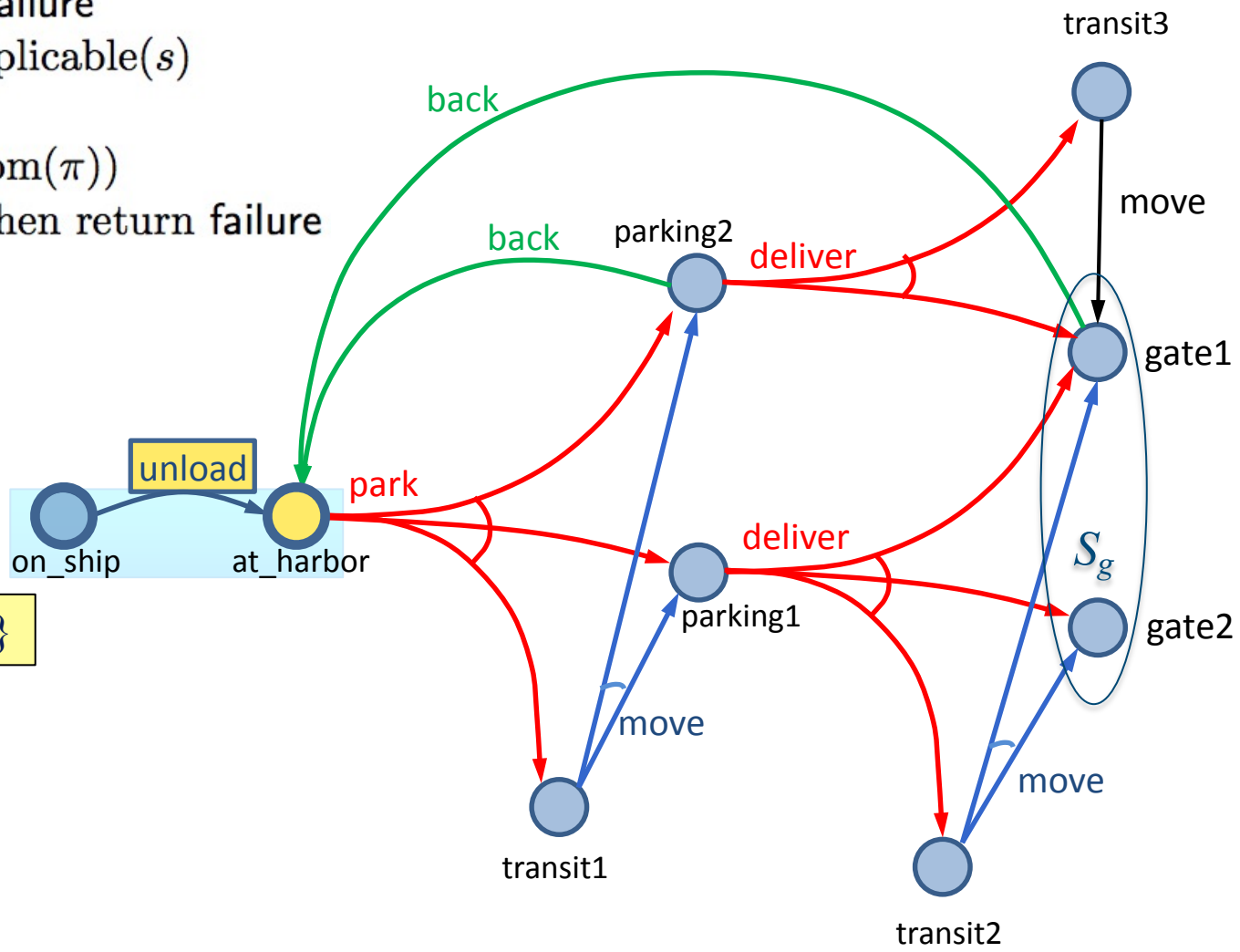


Example

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

$Frontier = \{$at_harbor$\}$

$\pi = \{$(on_ship, unload)$\}$

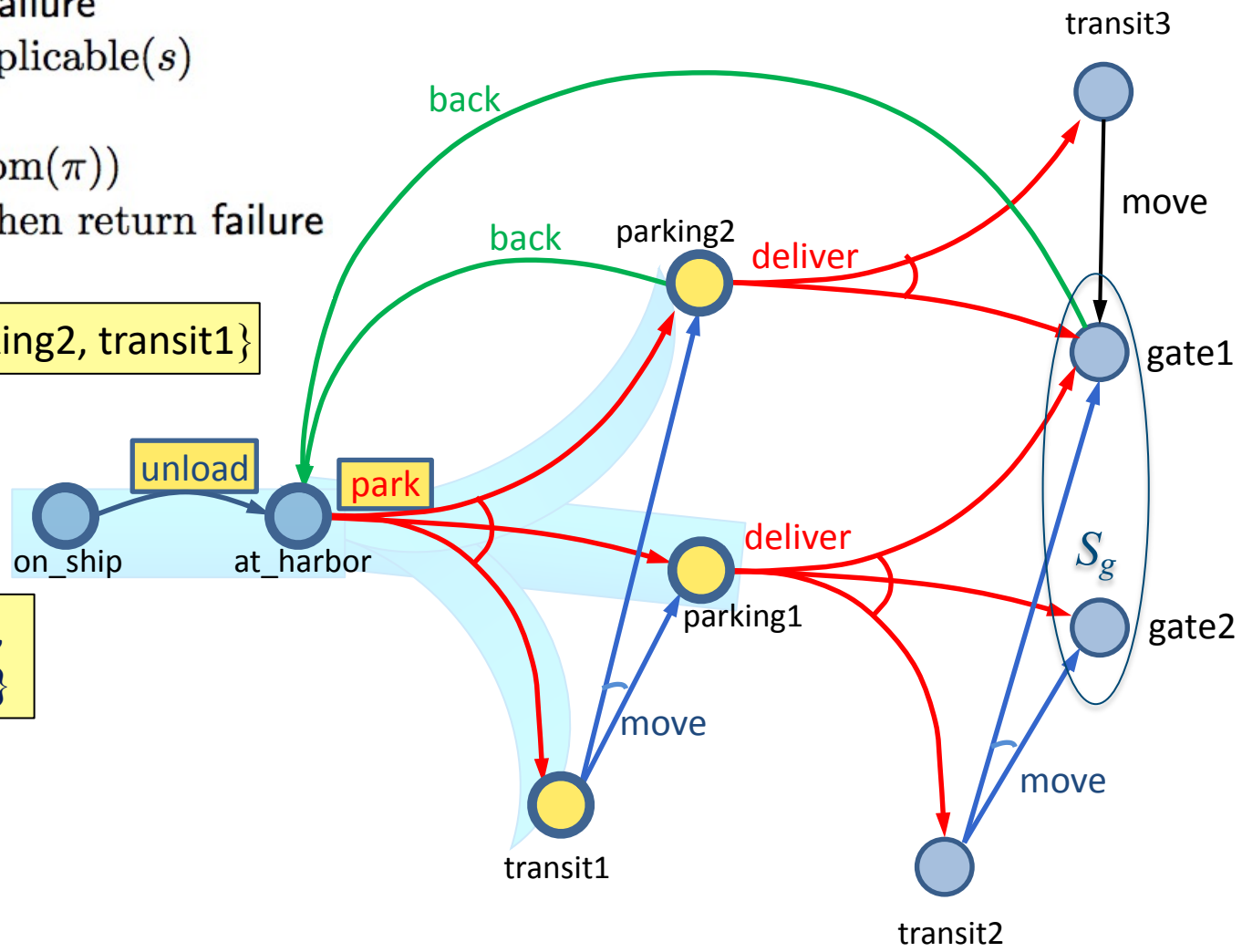


Example

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

$Frontier = \{$parking1, parking2, transit1$\}$

$\pi = \{$(on_ship, unload), (at_harbor, park)$\}$

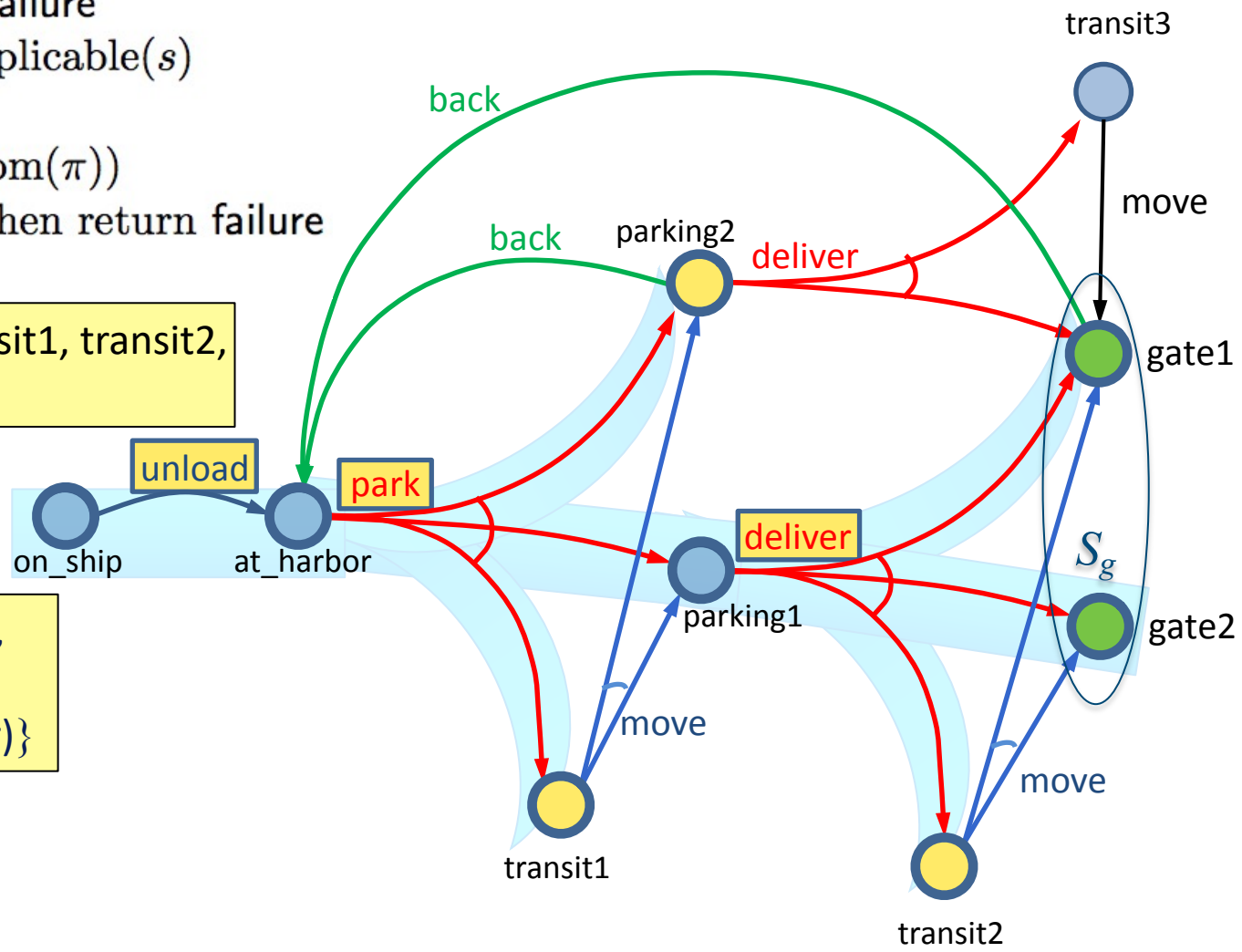


Example

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

$Frontier$ = {parking2, transit1, transit2, gate1, gate2}

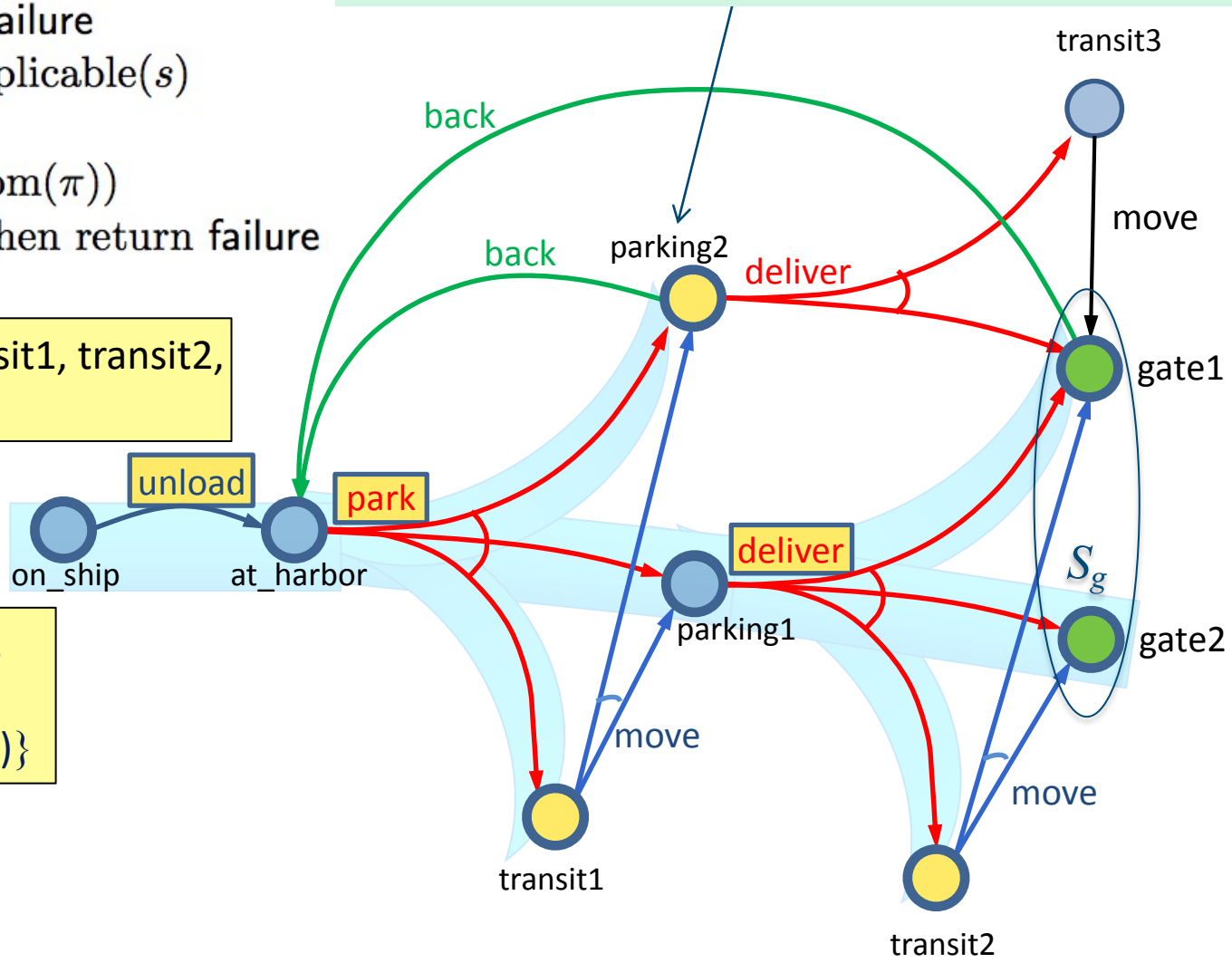
π = {(on_ship, unload), (at_harbor, park), (parking1, deliver)}



Example

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

Nondeterministically choose either back or deliver
- back is OK because cycle is escapable



$Frontier$ = {parking2, transit1, transit2, gate1, gate2}

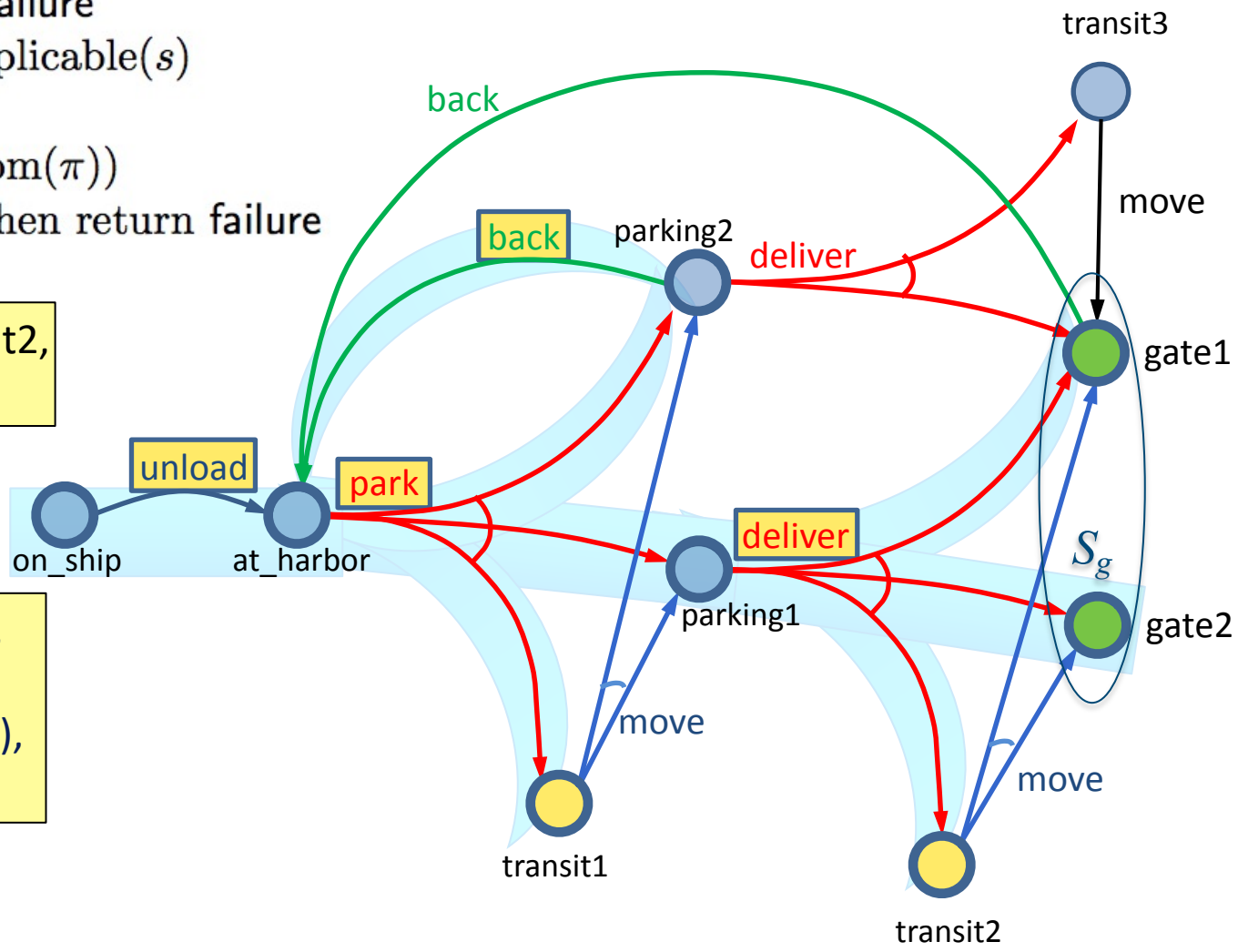
π = {(on_ship, unload), (at_harbor, park), (parking1, deliver)}

Example

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

$Frontier$ = {transit1, transit2, gate1, gate2}

π = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, back)}



Example

Find-Safe-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$Frontier \leftarrow \{s_0\}$

for every $s \in Frontier \setminus S_g$ do

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

  if Applicable$(s) = \varnothing$ then return failure

  nondeterministically choose $a \in$ Applicable(s)

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

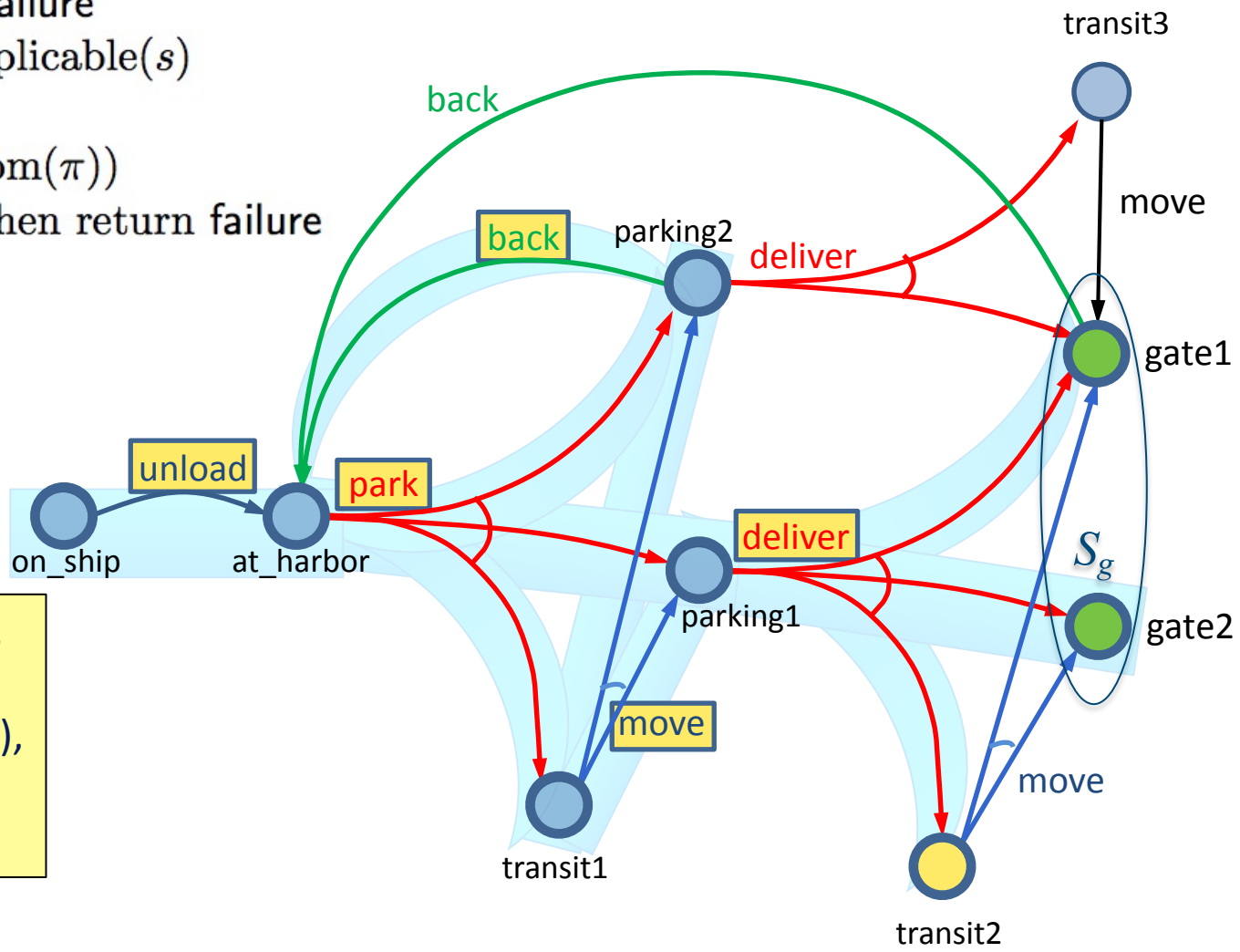
  $Frontier \leftarrow Frontier \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$

  if has-unsafe-loops$(\pi, a, Frontier)$ then return failure

return π

$Frontier = \{$transit2, gate1, gate2$\}$

$\pi = \{$(on_ship, unload), (at_harbor, park), (parking1, deliver), (parking2, back), (transit1, move)$\}$



Example

Find-Safe-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$Frontier \leftarrow \{s_0\}$

for every $s \in Frontier \setminus S_g$ do

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

  if $\text{Applicable}(s) = \varnothing$ then return failure

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

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

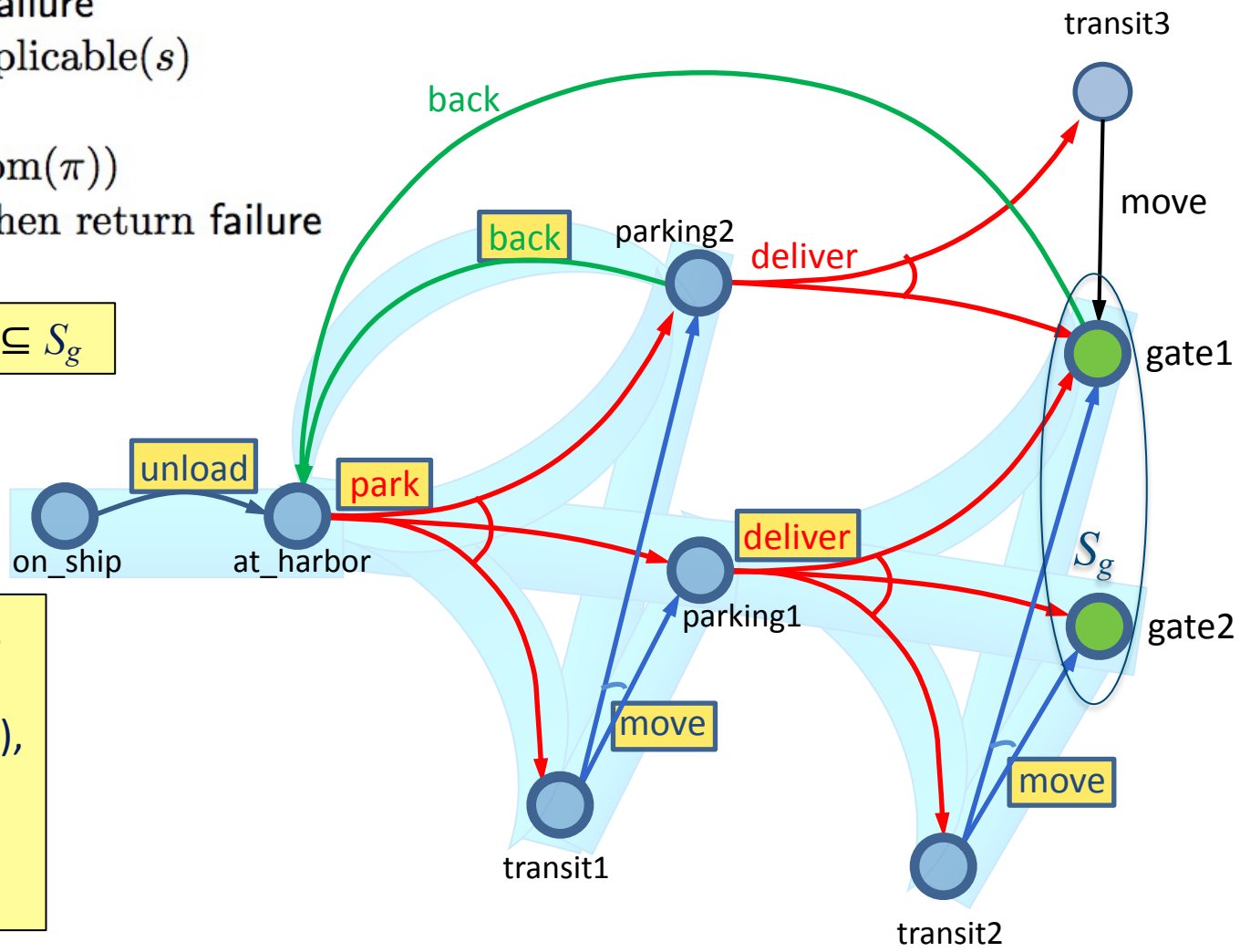
  $Frontier \leftarrow Frontier \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$

  if has-unsafe-loops$(\pi, a, Frontier)$ then return failure

return π

$Frontier = \{\text{gate1, gate2}\} \subseteq S_g$

$\pi = \{$(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, back),
(transit1, move),
(transit2, move)$\}$



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 (possibly unsafe) solution π
    - For each each leaf node of π
      - If the leaf node isn't a goal,
        - find a (possibly unsafe) solution and incorporate it into π

Guided-Find-Safe-Solution

Guided-Find-Safe-Solution (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  loop
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s,a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select arbitrarily $s \in Q$
    $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
    if $\pi' \neq$ failure then do
      $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make a not applicable in s'

π is a solution. Return the part that's reachable from s_0.

Choose any leaf s that isn't a goal. Find a (possibly unsafe) solution π' for s.

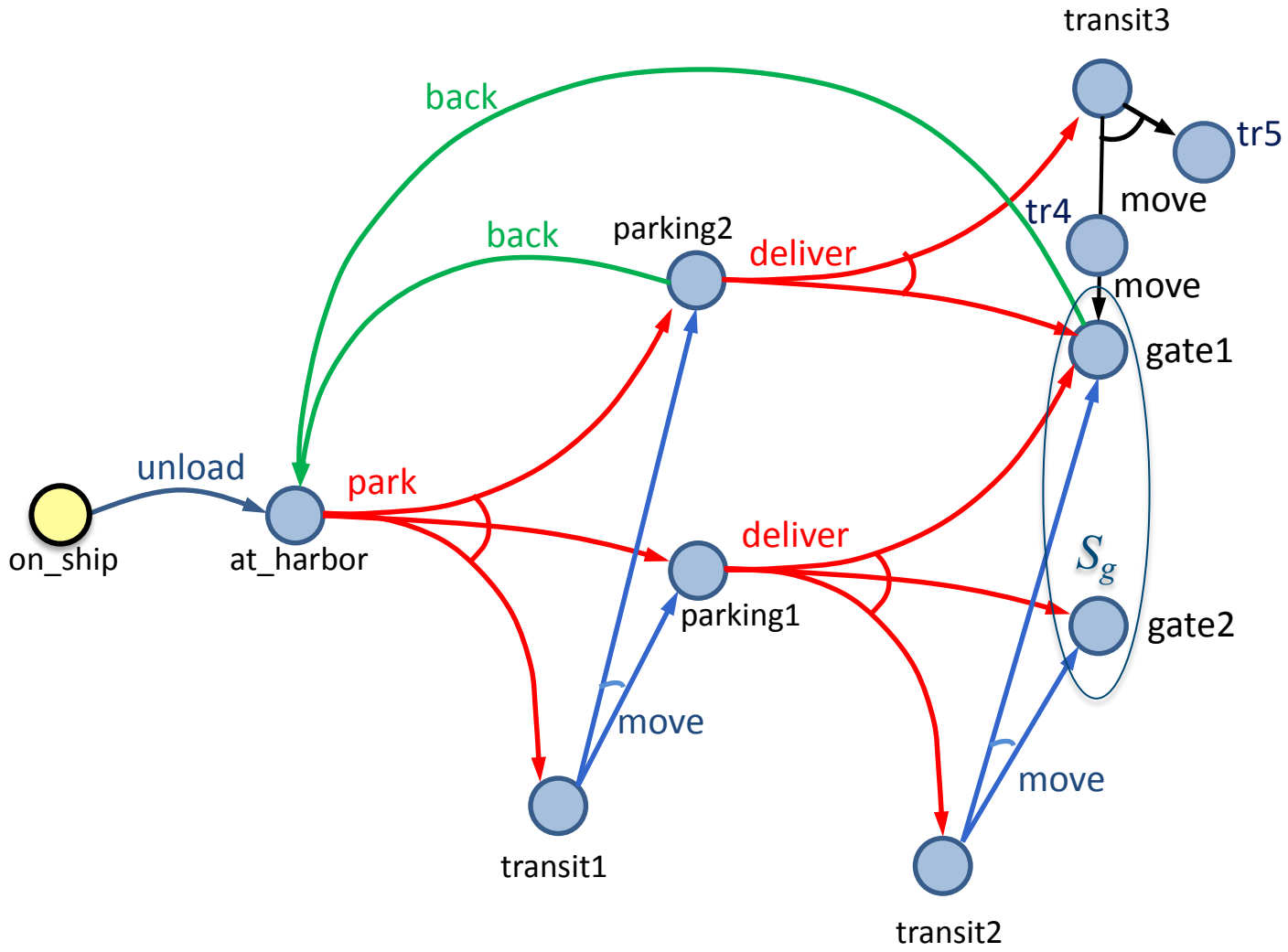
For each (s,a) in π', add to π unless π already has an action at s

s is unsolvable. For each (s',a) that can produce s, modify π and Σ so we'll never use a at s'

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

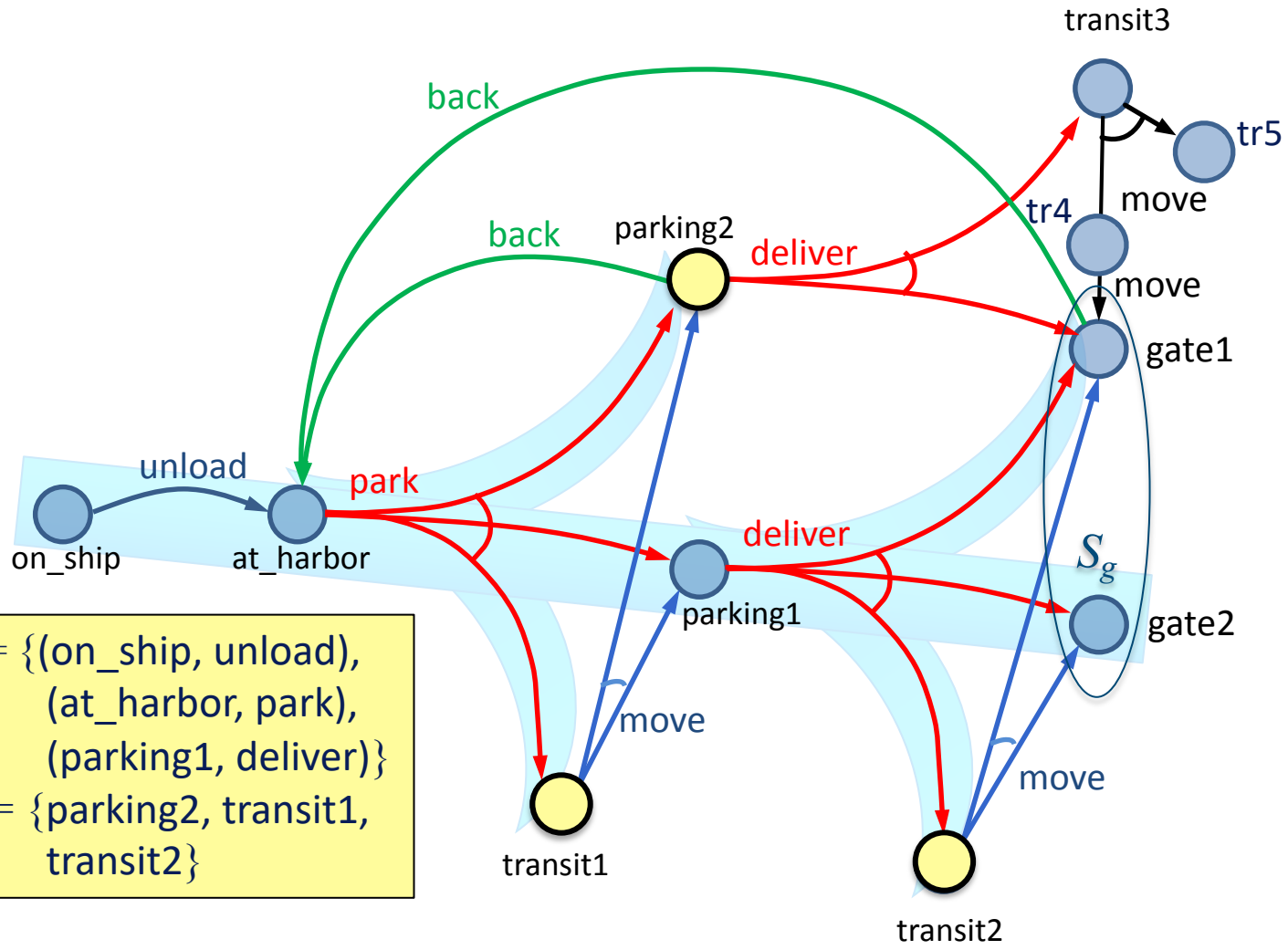
Example



Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

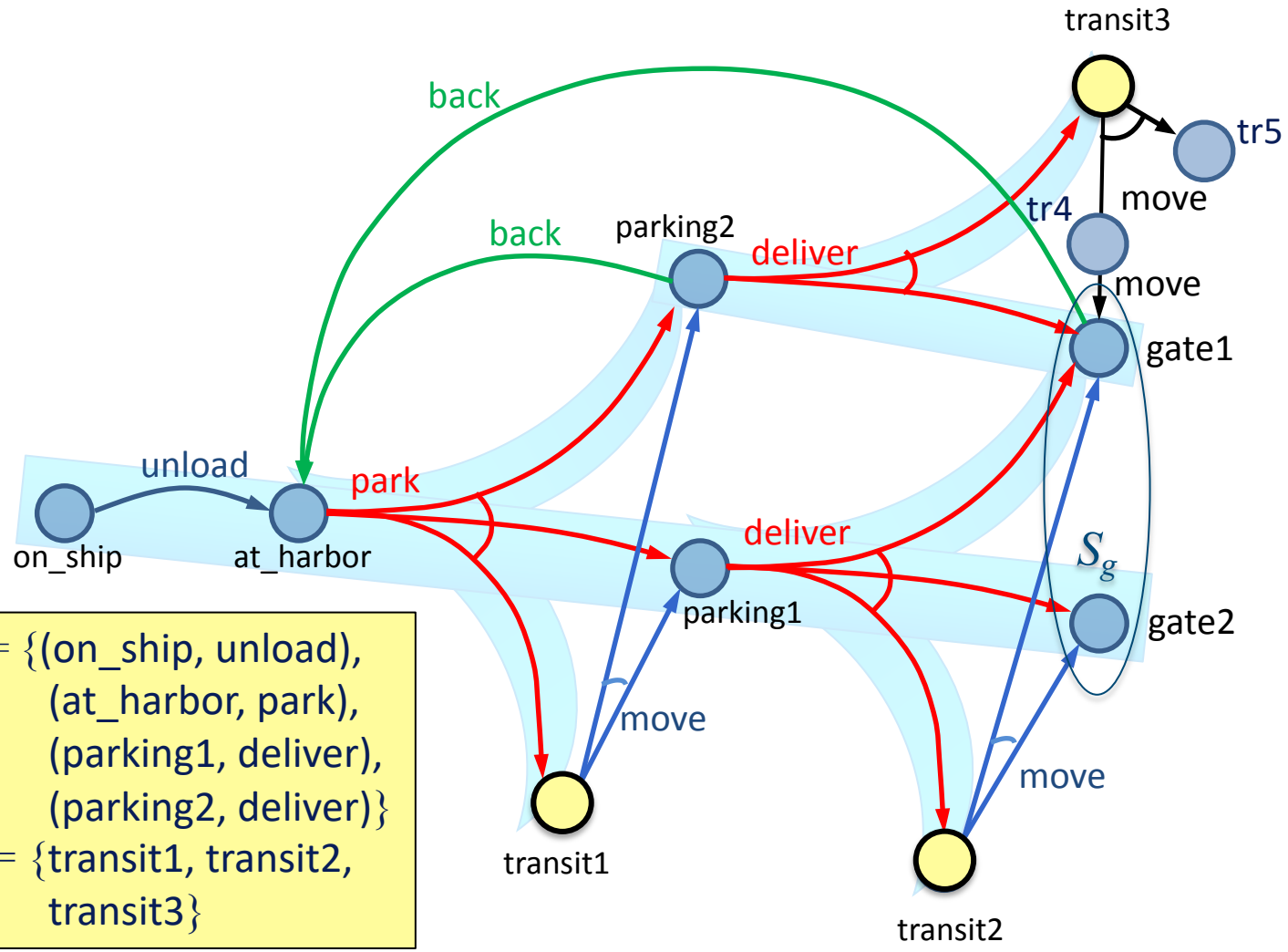


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver)}
Q = {parking2, transit1,
transit2}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

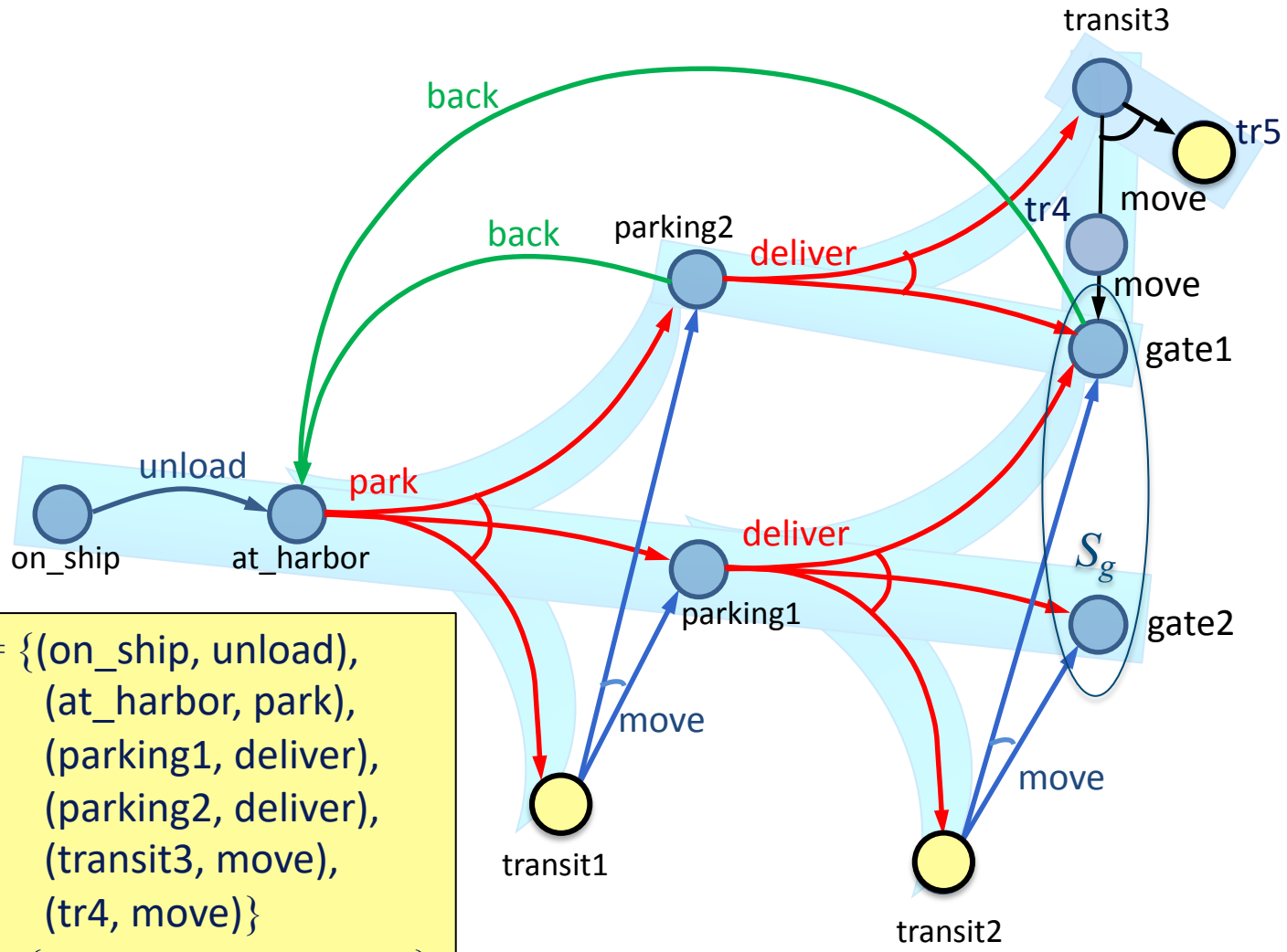


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver)}
Q = {transit1, transit2,
transit3}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return($\varnothing$)
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

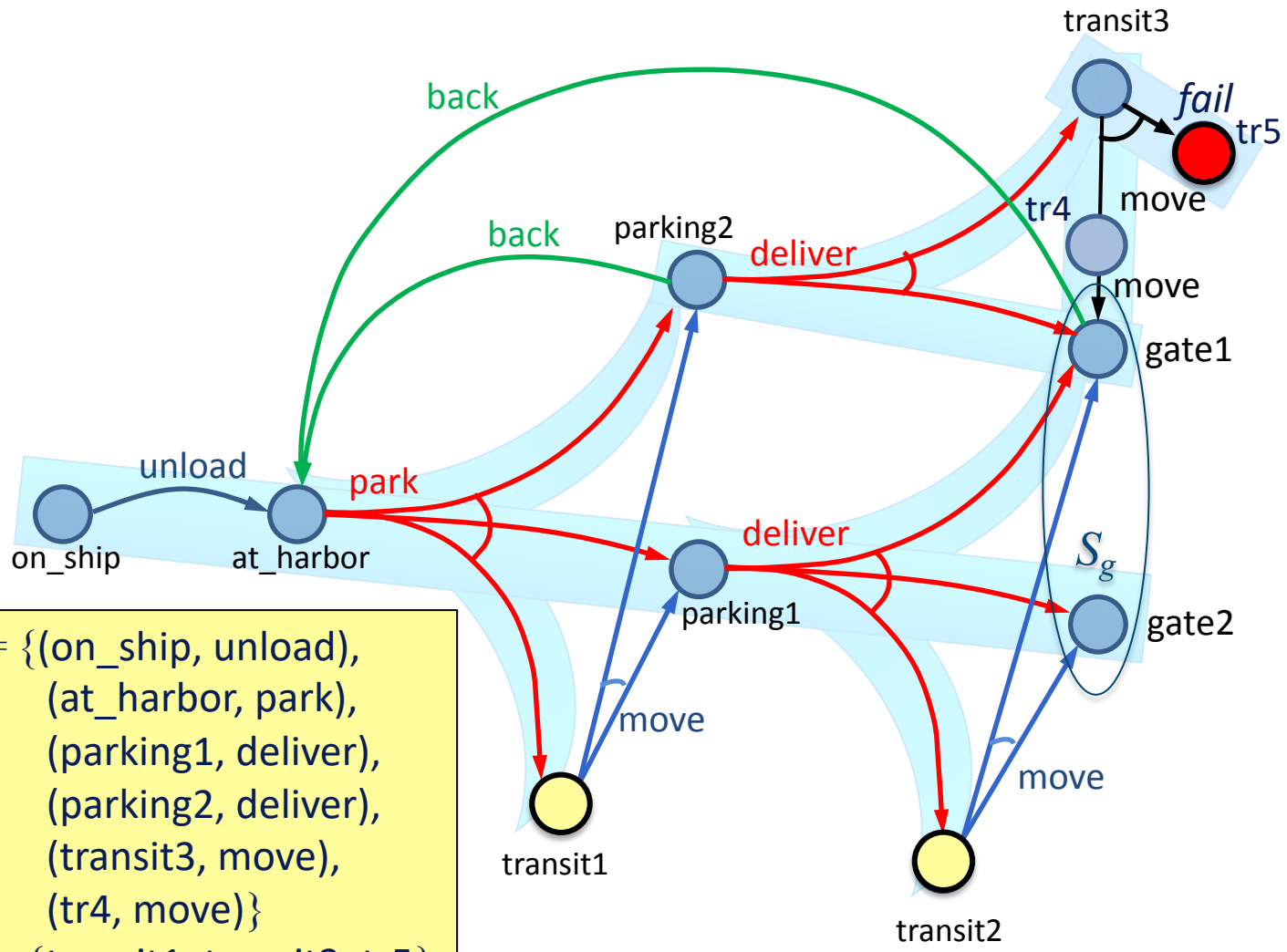


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver),
(transit3, move),
(tr4, move)}
Q = {transit1, transit2, tr5}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

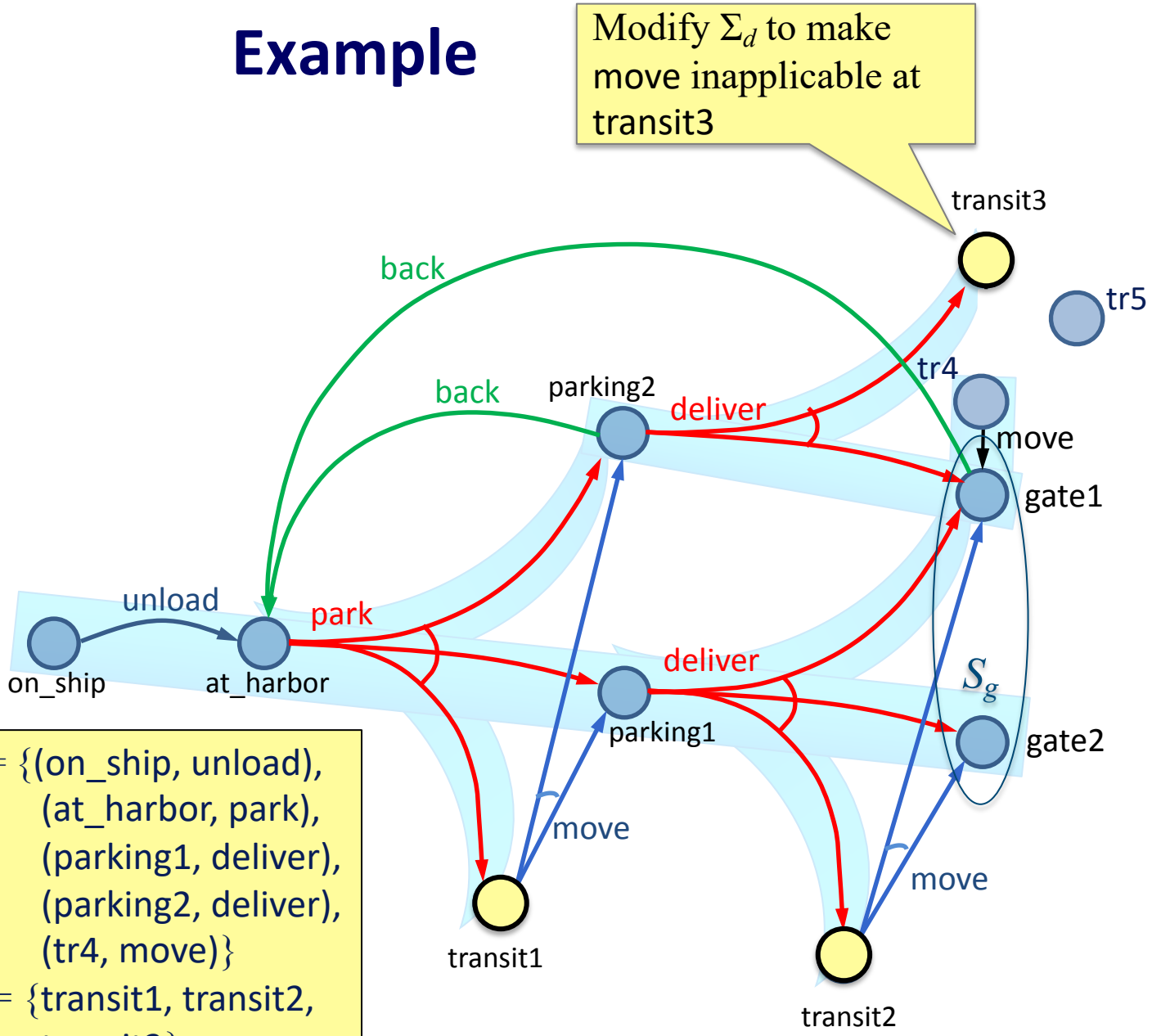


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver),
(transit3, move),
(tr4, move)}
Q = {transit1, transit2, tr5}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

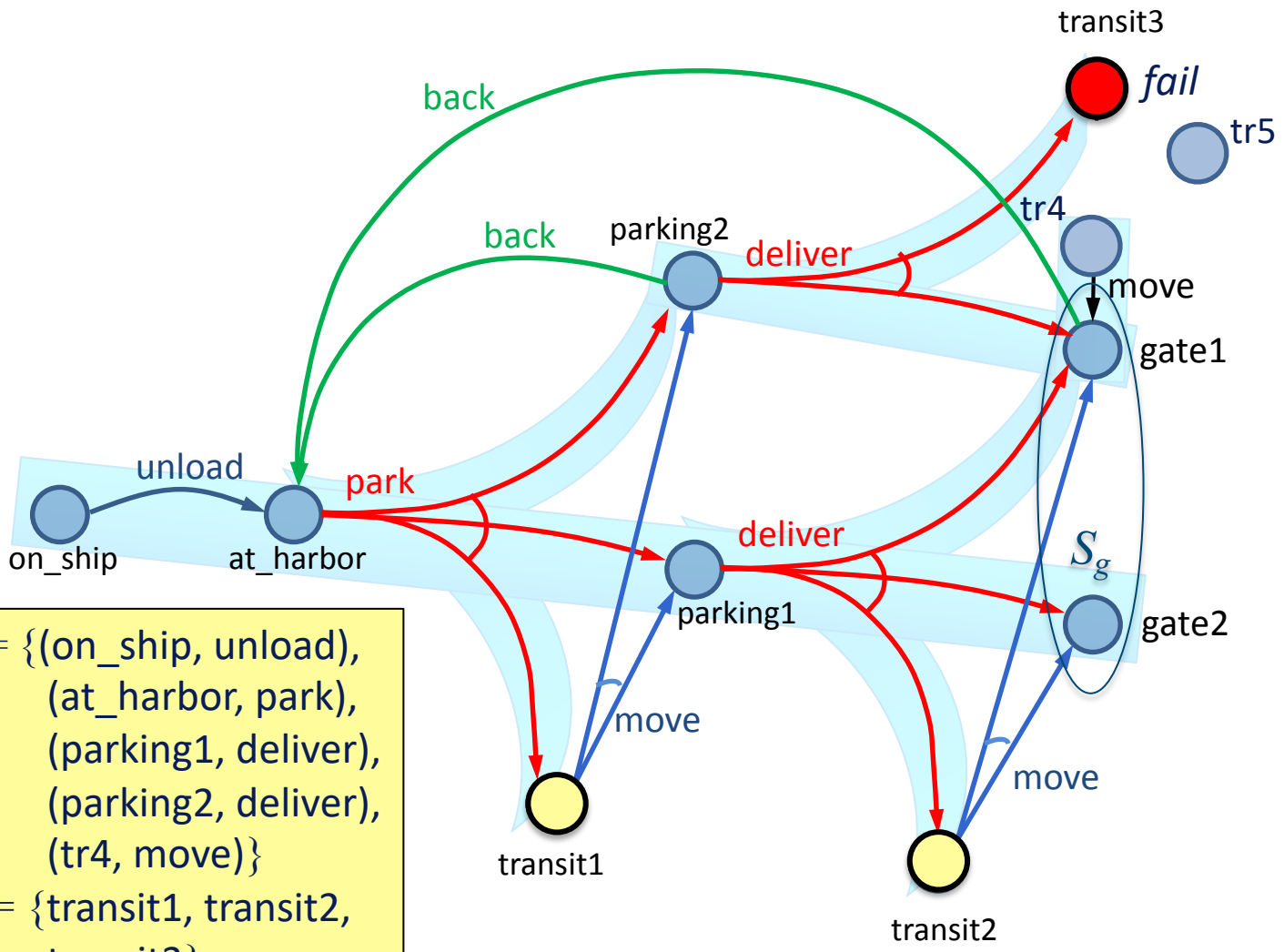


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver),
(tr4, move)}
Q = {transit1, transit2,
transit3}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return($\varnothing$)
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example



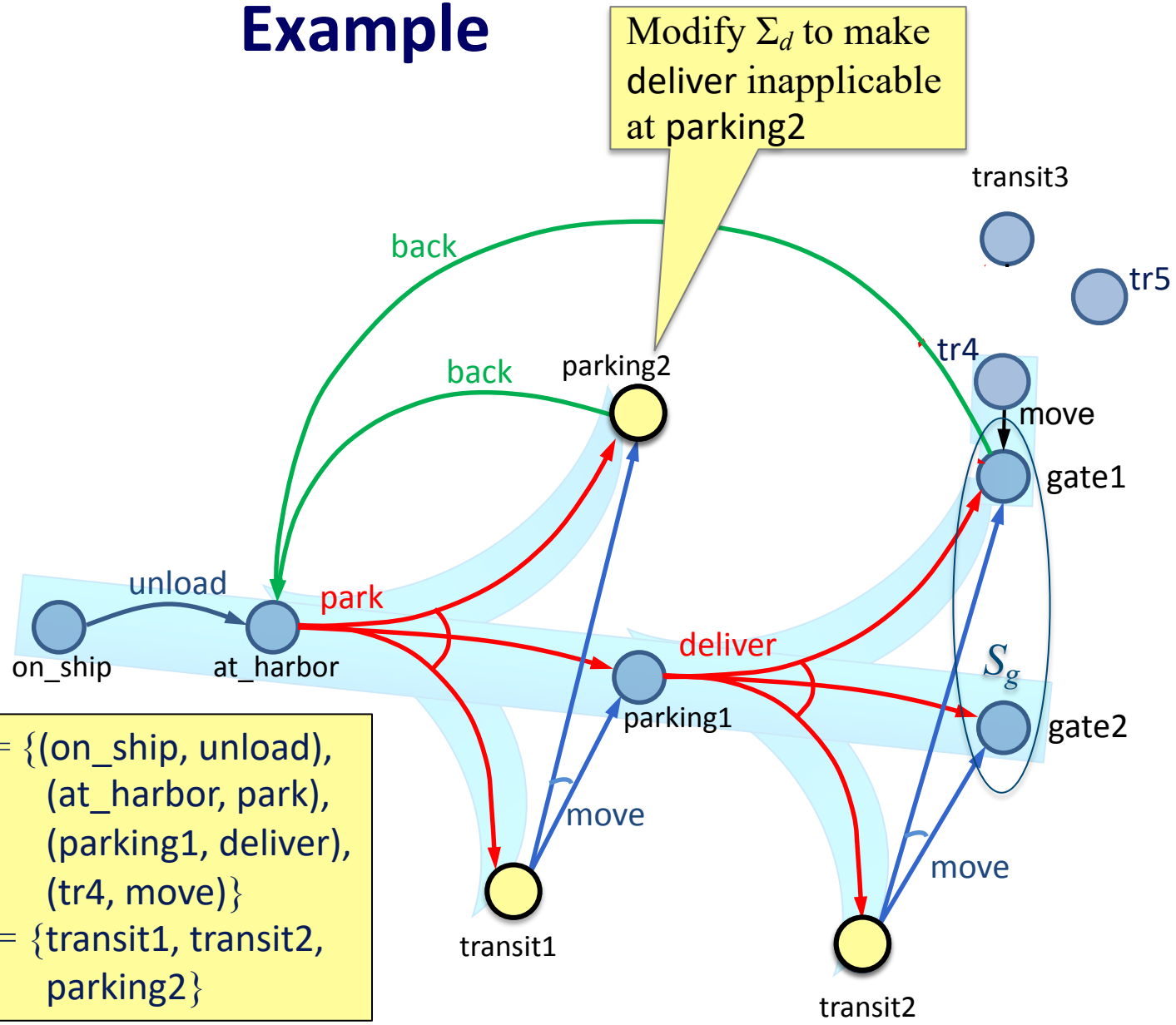
π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver),
(tr4, move)}

Q = {transit1, transit2,
transit3}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return($\varnothing$)
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

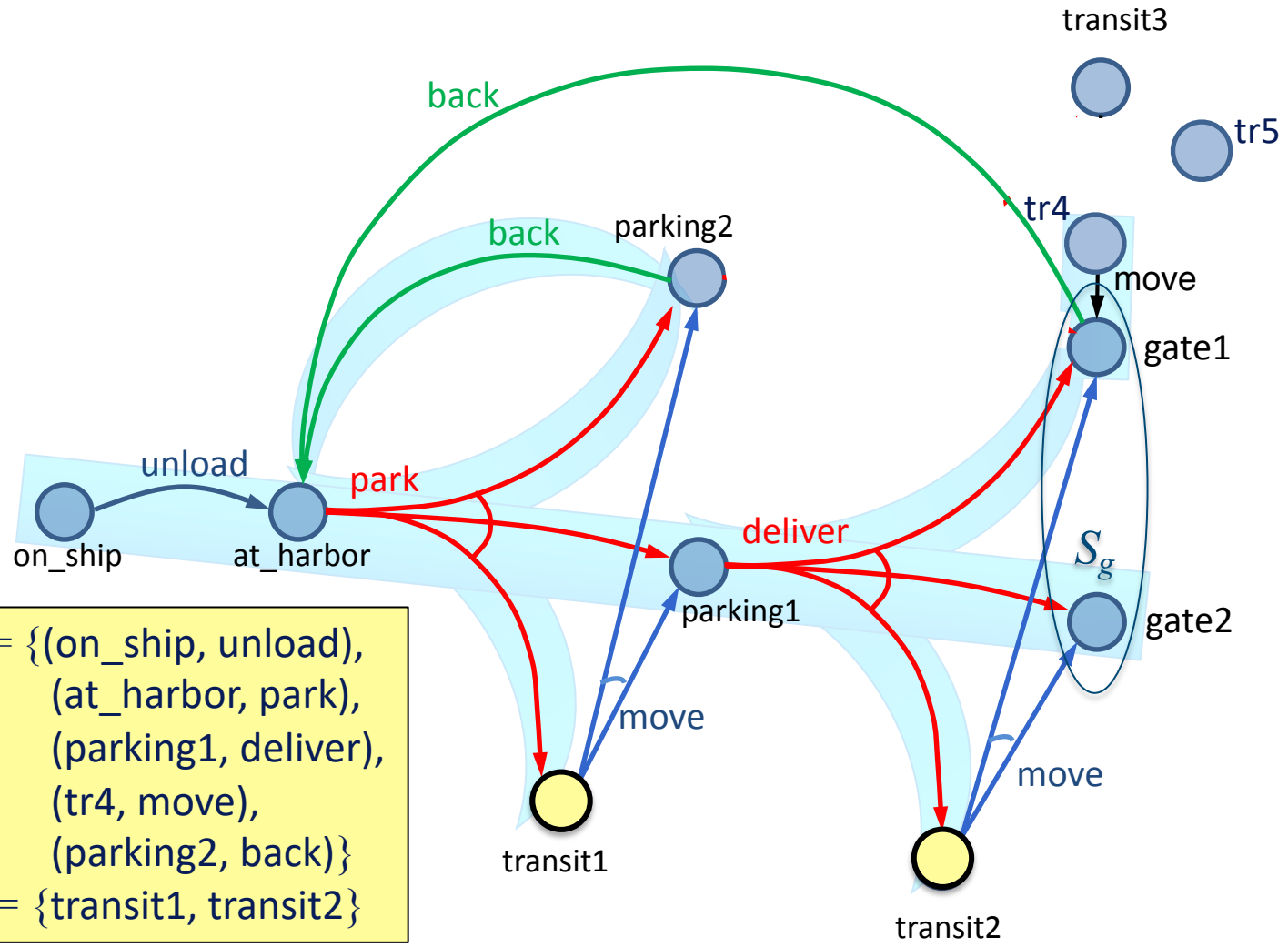


π = {(on_ship, unload), (at_harbor, park), (parking1, deliver), (tr4, move)}
Q = {transit1, transit2, parking2}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return($\varnothing$)
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

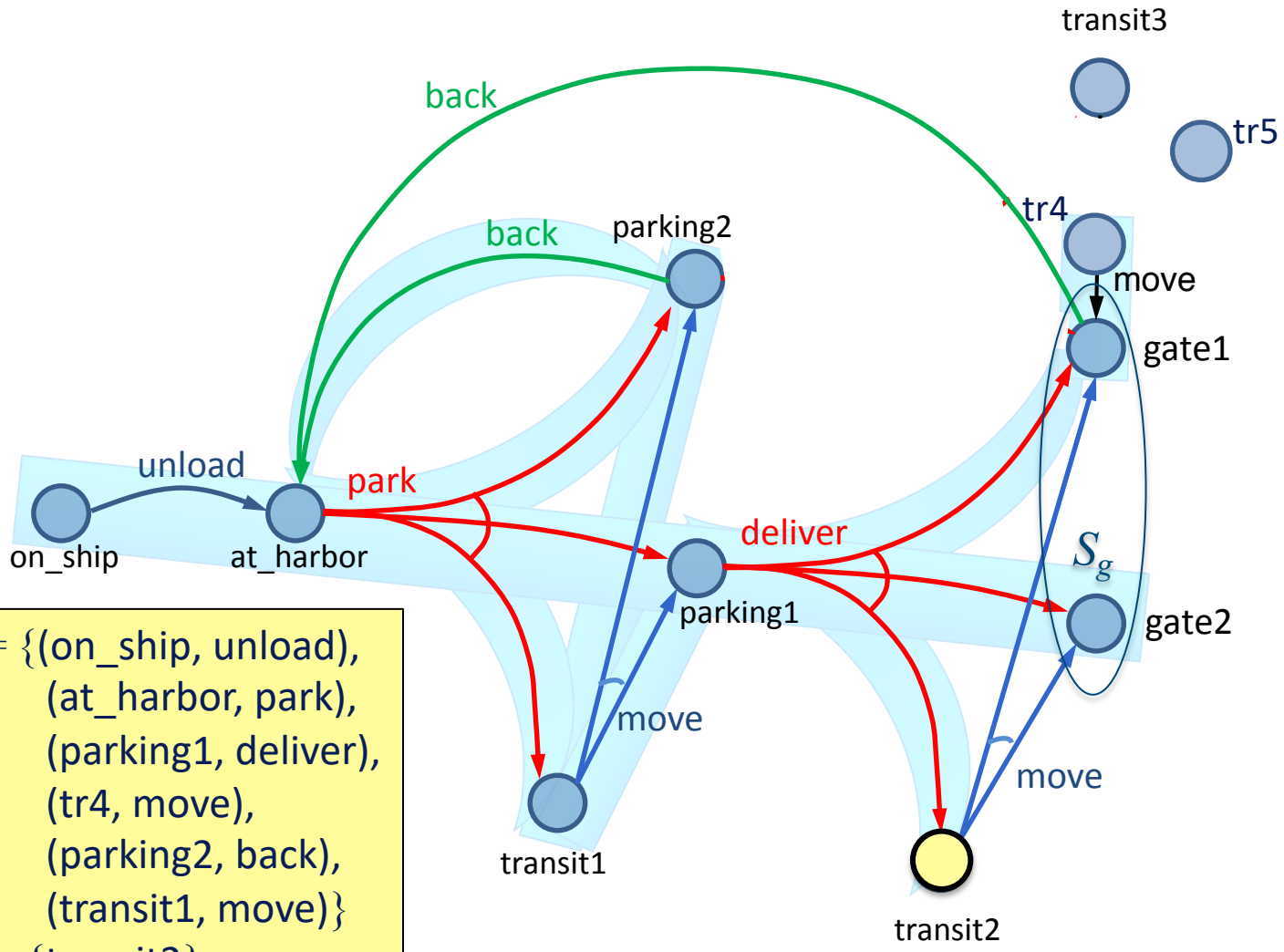


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(tr4, move),
(parking2, back)}
Q = {transit1, transit2}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

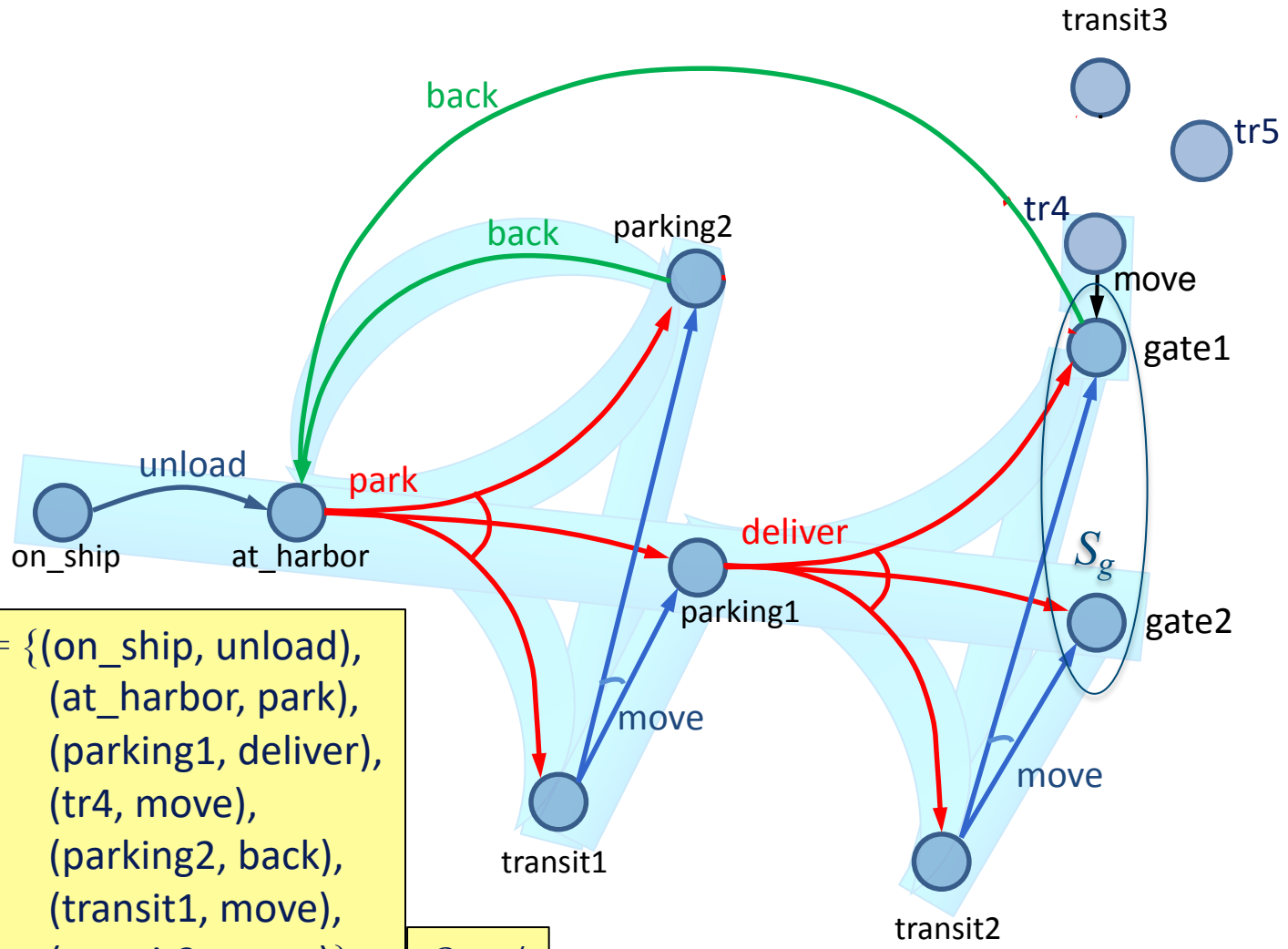


π = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(tr4, move),
(parking2, back),
(transit1, move)}
Q = {transit2}

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return($\varnothing$)
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example



$\pi = \{$(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(tr4, move),
(parking2, back),
(transit1, move),
(transit2, move)$\}$

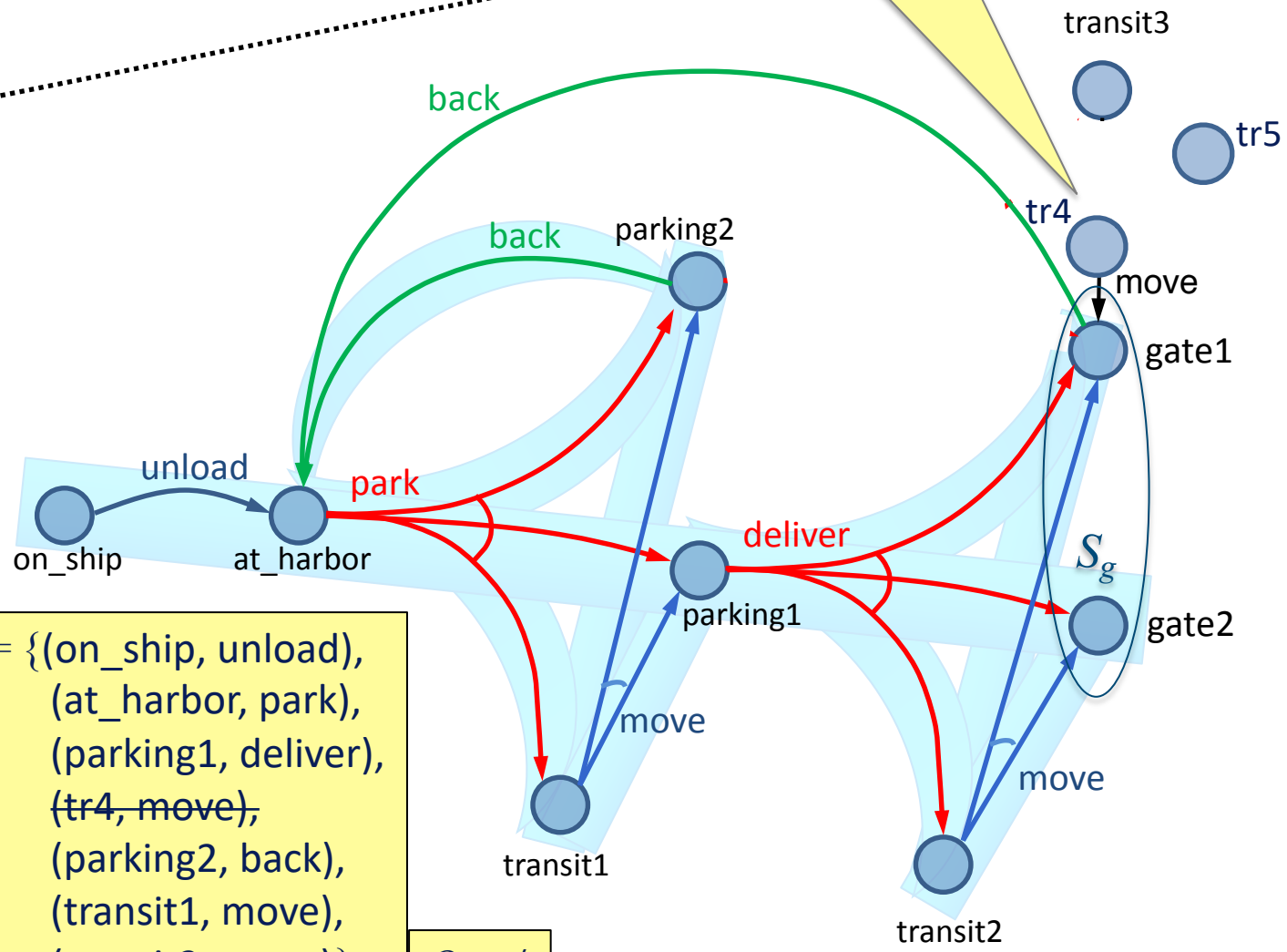
$Q = \emptyset$

Guided-Find-Safe-Solution (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
    $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    return(π)
  select arbitrarily $s \in Q$
  $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
  if $\pi' \neq$ failure then do
    $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
  else for every s' and a such that $s \in \gamma(s', a)$ do
    $\pi \leftarrow \pi \setminus \{(s', a)\}$
    make a not applicable in s'

Example

Remove (tr4,move) from π because π can't ever reach tr4



$\pi = \{$(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
~~(tr4, move),~~
(parking2, back),
(transit1, move),
(transit2, move)$\}$

$Q = \emptyset$

Guided-Find-Safe-Solution (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  loop
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select arbitrarily $s \in Q$
    $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
    if $\pi' \neq$ failure then do
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make a not applicable in s'

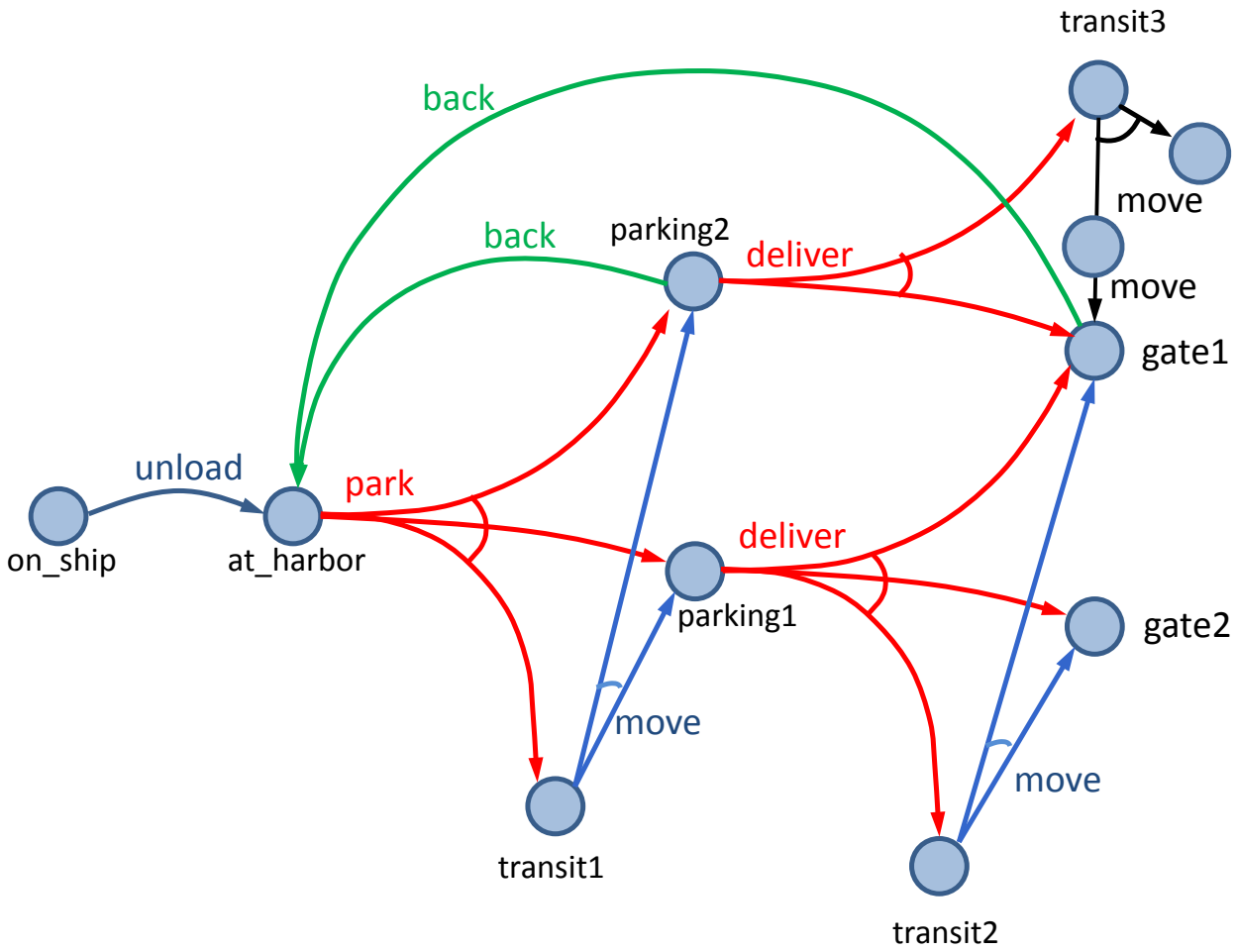
Discussion

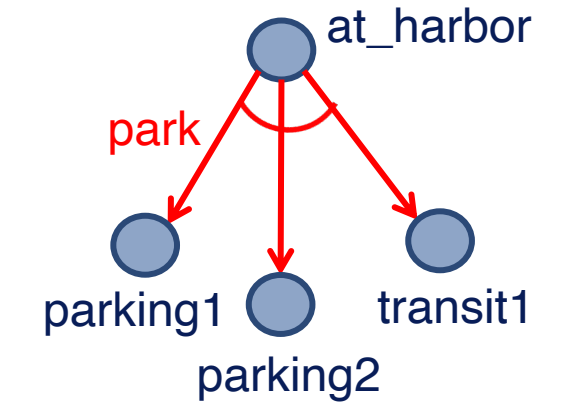
- How to implement it?
  - Need implementation of Find-Solution
  - Need it to be very efficient
    - We'll call it many times
- Idea: instead of Find-Solution, use a classical planner
  - Any of the algorithms from Chapter 2
  - Efficient algorithms, search heuristics
- Need to convert the actions into something the classical planner can use …

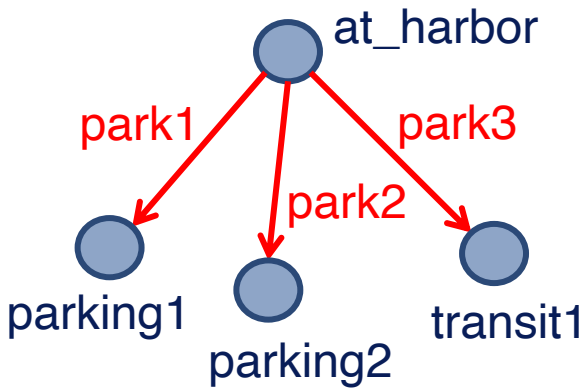
Determinization

- Let a_i be a nondeterministic action with n possible outcomes
- *Determinization* of a_i =

  {n deterministic actions, one for each outcome of a_i }



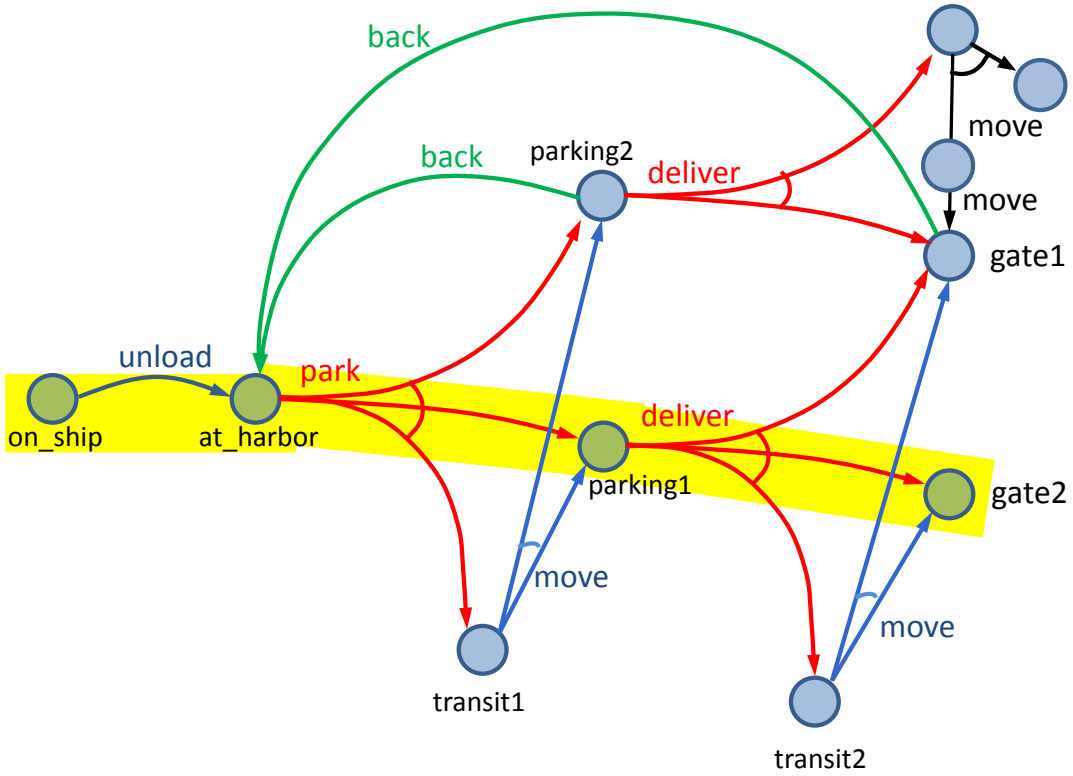


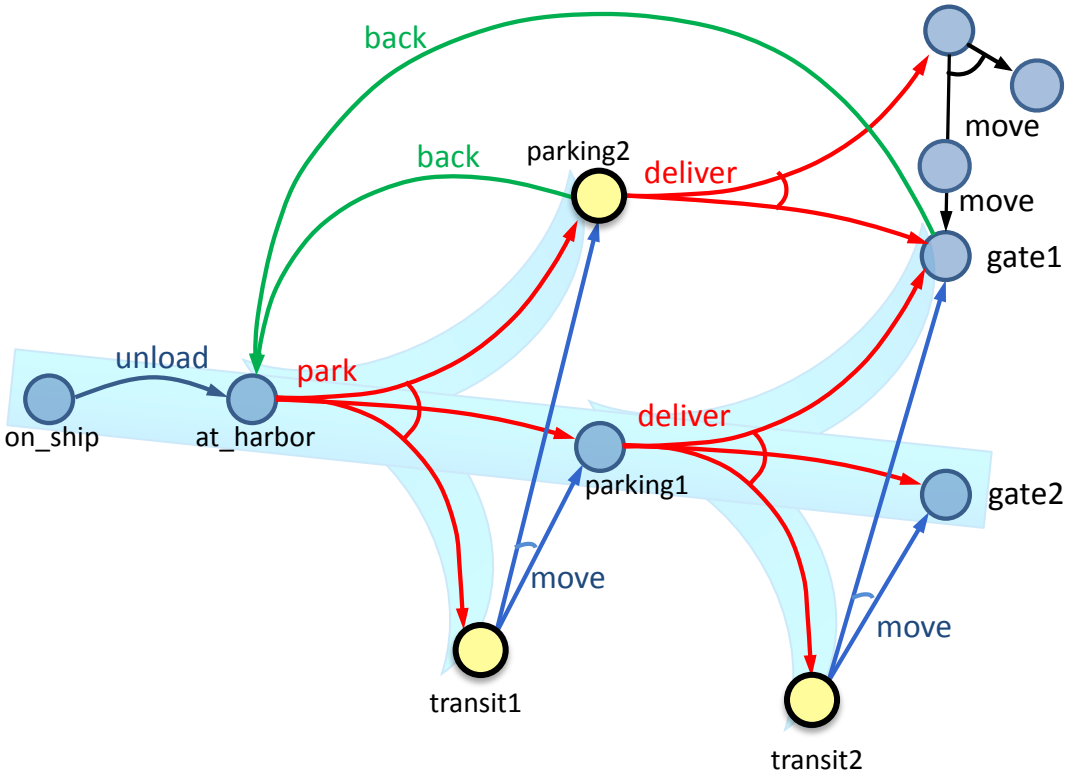


Determinization

- Suppose a classical planner returns an acyclic plan $p = \langle a_1, a_2, \ldots, a_n \rangle$
- Actions and states: $\langle s_0, a_1, s_1, a_2, s_2, a_3, \ldots, a_n, s_n \rangle$
- Convert p to a policy $\langle (s_0, \boldsymbol{a}_1), (s_1, \boldsymbol{a}_2), \ldots, (s_{n-1}, \boldsymbol{a}_n) \rangle$
  - $\boldsymbol{a}_1$ = the nondeterministic action whose determinization includes a_i

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





Find-Safe-Solution-by-Determinization

Guided-Find-Safe-Solution (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  loop
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select arbitrarily $s \in Q$
    $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)
    if $\pi' \neq$ failure then do
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make a not applicable in s'

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a not applicable in s'

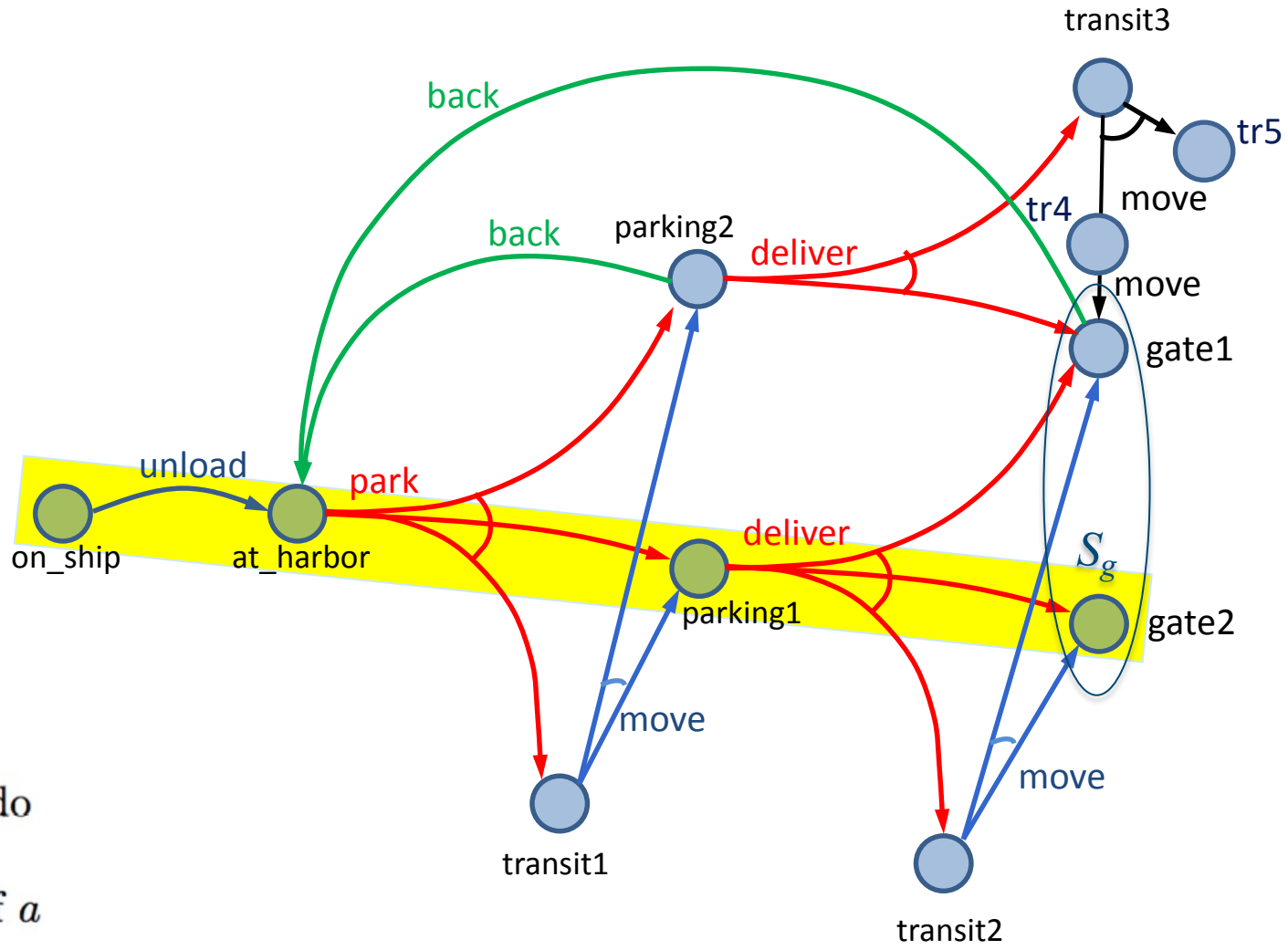
Any classical planner that doesn't return cyclic plans

Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'

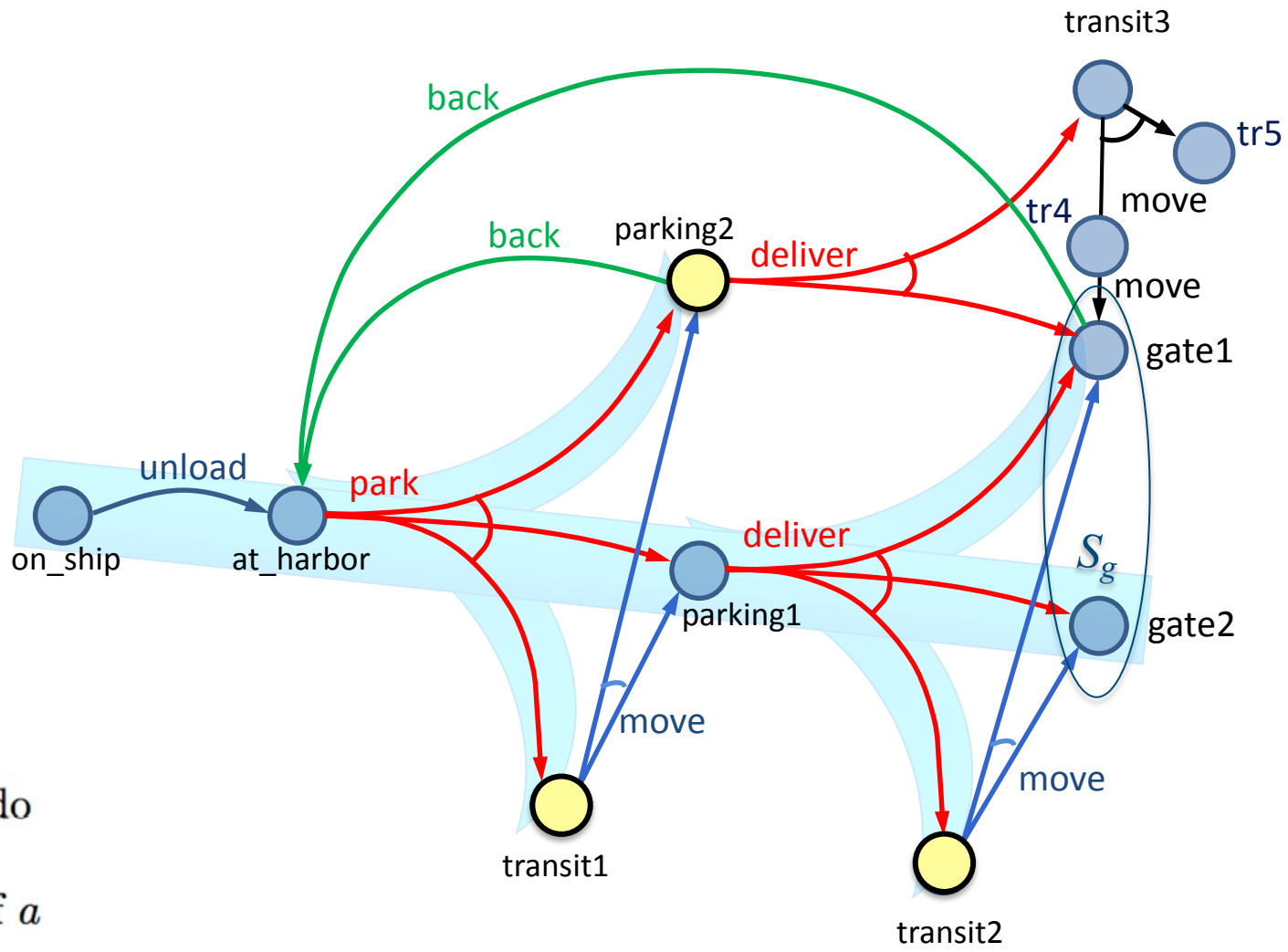


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'

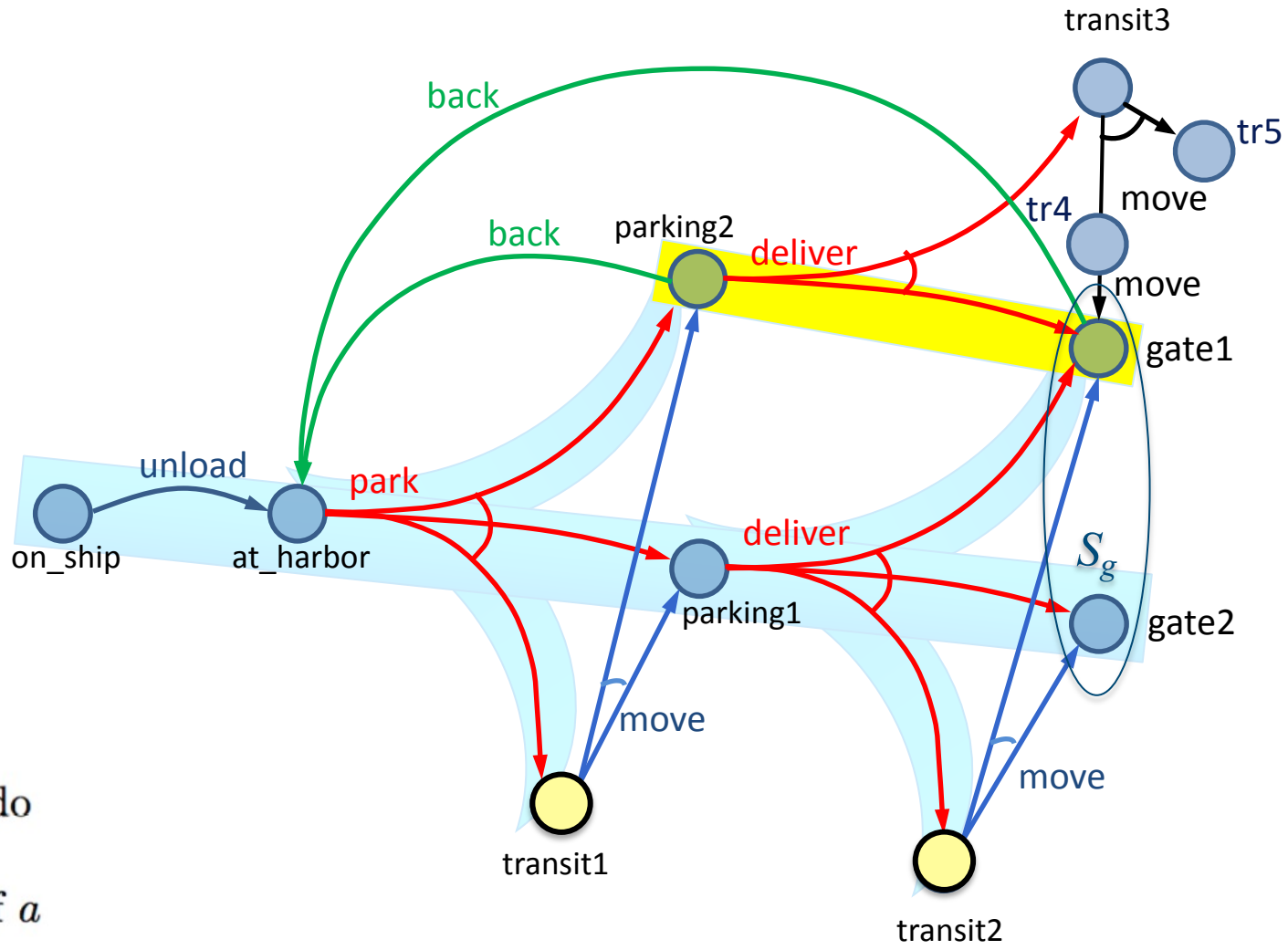


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'

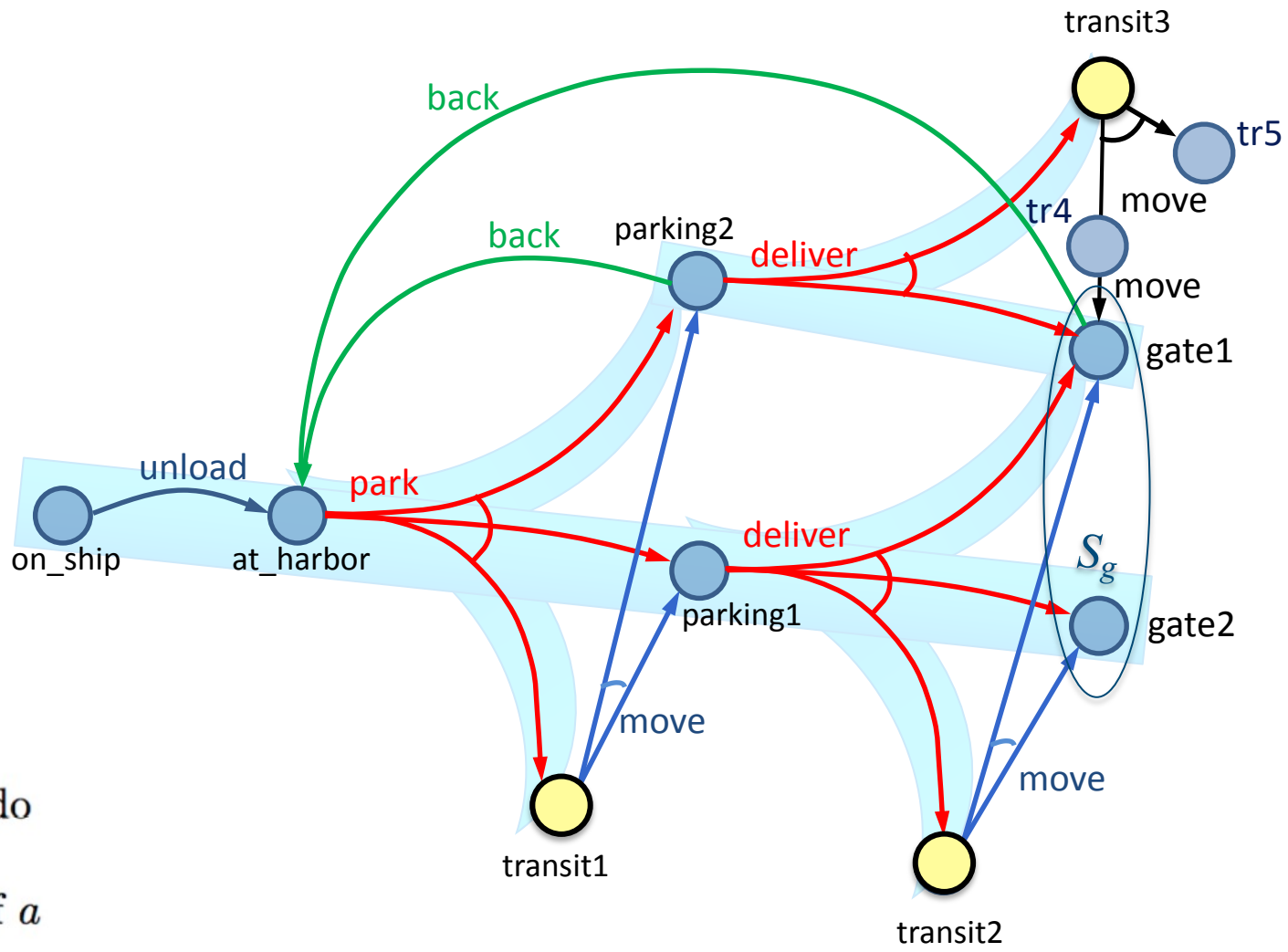


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'

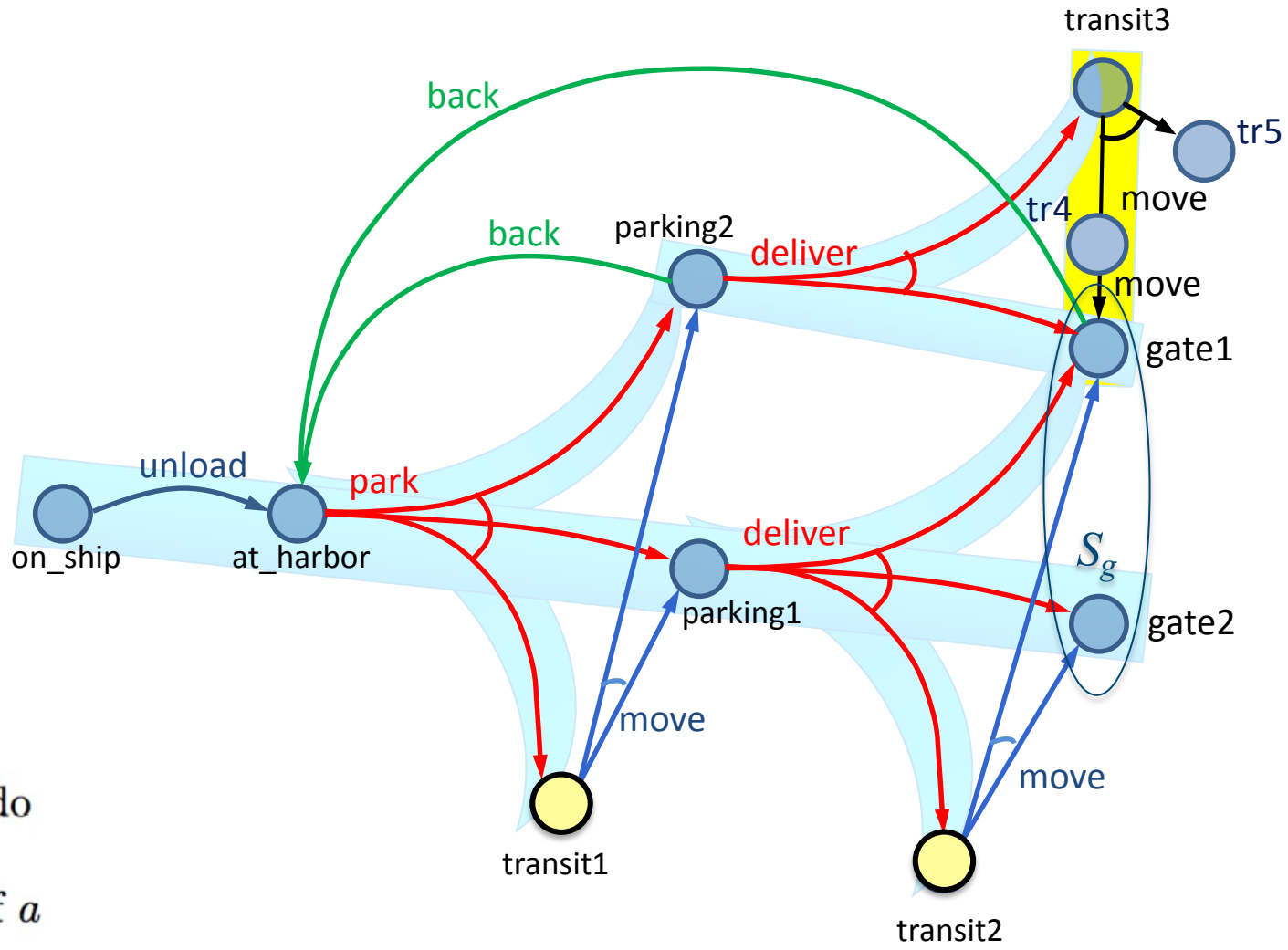


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'

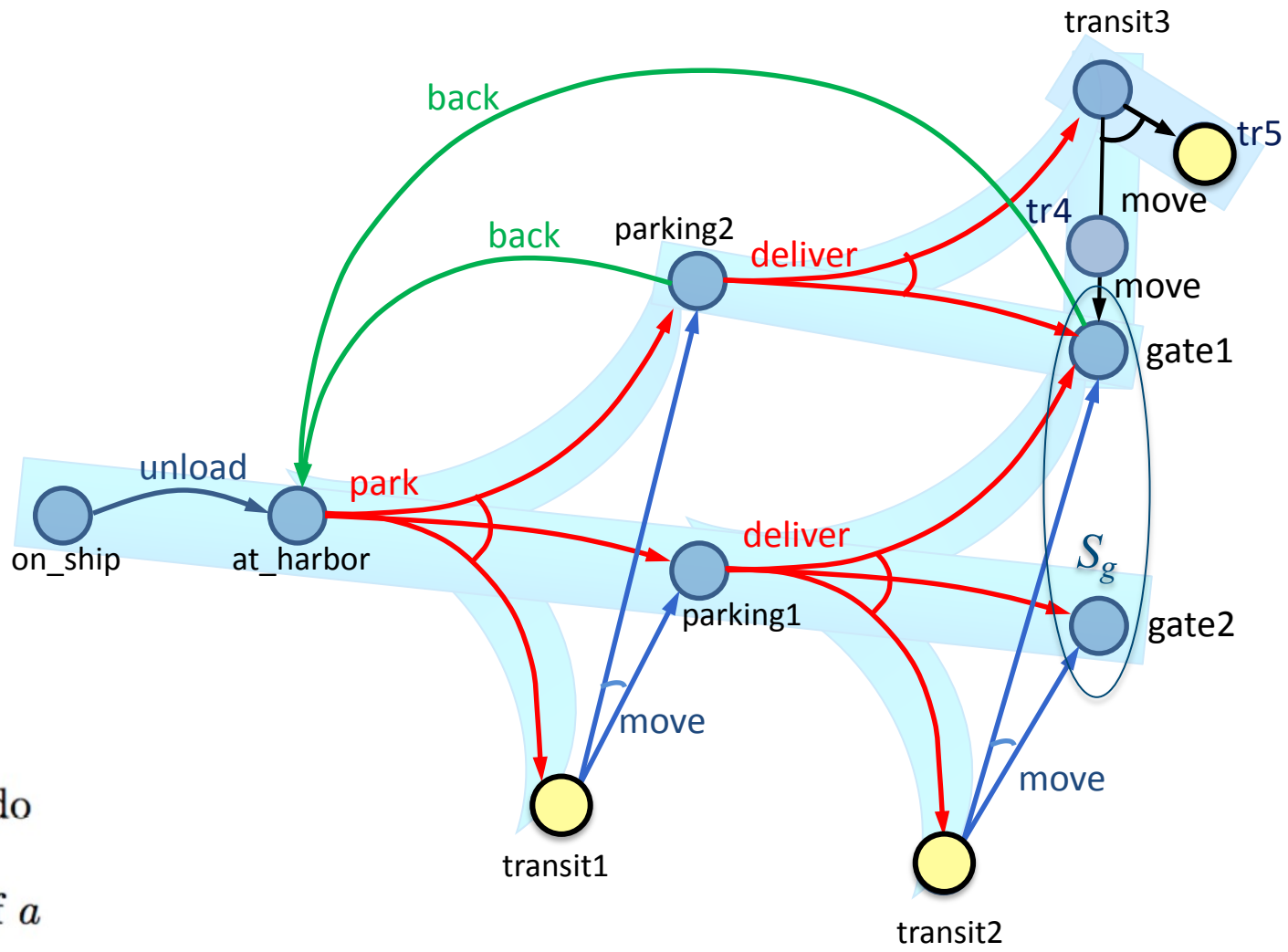


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'

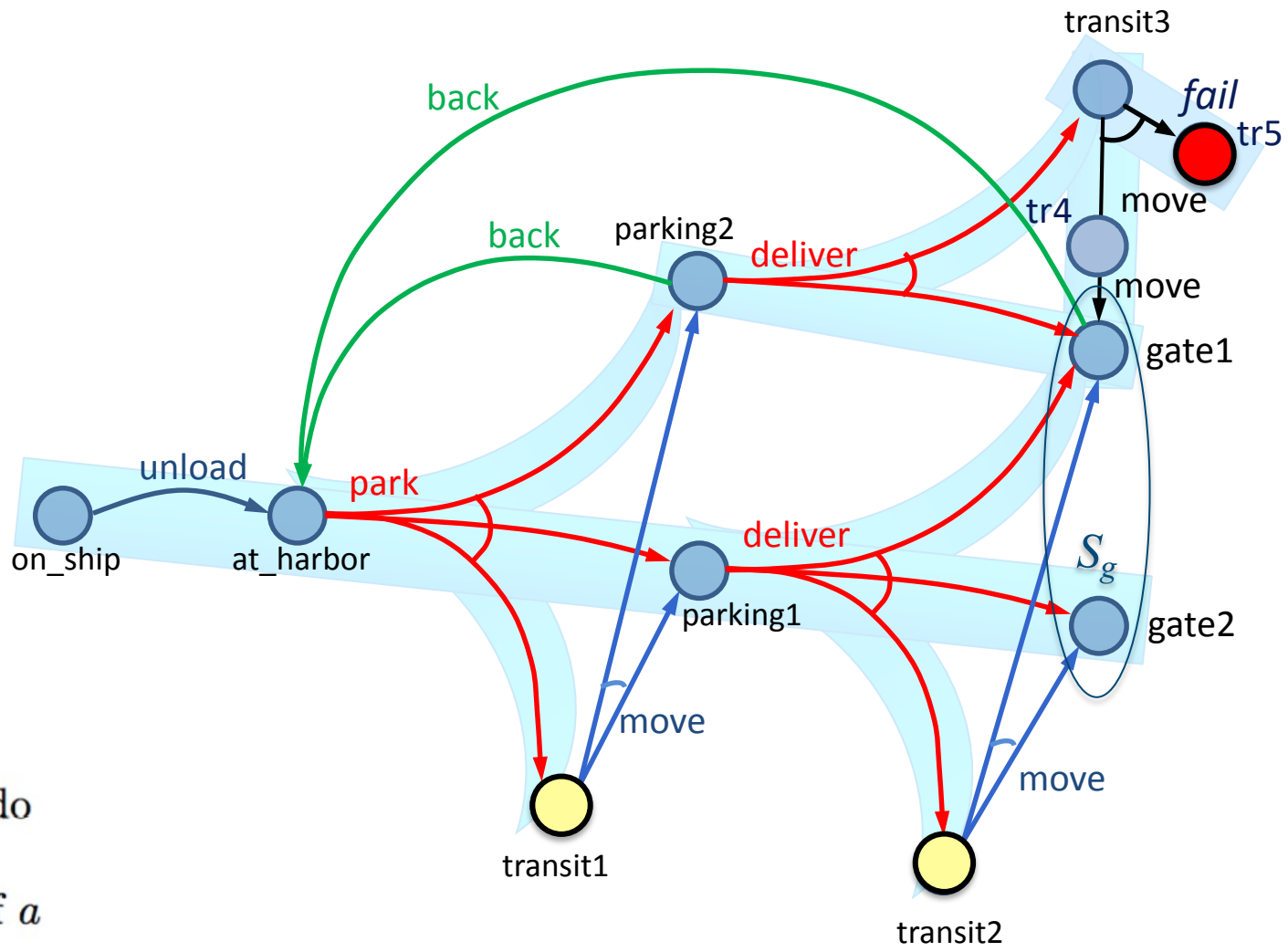


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

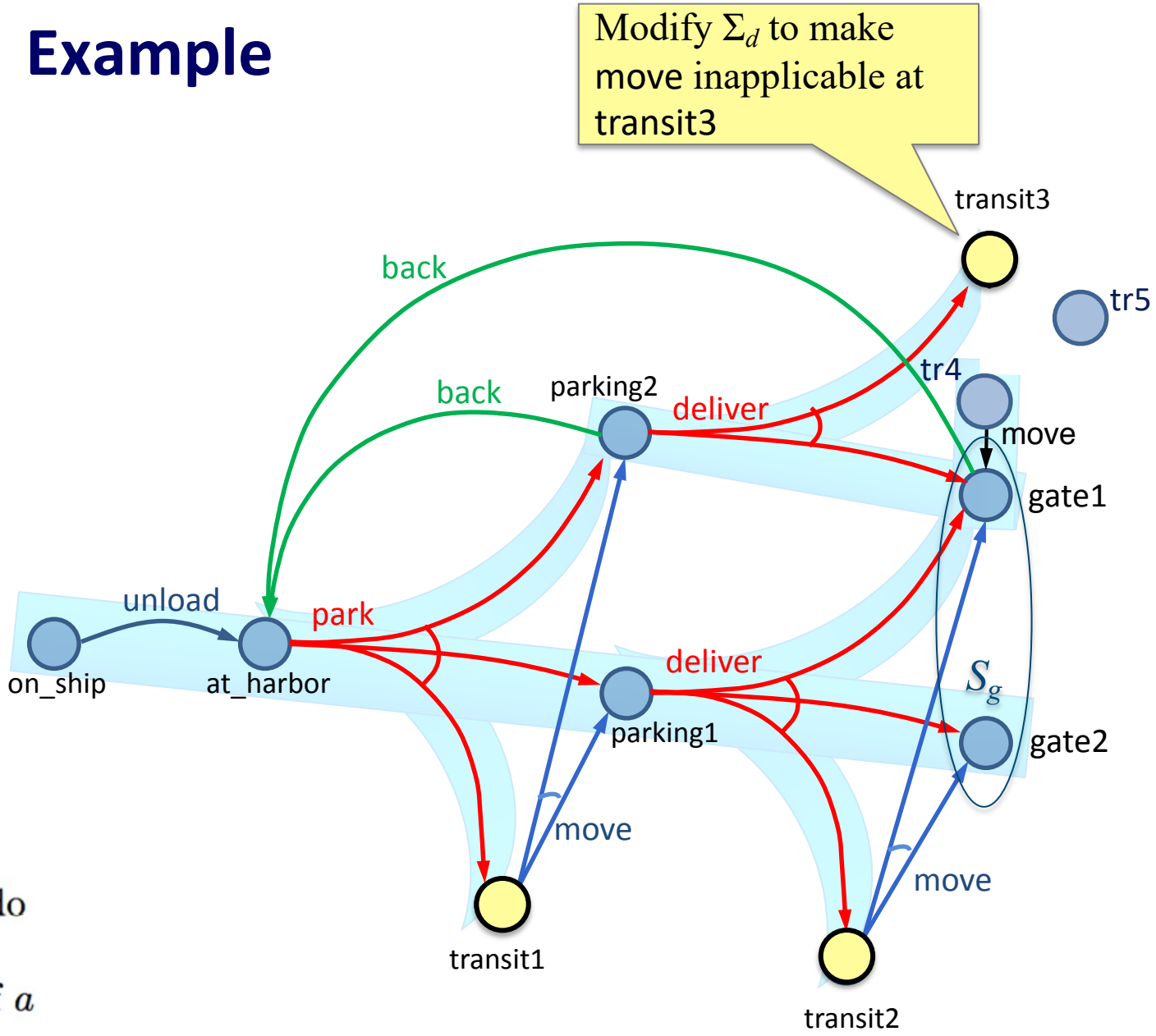
$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'



Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop

    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a
      not applicable in s'

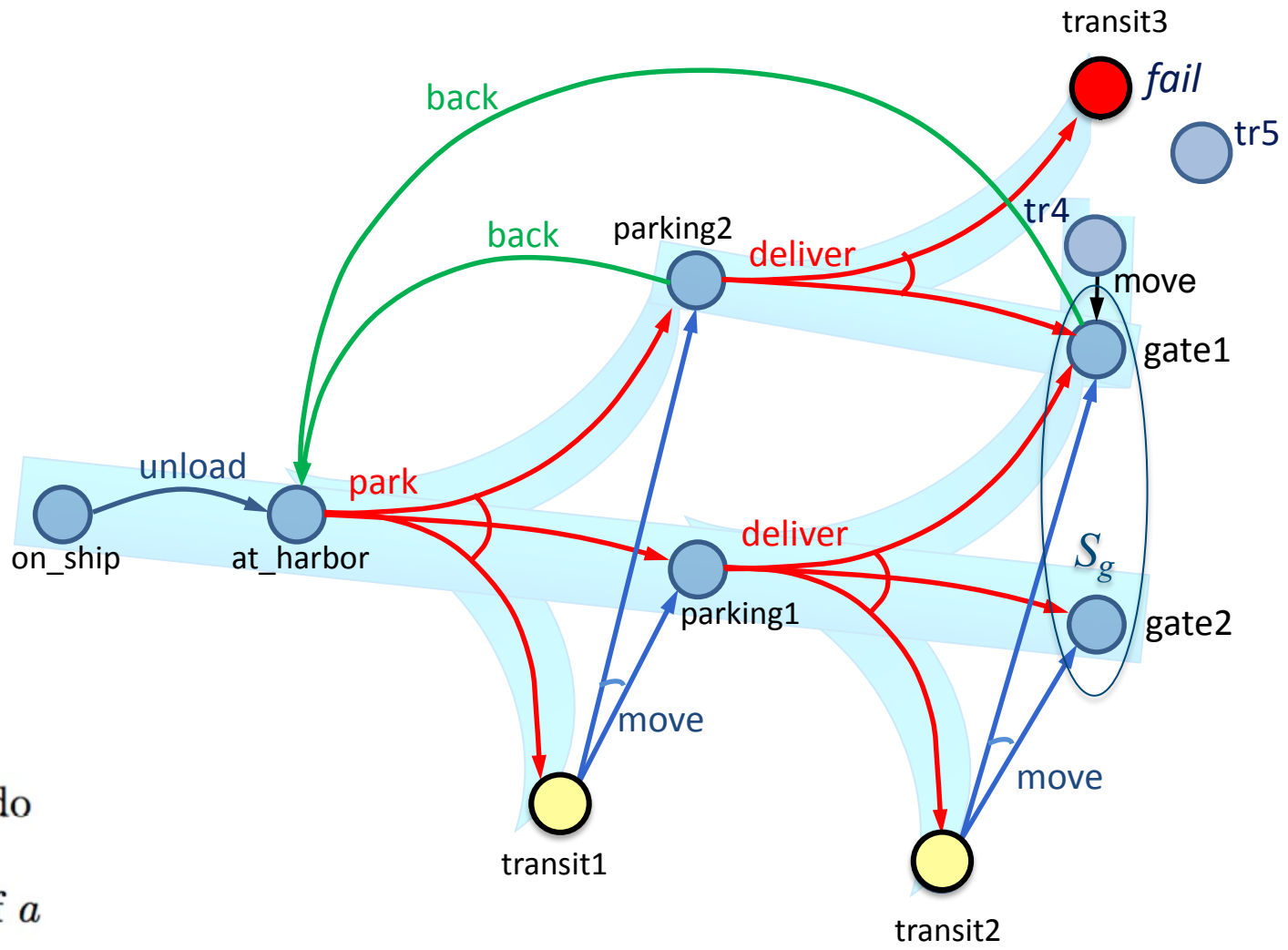


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

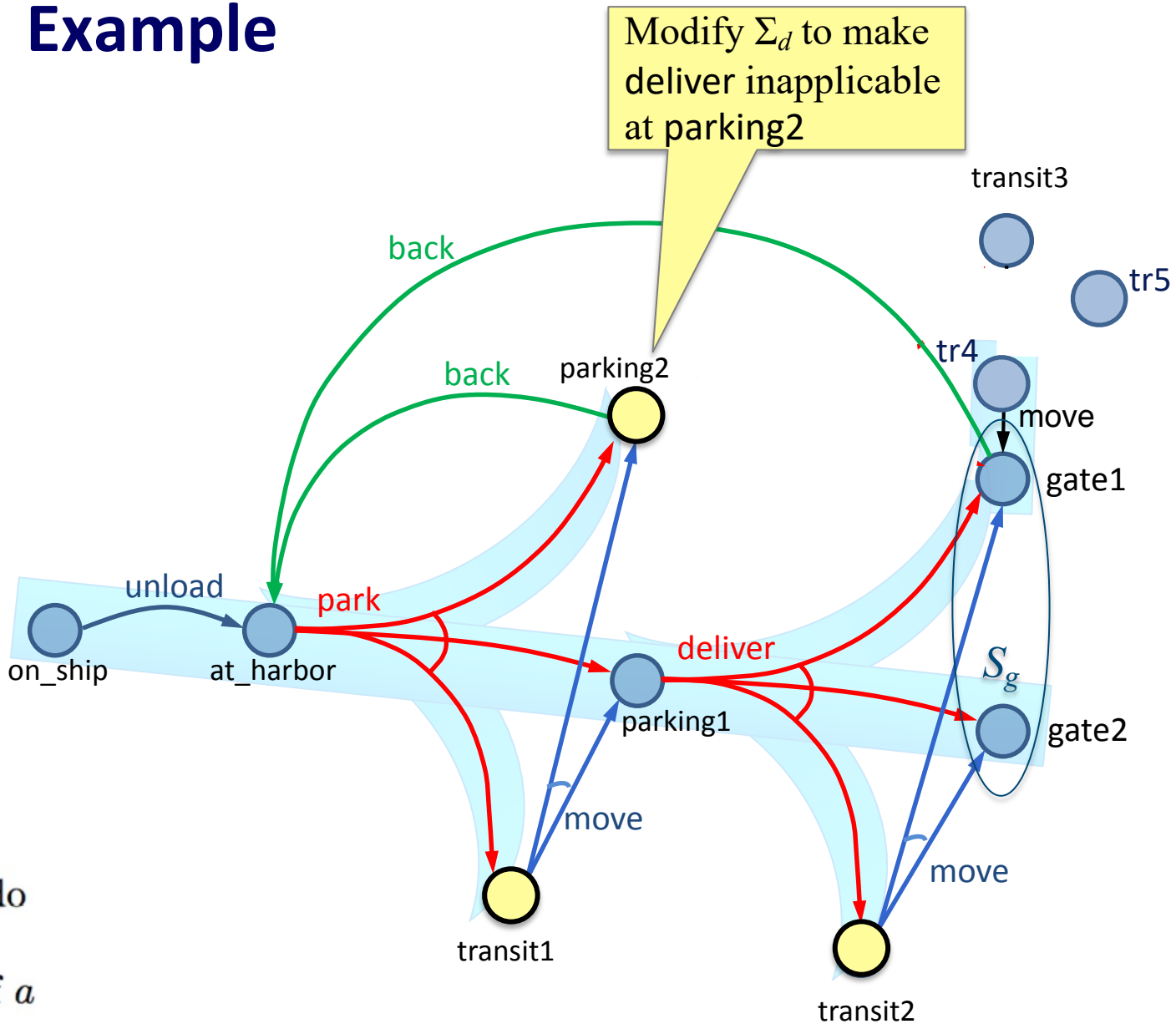
$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'



Example

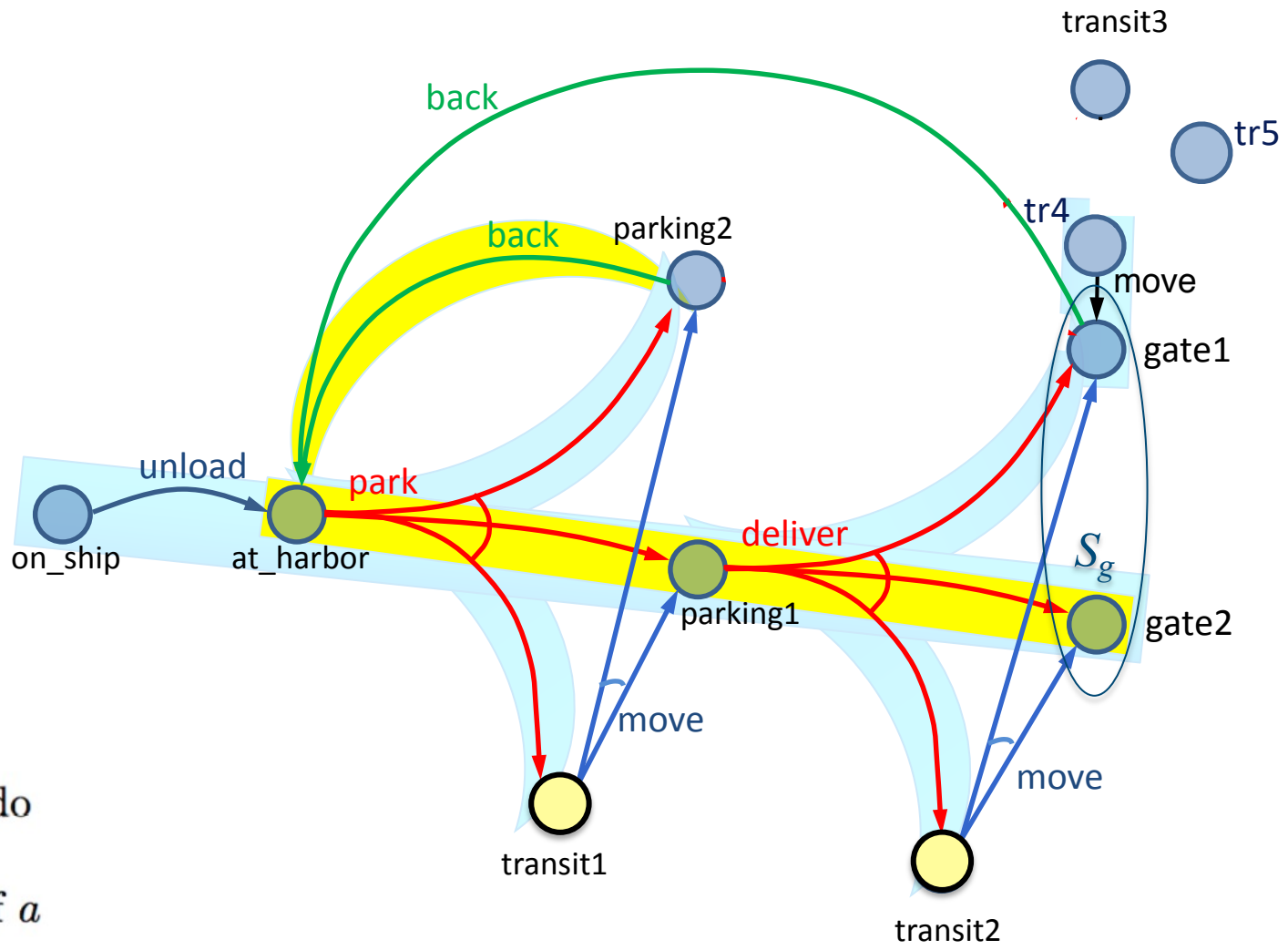
Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop

    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s,a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a
      not applicable in s'



Example

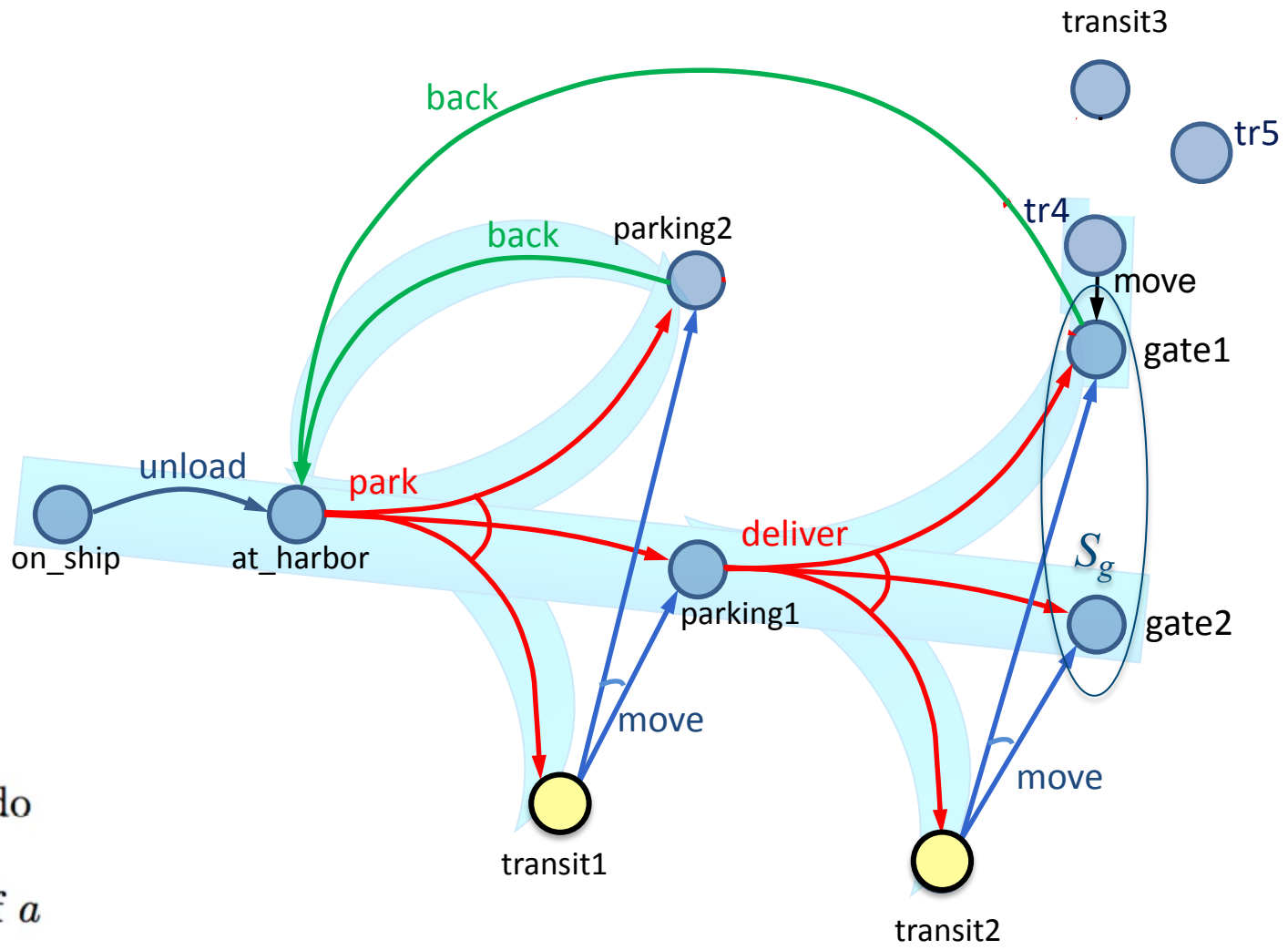
Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
- if $s_0 \in S_g$ then return$(\varnothing)$
- if $Applicable(s_0) = \varnothing$ then return(failure)
- $\pi \leftarrow \varnothing$
- $\Sigma_d \leftarrow$ mk-deterministic(Σ)
- loop
  - $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  - if $Q = \varnothing$ then do
    - $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    - return(π)
  - select $s \in Q$
  - $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
  - if $p' \neq fail$ then do
    - $\pi' \leftarrow$ Plan2policy(p', s)
    - $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
  - else for every s' and a such that $s \in \gamma(s', a)$ do
    - $\pi \leftarrow \pi \setminus \{(s', a)\}$
    - make the actions in the determinization of a not applicable in s'



Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop

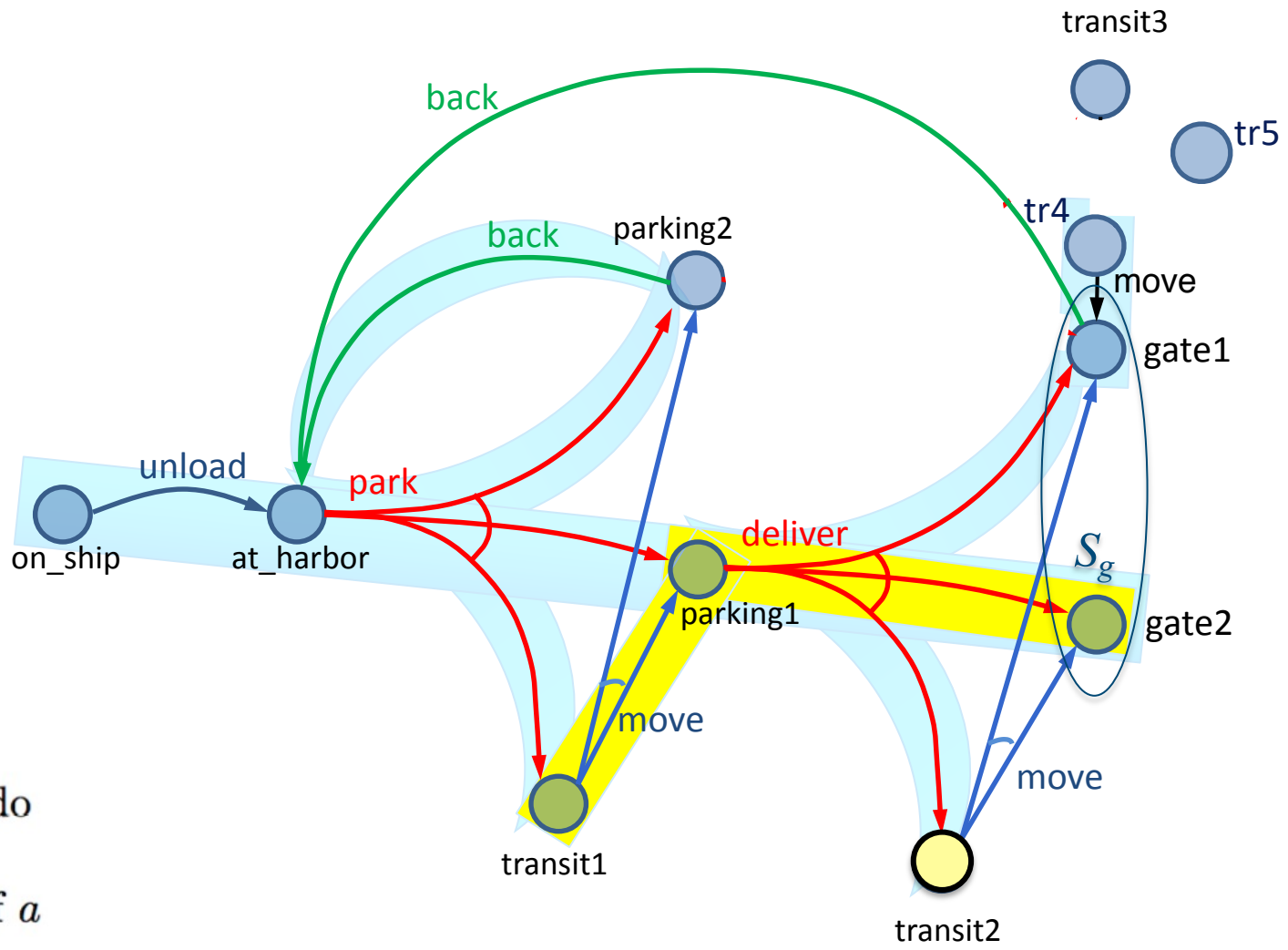
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a
      not applicable in s'



Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop

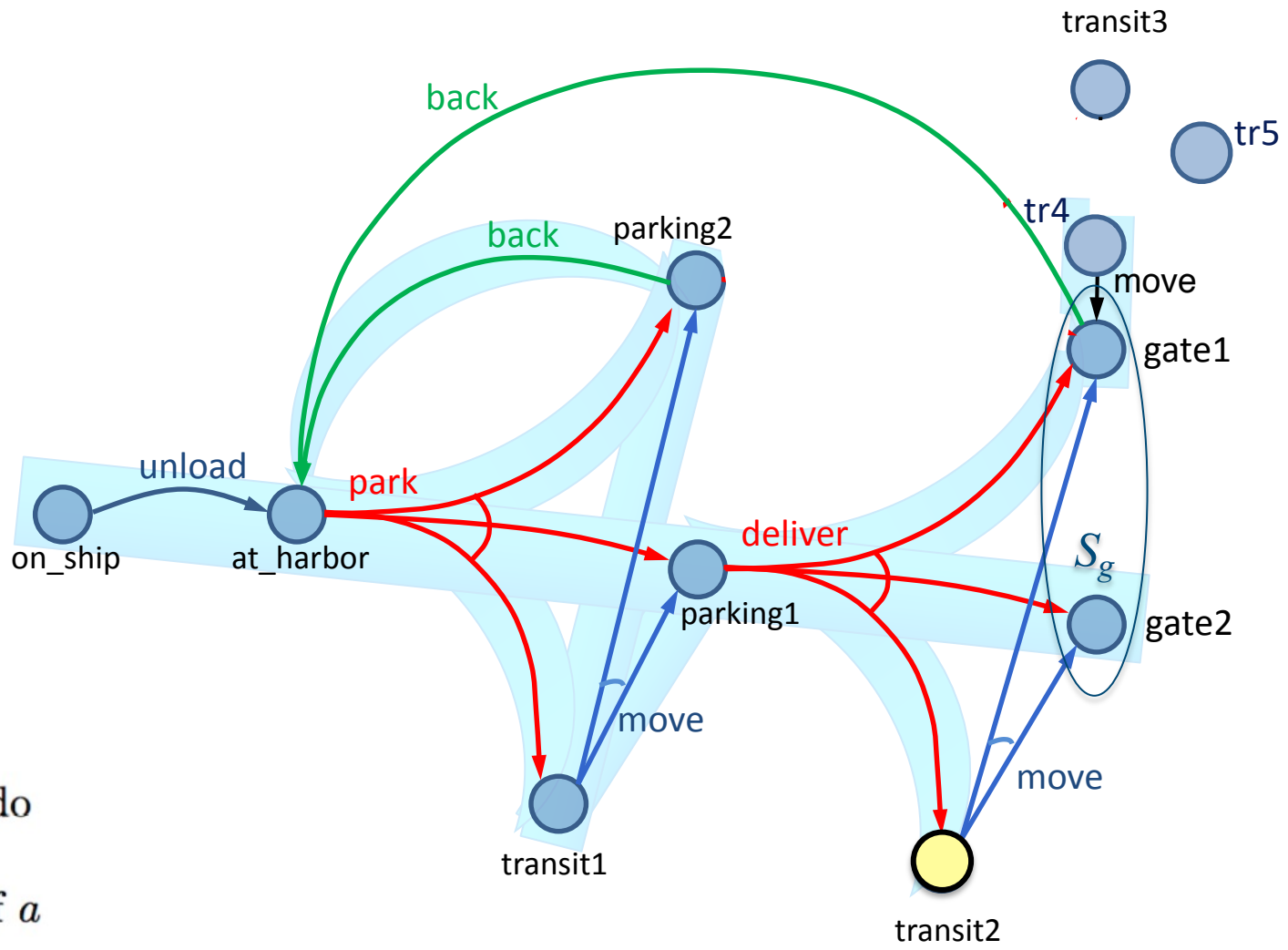
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a
      not applicable in s'



Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop

    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a
      not applicable in s'

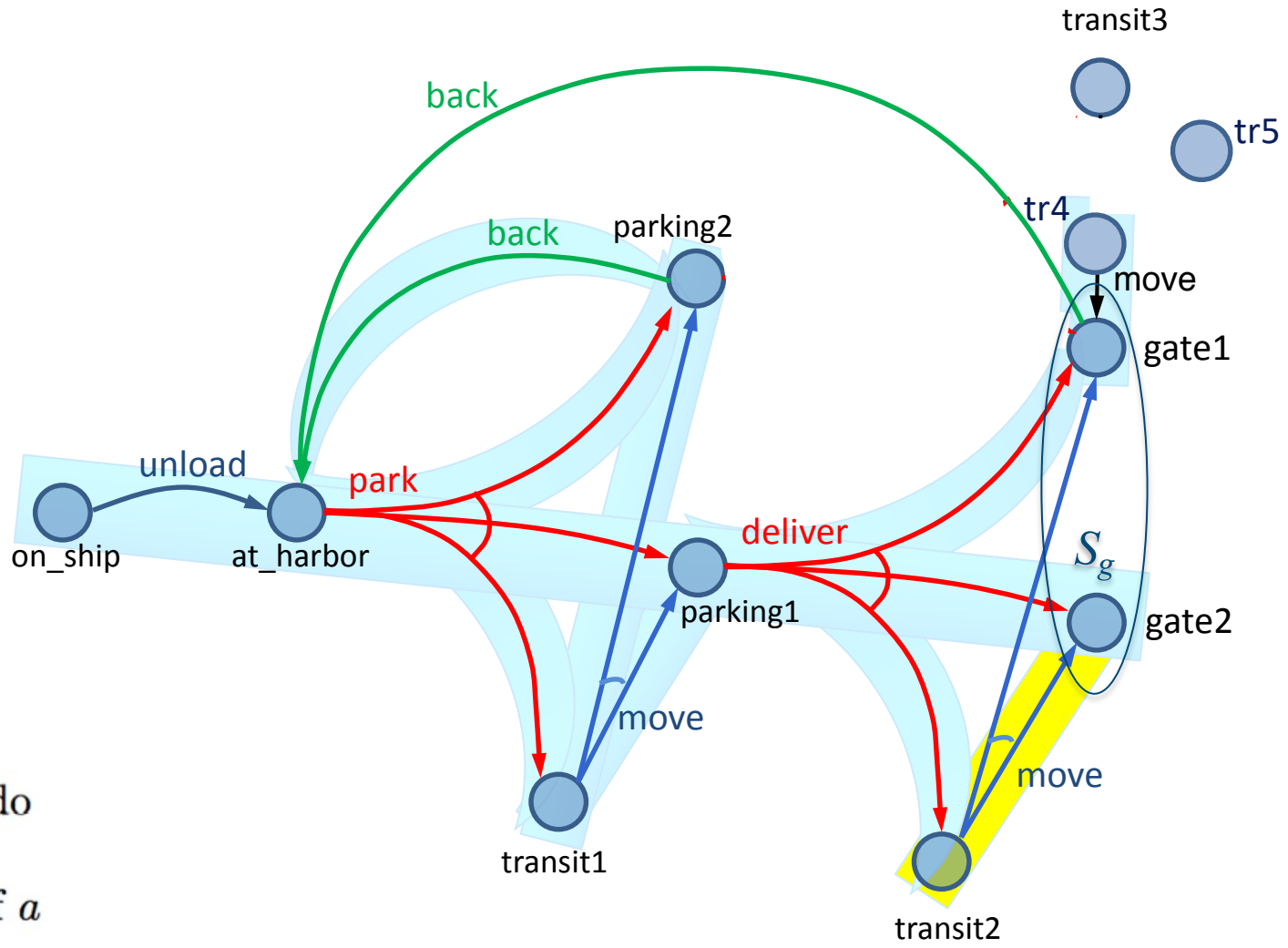


Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)

if $s_0 \in S_g$ then return$(\varnothing)$
if $Applicable(s_0) = \varnothing$ then return(failure)
$\pi \leftarrow \varnothing$
$\Sigma_d \leftarrow$ mk-deterministic(Σ)
loop

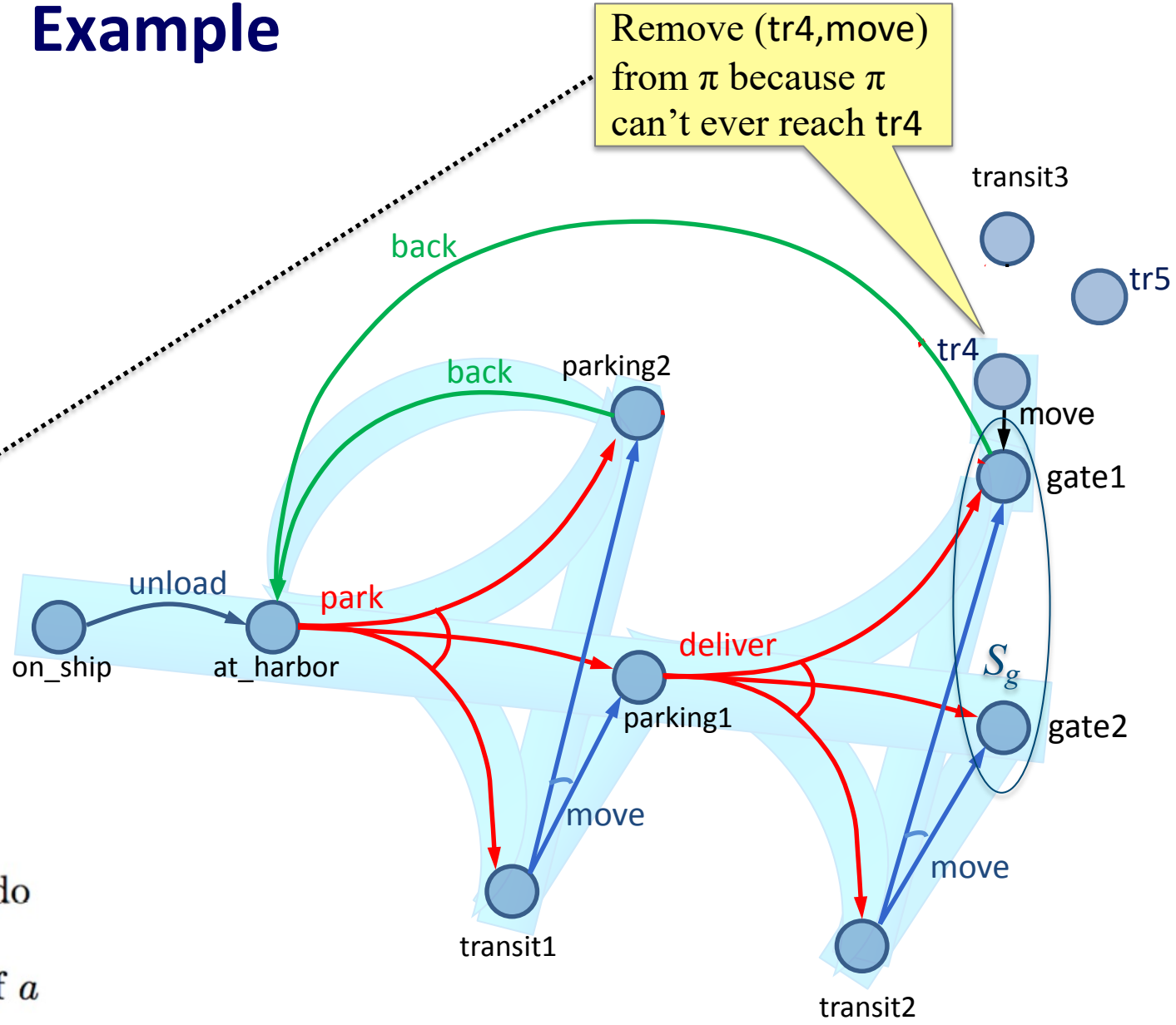
$Q \leftarrow leaves(s_0, \pi) \setminus S_g$
if $Q = \varnothing$ then do
$\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
return(π)
select $s \in Q$
$p' \leftarrow$ Forward-search (Σ_d, s, S_g)
if $p' \neq fail$ then do
$\pi' \leftarrow$ Plan2policy(p', s)
$\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
else for every s' and a such that $s \in \gamma(s', a)$ do
$\pi \leftarrow \pi \setminus \{(s', a)\}$
make the actions in the determinization of a
not applicable in s'



Example

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop

    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \text{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a
      not applicable in s'



Making Actions Inapplicable

Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return$(\varnothing)$
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  $\Sigma_d \leftarrow$ mk-deterministic(Σ)
  loop
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s,a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      return(π)
    select $s \in Q$
    $p' \leftarrow$ Forward-search (Σ_d, s, S_g)
    if $p' \neq fail$ then do
      $\pi' \leftarrow$ Plan2policy(p', s)
      $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$
    else for every s' and a such that $s \in \gamma(s', a)$ do
      $\pi \leftarrow \pi \setminus \{(s', a)\}$
      make the actions in the determinization of a not applicable in s'

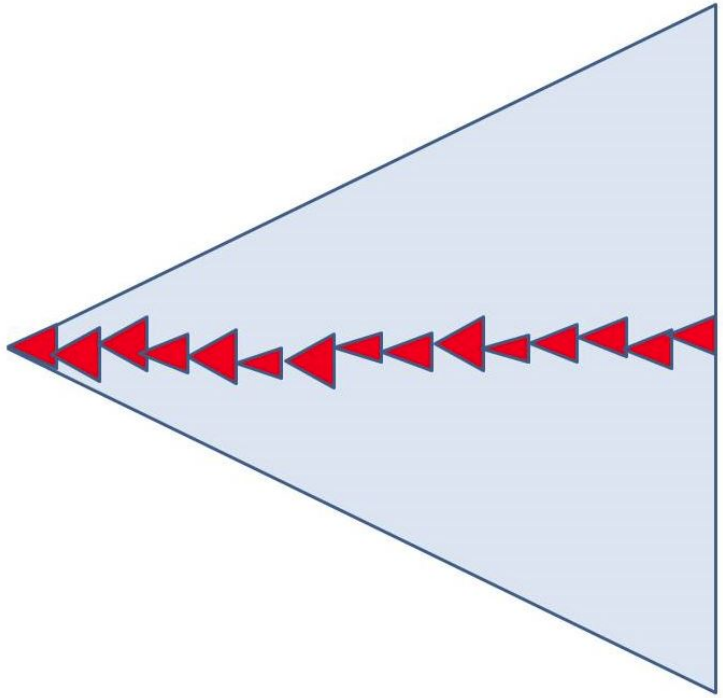
- Modify Σ_d to make a inapplicable at s
  - worst-case exponential time
- Better: hash table of bad state-action pairs
  - For every (s',a) such that $s \in \gamma(s',a)$, $Bad[s'] \leftarrow Bad[s'] \cup$ determinization(a)
  - Modify classical planner to take the table as an argument
    - if s is current state, only choose actions in $Applicable(s) \setminus Bad[s]$

Skip Ahead

- Several topics I'll skip for now
  - will come back later if there's time
  - Other kinds of search algorithms
    - min-max search
  - Symbolic model checking techniques
    - Backward search
    - BDD representation
      - Reduce search-space size by planning over sets of states

5.6 Online Approaches

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



Offline vs Runtime Search Spaces

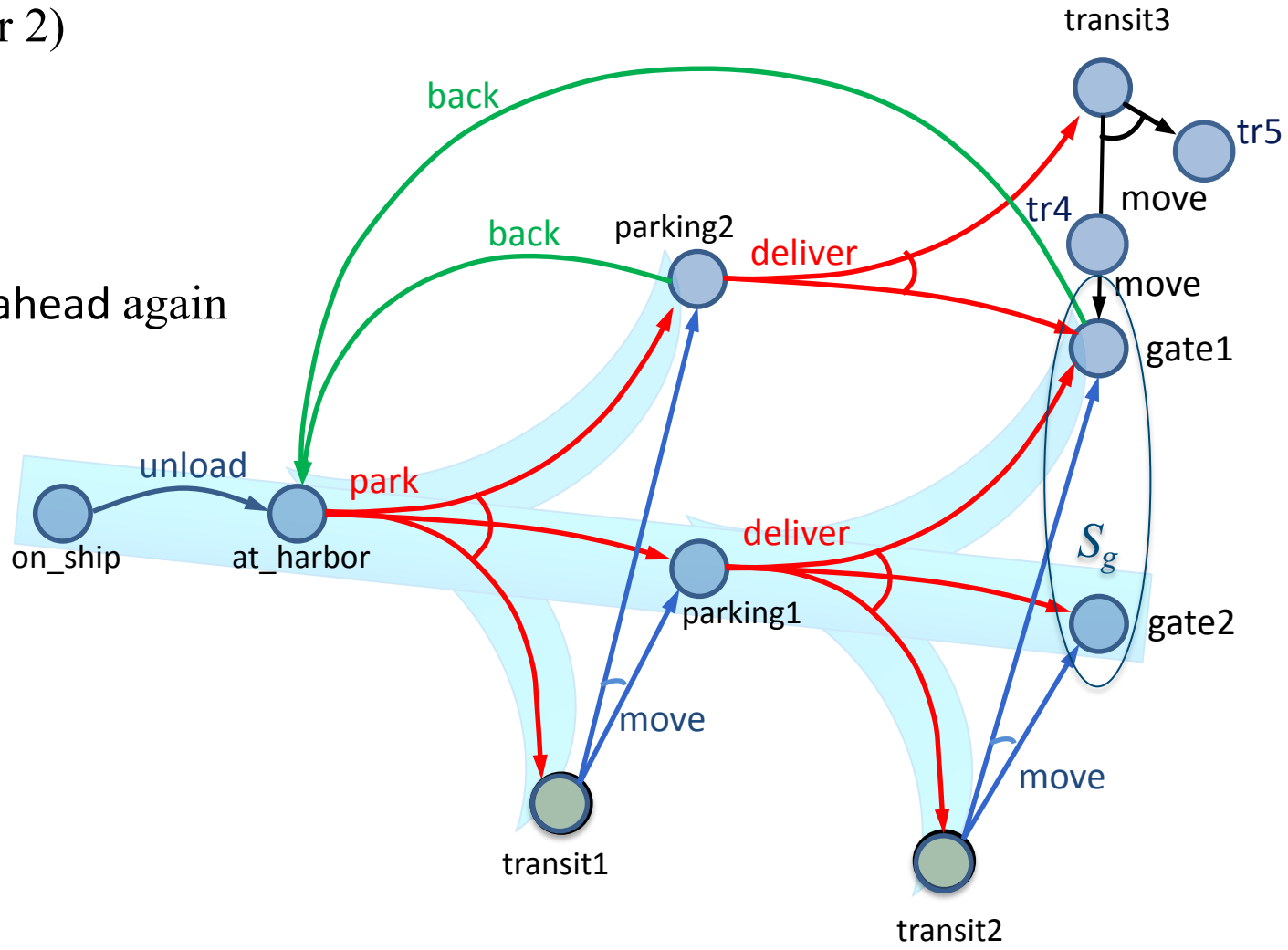
Online Approaches

- 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 always reachable
- In domains with dead-ends, partial planning won't guarantee success
  - Can get trapped in dead ends that we would have detected if we had planned fully
    - No applicable actions
      - robot's battery goes dead
    - Applicable actions, but caught in a loop
      - robot goes into a collection of rooms from which there's no exit
  - But partial planning can still make success more likely

Lookahead-Partial-Plan

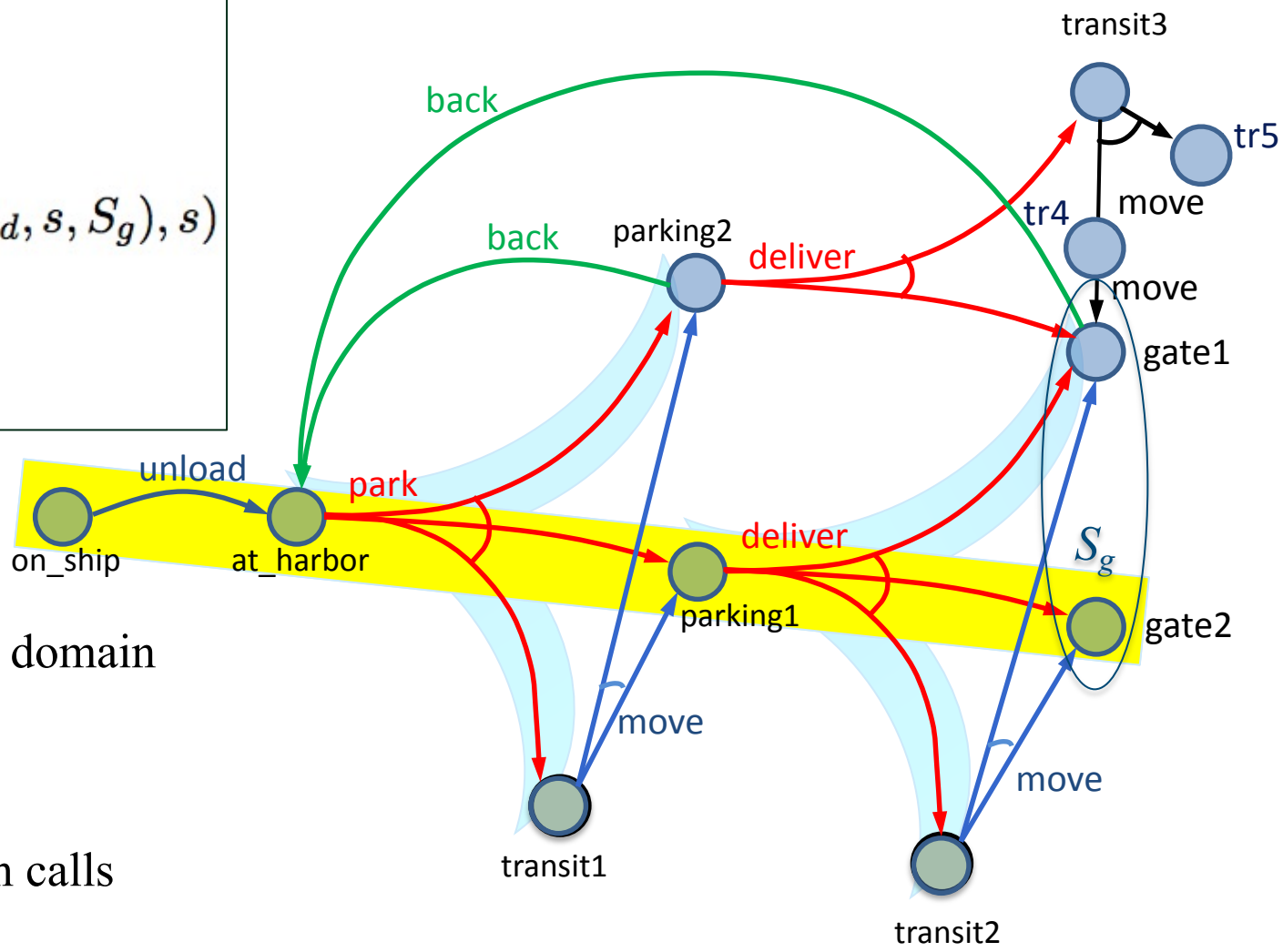
- Adaptation of Run-Lazy-Lookahead (Chapter 2)
- Lookahead is any planning algorithm that returns a policy π
  - π may be partial or unsafe solution
- Execute π as far as it will go, then call Lookahead again

```
Lookahead-Partial-Plan(Σ, s_0, S_g)
    s ← s_0
    while s ∉ S_g and Applicable(s) ≠ ∅ do
        π ← Lookahead(s, θ)
        if π = ∅ then return failure
        else do
            perform partial plan π
            s ← observe current state
```



FS-Replan

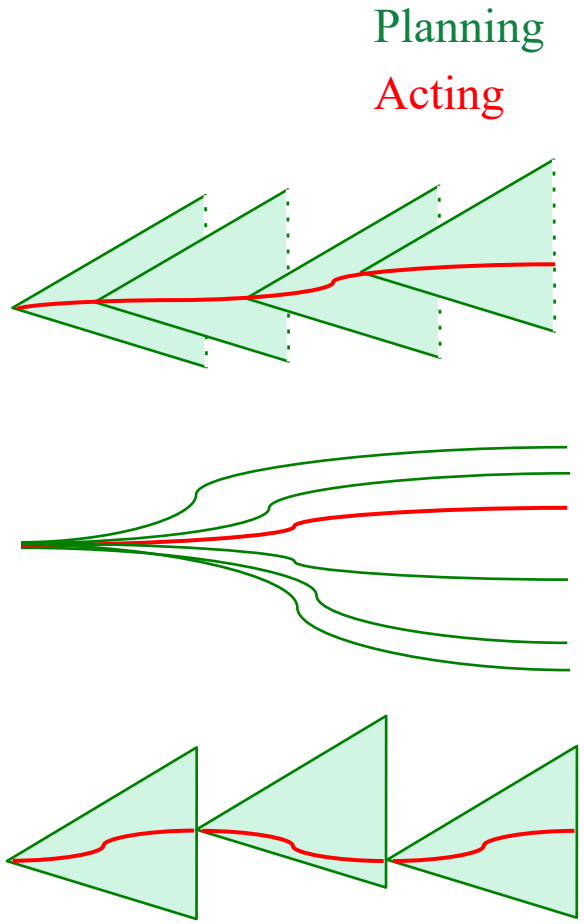
FS-Replan (Σ, s, S_g)
  $\pi_d \leftarrow \varnothing$
  while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do
    if π_d undefined for s then do
      $\pi_d \leftarrow$ Plan2policy(Forward-search$(\Sigma_d, s, S_g), s)$
      if $\pi_d =$ failure then return failure
    perform action $\pi_d(s)$
    $s \leftarrow$ observe resulting state



- Adaptation of Lookahead-Partial-Plan
  - Calls classical planner on determinized domain
  - Gets plan, converts converts to policy
    - Unsafe solution
  - Executes policy as far as it will go, then calls classical planner again

More Possibilities for Lookahead

- What if Lookahead doesn't have time to run to completion?
  - Can use the same techniques we discussed in Chapter 3
    - Receding horizon
    - Sampling
    - Subgoaling
    - Iterative deepening
  - A few others …



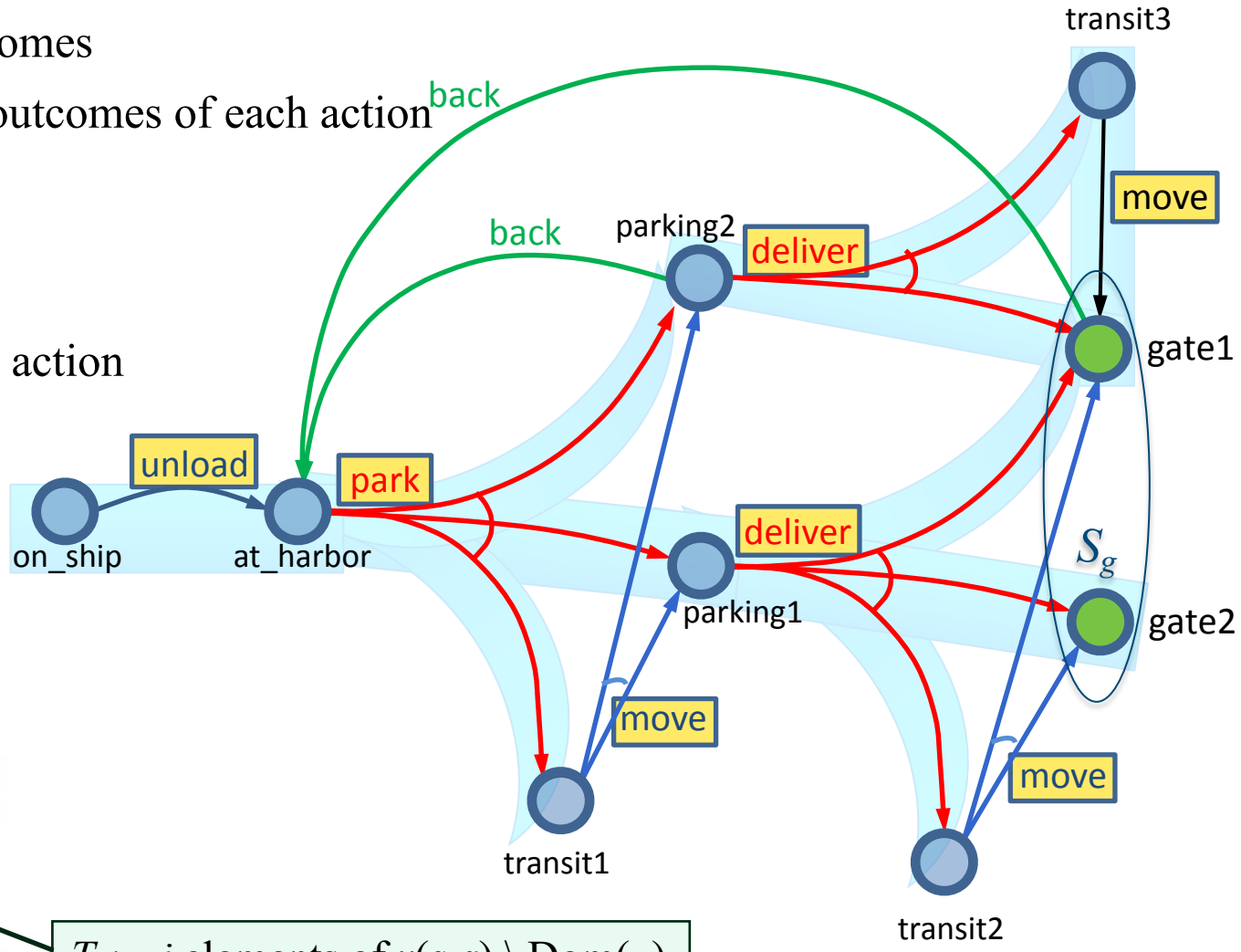
More Possibilities for Lookahead

- *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 each action
- *Iterative broadening*:

  for i = 1 by 1 until time runs out

  look for a solution that handles i outcomes per action

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



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

Safely Explorable Domains

- *Safely explorable* domain
  - for every state s, at least one goal state is reachable from s

- For Lookahead, suppose we use Lookahead-Partial-Plan or FS-Replan
  - Then Lookahead never returns failure
- Every "fair" execution will eventually reach a goal

Poll: Suppose we just choose a random action each time. Is every "fair" execution guaranteed to reach a goal?

Min-Max LRTA*

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ 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

If some actions have cost ≠ 1, then use cost(s,a) here

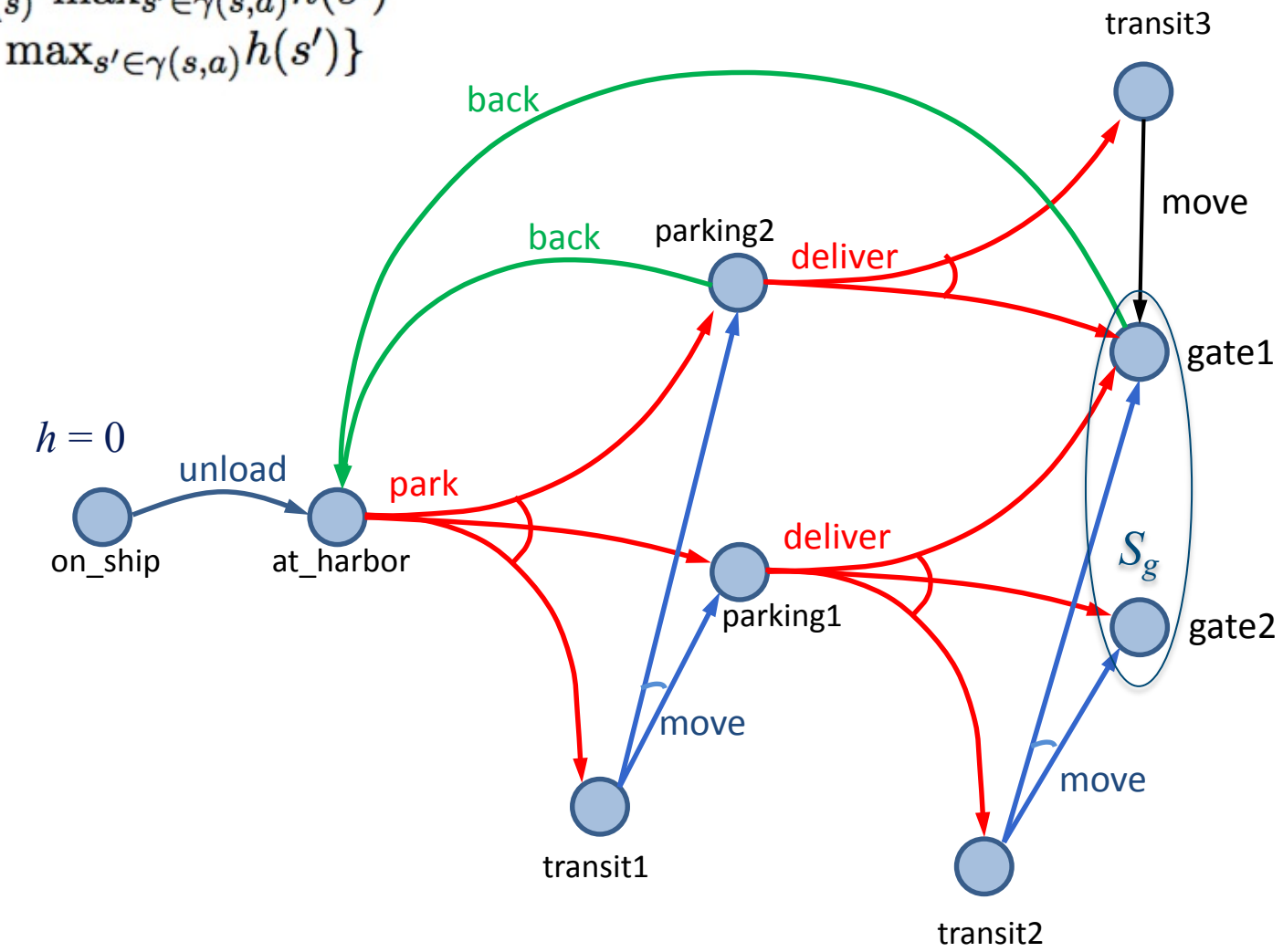
- loop
  - choose an action a that (according to h) has optimal worst-case cost
    - Update $h(s)$ to use a's worst-case cost
    - Perform a
- In safely explorable domains with no "unfair" executions, guaranteed to reach a goal

Example

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ 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

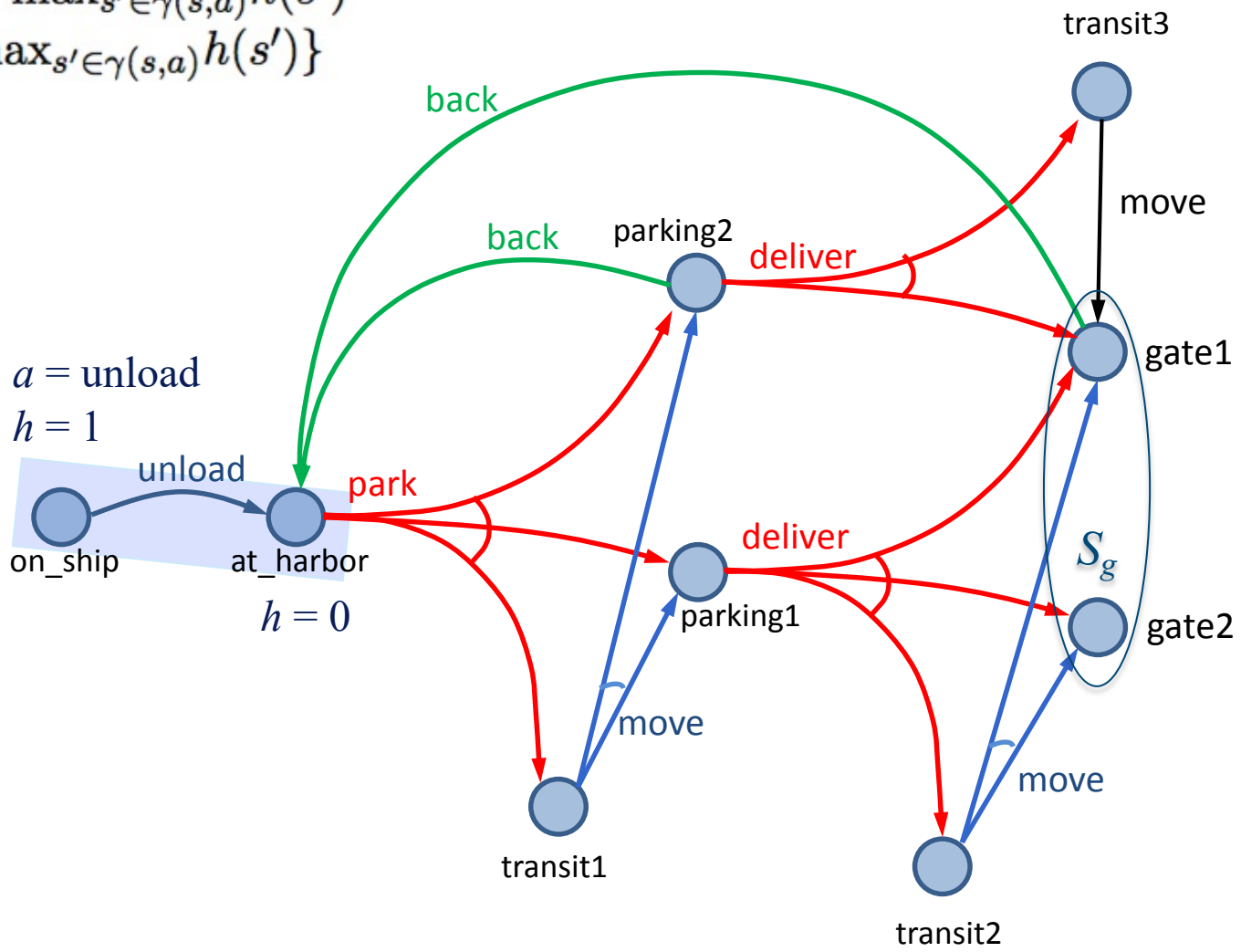


Example

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ 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



Example

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

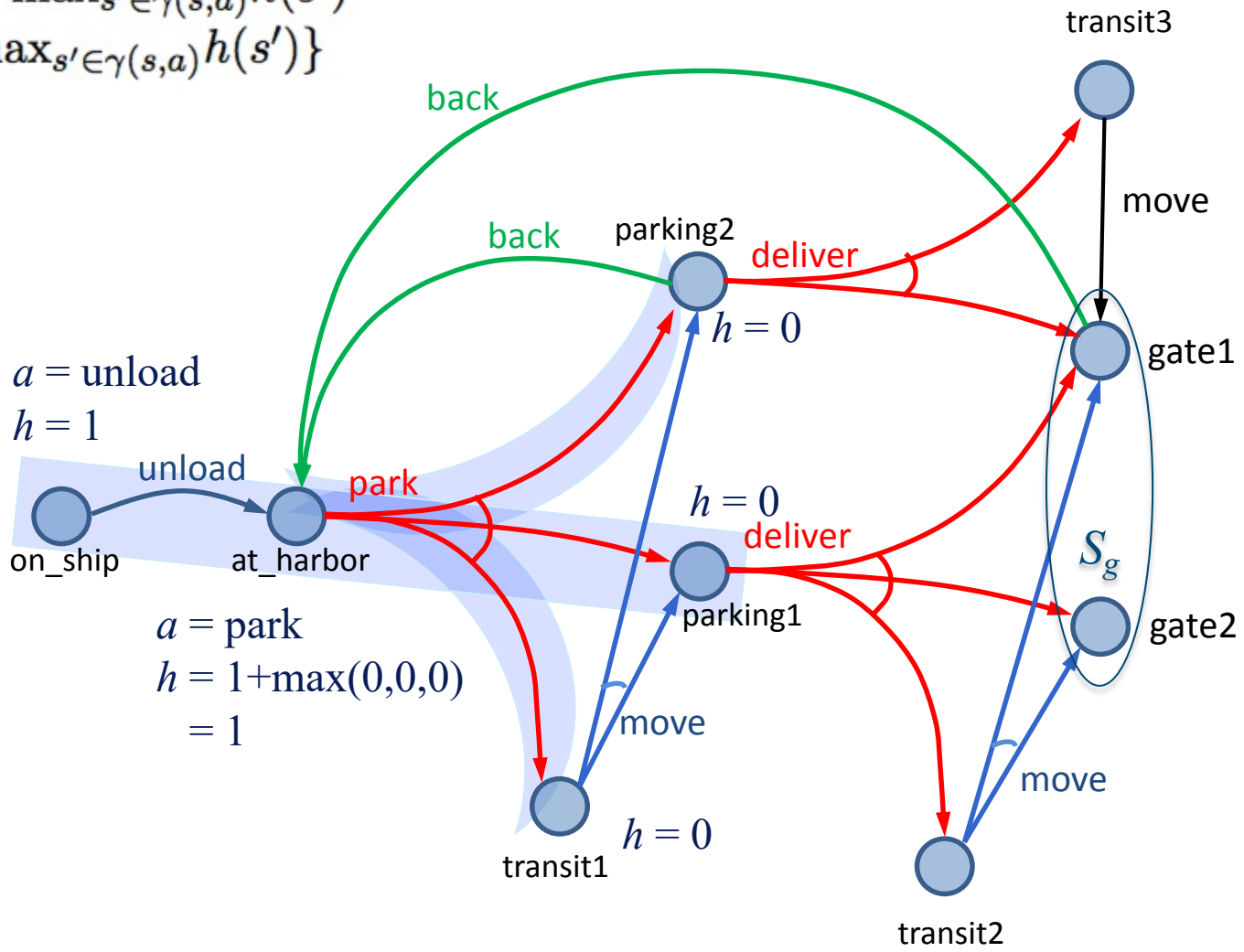
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

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

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

perform action a

$s \leftarrow$ the current state



Example

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

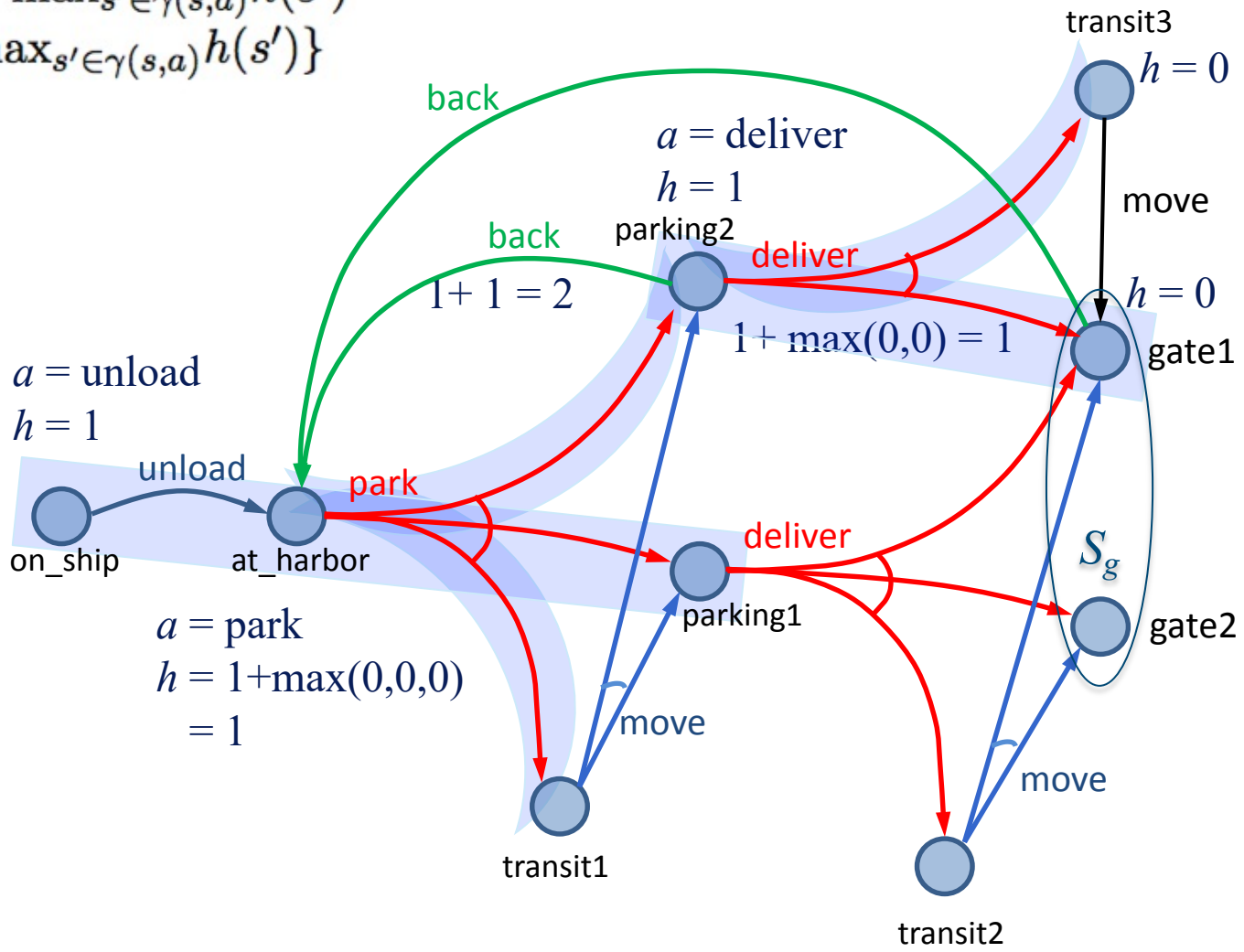
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

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

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

perform action a

$s \leftarrow$ the current state



5.7 Refinement Methods

- Differences to refinement methods in Chapter 3:
  - Tasks refine into automata
  - Need to combine the automata
- Important work, but the concepts are complicated
  - We won't have time to cover them

Summary

- Actions, plans, policies, planning problems
- types of solutions: unsafe, safe (acyclic, cyclic)
  - Find-solution, Find-acyclic-solution, Find-safe-solution
- Guided-find-safe-solution
  - call find-solution to get an unsafe solution
  - call find-solution again on the leaves
  - if dead-ends are encountered, modify actions that lead to them
- Find-safe-solution-by-determinization
  - Like Guided-find-safe-solution, but call classical planner on determinized domain, convert plan into policy
- 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
  - convergence in safely explorable domains