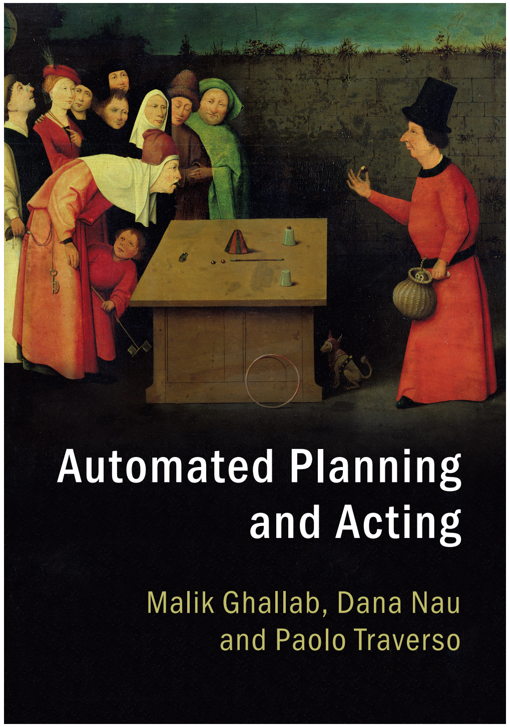
Last update: April 18, 2019

Chapter 5

Deliberation with Nondeterministic Domain Models

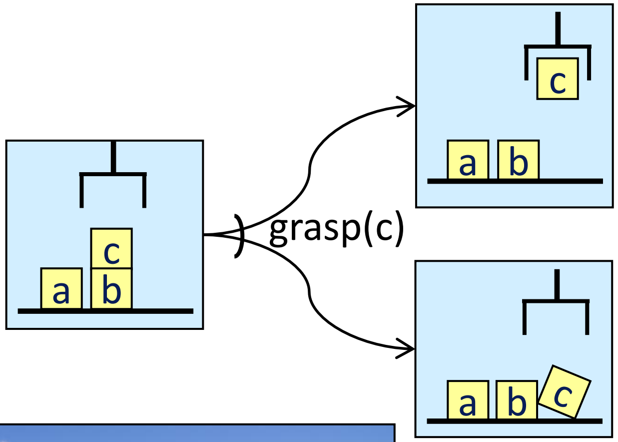


http://www.laas.fr/planning

Dana S. Nau
University of Maryland

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

Nondeterministic Planning Domains

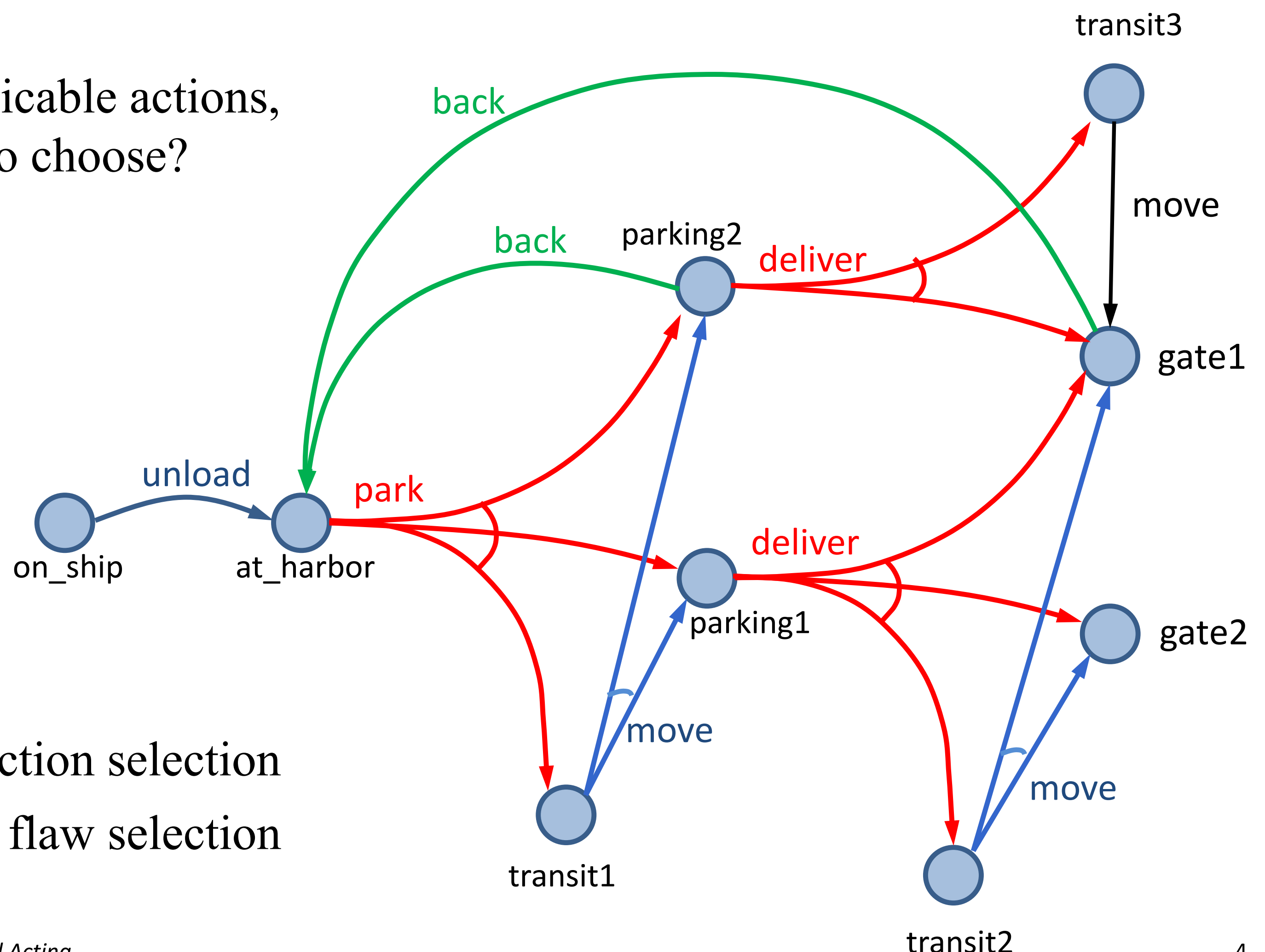
- 3-tuple (S, A, γ)
  - S and A – finite sets of states and actions
  - $\gamma: S \times A \to 2^S$
- $\gamma(s,a)$ = {all possible "next states" after applying action a in state s}
  - a is applicable in state s iff $\gamma(s,a) \neq \emptyset$
- Applicable(s) = {all actions applicable in s} = $\{a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

$a(z_1, \ldots, z_k)$
pre: $p_1, \ldots, p_m$
eff$_1$: $e_{11}, e_{12}, \ldots$
eff$_2$: $e_{21}, e_{22}, \ldots$
$\ldots$
eff$_n$: $e_{n1}, e_{n2}, \ldots$

- Problem: n may be combinatorially large
  - Suppose a can cause any possible combination of effects $e_1, e_2, \ldots, e_k$
  - Need eff$_1$, eff$_2$, …, eff$_{2^k}$
    - One for for each combination
  - Section 5.4: a way to alleviate this
- For now, ignore most of that
  - states, actions ⇔ nodes, edges in a graph

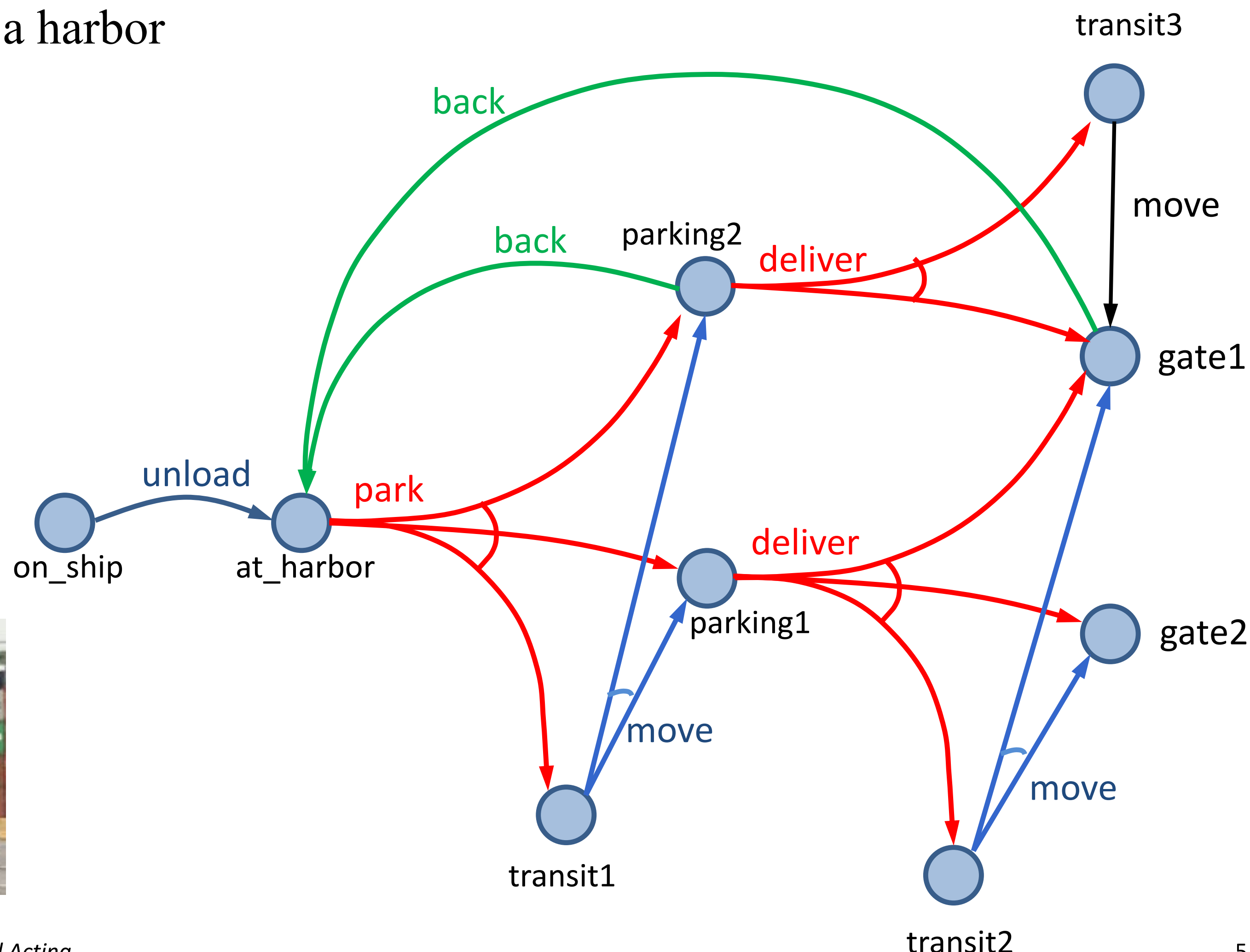
Nondeterministic Planning Domains

- 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
  - *OR* branch ⇔ action selection
  - *AND* branch ⇔ flaw selection



Example

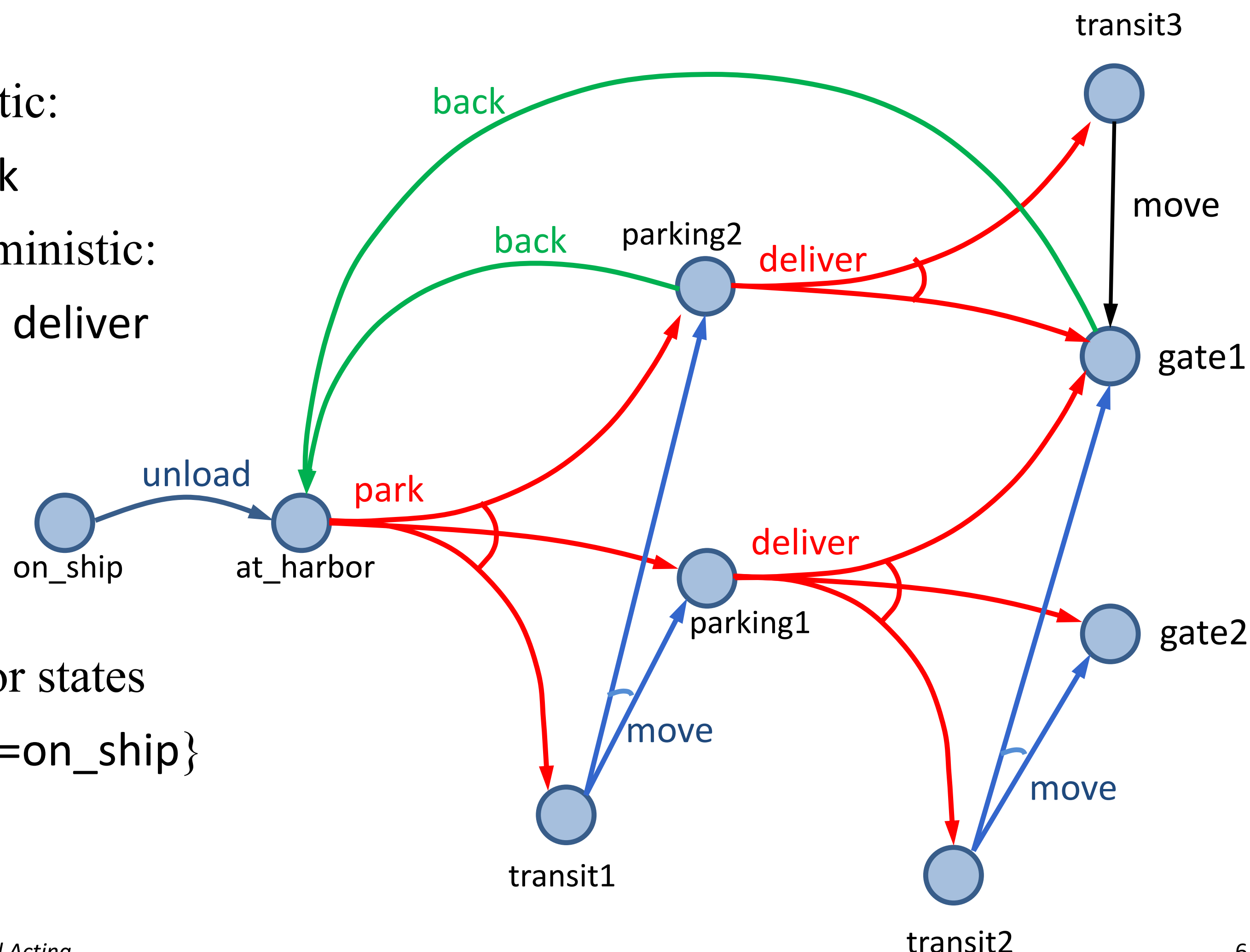
- Very simple harbor management domain
  - Unload a single item from a ship
  - Move it around a harbor



Example

- One state variable: pos(item)

- Five actions
  - Two deterministic:
    - unload, back
  - Three nondeterministic:
    - park, move, deliver



- Simplified names for states
  - For {pos(item)=on_ship} write on_ship

Actions

- park

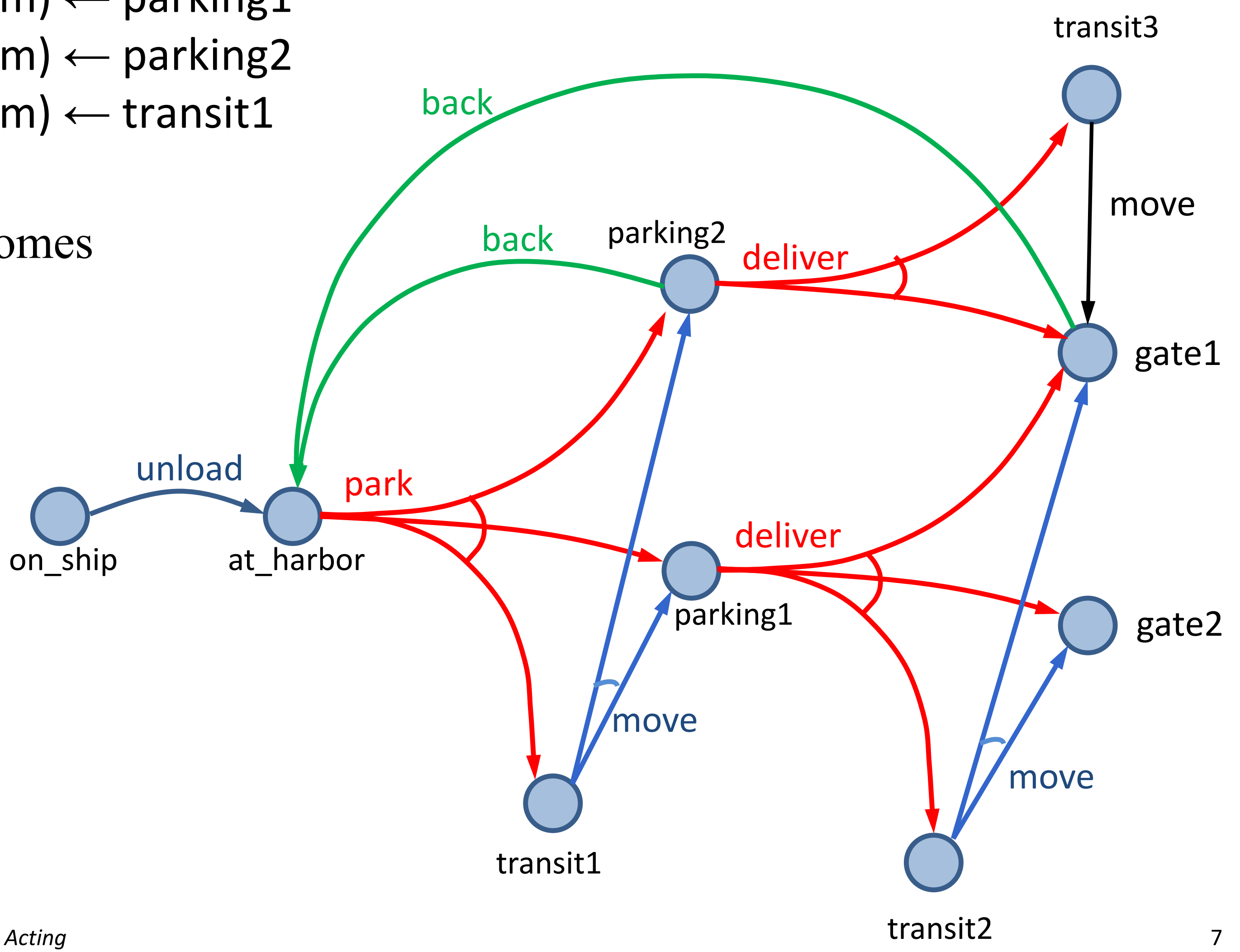
  pre: pos(item) = at_harbor

  eff_1: pos(item) ← parking1

  eff_2: pos(item) ← parking2

  eff_3: pos(item) ← transit1

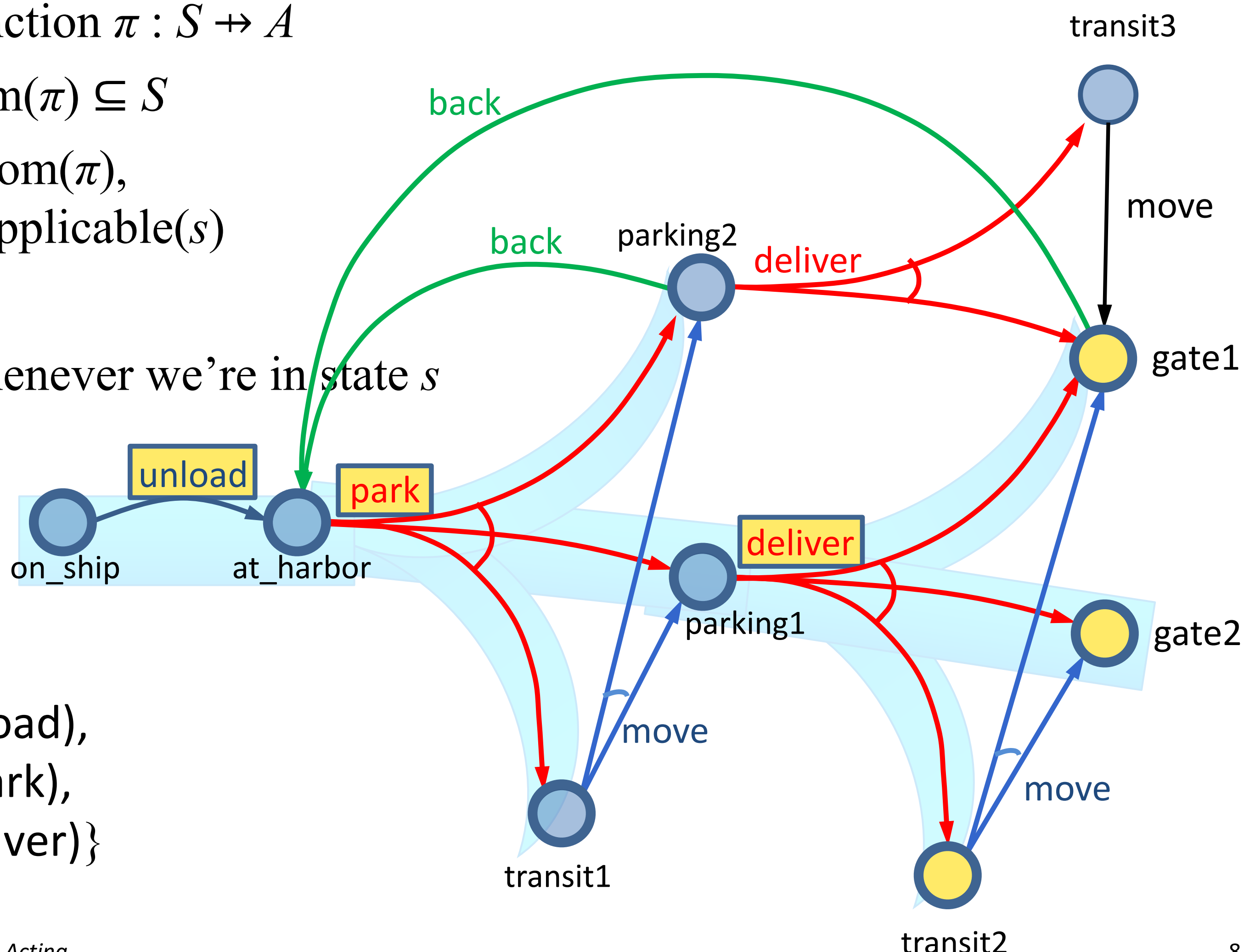
- Three possible outcomes
  - put item in parking1 or parking2 if one of them has space
  - or in transit1 if there's no parking space



~~Plans~~ Policies

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

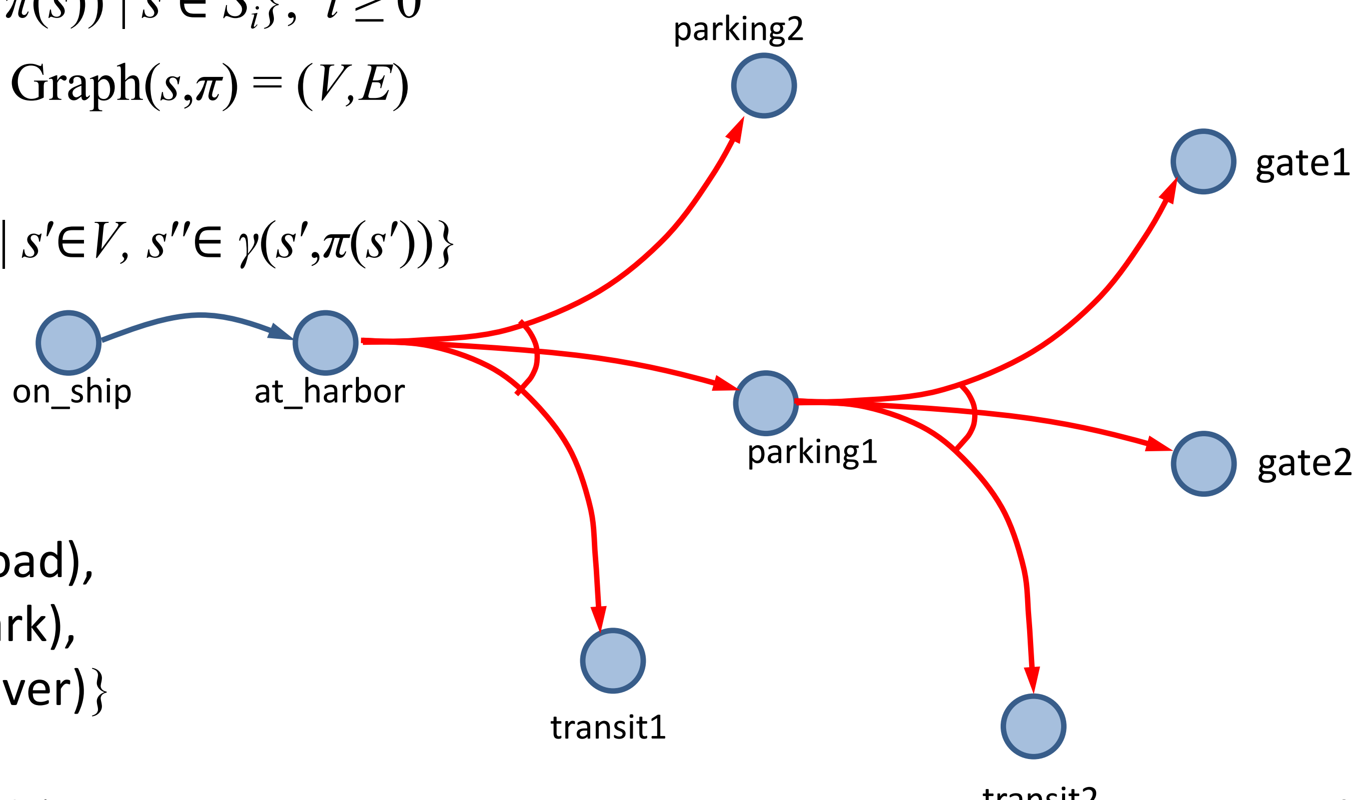
- *Policy*: a *partial* function $\pi : S \nrightarrow A$
  - i.e., $\mathrm{Dom}(\pi) \subseteq S$
  - For every $s \in \mathrm{Dom}(\pi)$, require $\pi(s) \in \mathrm{Applicable}(s)$
- Meaning:
  - perform $\pi(s)$ whenever we're in state s



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

Definitions Over Policies

- *Transitive closure*:
  {all states reachable from s using π}
  - $\hat{\gamma}(s,\pi) = S_0 \cup S_1 \cup S_2 \cup \ldots$
    - $S_0 = \{s\}$
    - $S_{i+1} = \cup\{\gamma(s,\pi(s)) \mid s \in S_i\},\ \ i \geq 0$
- *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*$(s,\pi) = \hat{\gamma}(s,\pi) \setminus \mathrm{Dom}(\pi)$
  - may be empty



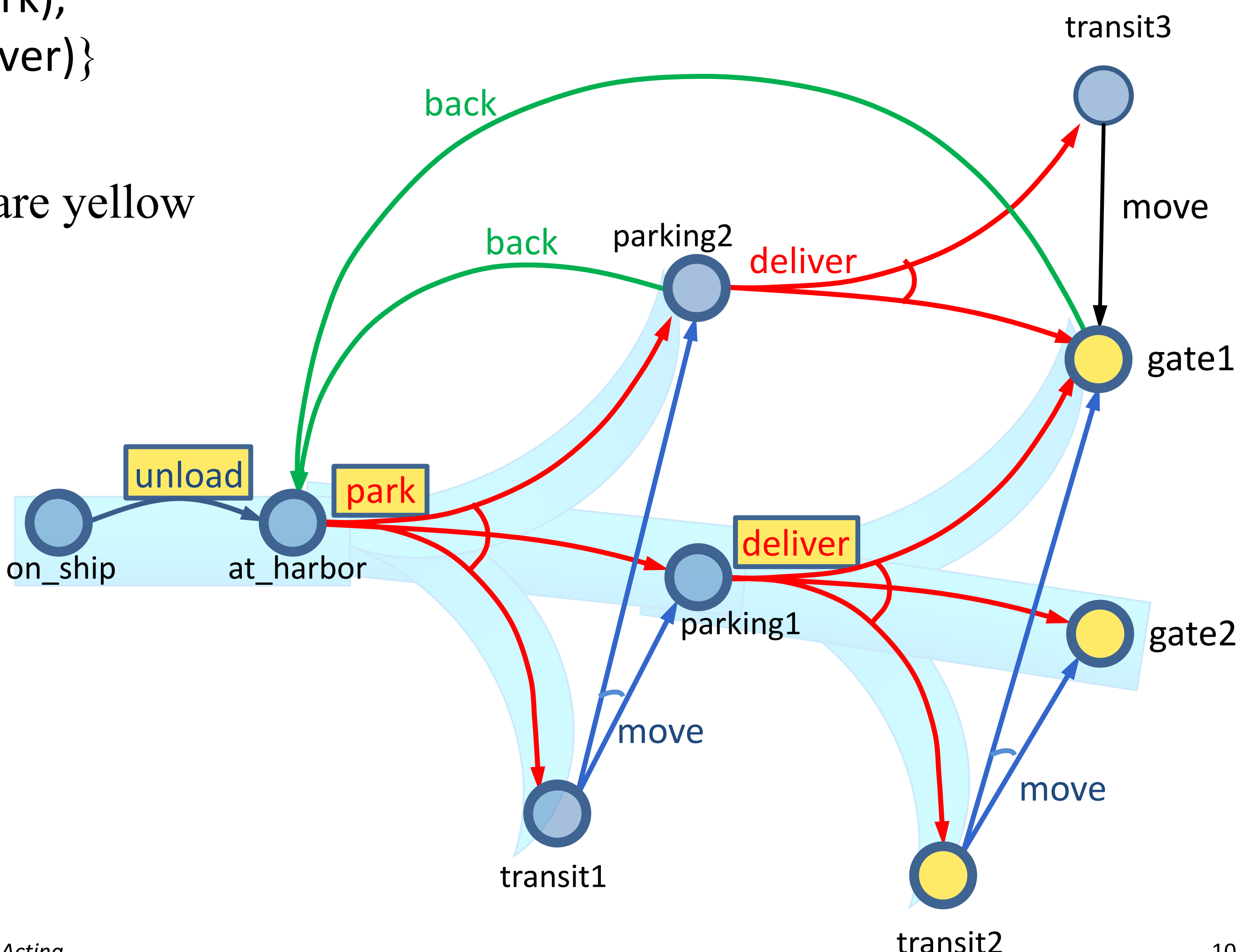
- $\pi_1 = \{$(on_ship, unload),
  (at_harbor, park),
  (parking1, deliver)$\}$

Definitions Over Policies

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

- *leaves*(on_ship, π_1) are yellow

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



Performing a Policy

- PerformPolicy(π)

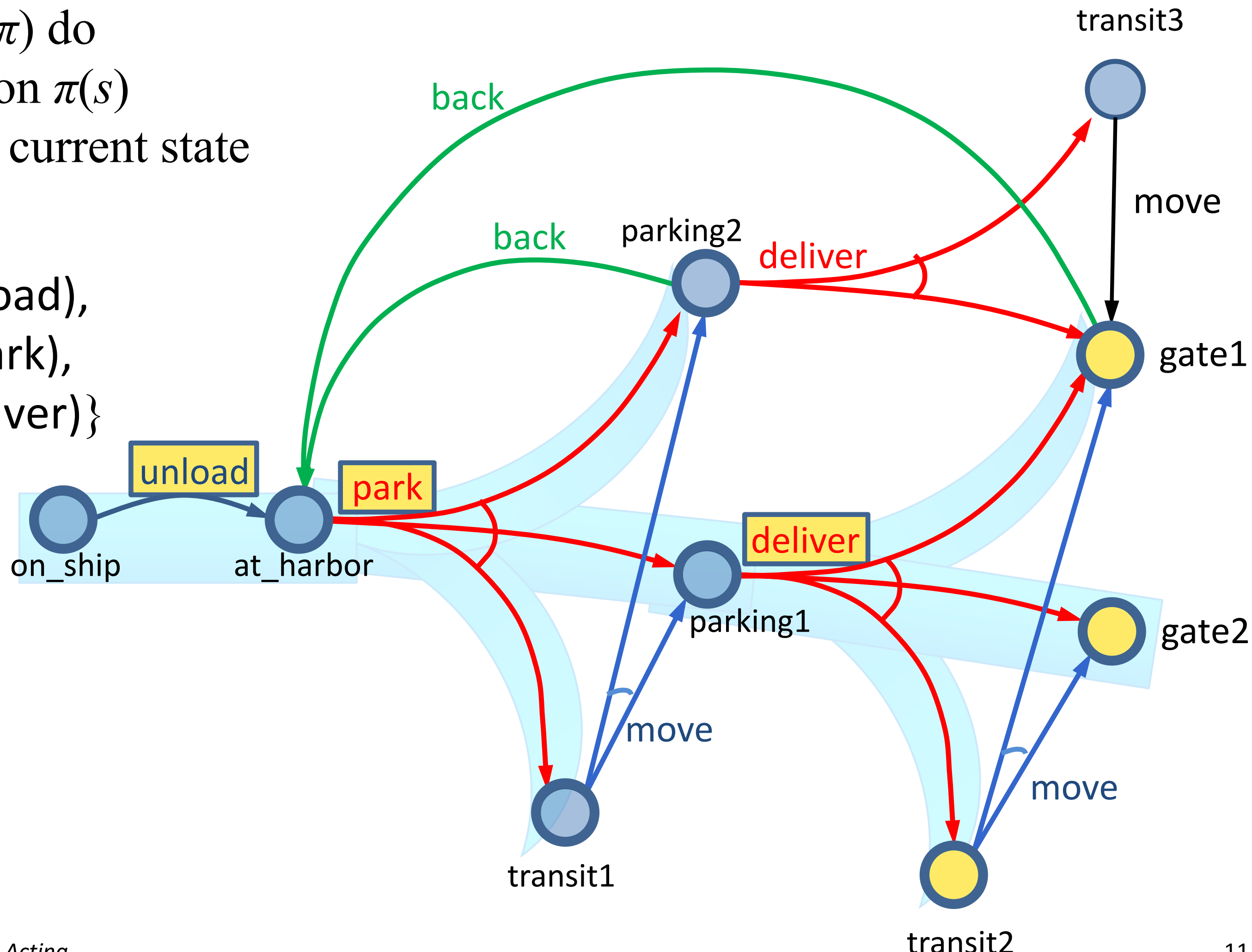
  $s \leftarrow$ observe current state

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

  perform action $\pi(s)$

  $s \leftarrow$ observe current state

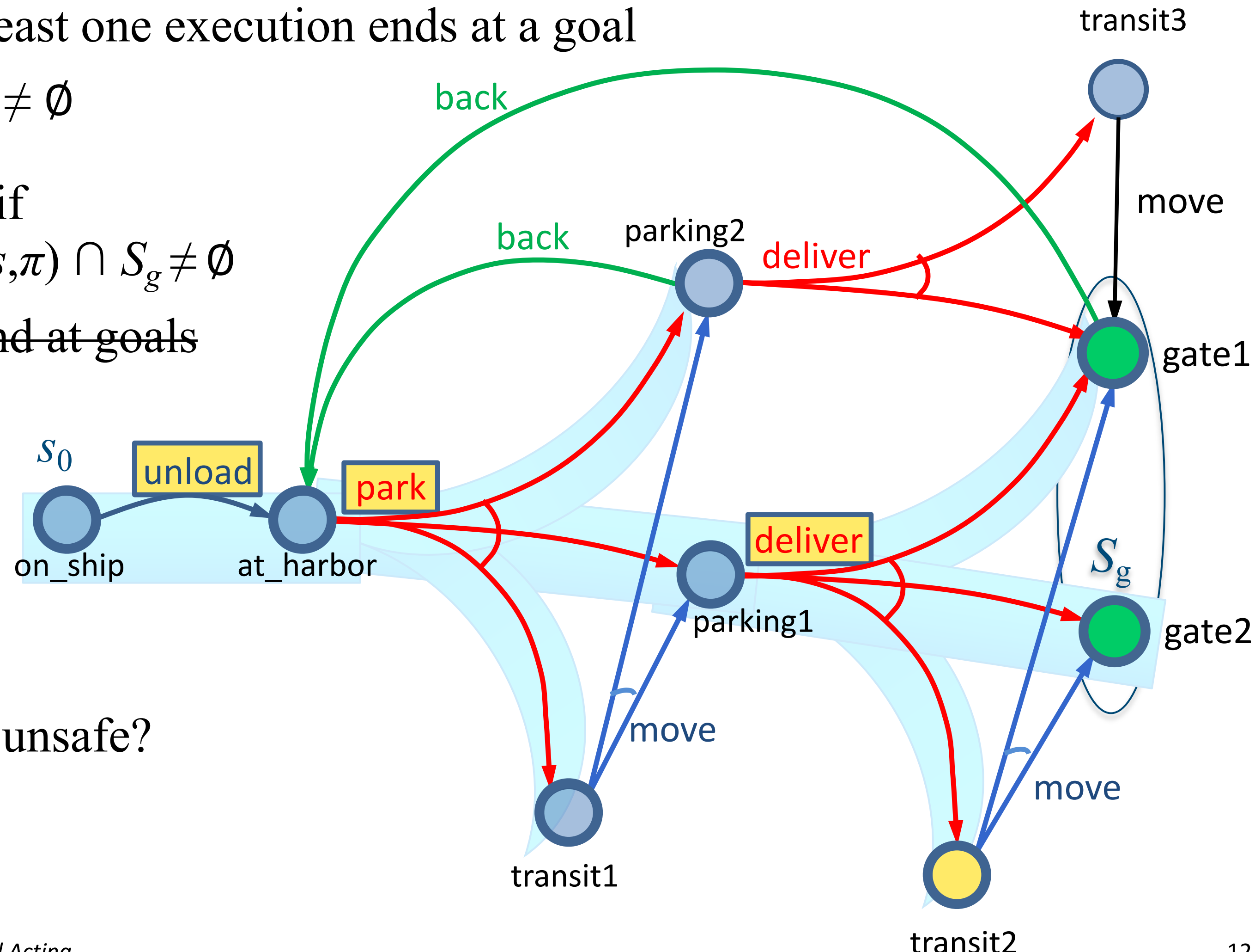
- π_1 = {(on_ship, unload),
  (at_harbor, park),
  (parking1, deliver)}



Planning Problems and Solutions

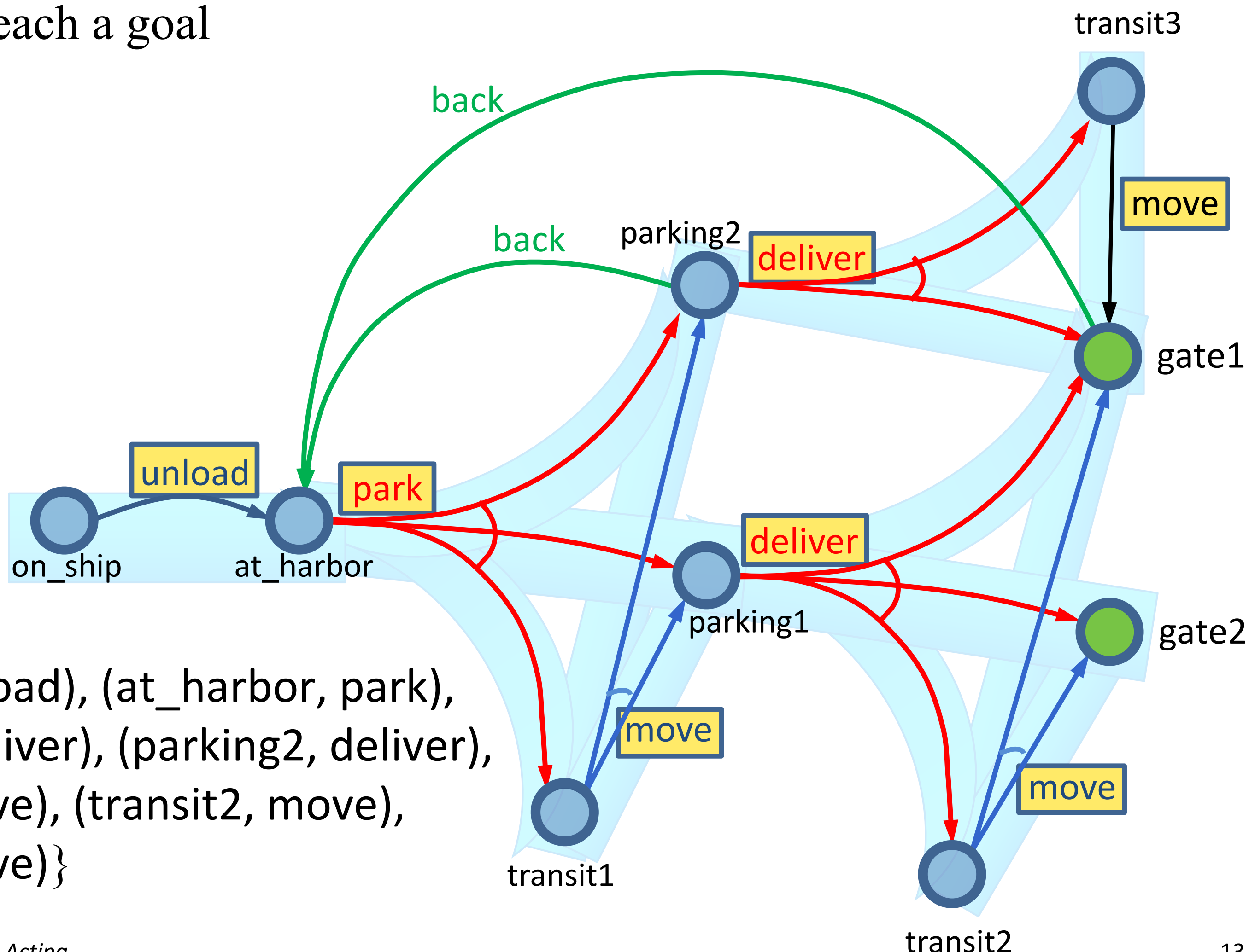
π_1 = {(on_ship, unload),
(at_harbor, park),
(parking1, deliver)}

- 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 $\forall s \in \hat{\gamma}(s_0,\pi)$, $leaves(s,\pi) \cap S_g \neq \emptyset$
  - ~~all executions end at goals~~
  - at every node of Graph(s_0,π), the goal is reachable
- Otherwise, *unsafe*
  - Is π_1 safe or unsafe?



Safe Solutions

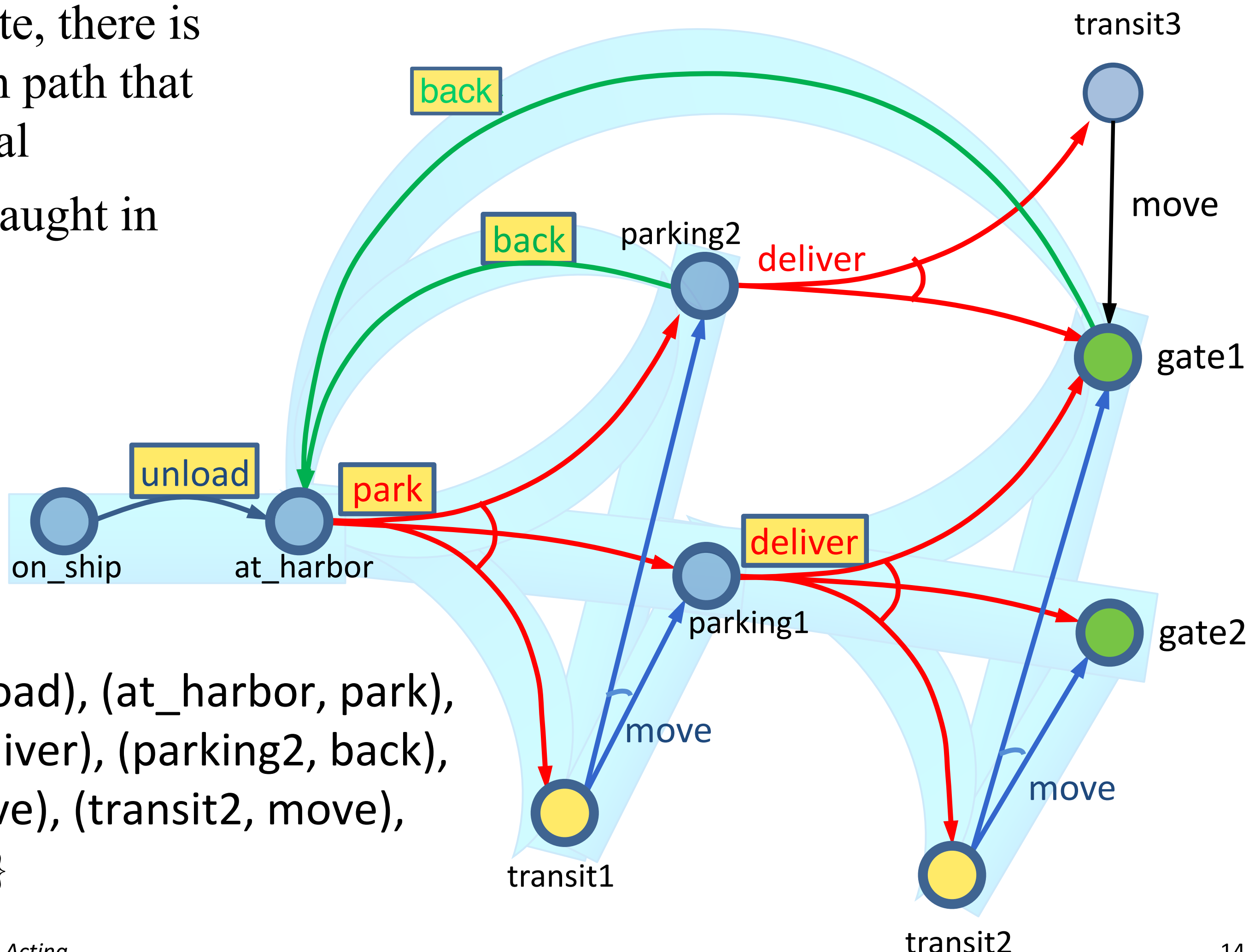
- *Acyclic* safe solution
  - Graph(s_0,π) is acyclic, and *leaves*(s,π) ⊆ S_g
  - Guaranteed to reach a goal



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

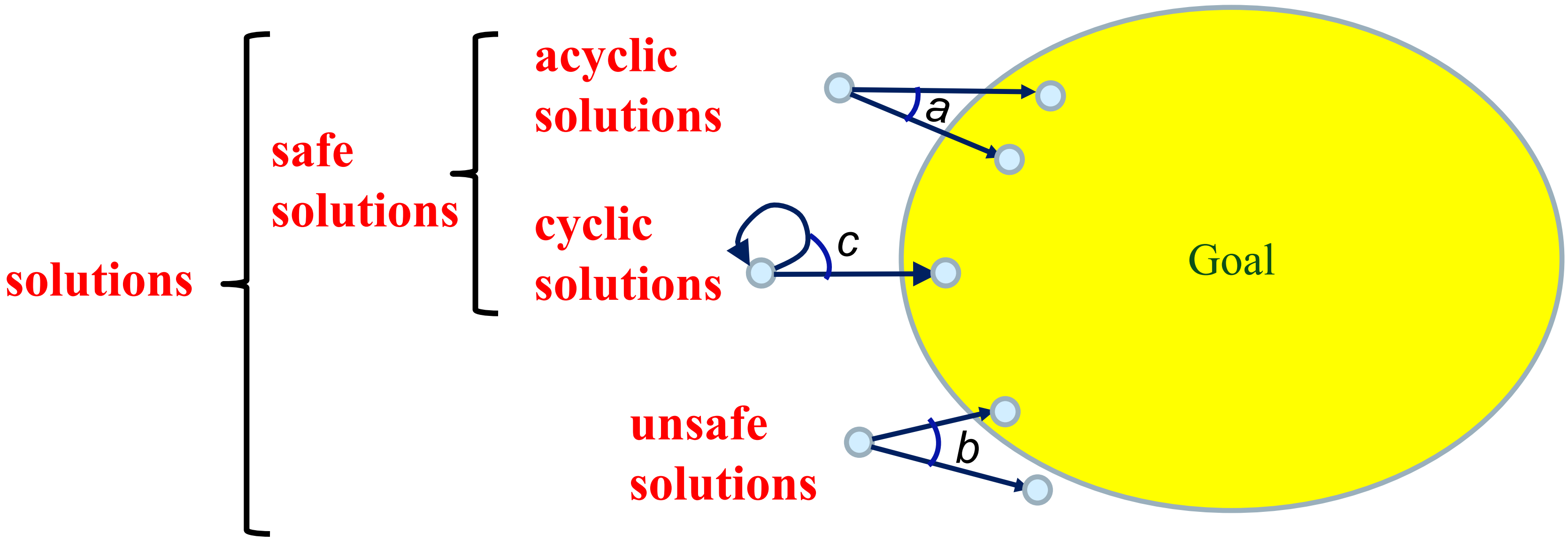
Safe Solutions

- *Cyclic* safe solution
  - Graph(s_0, π) is cyclic, $leaves(s,\pi) \subseteq S_g$, $\forall s \in \hat{\gamma}(s_0,\pi)$, $leaves(s,\pi) \cap S_g \neq \emptyset$
    - At every state, there is an execution path that ends at a goal
  - Will never get caught in a dead end



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

Kinds of Solutions



15

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)
$\pi \leftarrow \varnothing$; $s \leftarrow s_0$; $\mathit{Visited} \leftarrow \{s_0\}$
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 $s' \in \gamma(s, a)$ ← Decide which state to plan for
if $s' \in \mathit{Visited}$ then return failure ← Cycle-checking
$\pi(s) \leftarrow a$; $\mathit{Visited} \leftarrow \mathit{Visited} \cup \{s'\}$; $s \leftarrow s'$

Poll: which should (*) be?
1. nondeterministically choose
2. arbitrarily choose

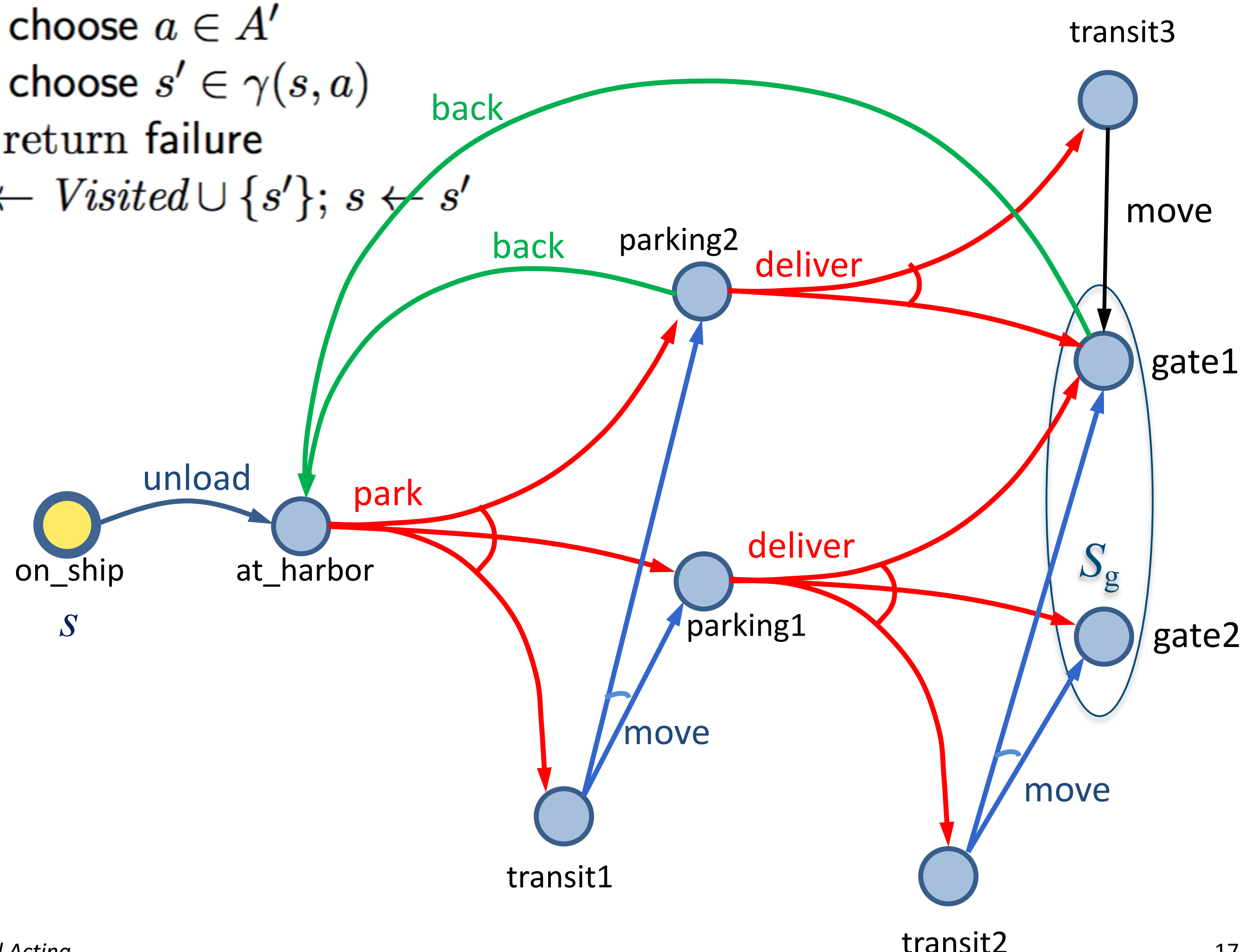
Example

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

$\pi = \{\}$

$Visited = \{$on_ship$\}$



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

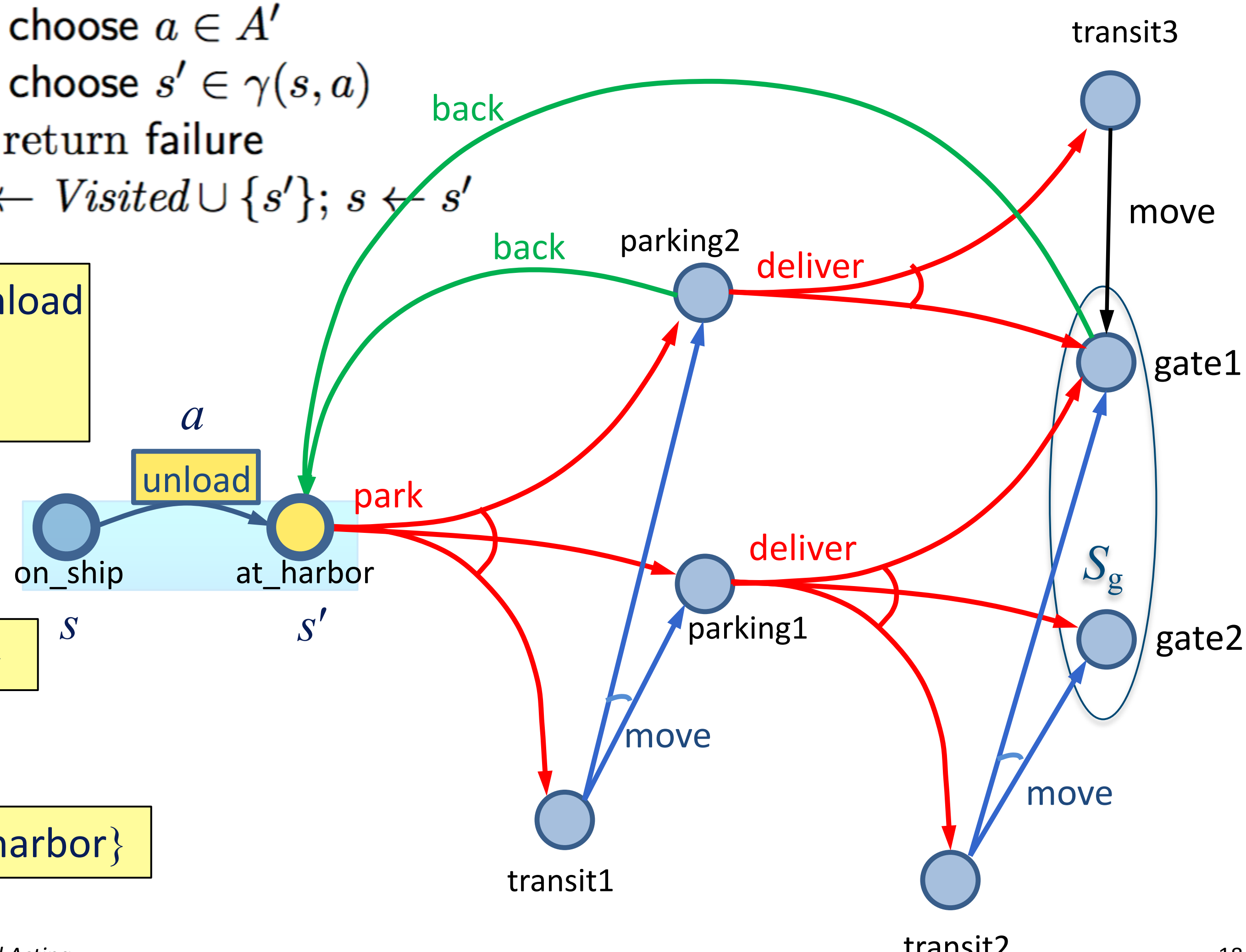
$\pi(s) \leftarrow a;\ \ Visited \leftarrow Visited \cup \{s'\};\ s \leftarrow s'$

Example

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

π = {(on_ship, unload)}

$Visited$ = {on_ship, at_harbor}



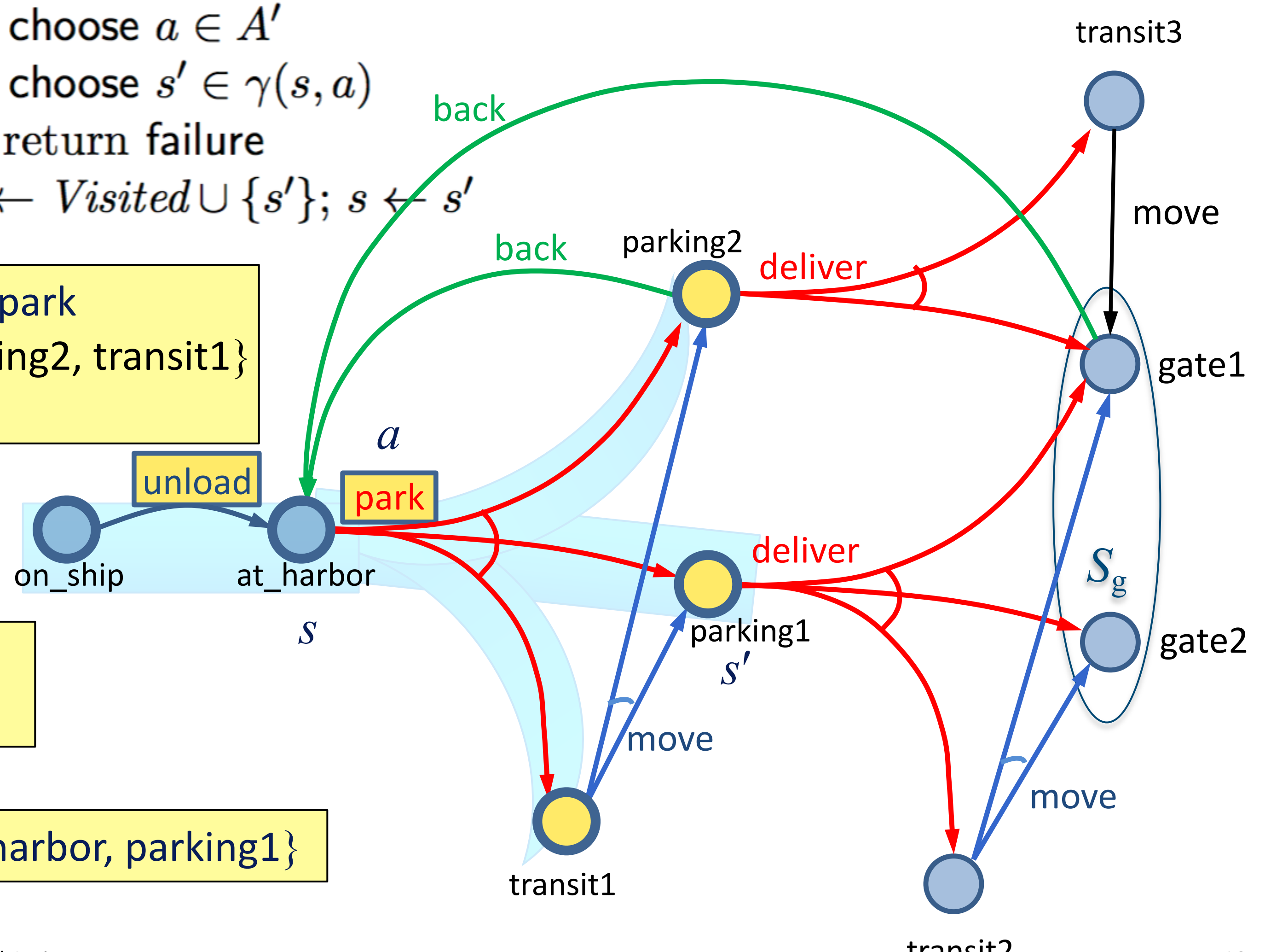
Example

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' = parking1

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

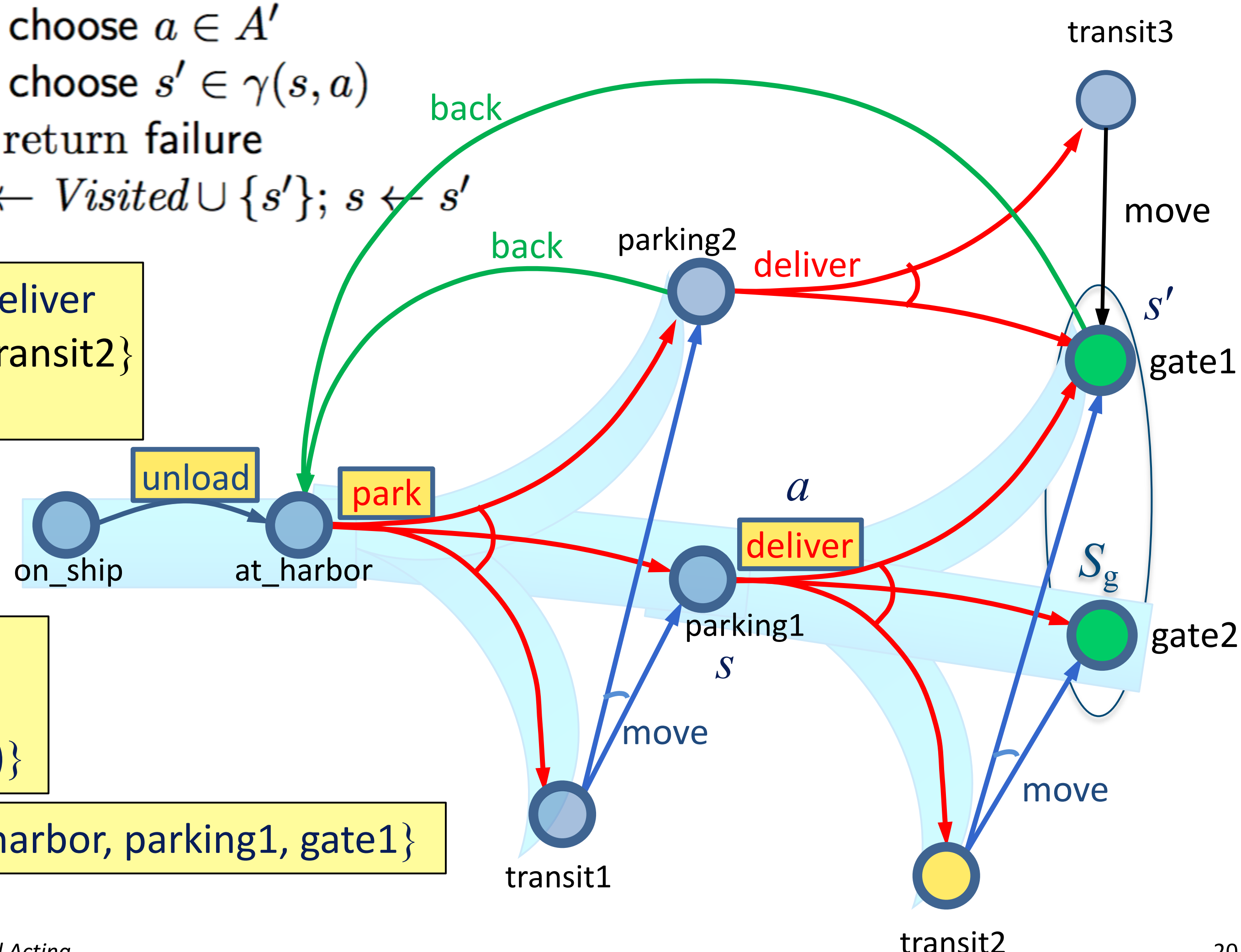
$Visited$ = {on_ship, at_harbor, parking1}



Example

Find-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing;\ \ s \leftarrow s_0;\ \ \mathit{Visited} \leftarrow \{s_0\}$
  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 $s' \in \mathit{Visited}$ then return failure
    $\pi(s) \leftarrow a;\ \ \mathit{Visited} \leftarrow \mathit{Visited} \cup \{s'\};\ s \leftarrow s'$

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



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

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

Example

Find-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing;\ \ s \leftarrow s_0;\ \ \mathit{Visited} \leftarrow \{s_0\}$

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 $s' \in \mathit{Visited}$ then return failure

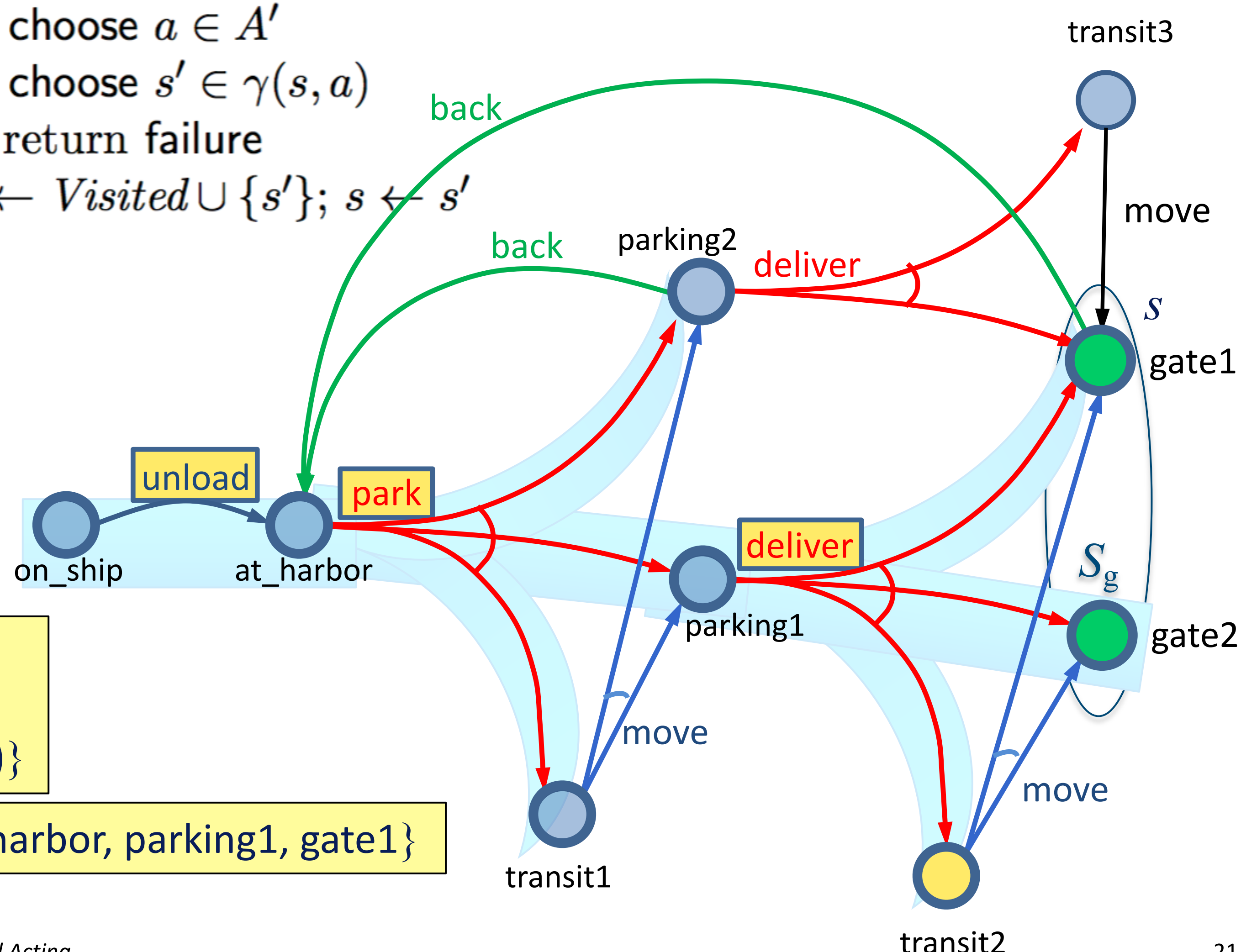
  $\pi(s) \leftarrow a;\ \ \mathit{Visited} \leftarrow \mathit{Visited} \cup \{s'\};\ s \leftarrow s'$

s = gate1

gate1 is a goal, so return π

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

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



Finding Acyclic Safe Solutions

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

Check for cycles:

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)

$\pi \leftarrow \varnothing$

$\mathit{Frontier} \leftarrow \{s_0\}$

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

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

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

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

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

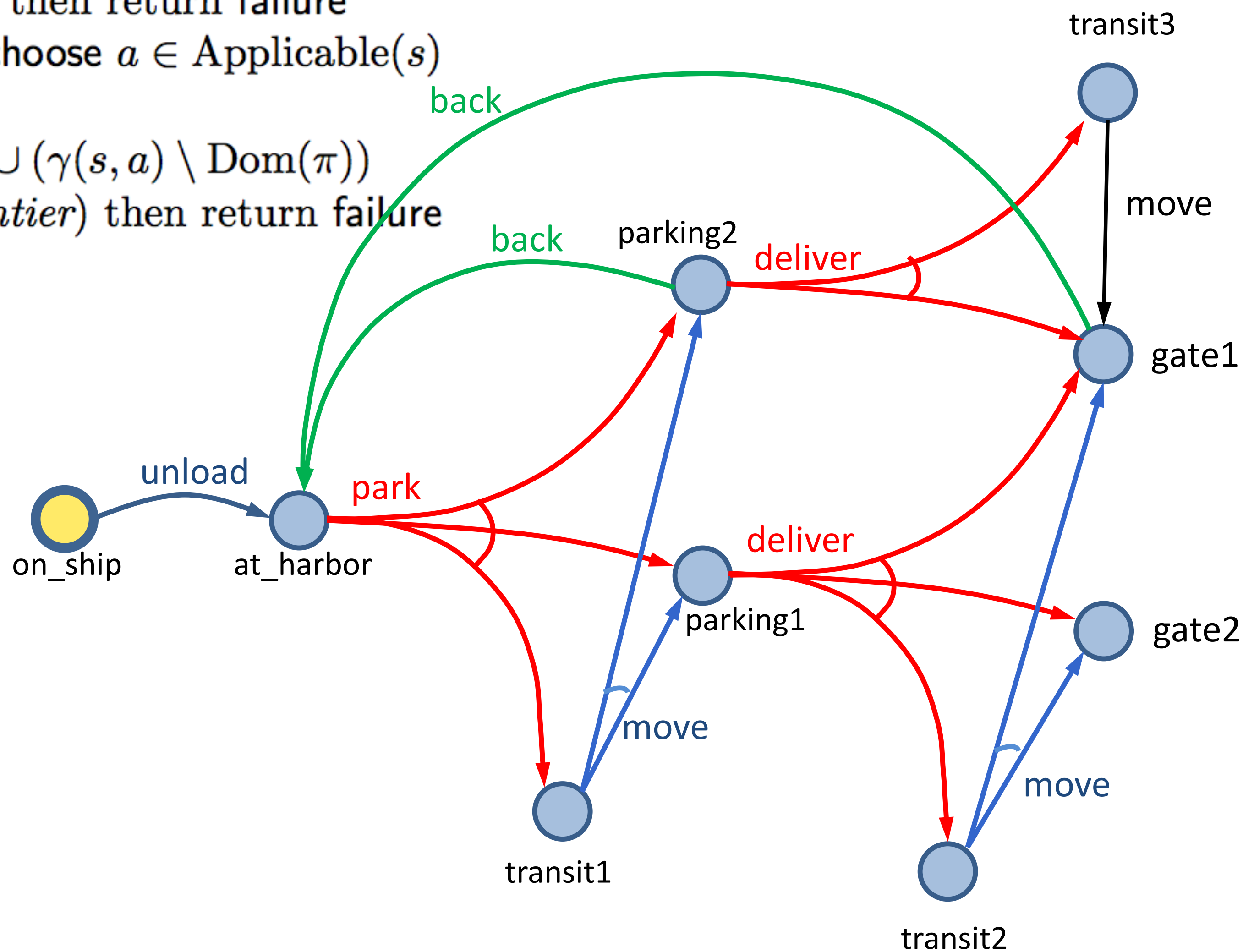
  $\mathit{Frontier} \leftarrow \mathit{Frontier} \cup (\gamma(s,a) \setminus \text{Dom}(\pi))$

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

return π

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

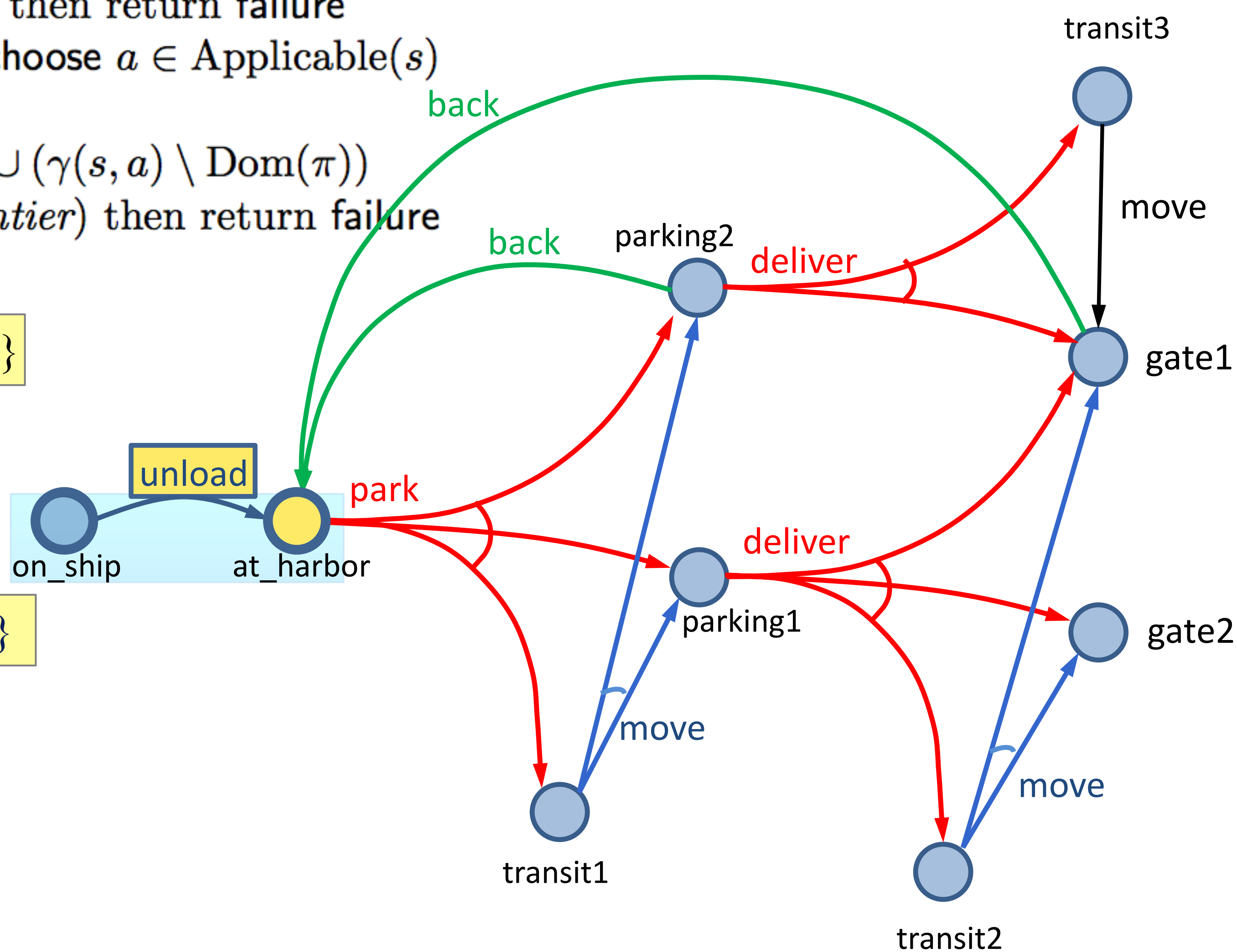
$\pi = \{\}$



Example

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

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

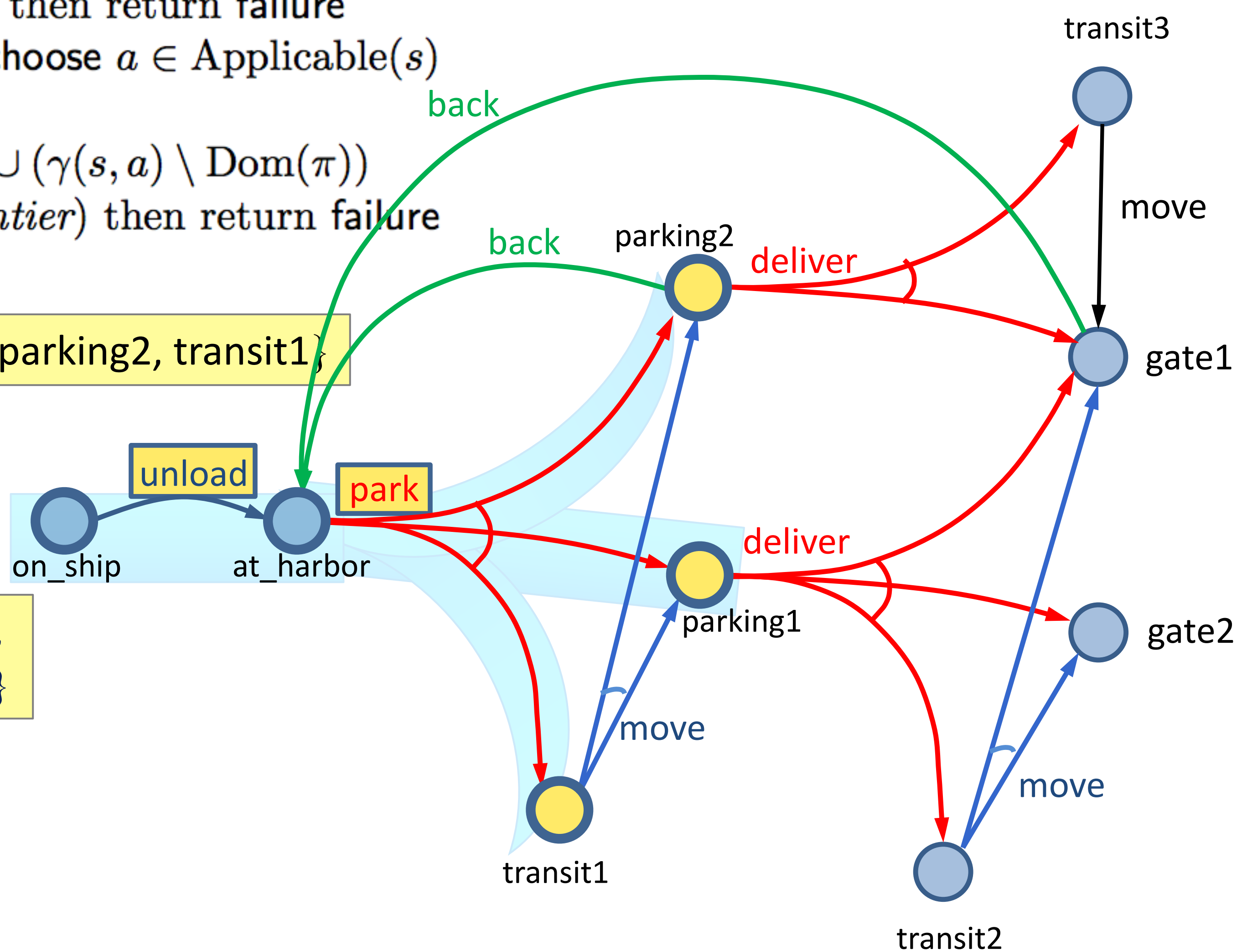


$\pi = \{(\text{on\_ship, unload})\}$

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

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



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

Example

Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$\mathit{Frontier} \leftarrow \{s_0\}$

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

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

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

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

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

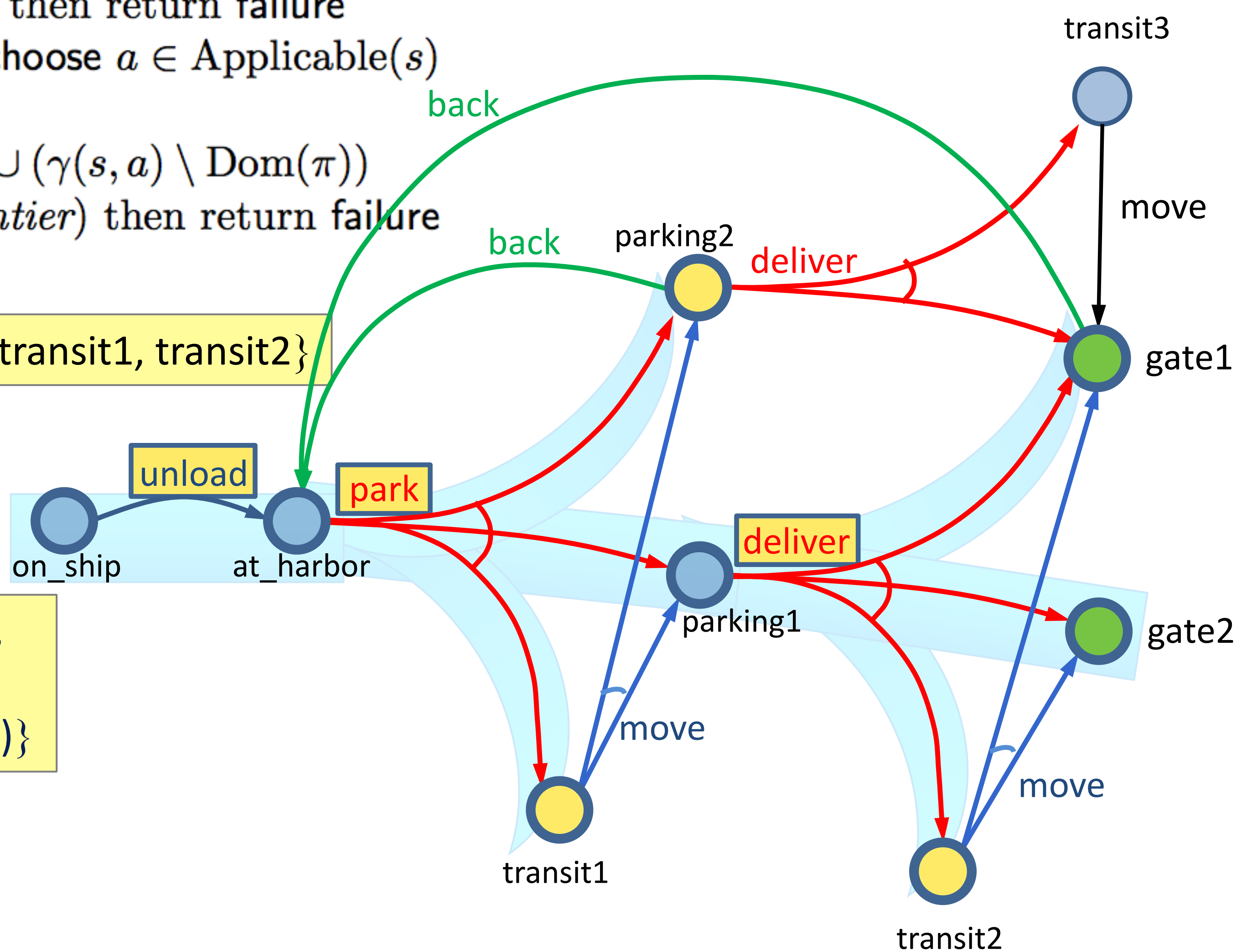
  $\mathit{Frontier} \leftarrow \mathit{Frontier} \cup (\gamma(s,a) \setminus \mathrm{Dom}(\pi))$

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

return π

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

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



Example

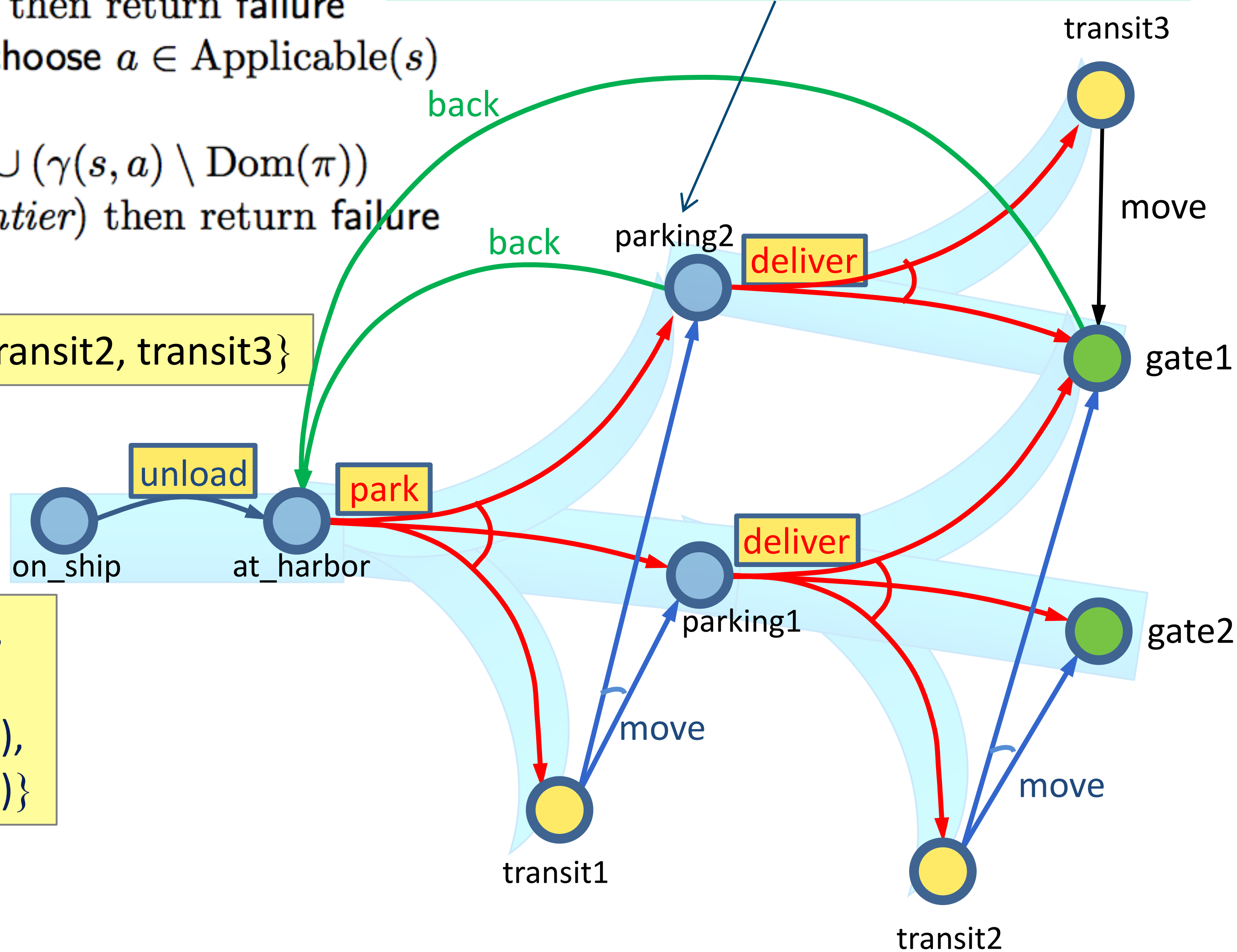
Find-Acyclic-Solution (Σ, s_0, S_g)
$\pi \leftarrow \varnothing$
$Frontier \leftarrow \{s_0\}$
for every $s \in Frontier \setminus S_g$ do
$Frontier \leftarrow Frontier \setminus \{s\}$
if $\text{Applicable}(s) = \varnothing$ then return failure
nondeterministically choose $a \in \text{Applicable}(s)$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

$Frontier \setminus S_g = \{$transit1, transit2, transit3$\}$

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

nondeterministically choose back or deliver

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



Example

Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$\mathit{Frontier} \leftarrow \{s_0\}$

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

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

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

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

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

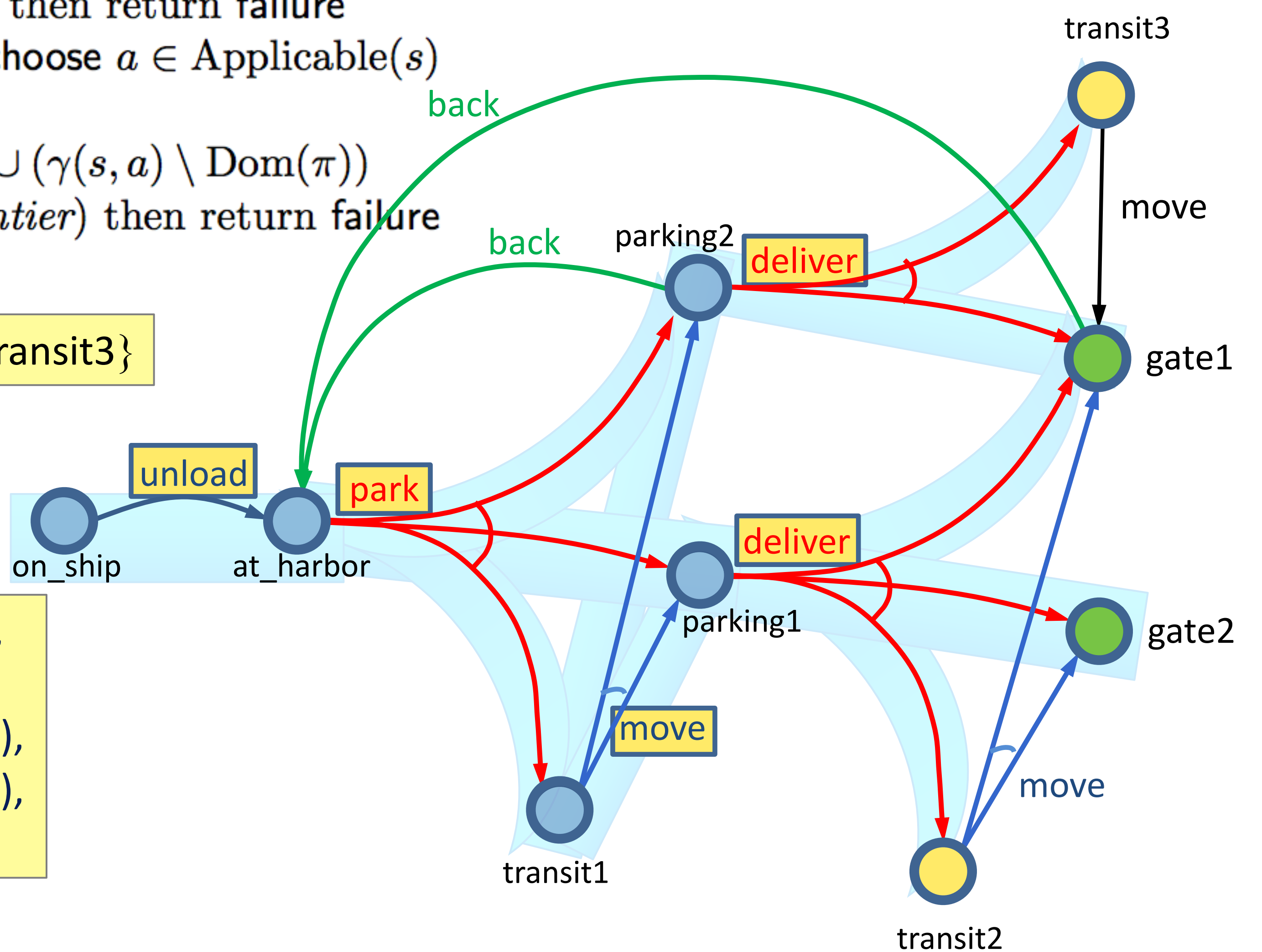
  $\mathit{Frontier} \leftarrow \mathit{Frontier} \cup (\gamma(s,a) \setminus \text{Dom}(\pi))$

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

return π

$\mathit{Frontier} \setminus S_g = \{$transit2, transit3$\}$

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

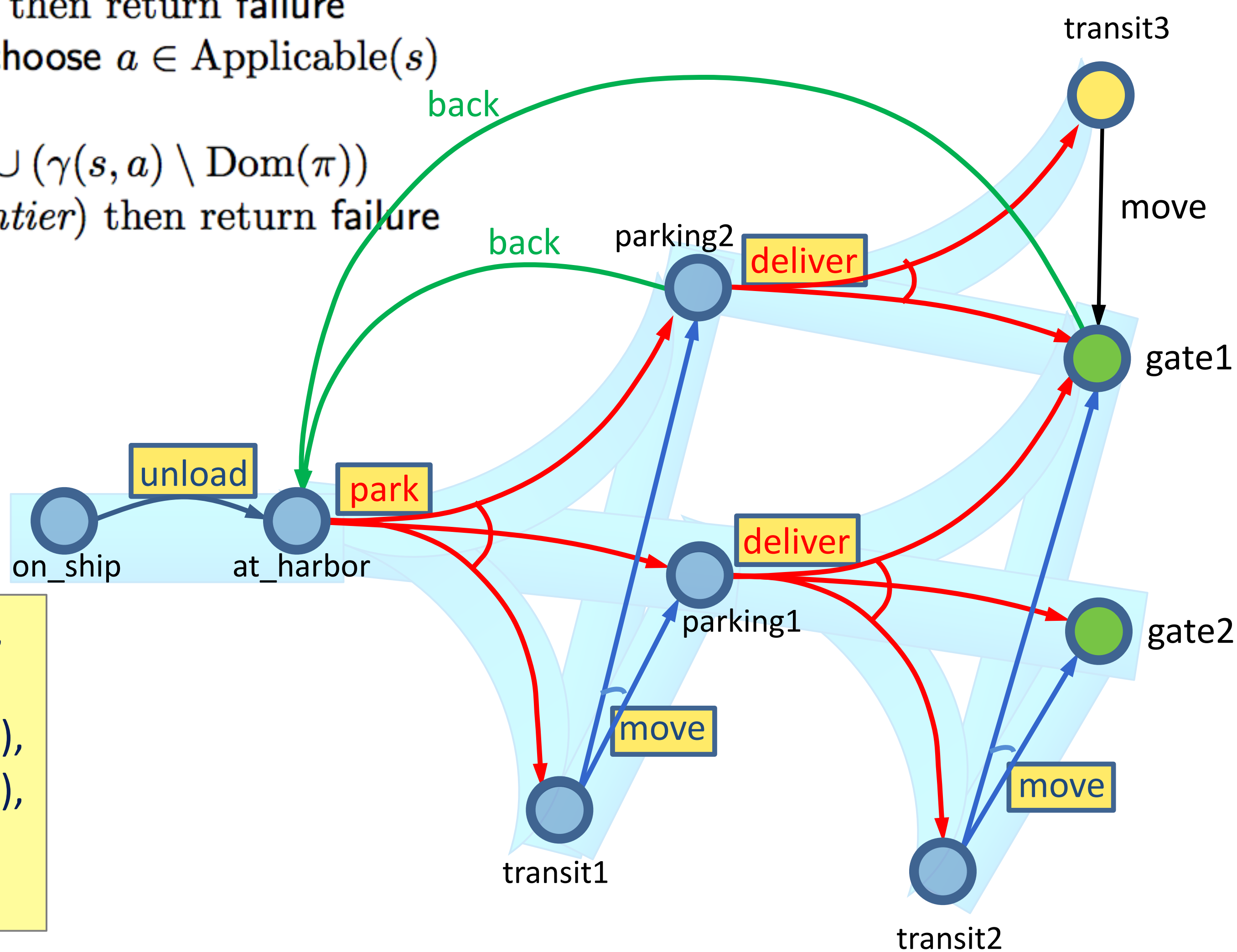


Example

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

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

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



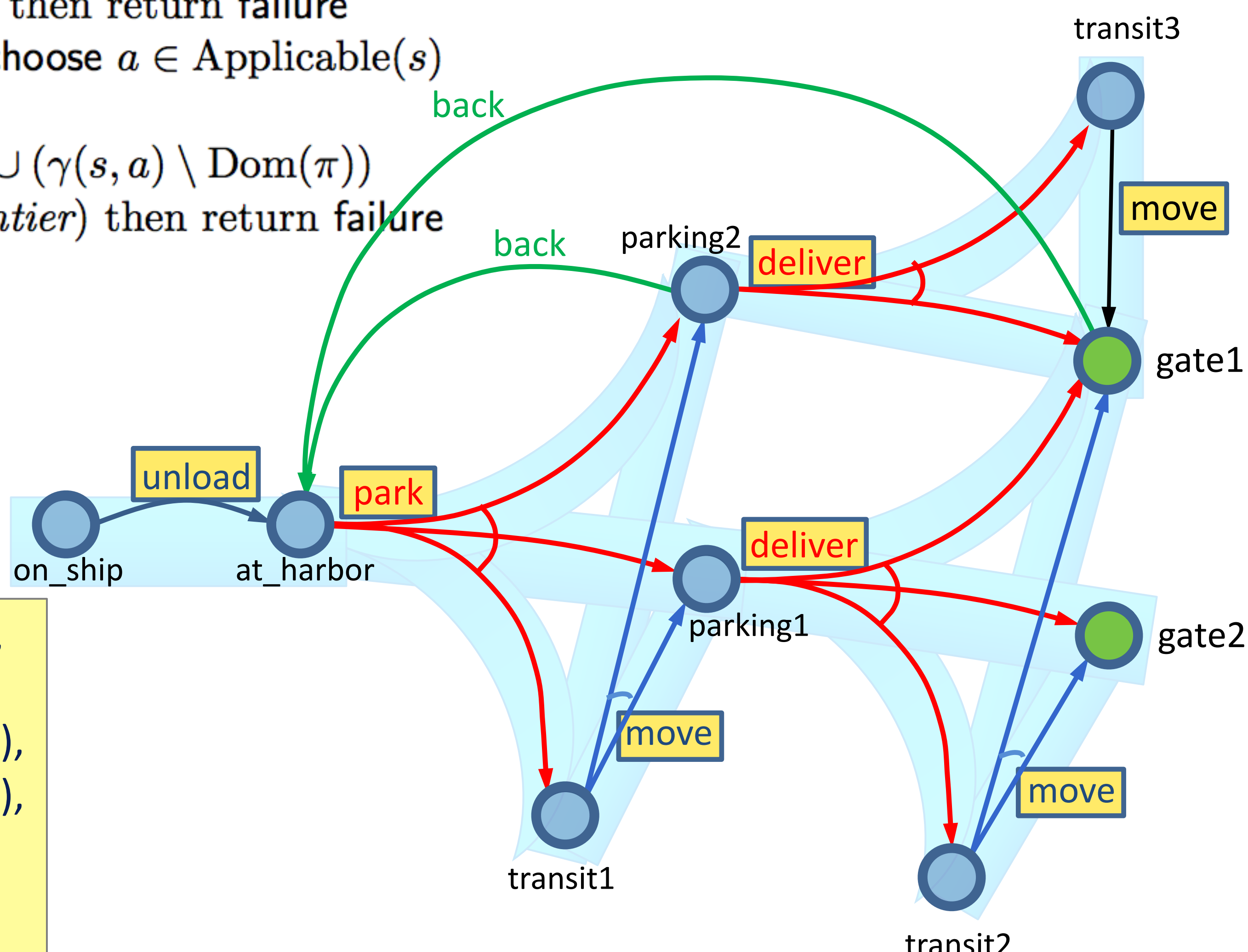
Example

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)
    $\pi \leftarrow \pi \cup (s, a)$
    $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \mathrm{Dom}(\pi))$
    if has-loops$(\pi, a, Frontier)$ then return failure
return π

$Frontier \setminus S_g = \emptyset$

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)
    $\pi \leftarrow \varnothing$
    $Frontier \leftarrow \{s_0\}$ ← *Keep track of unexpanded states, like A**
    for every $s \in Frontier \setminus S_g$ do
        $Frontier \leftarrow Frontier \setminus \{s\}$
        if $\text{Applicable}(s) = \varnothing$ then return failure
        nondeterministically choose $a \in \text{Applicable}(s)$
        $\pi \leftarrow \pi \cup (s, a)$
        $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \text{Dom}(\pi))$ ← *Add all outcomes that π doesn't already handle*
        if has-unsafe-loops$(\pi, a, Frontier)$ then return failure
    return π

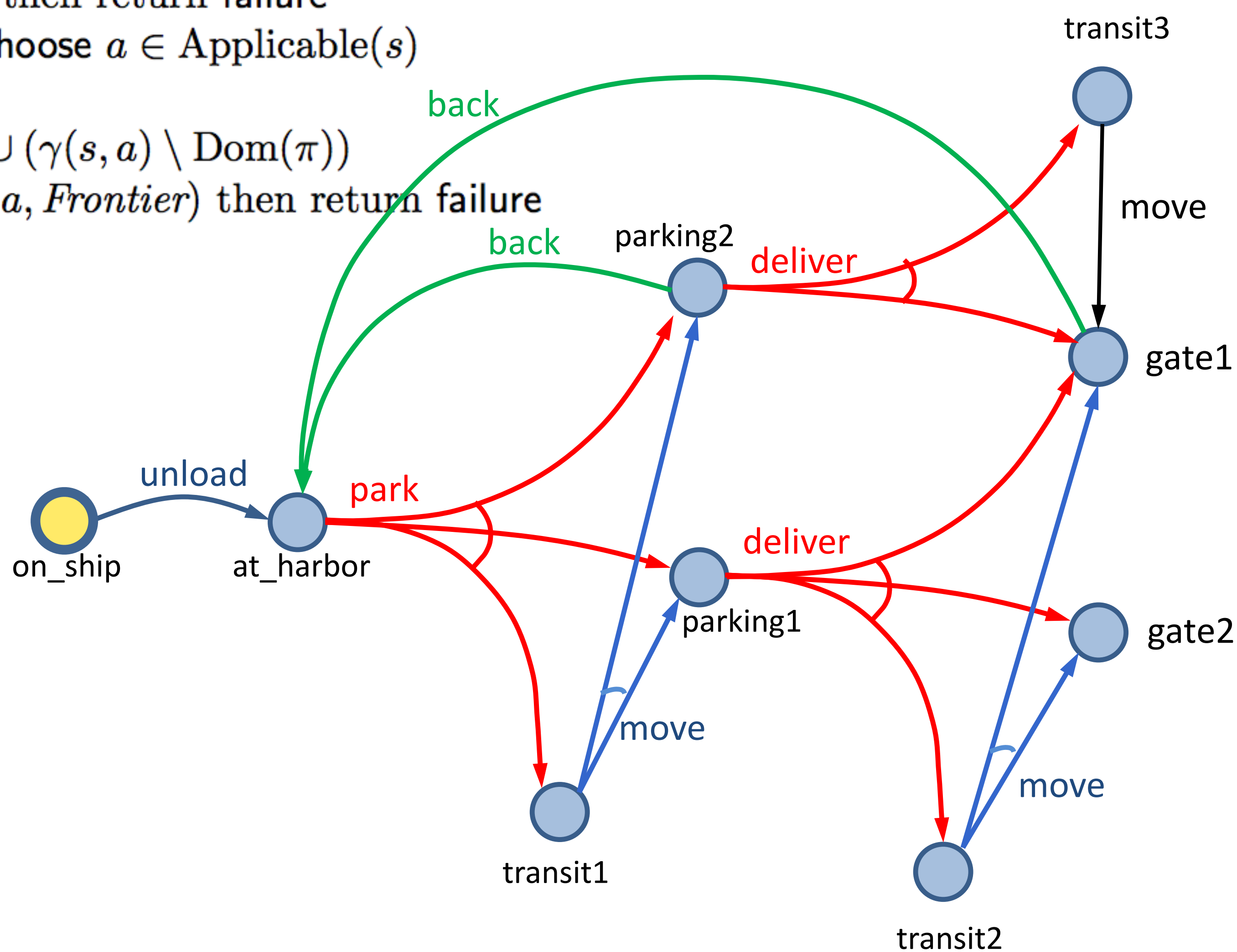
- Same as Find-Acyclic-Solution except for one difference:
- has-unsafe-loops instead of has-loops
    - Check whether π contains any cycles that can't be escaped:
    - For each $s' \in \gamma(s,a) \cap \text{Dom}(\pi)$, is $\hat{\gamma}(s',\pi) \cap Frontier = \varnothing$?

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)
    $\pi \leftarrow \pi \cup (s, a)$
    $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \mathrm{Dom}(\pi))$
    if has-unsafe-loops$(\pi, a, Frontier)$ then return failure
  return π

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

$\pi = \{\}$



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

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

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

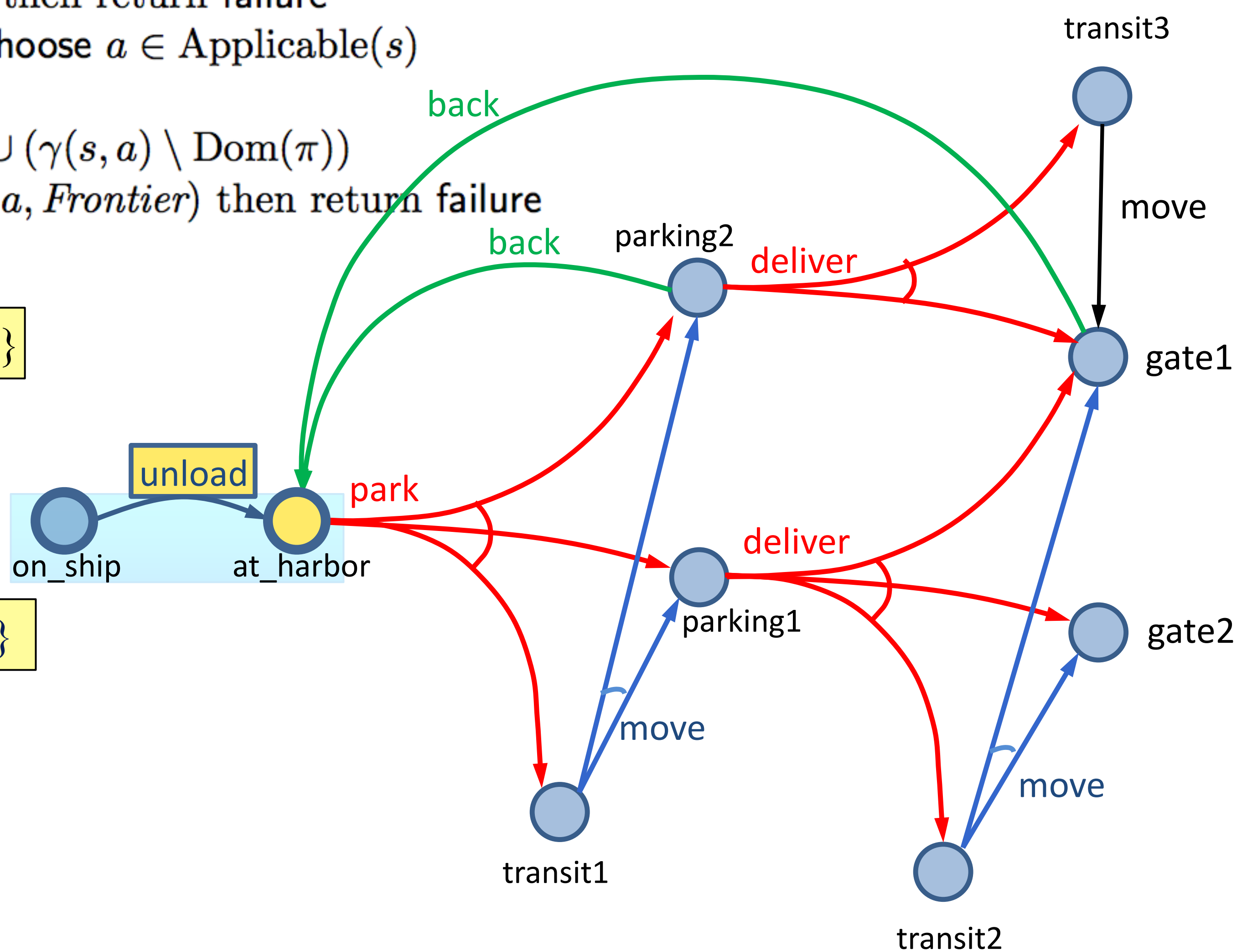
  $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \text{Dom}(\pi))$

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

return π

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

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



Example

Find-Safe-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$\mathit{Frontier} \leftarrow \{s_0\}$

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

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

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

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

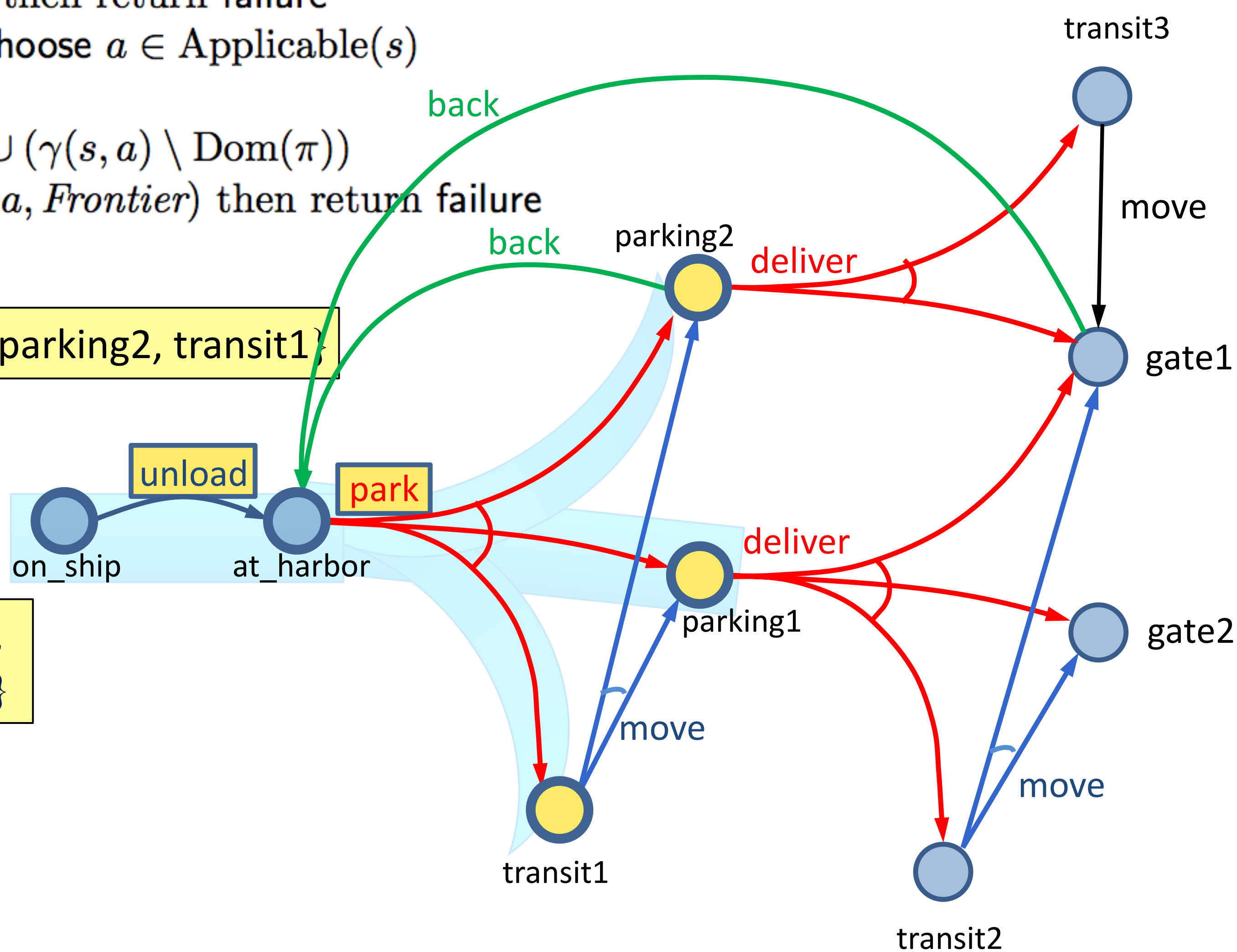
  $\pi \leftarrow \pi \cup (s, a)$

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

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

return π

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



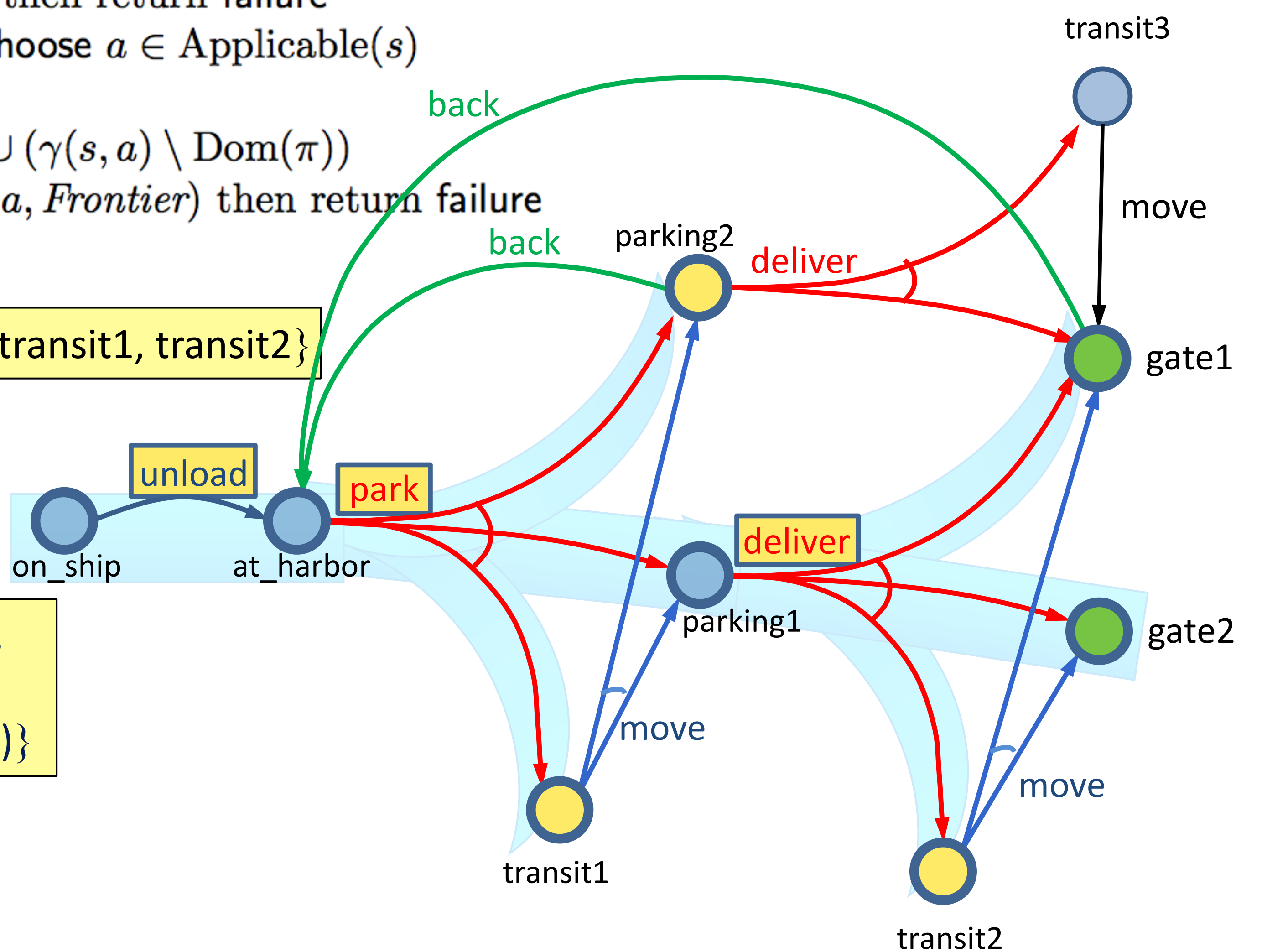
$\pi = \{$(on_ship, unload), (at_harbor, park)$\}$

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

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

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



Example

Find-Safe-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$\mathit{Frontier} \leftarrow \{s_0\}$

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

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

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

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

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

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

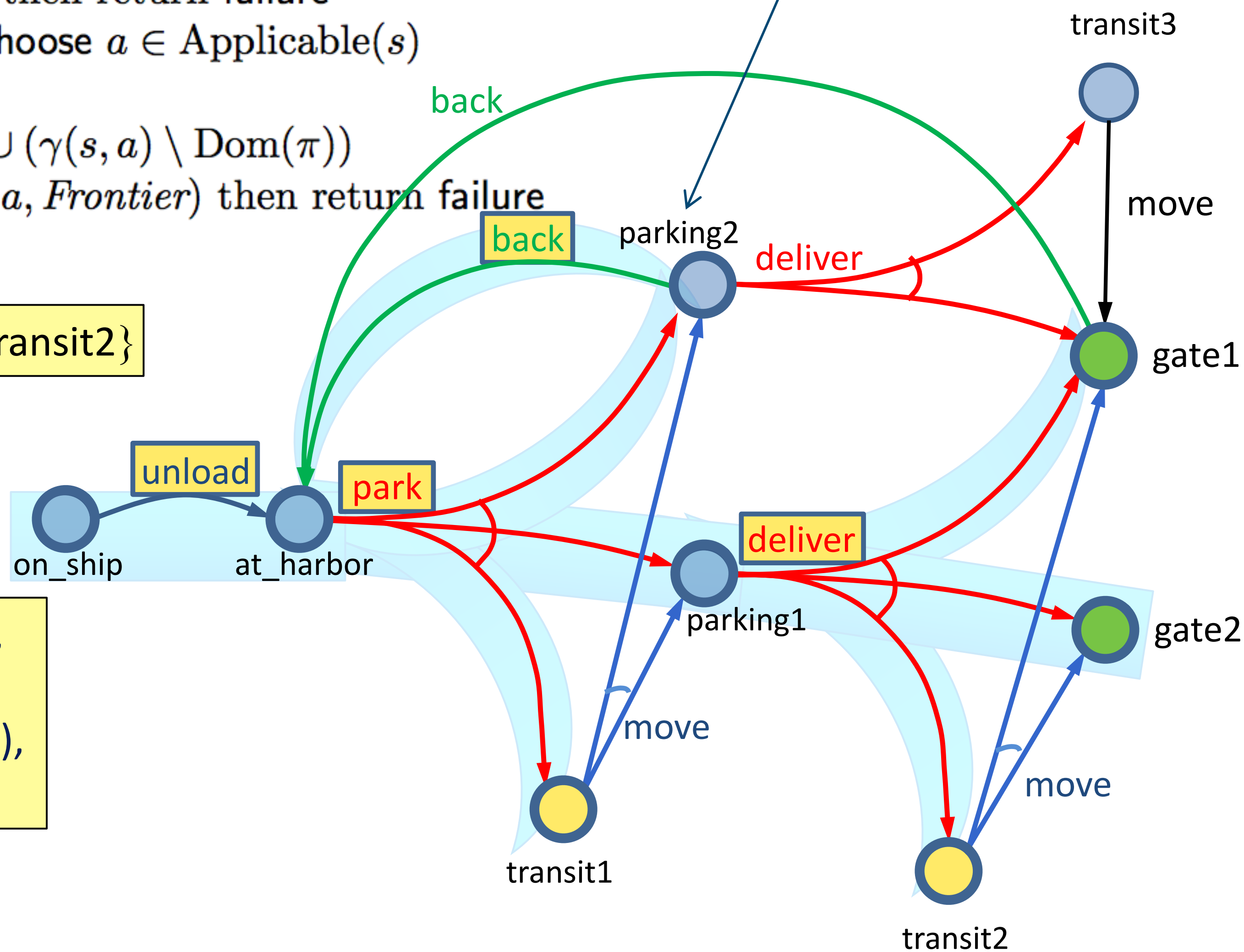
  if has-unsafe-loops$(\pi, a, \mathit{Frontier})$ then return failure

return π

Nondeterministically choose back or deliver

- back is OK: escapable cycle

$\mathit{Frontier} \setminus S_g = \{$transit1, transit2$\}$



$\pi = \{$(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)

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

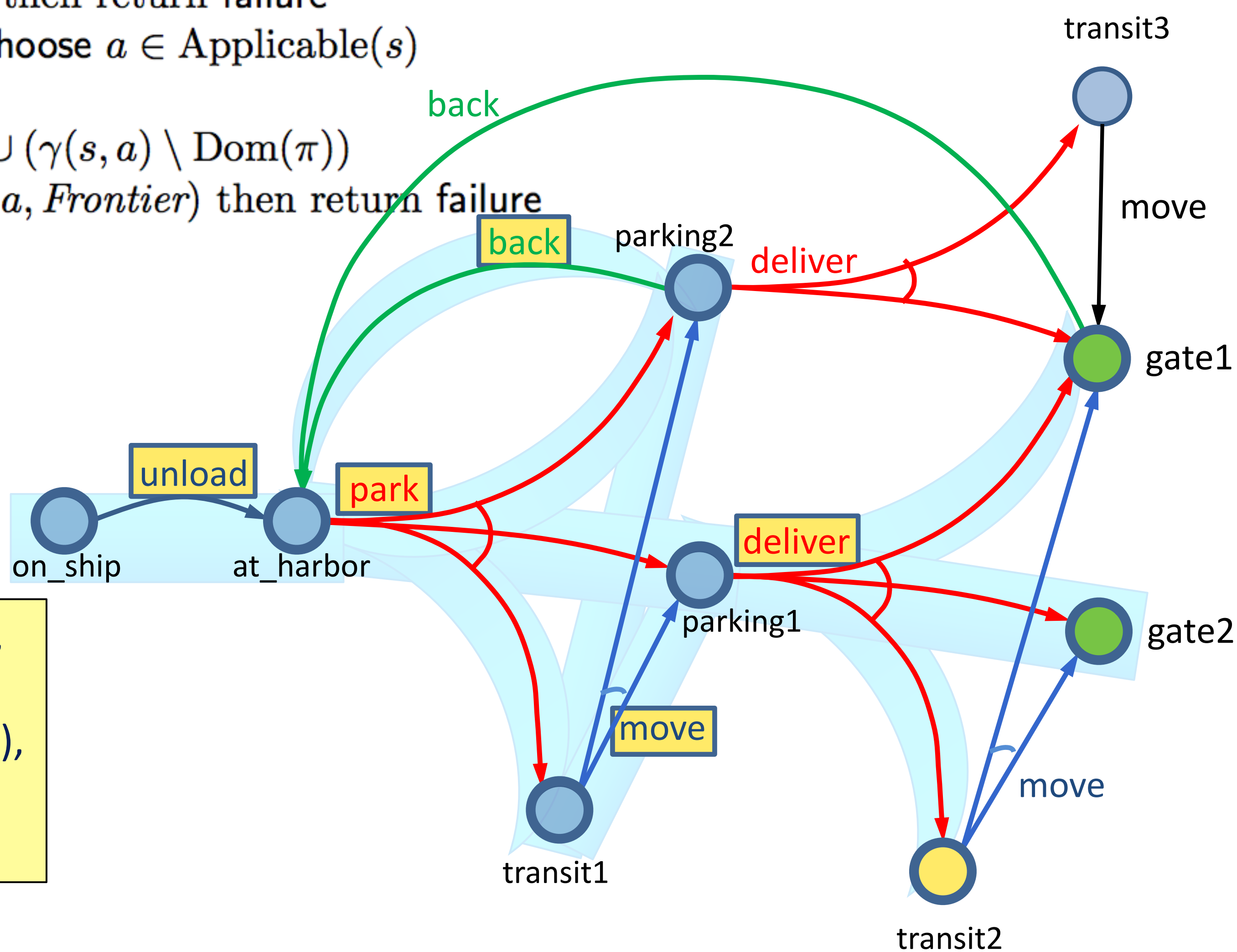
  $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \mathrm{Dom}(\pi))$

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

return π

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

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

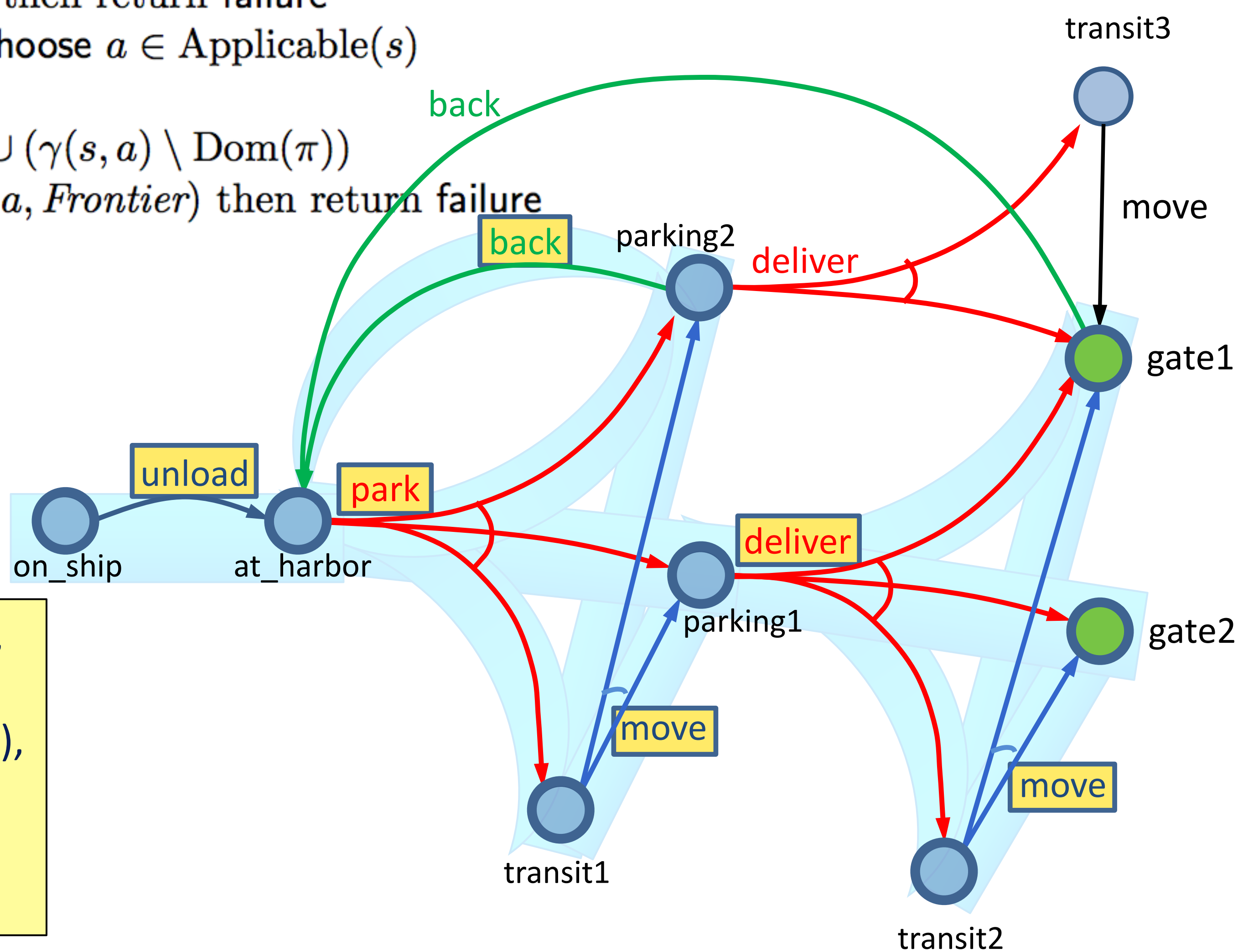


Example

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 Applicable$(s) = \varnothing$ then return failure
  nondeterministically choose $a \in$ Applicable(s)
  $\pi \leftarrow \pi \cup (s, a)$
  $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \mathrm{Dom}(\pi))$
  if has-unsafe-loops$(\pi, a, Frontier)$ then return failure
return π

$Frontier \setminus S_g = \emptyset$

$\pi = \{$(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 solution π
    - Look at each leaf node of π
      - If the leaf node isn't a goal, find a 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 if $s = s_0$ then return failure *(not in book)*
    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 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'

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)

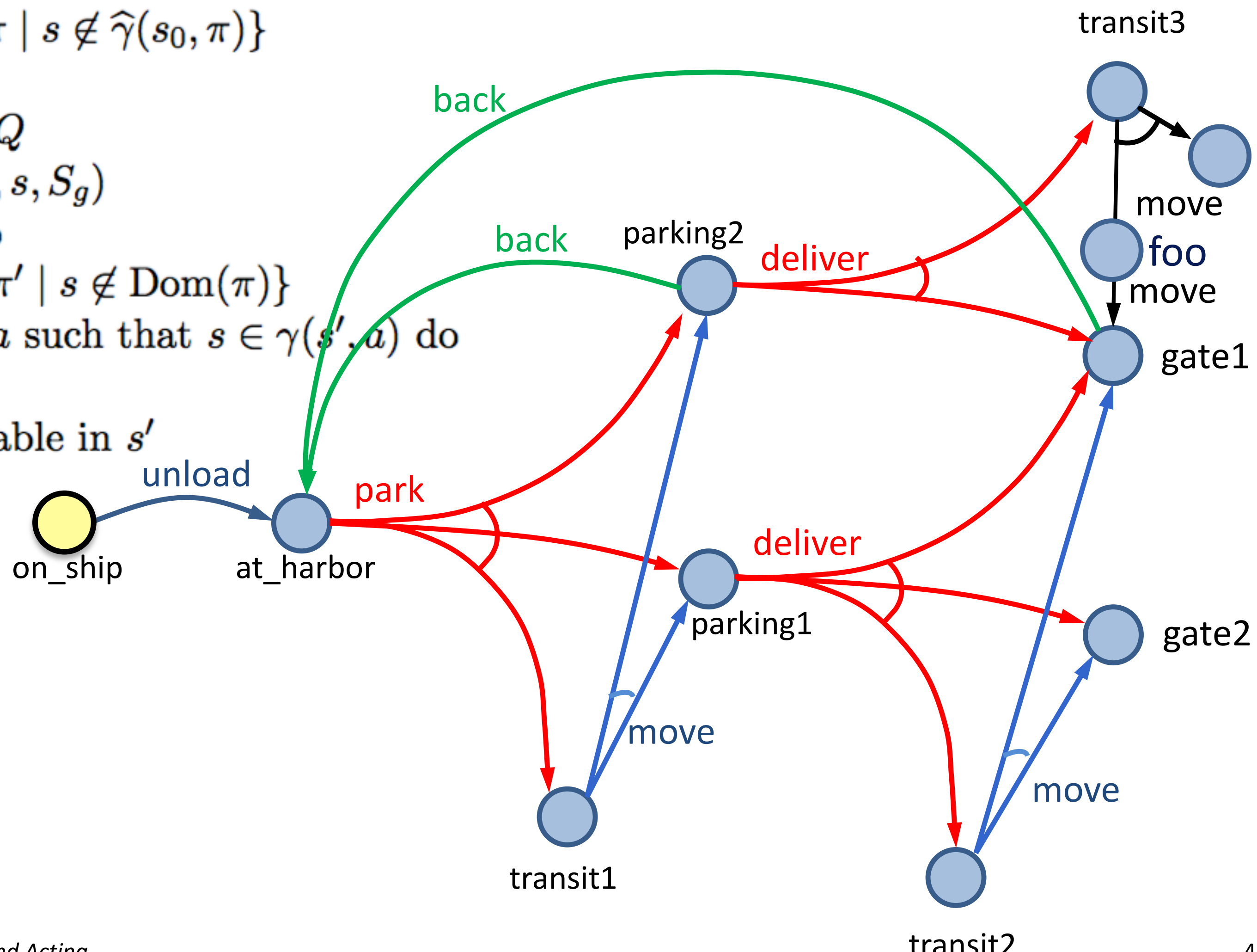
    if $\pi' \neq$ failure then do

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

    else for every s' and a such that $s \in \gamma(s', a)$ do

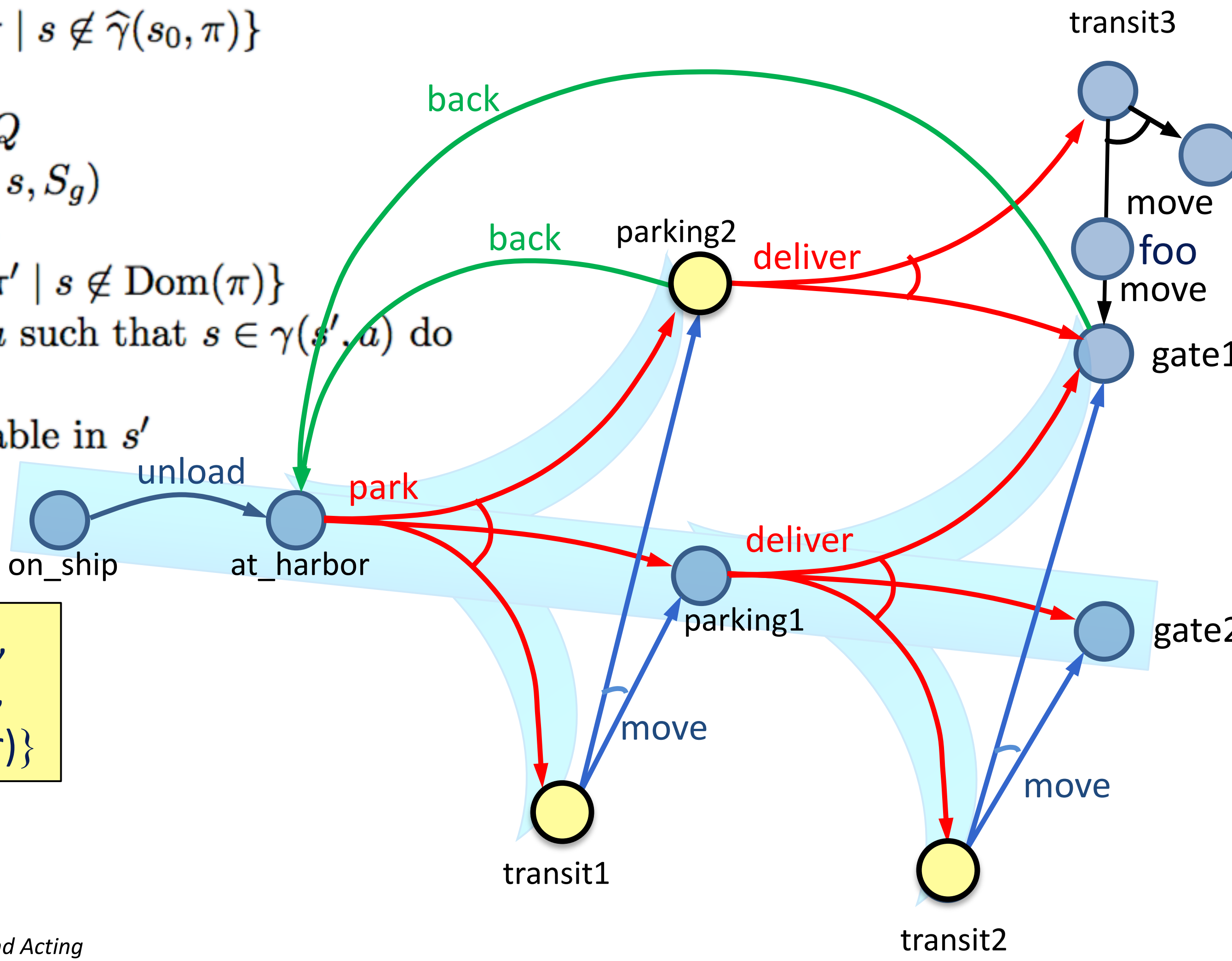
      $\pi \leftarrow \pi \setminus \{(s', a)\}$

      make a not applicable in s'



Example

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'



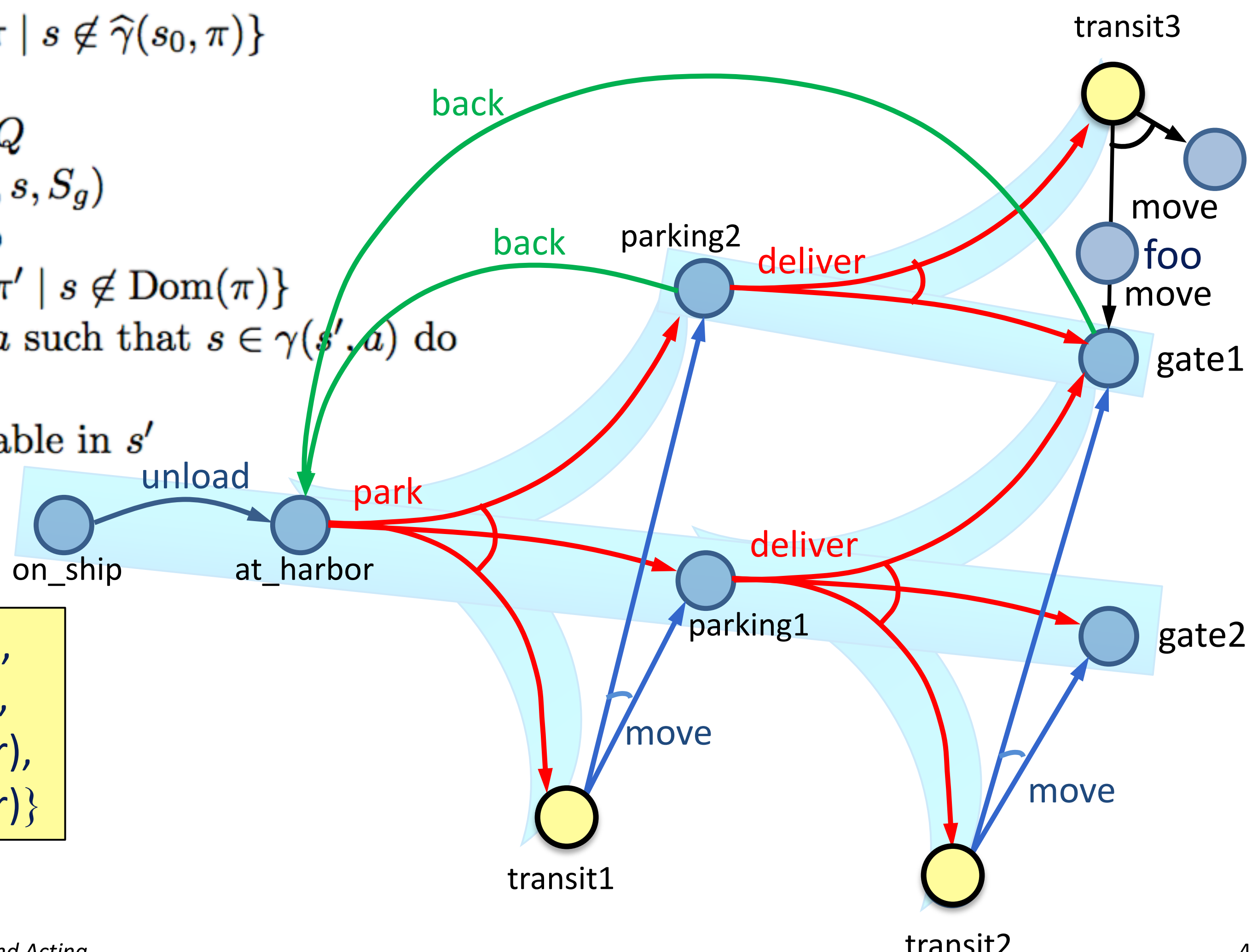
$\pi = \{$(on_ship, unload),
(at_harbor, park),
(parking1, deliver)$\}$

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

```
if s_0 ∈ S_g then return(∅)
if Applicable(s_0) = ∅ then return(failure)
π ← ∅
loop
    Q ← leaves(s_0, π) \ S_g
    if Q = ∅ then do
        π ← π \ {(s, a) ∈ π | s ∉ γ̂(s_0, π)}
        return(π)
    select arbitrarily s ∈ Q
    π′ ← Find-Solution(Σ, s, S_g)
    if π′ ≠ failure then do
        π ← π ∪ {(s, a) ∈ π′ | s ∉ Dom(π)}
    else for every s′ and a such that s ∈ γ(s′, a) do
        π ← π \ {(s′, a)}
        make a not applicable in s′
```

Example

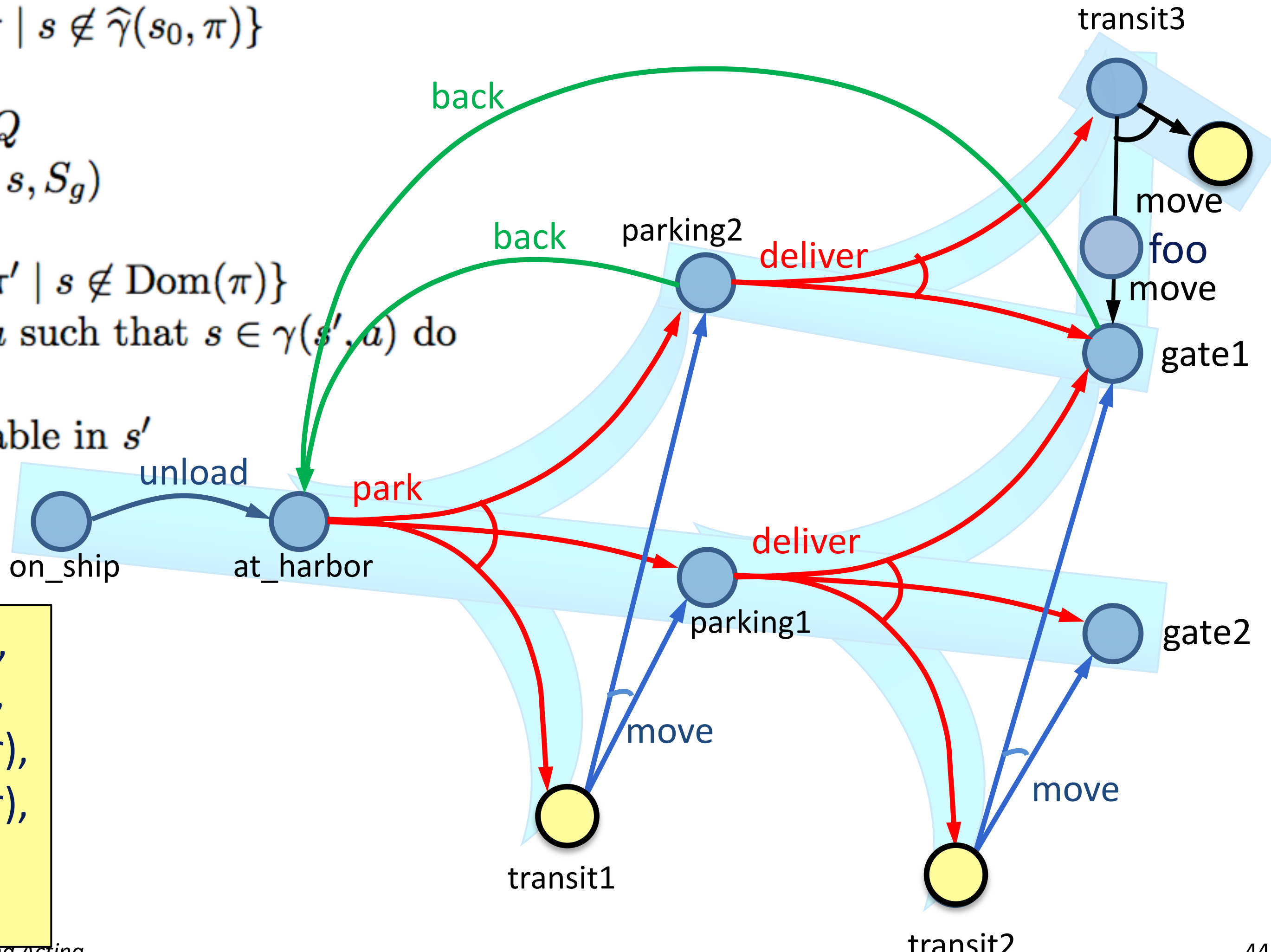
$\pi = \{$(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver)$\}$



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'

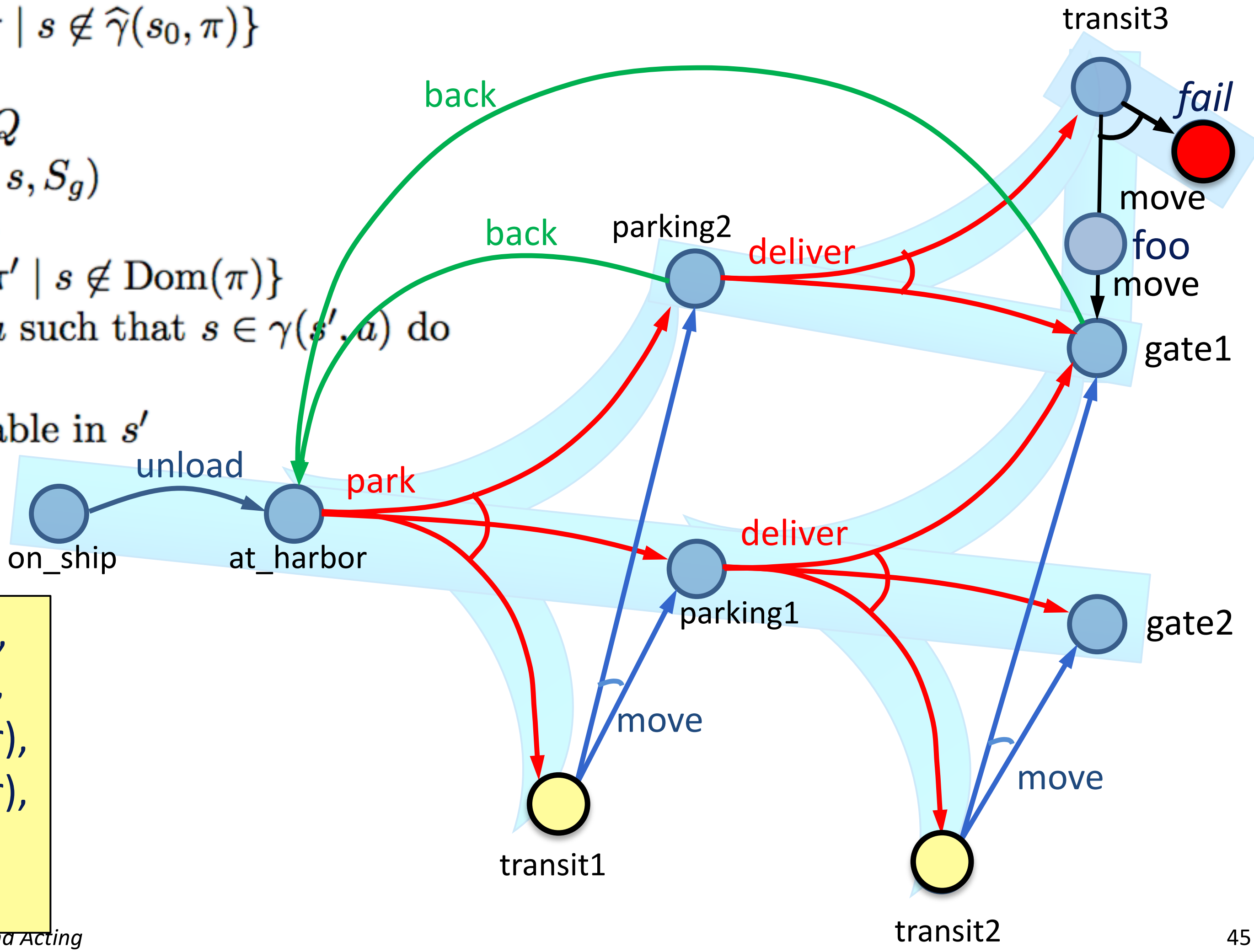
Example



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

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

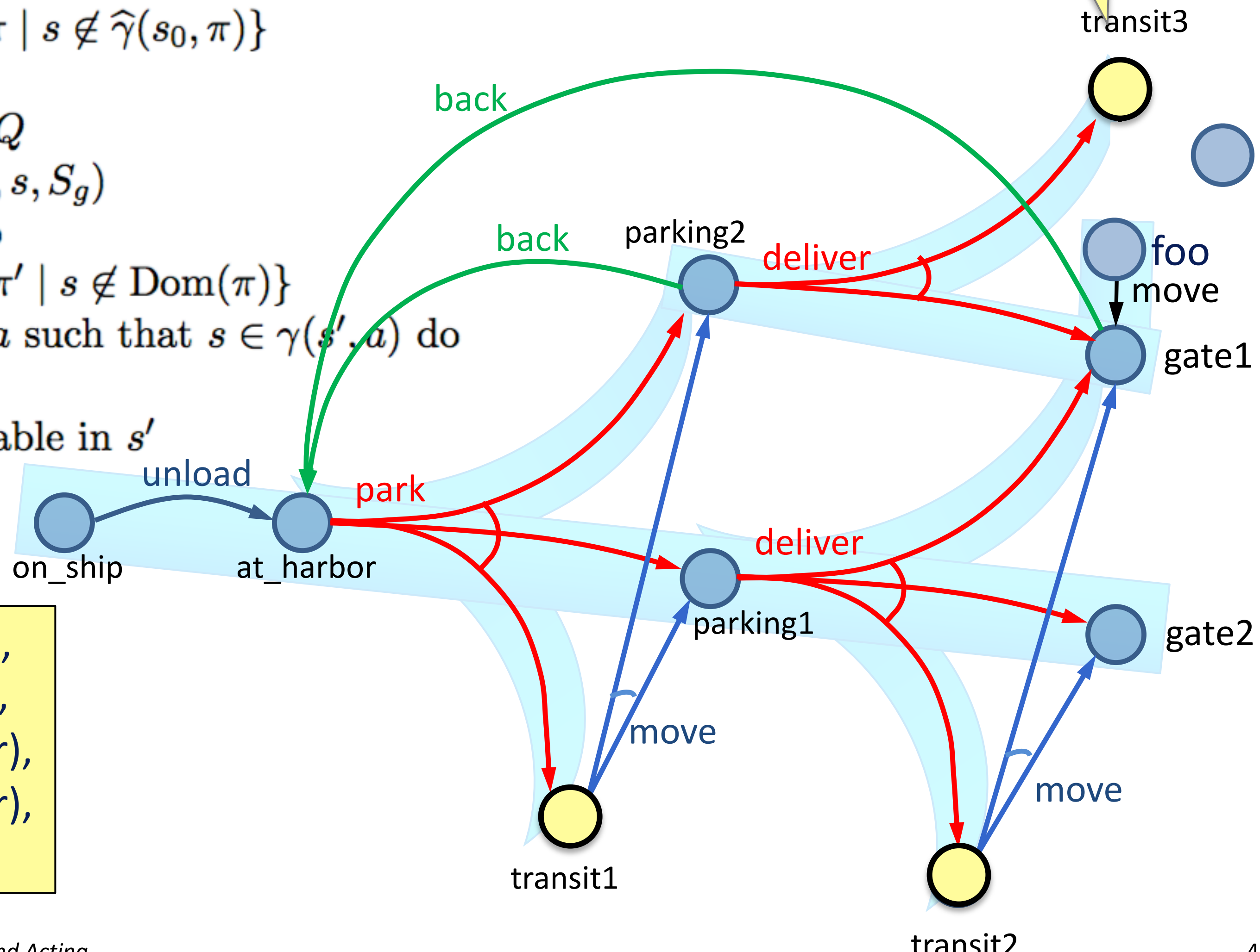


$\pi = \{$(on_ship, unload),
(at_harbor, park),
(parking1, deliver),
(parking2, deliver),
(transit3, move),
(foo, move)$\}$

Example

```
Guided-Find-Safe-Solution (Σ, s0, Sg)
  if s0 ∈ Sg then return(∅)
  if Applicable(s0) = ∅ then return(failure)
  π ← ∅
  loop
    Q ← leaves(s0, π) \ Sg
    if Q = ∅ then do
      π ← π \ {(s, a) ∈ π | s ∉ γ̂(s0, π)}
      return(π)
    select arbitrarily s ∈ Q
    π′ ← Find-Solution(Σ, s, Sg)
    if π′ ≠ failure then do
      π ← π ∪ {(s, a) ∈ π′ | s ∉ Dom(π)}
    else for every s′ and a such that s ∈ γ(s′, a) do
      π ← π \ {(s′, a)}
      make a not applicable in s′
```

Modify Σ_d to make move inapplicable



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

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

$\quad$ if $s_0 \in S_g$ then return($\varnothing$)

$\quad$ if $Applicable(s_0) = \varnothing$ then return(failure)

$\quad$ $\pi \leftarrow \varnothing$

$\quad$ loop

$\quad\quad$ $Q \leftarrow leaves(s_0, \pi) \setminus S_g$

$\quad\quad$ if $Q = \varnothing$ then do

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

$\quad\quad\quad$ return(π)

$\quad\quad$ select arbitrarily $s \in Q$

$\quad\quad$ $\pi' \leftarrow$ Find-Solution(Σ, s, S_g)

$\quad\quad$ if $\pi' \neq$ failure then do

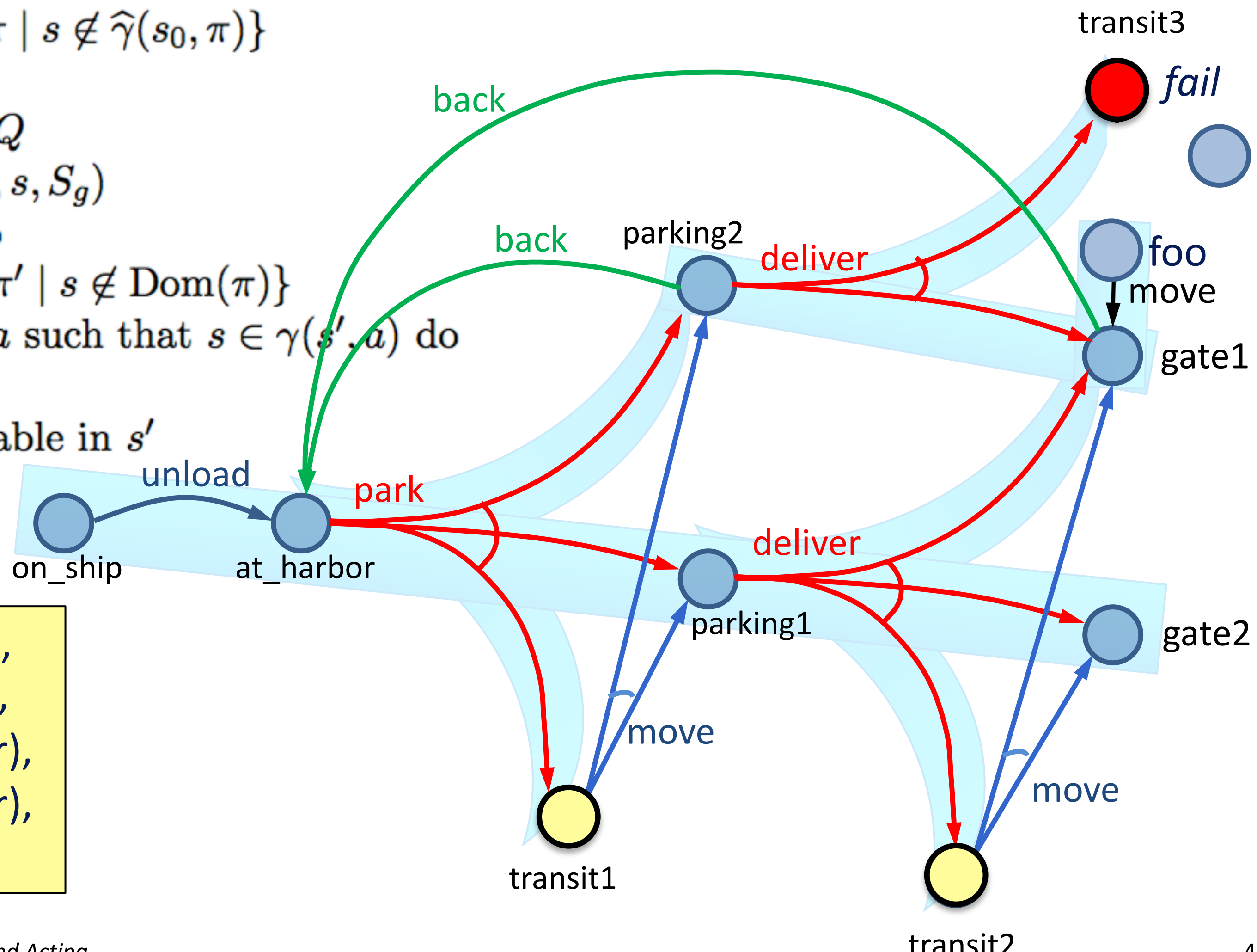
$\quad\quad\quad$ $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$

$\quad\quad$ else for every s' and a such that $s \in \gamma(s', a)$ do

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

$\quad\quad\quad$ make a not applicable in s'

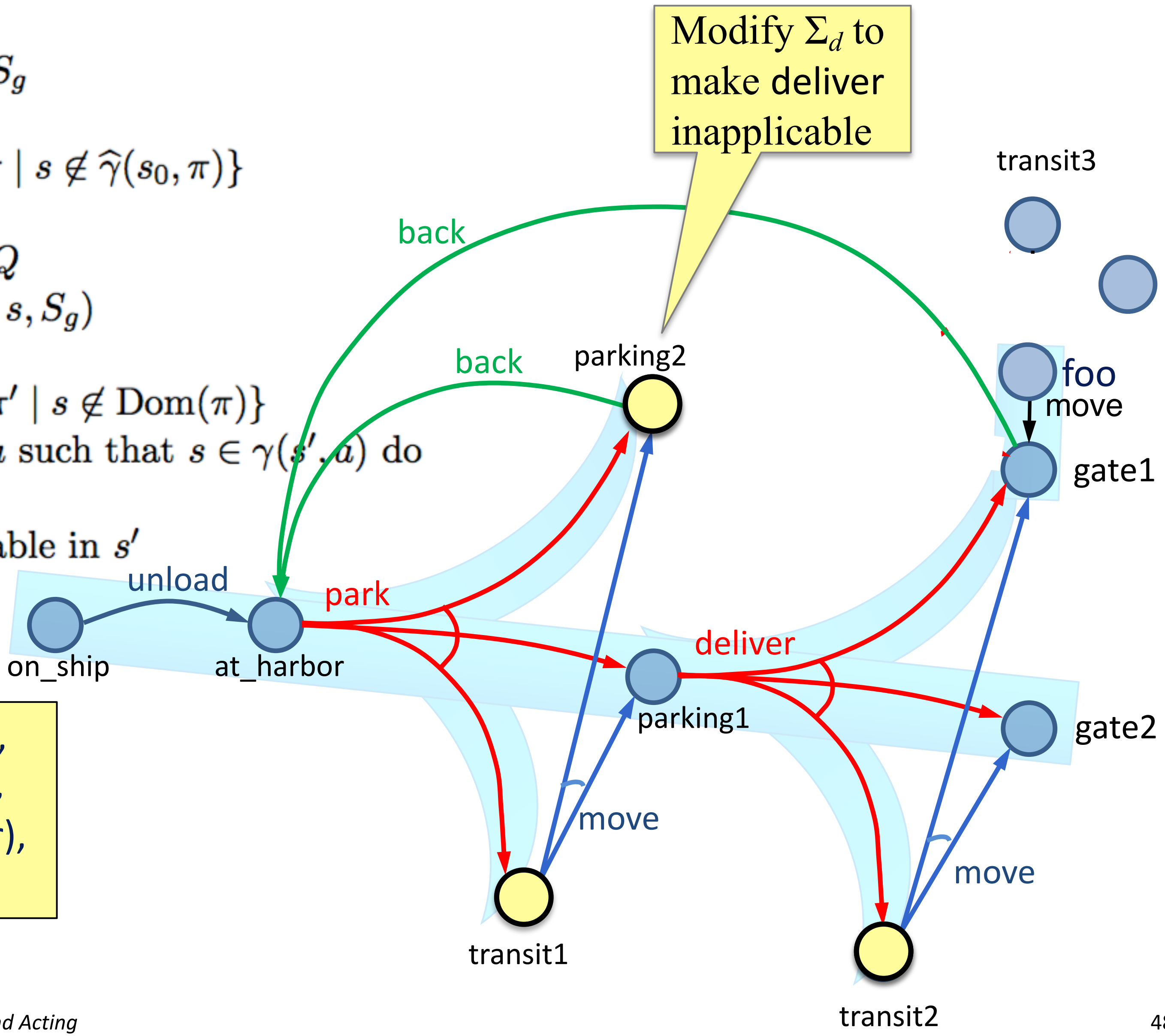
Example



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

Example

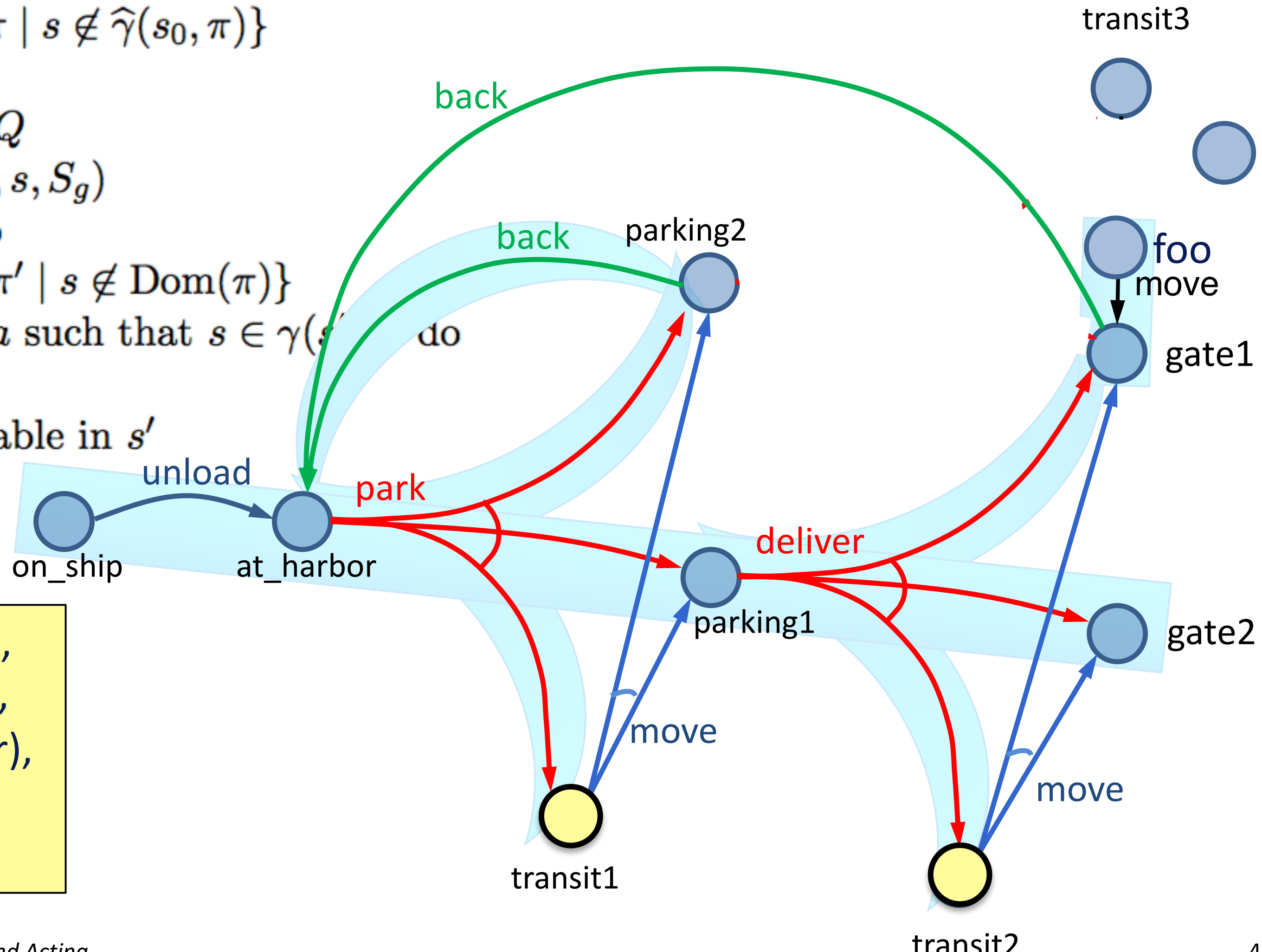
Guided-Find-Safe-Solution (Σ, s_0, S_g)
 if $s_0 \in S_g$ then return($\varnothing$)
 if $Applicable(s_0) = \varnothing$ then return(failure)
 $\pi \leftarrow \varnothing$
 loop
  $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  if $Q = \varnothing$ then do
   $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
   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'



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

Example

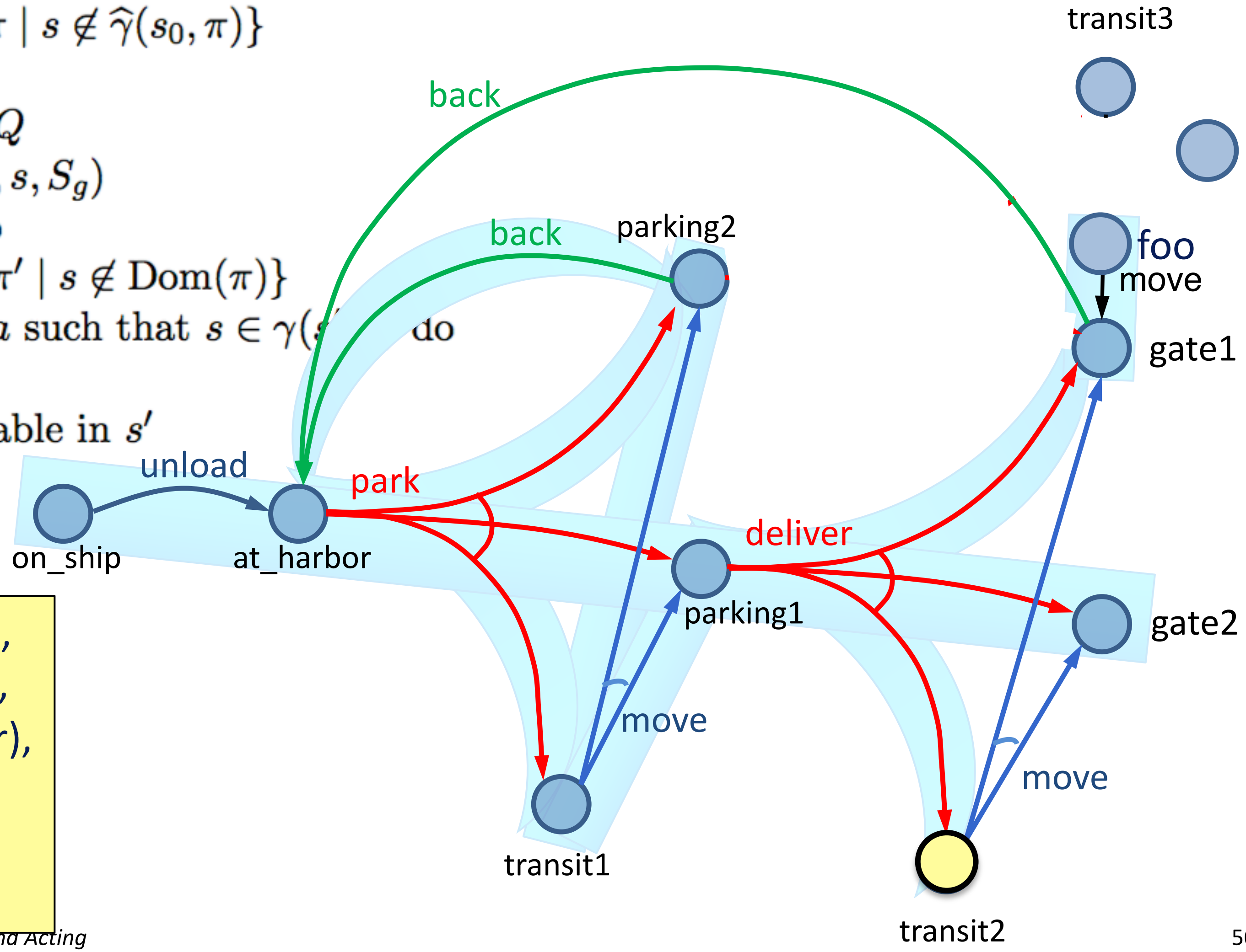
Guided-Find-Safe-Solution (Σ, s_0, S_g)
  if $s_0 \in S_g$ then return($\varnothing$)
  if $Applicable(s_0) = \varnothing$ then return(failure)
  $\pi \leftarrow \varnothing$
  loop
    $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
    if $Q = \varnothing$ then do
      $\pi \leftarrow \pi \setminus \{(s, a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
      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'



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

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'

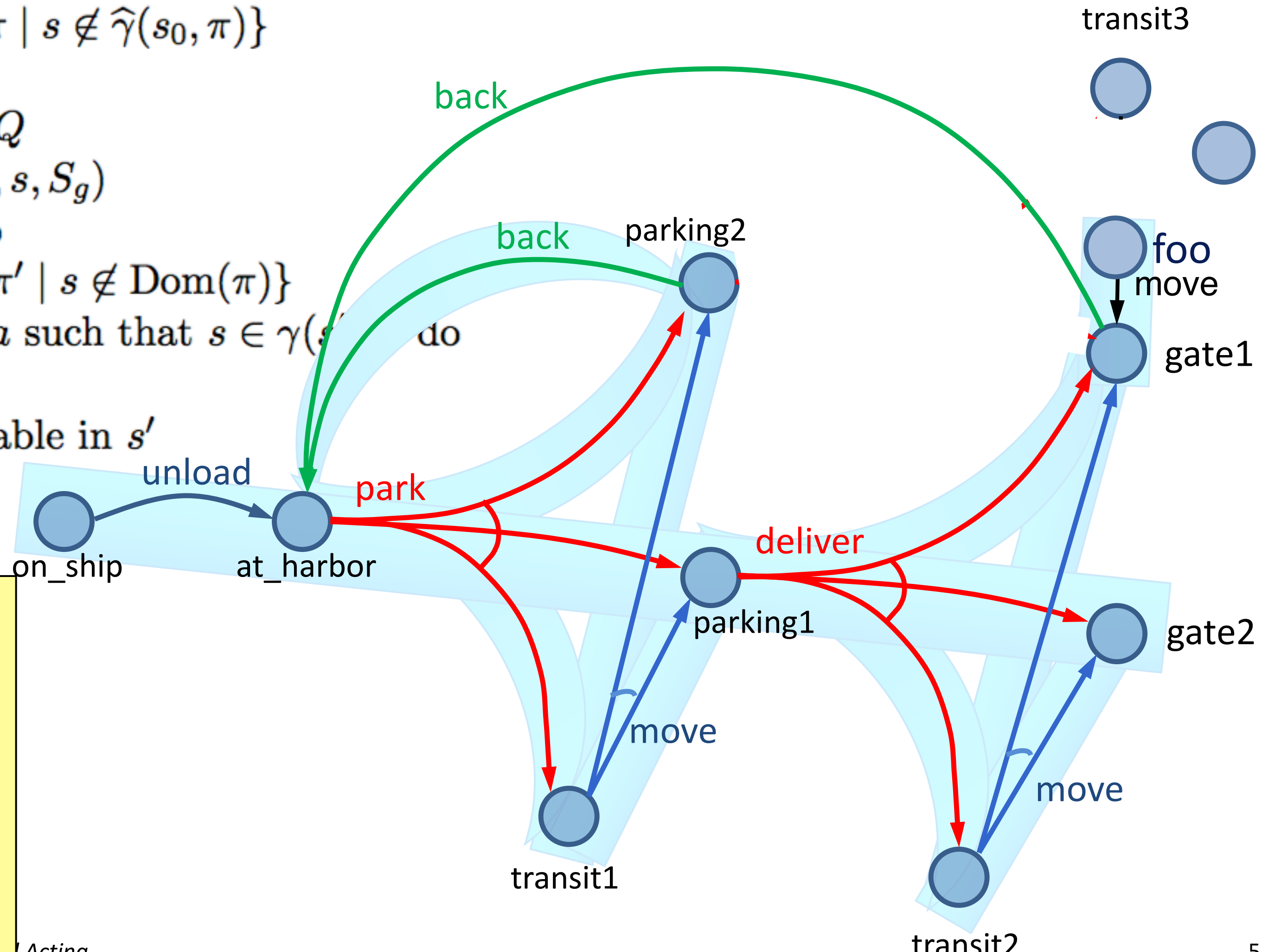
Example



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

Example

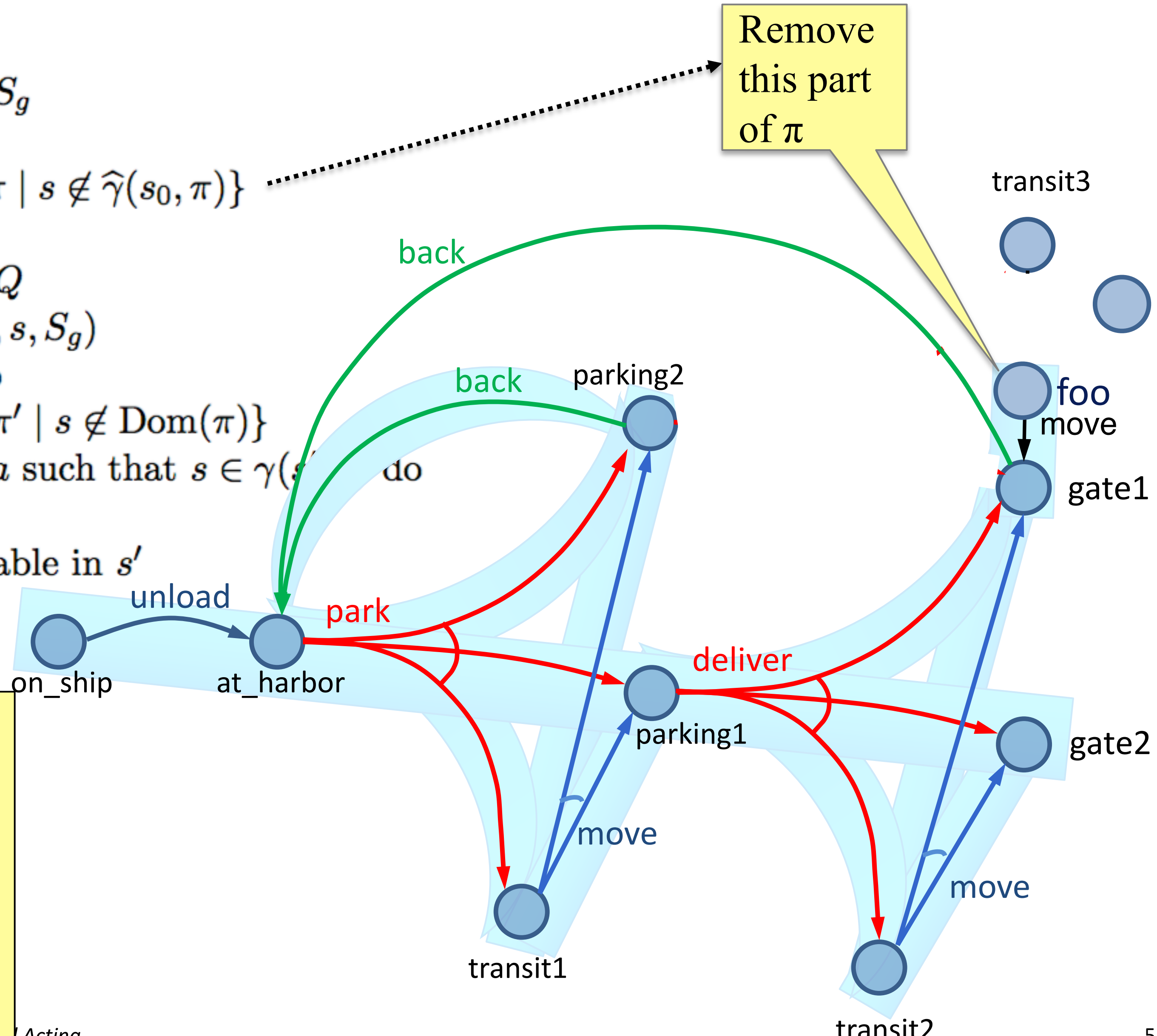
Guided-Find-Safe-Solution (Σ, s_0, S_g)
- if $s_0 \in S_g$ then return($\varnothing$)
- if $Applicable(s_0) = \varnothing$ then return(failure)
- $\pi \leftarrow \varnothing$
- loop
  - $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  - if $Q = \varnothing$ then do
    - $\pi \leftarrow \pi \setminus \{(s,a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    - 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'



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

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'



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

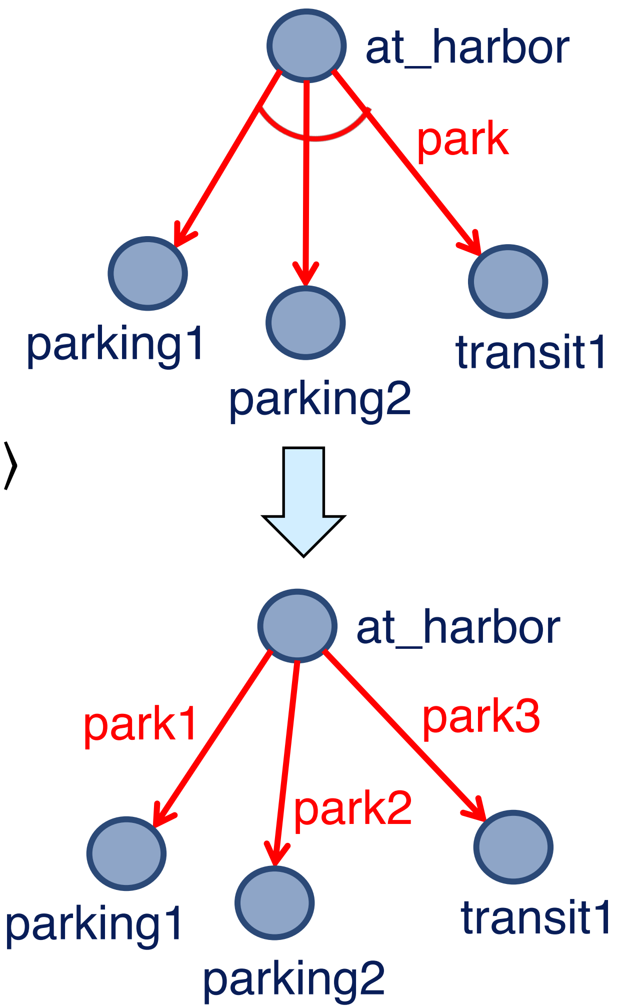
Determinization

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

- 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

Determinization

- Convert the nondeterministic actions into something the classical planner can use
- *Determinize*
  - Suppose a_i has n possible outcomes
  - n deterministic actions, one for each outcome



- Classical planner returns a plan $p = \langle a_1, a_2, \ldots, a_n \rangle$
- If p is acyclic, can convert it to a policy
  - (unsafe) solution for P
  - $\{(s_0,\boldsymbol{a}_1), (s_1,\boldsymbol{a}_2), \ldots, (s_{n-1},\boldsymbol{a}_n)\rangle$

    where

    - each $\boldsymbol{a}_i$ is the nondeterministic action whose determinization includes a_i
    - $s_i \in \gamma(s_{i-1},\boldsymbol{a}_i)$

Determinization

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

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

Determinization

```
Find-Safe-Solution-by-Determinization (Σ, s0, Sg)
    if s0 ∈ Sg then return(∅)
    if Applicable(s0) = ∅ then return(failure)
    π ← ∅
    Σd ← mk-deterministic(Σ)
    loop
        Q ← leaves(s0, π) \ Sg
        if Q = ∅ then do
            π ← π \ {(s, a) ∈ π | s ∉ γ̂(s0, π)}
            return(π)
        select s ∈ Q
        p′ ← Forward-search(Σd, s, Sg)
        if p′ ≠ fail then do
            π′ ← Plan2policy(p′, s)
            π ← π ∪ {(s, a) ∈ π′ | s ∉ Dom(π)}
        else for every s′ and a such that s ∈ γ(s′, a) do
            π ← π \ {(s′, a)}
            make the actions in the determinization of a
            not applicable in s′
```

Same as Guided-Find-Safe-Solution

Any classical planner that doesn't return cyclic plans

Convert p' to a policy. Add each (s,a) 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_d so we'll never use a at 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 $(\Sigma,$

    if $p' \neq fail$ then do

      $\pi' \leftarrow$ Plan2policy(p', s)

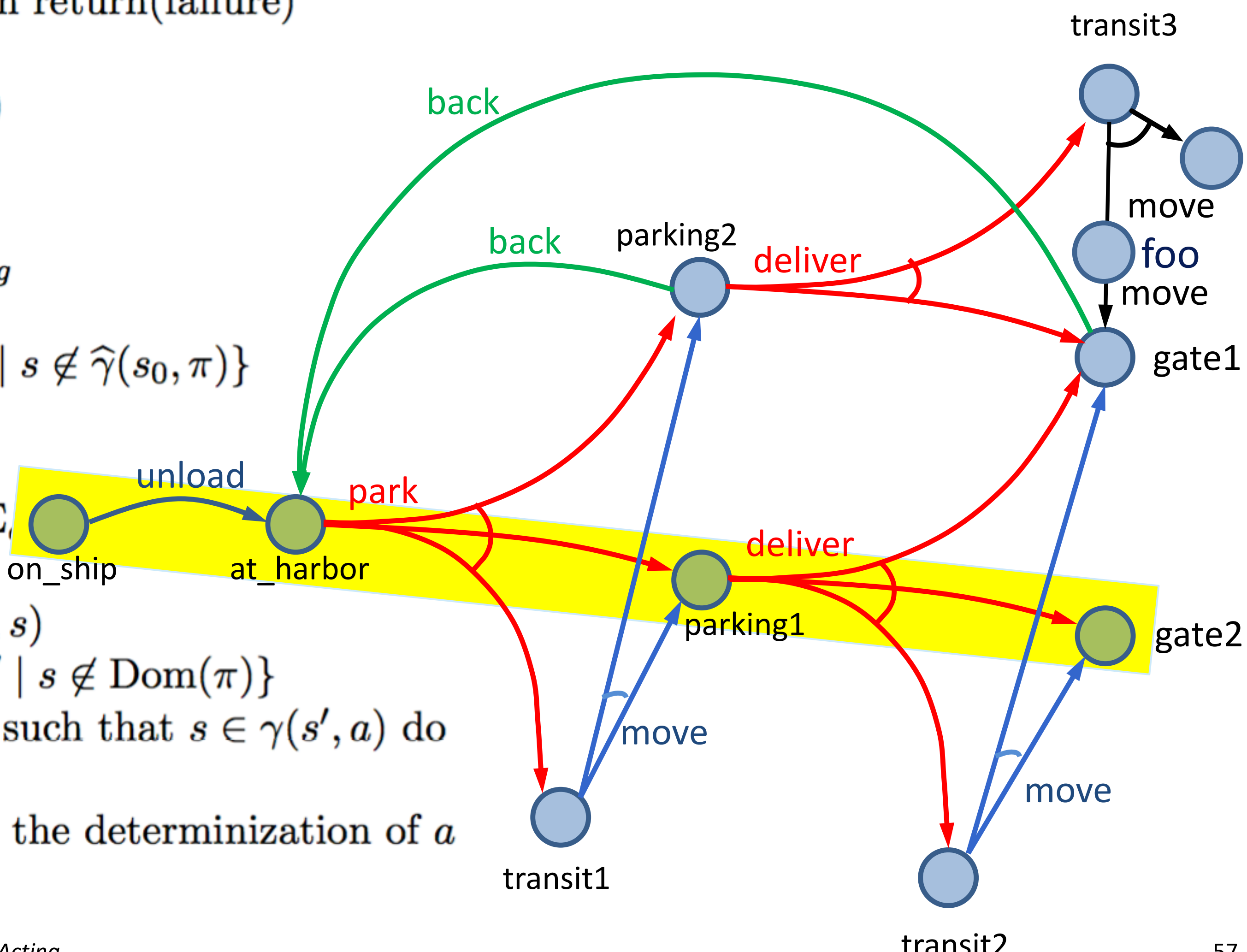
      $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$

    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'



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 $(\Sigma,$

    if $p' \neq fail$ then do

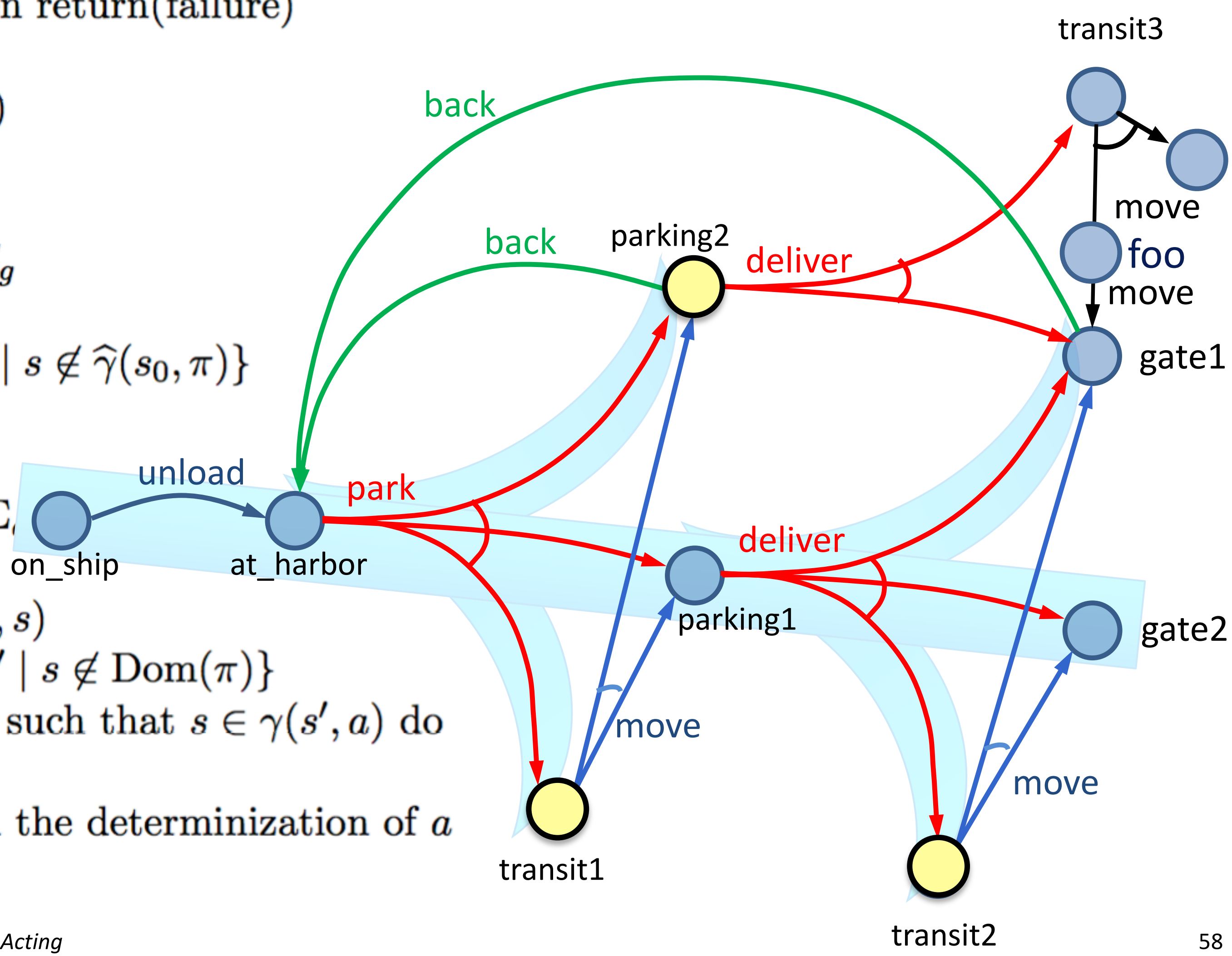
      $\pi' \leftarrow$ Plan2policy(p', s)

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

    else for every s' and a such that $s \in \gamma(s', a)$ do

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

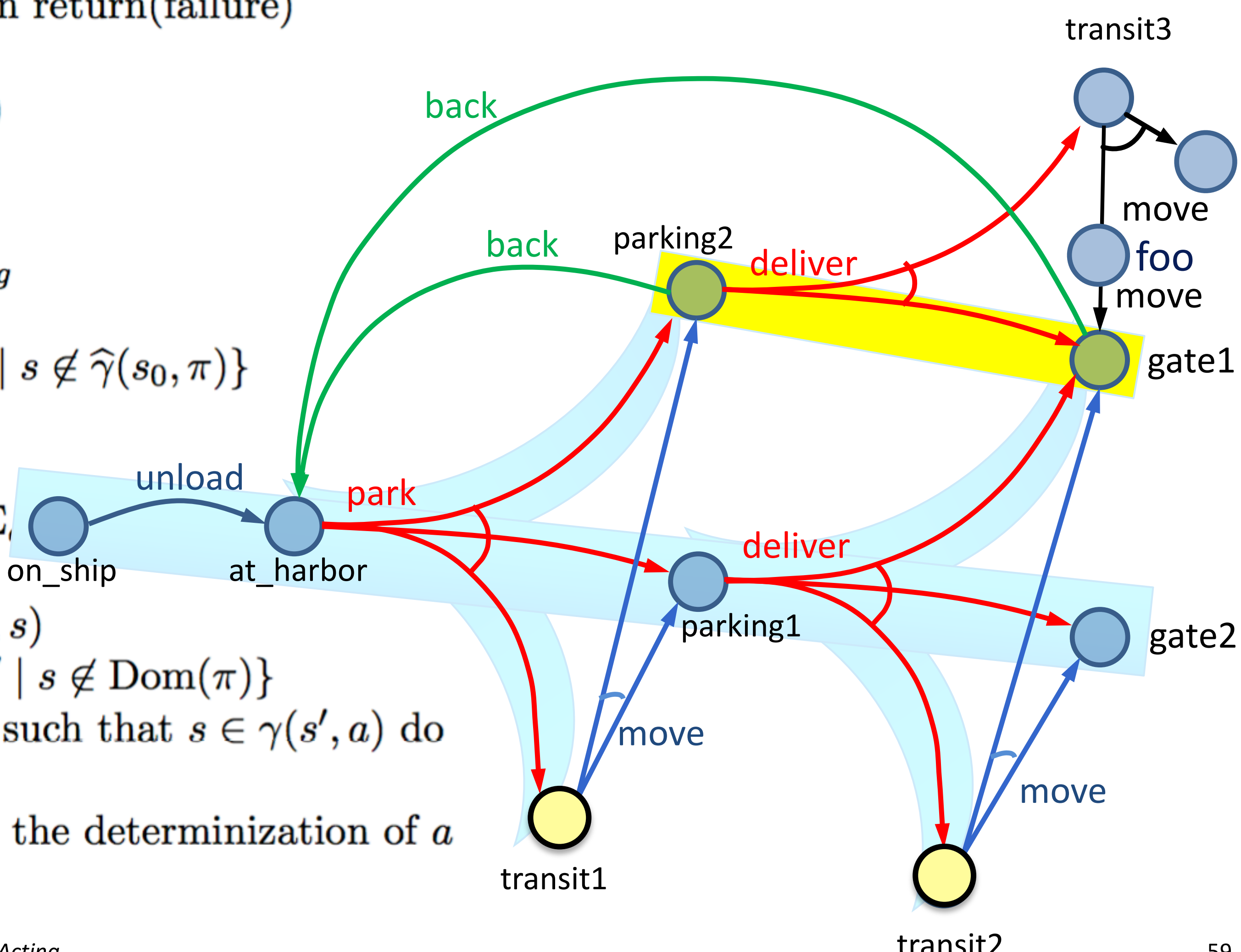
      make the actions in the determinization of a not applicable in s'



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 $(\Sigma,$
  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'



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 $(\Sigma,$

    if $p' \neq fail$ then do

      $\pi' \leftarrow$ Plan2policy(p', s)

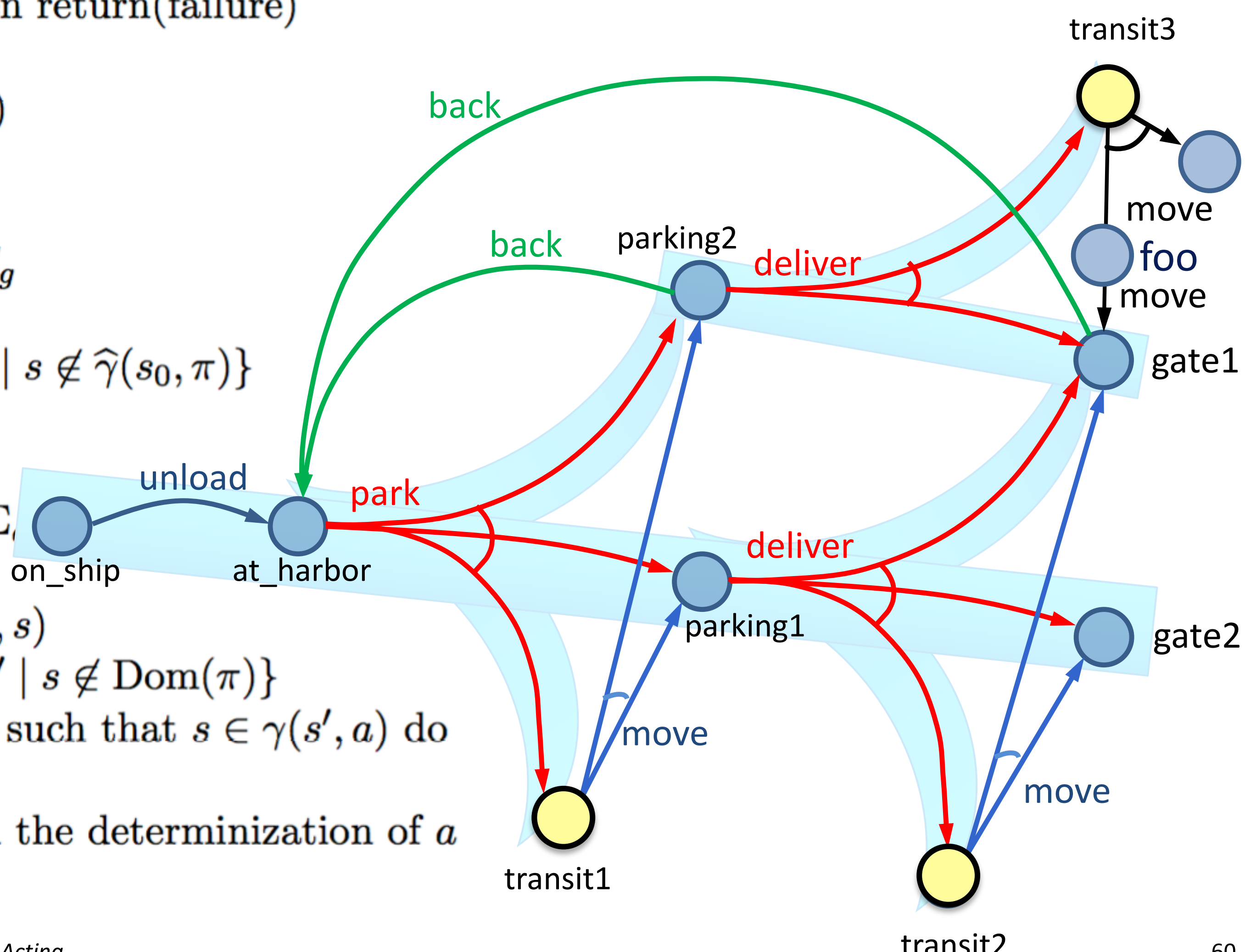
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$

    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'



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 $(\Sigma,$

if $p' \neq fail$ then do

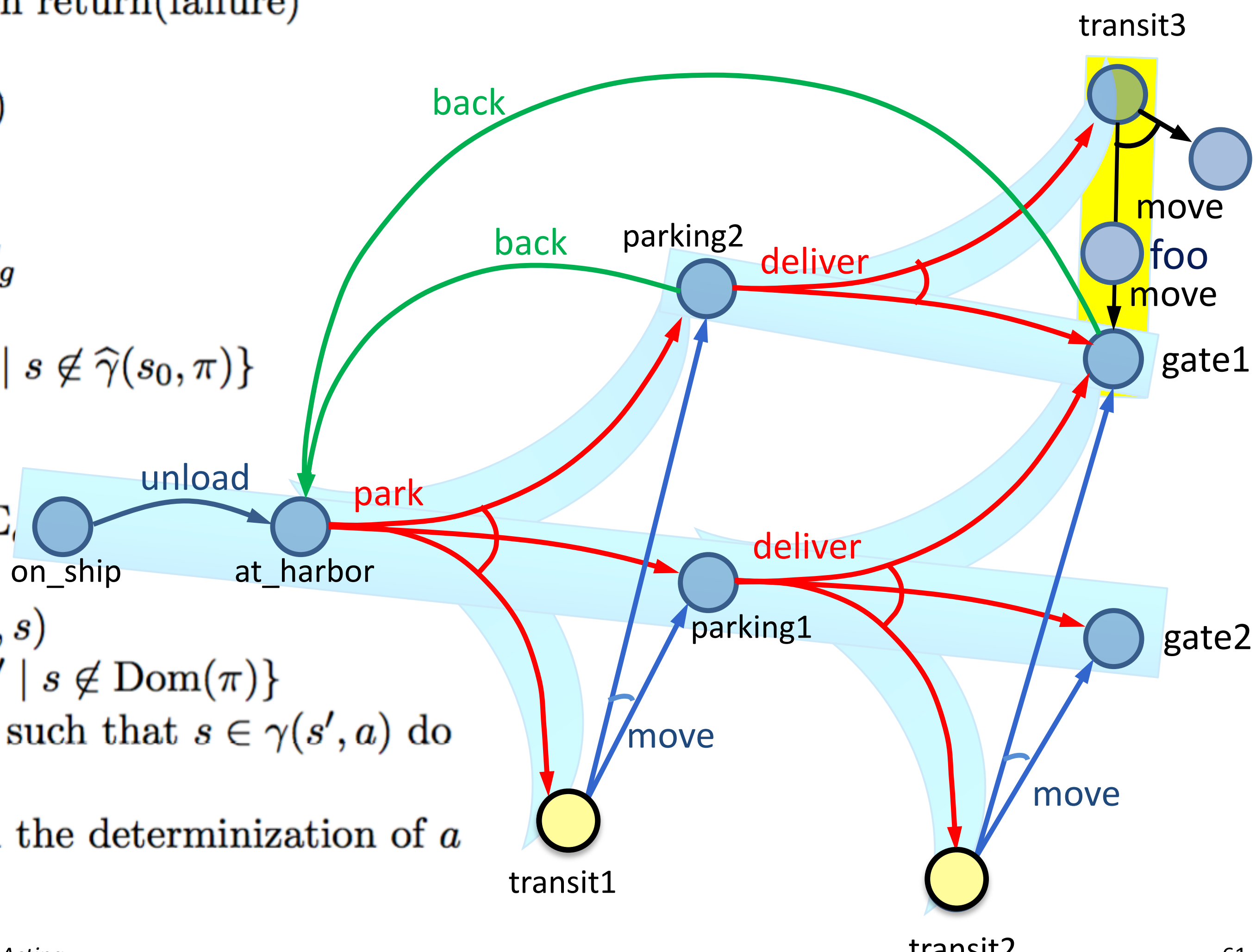
$\pi' \leftarrow$ Plan2policy(p', s)

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

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 $(\Sigma,$

      if $p' \neq fail$ then do

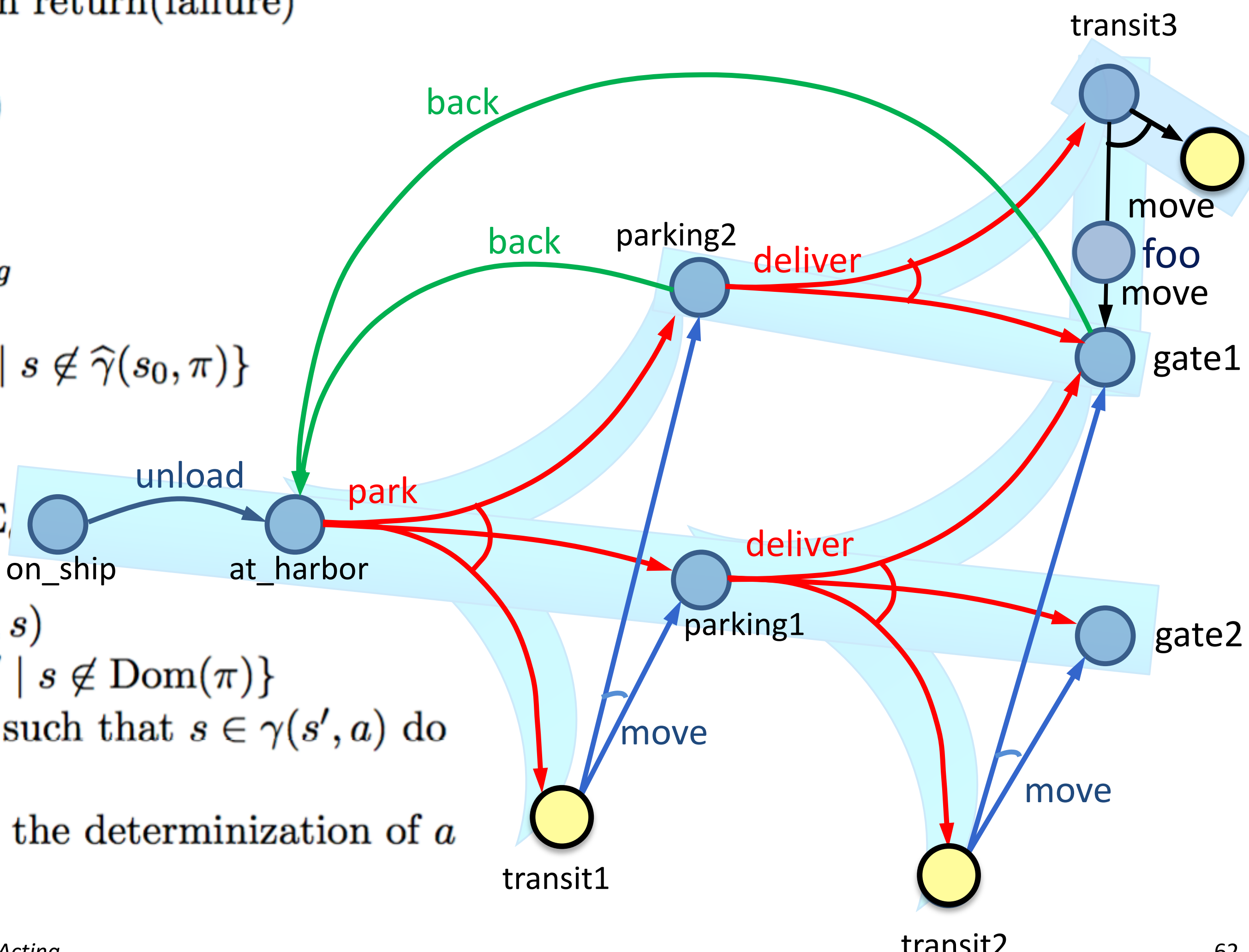
         $\pi' \leftarrow$ Plan2policy(p', s)

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

      else for every s' and a such that $s \in \gamma(s', a)$ do

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

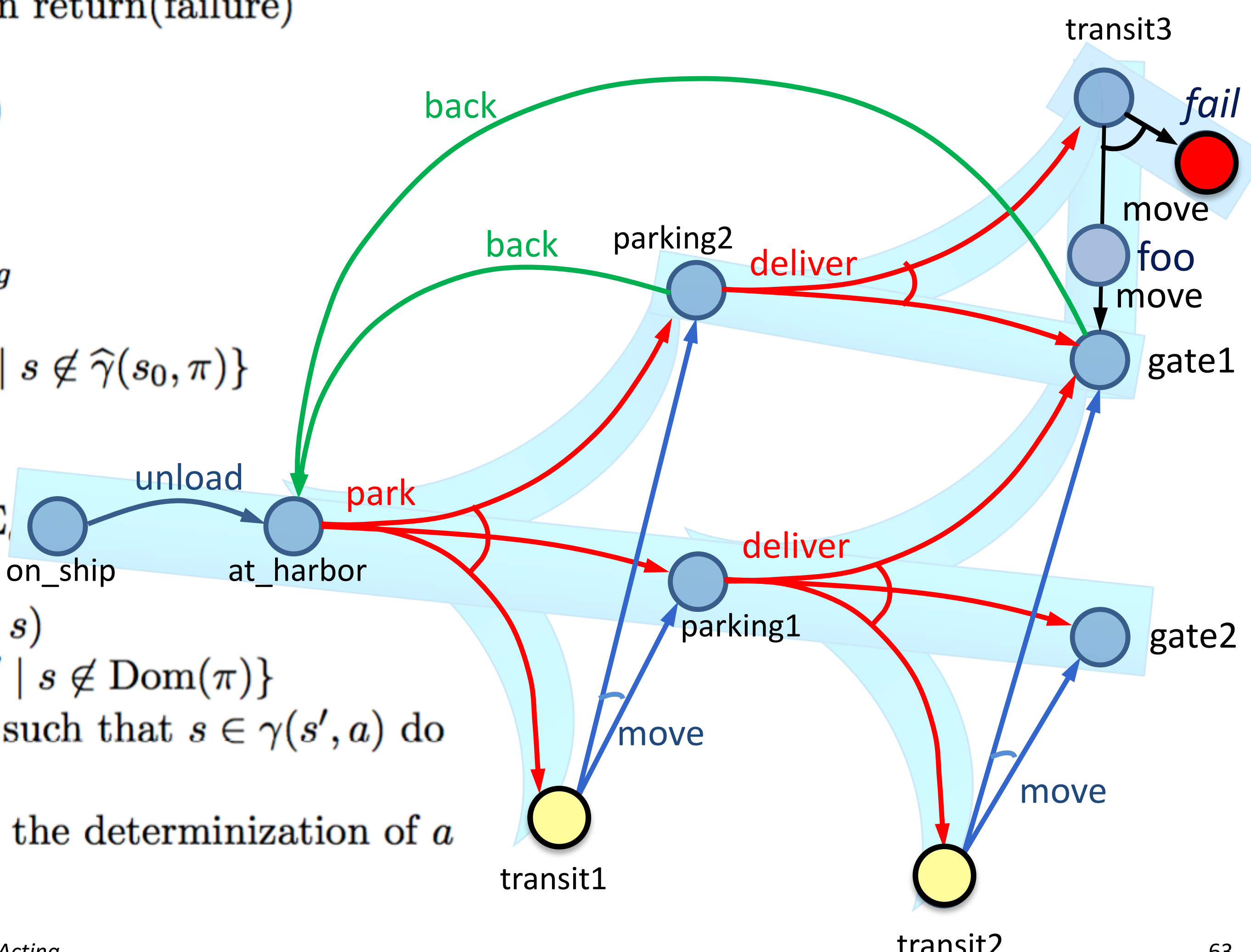
         make the actions in the determinization of a not applicable in s'



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 $(\Sigma$
    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'



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 $(\Sigma,$

    if $p' \neq fail$ then do

      $\pi' \leftarrow$ Plan2policy(p', s)

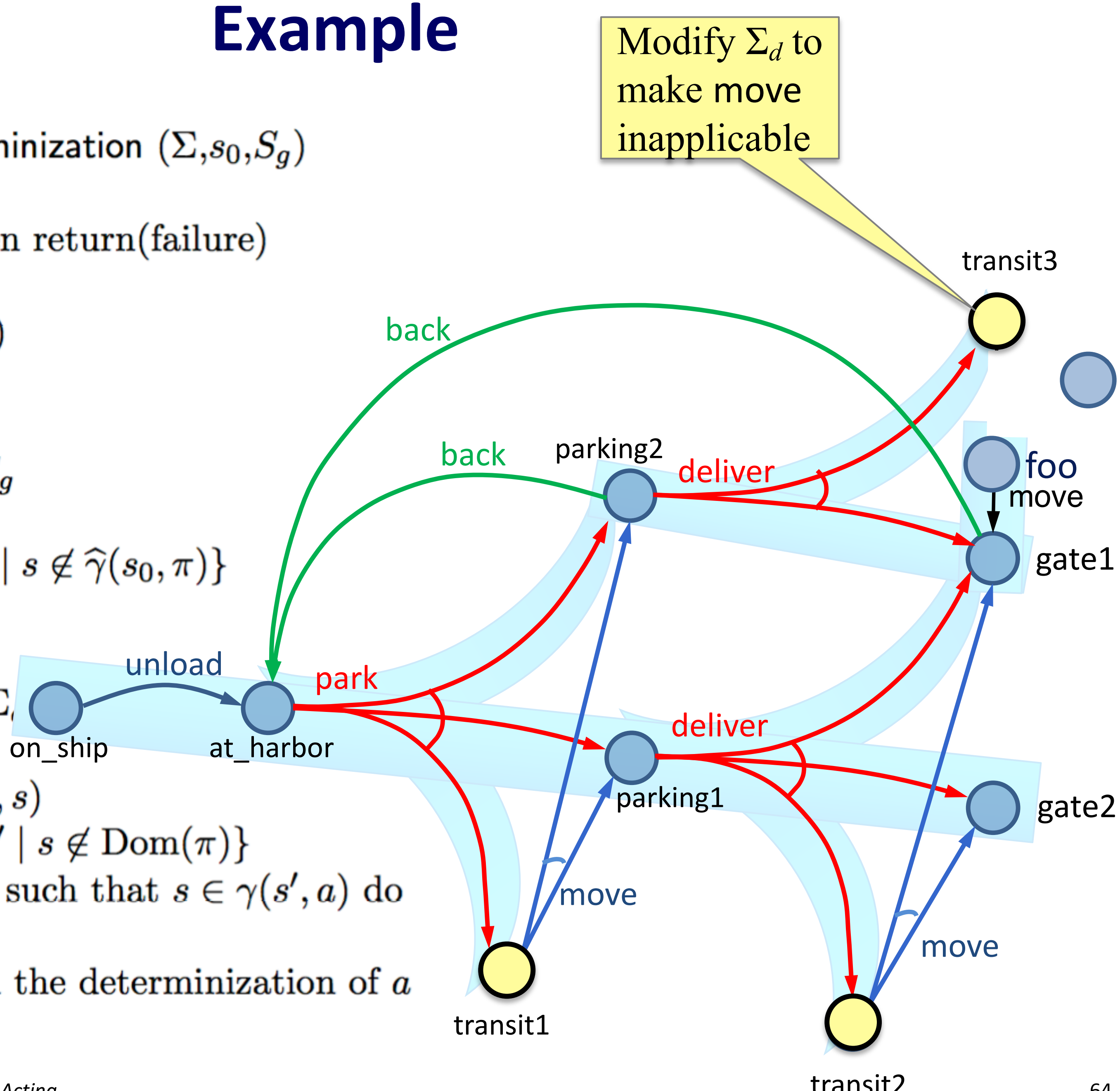
      $\pi \leftarrow \pi \cup \{(s, a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$

    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'



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 $(\Sigma,$

  if $p' \neq fail$ then do

   $\pi' \leftarrow$ Plan2policy(p', s)

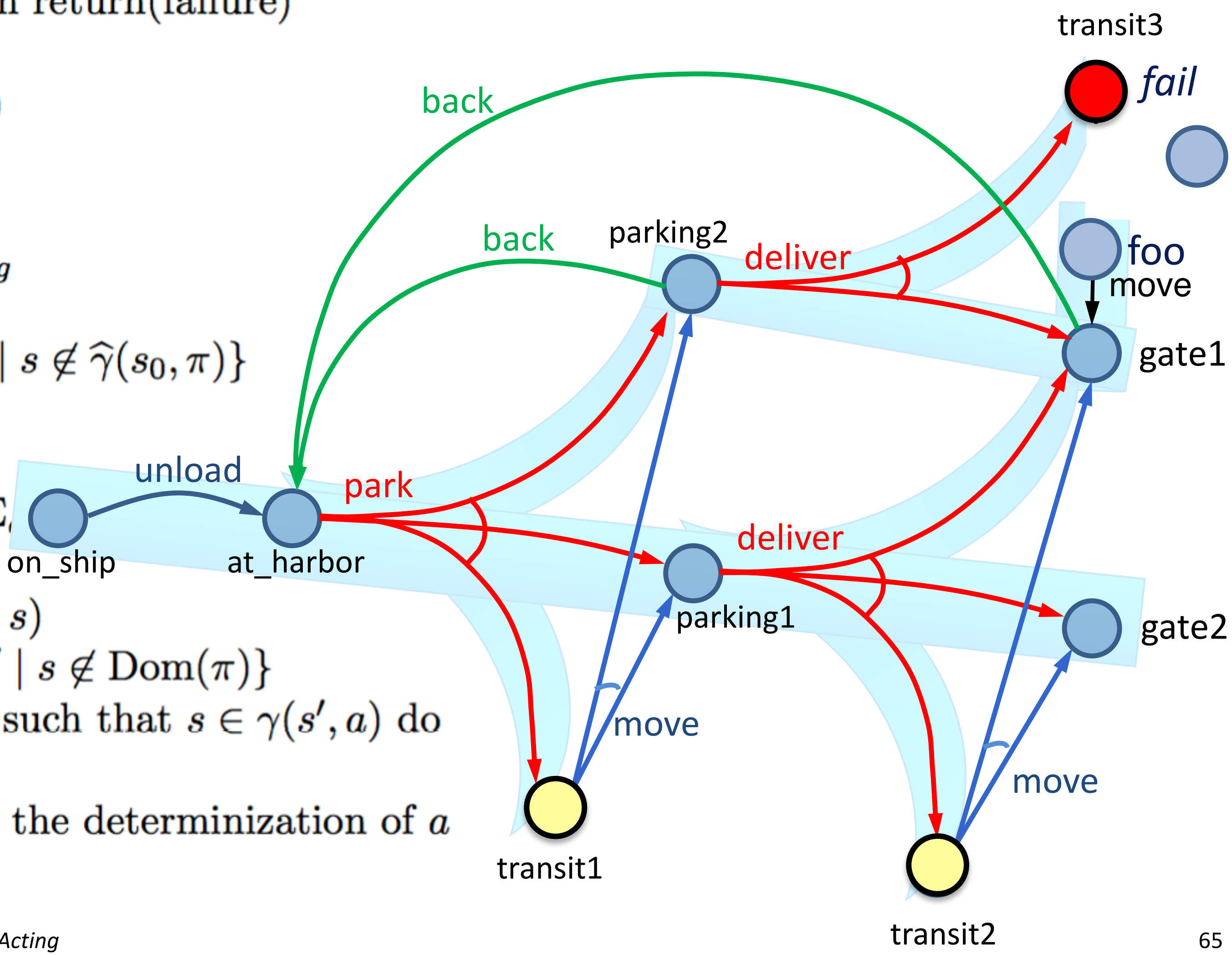
   $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$

  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

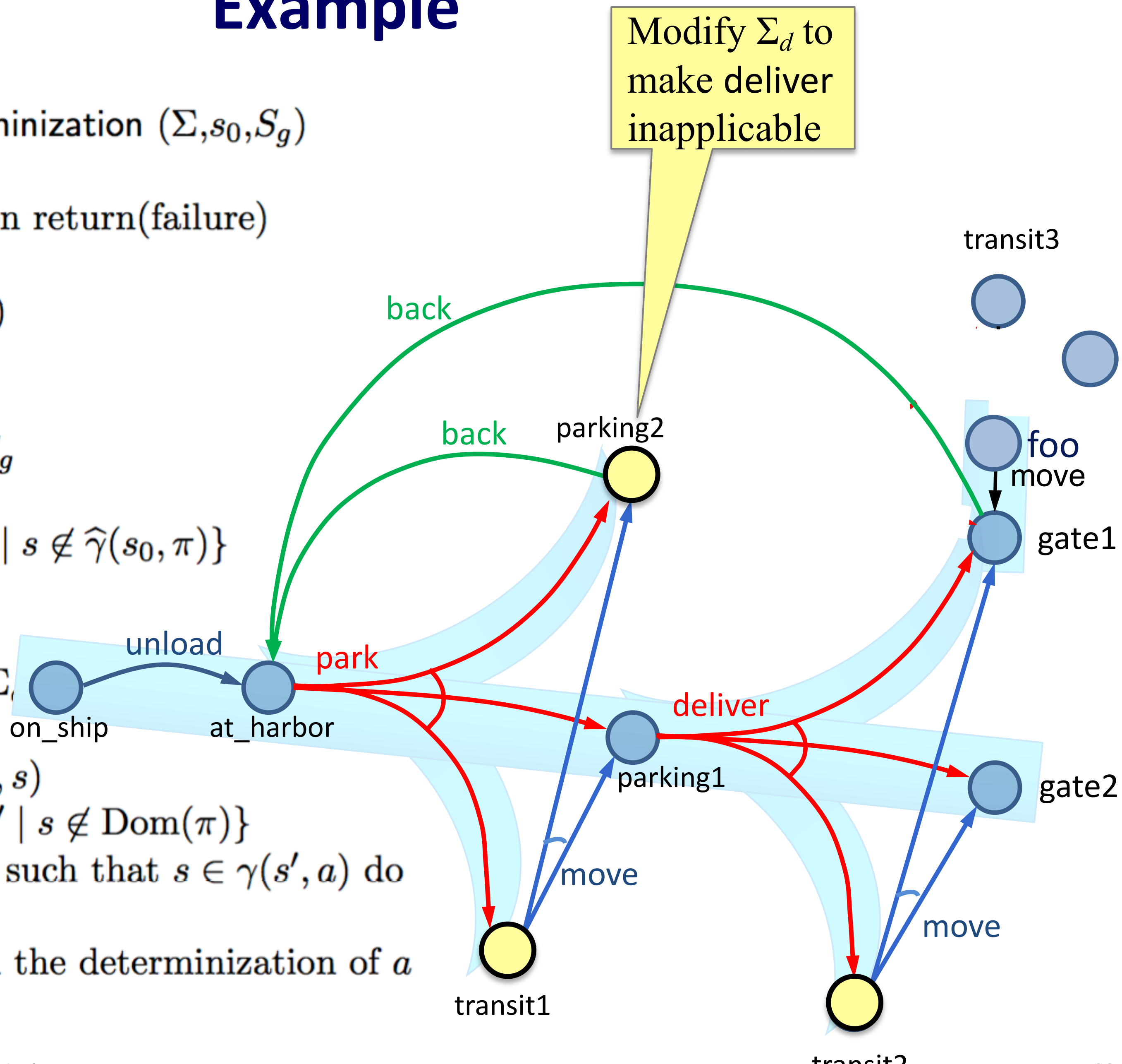
   not applicable in s'



Example

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

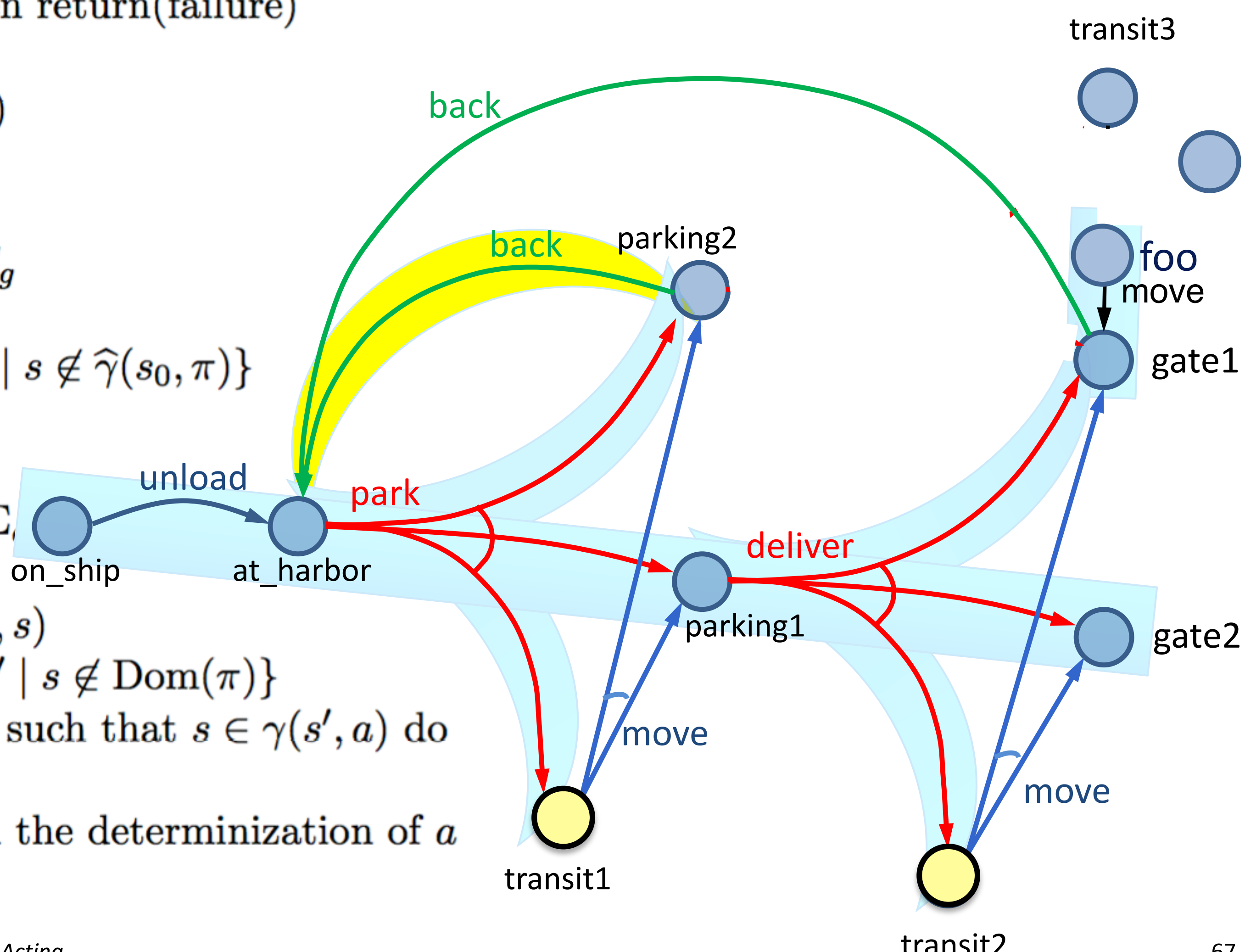
- if $s_0 \in S_g$ then return($\varnothing$)
- if $Applicable(s_0) = \varnothing$ then return(failure)
- $\pi \leftarrow \varnothing$
- $\Sigma_d \leftarrow$ mk-deterministic(Σ)
- loop
  - $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  - if $Q = \varnothing$ then do
    - $\pi \leftarrow \pi \setminus \{(s,a) \in \pi \mid s \notin \widehat{\gamma}(s_0, \pi)\}$
    - return(π)
  - select $s \in Q$
  - $p' \leftarrow$ Forward-search $(\Sigma,$
  - 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'



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 $(\Sigma,$
  - 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'



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 $(\Sigma,$

        if $p' \neq fail$ then do

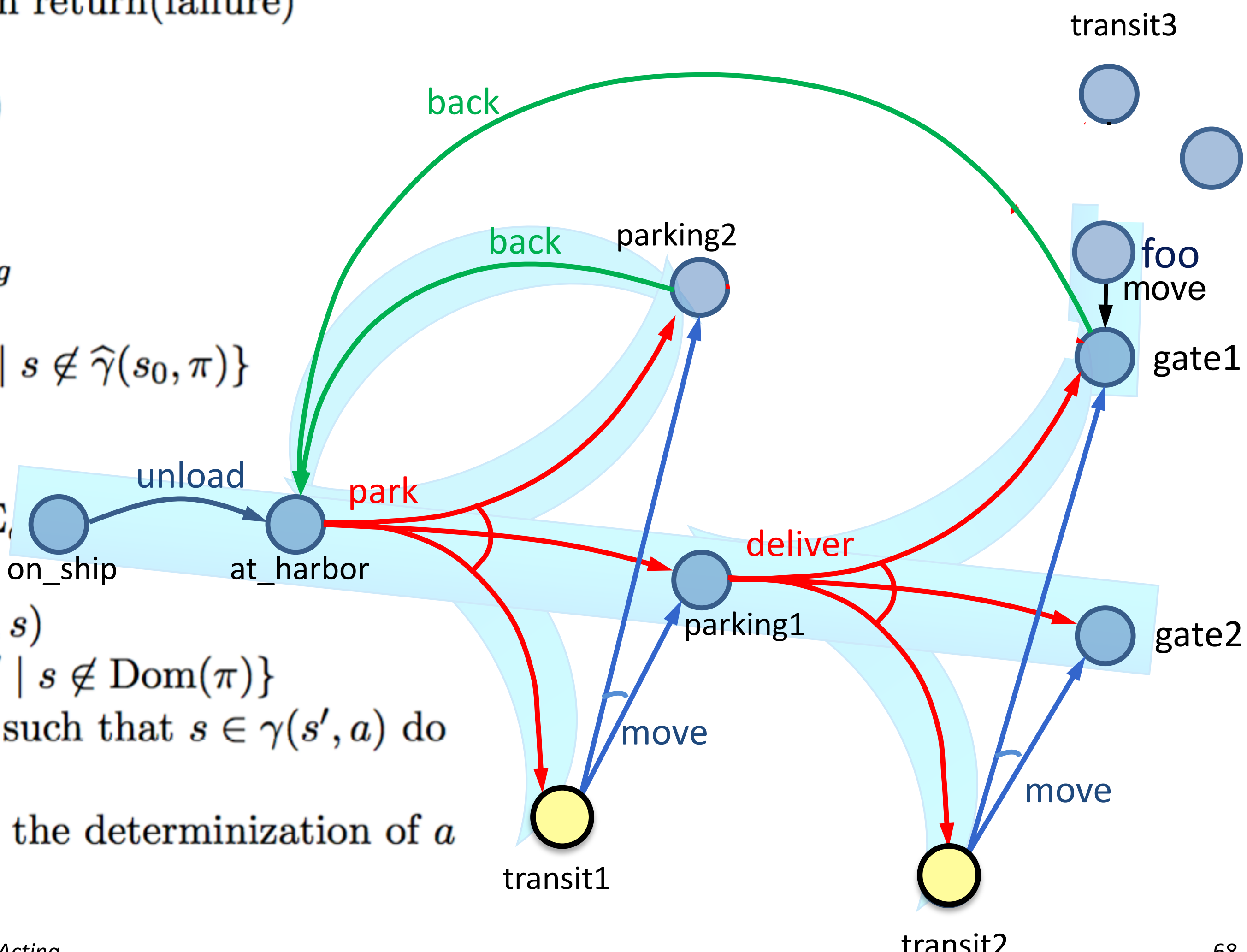
            $\pi' \leftarrow$ Plan2policy(p', s)

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

        else for every s' and a such that $s \in \gamma(s', a)$ do

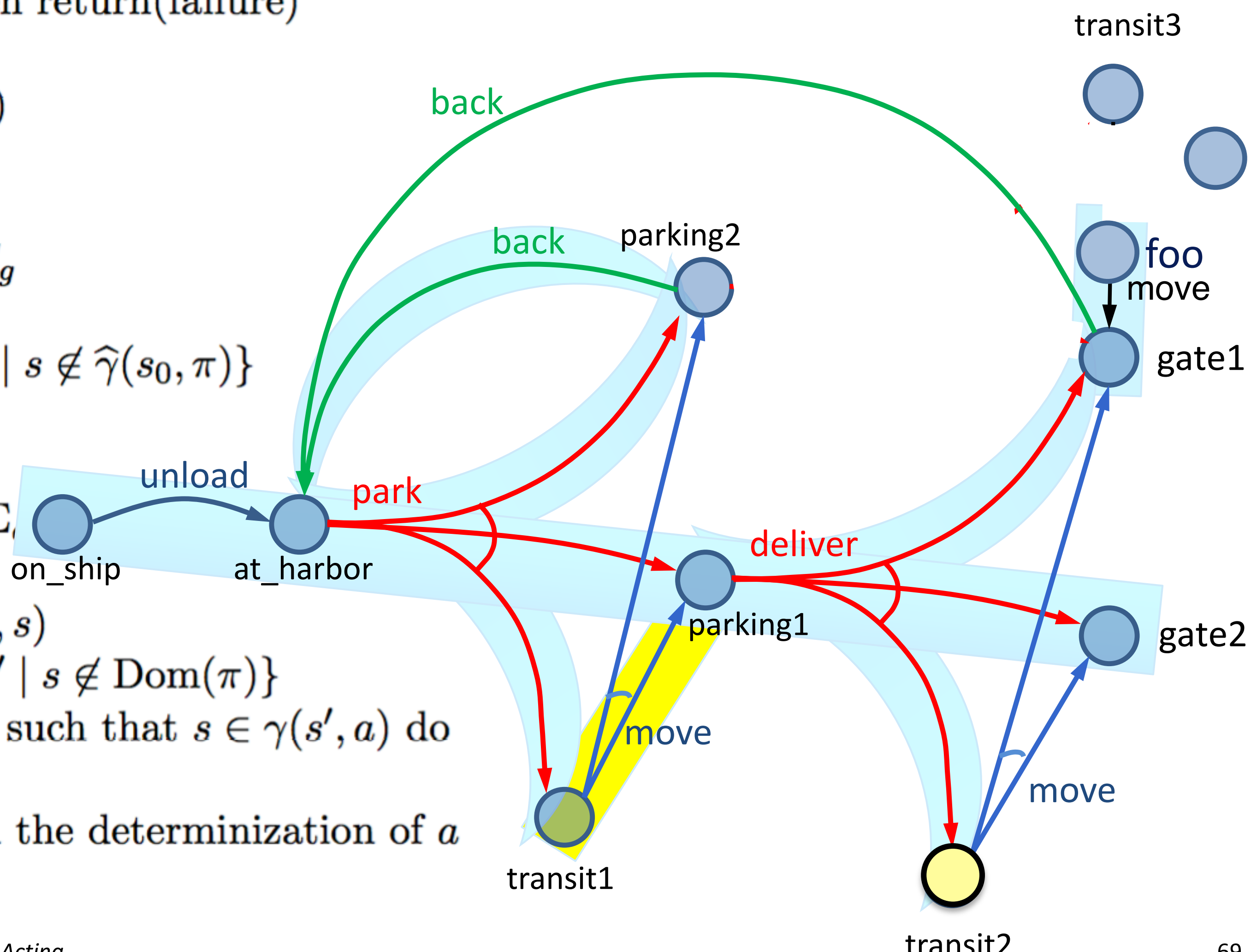
            $\pi \leftarrow \pi \setminus \{(s', a)\}$

            make the actions in the determinization of a not applicable in s'



Example

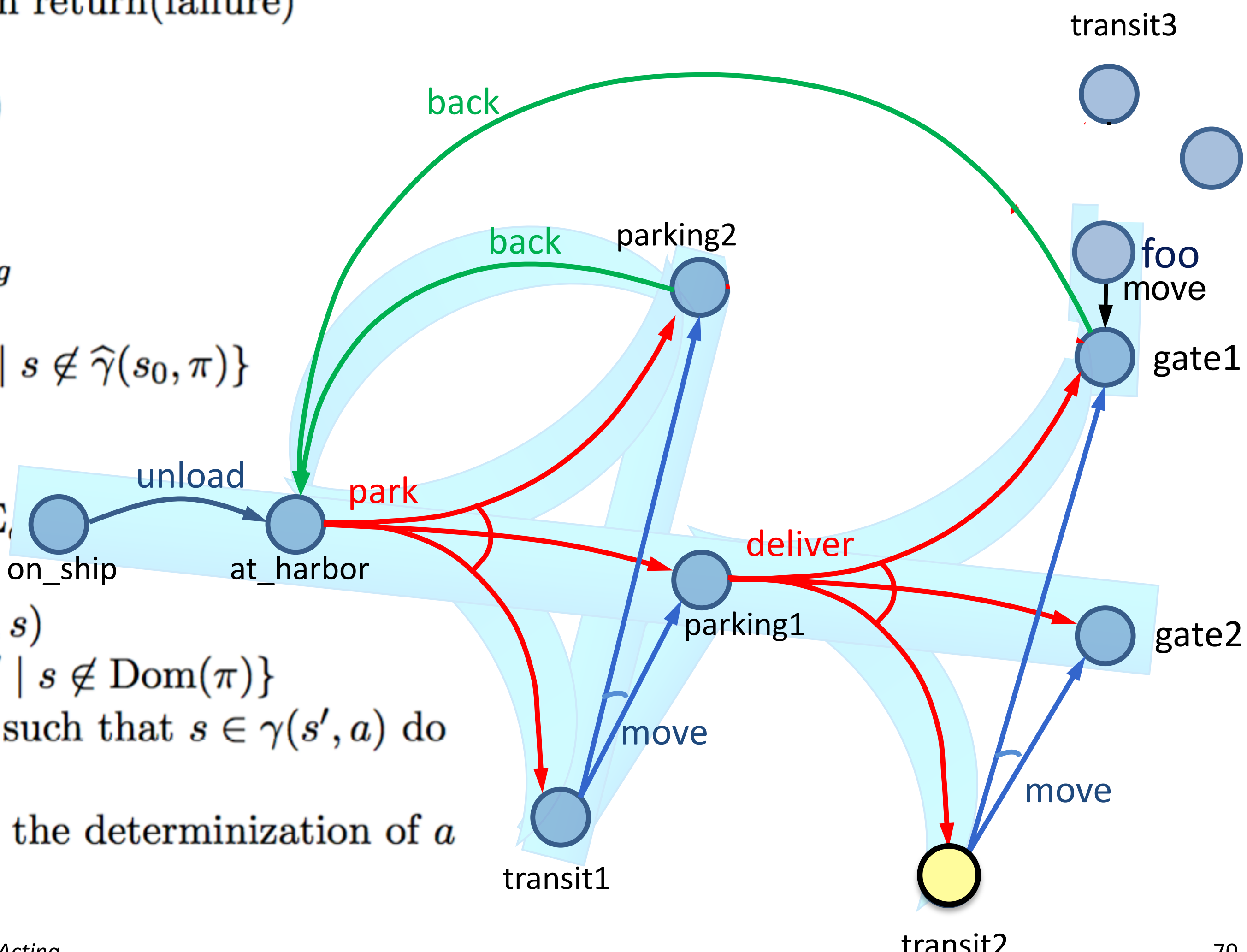
Find-Safe-Solution-by-Determinization (Σ, s_0, S_g)
- if $s_0 \in S_g$ then return($\varnothing$)
- if $Applicable(s_0) = \varnothing$ then return(failure)
- $\pi \leftarrow \varnothing$
- $\Sigma_d \leftarrow$ mk-deterministic(Σ)
- loop
  - $Q \leftarrow leaves(s_0, \pi) \setminus S_g$
  - if $Q = \varnothing$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



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 $(\Sigma,$

    if $p' \neq fail$ then do

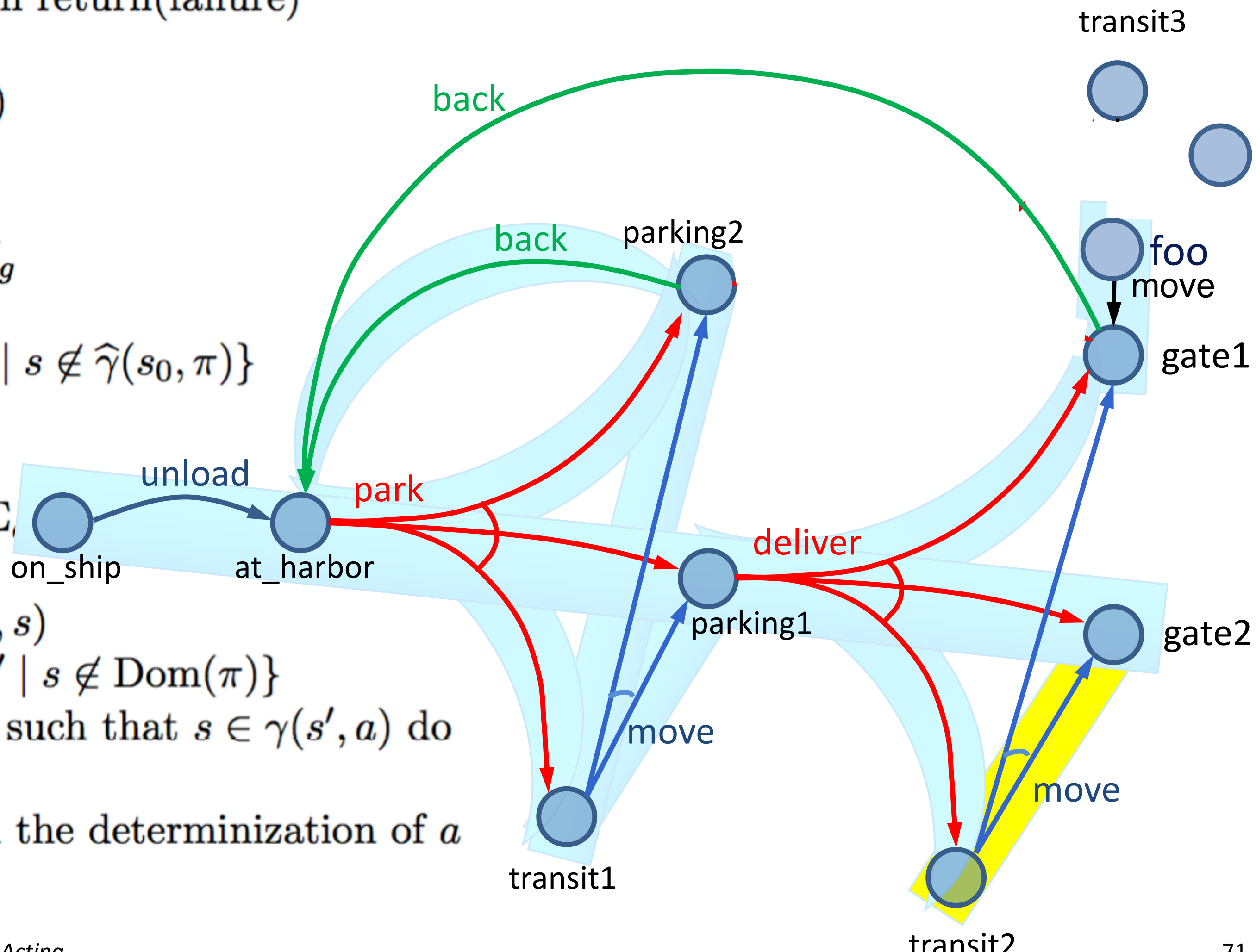
      $\pi' \leftarrow$ Plan2policy(p', s)

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

    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'



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 $(\Sigma,$

  if $p' \neq fail$ then do

   $\pi' \leftarrow$ Plan2policy(p', s)

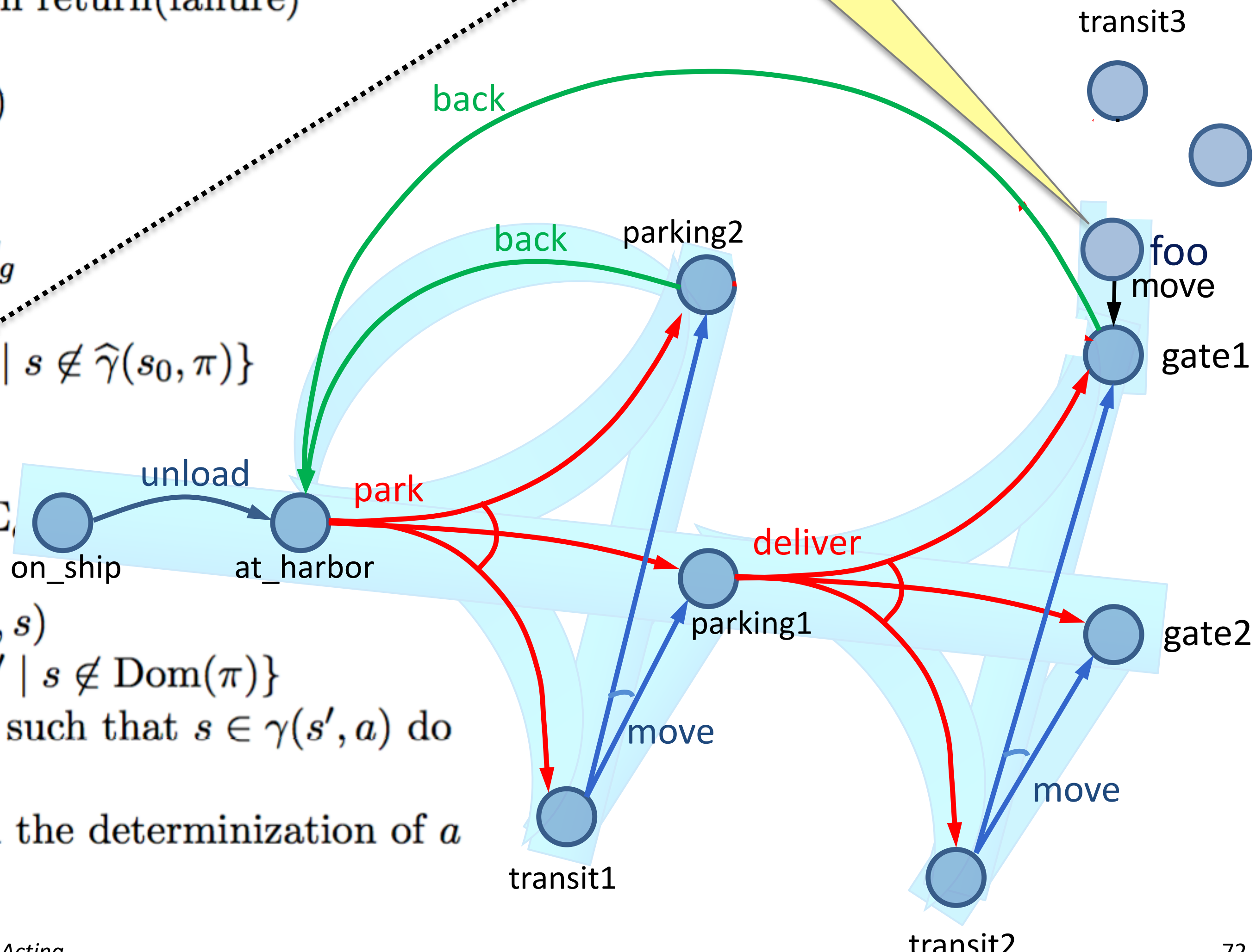
   $\pi \leftarrow \pi \cup \{(s,a) \in \pi' \mid s \notin \mathrm{Dom}(\pi)\}$

  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'

Remove unreachable parts of π



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'

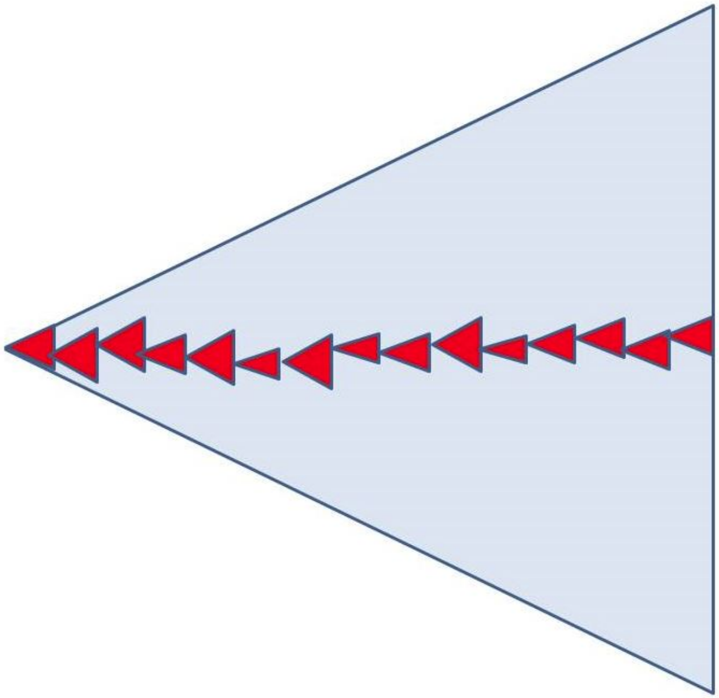
- Modify Σ_d to make actions inapplicable
  - worst-case exponential time
- Better: table of bad state-action pairs
  - For every (s',a) such that $s \in \gamma(s',a)$, $Bad[s'] \leftarrow Bad[s'] \cup$ determinization(a)
  - Modify classical planner to take the table as an argument
    - if s is current 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 reachable from all situations
- Other domains contain 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 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 (Chapter 2)

- Lookahead is any planning algorithm that returns a policy π

    - π may be partial solution, or unsafe solution

    - Lookahead-Partial-Plan executes π as far as it will go, then calls Lookahead again

```
Lookahead-Partial-Plan(Σ, s0, Sg)
    s ← s0
    while s ∉ Sg 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

- Adaptation of Run-Lookahead (Chapter 2)
- Calls Forward-Search (Chapter 2) on determinized domain, converts to a policy
  - Unsafe solution

```
FS-Replan (Σ, s, S_g)
    π_d ← ∅
    while s ∉ S_g and Applicable(s) ≠ ∅ do
        if π_d undefined for s then do
            π_d ← Plan2policy(Forward-search(Σ_d, s, S_g), s)
            if π_d = failure then return failure
        perform action π_d(s)
        s ← observe resulting state
```

- Generalization:
  - Lookahead can be any planning algorithm that returns a policy π

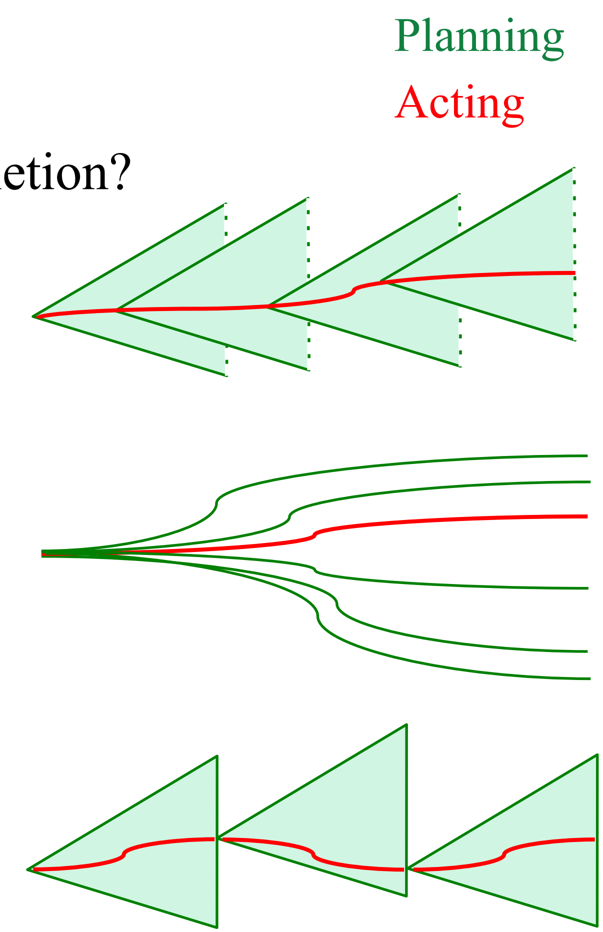
```
FS-Replan (Σ, s, S_g)    (generalize)
    π_d ← ∅
    while s ∉ S_g and Applicable(s) ≠ ∅ do
        if π_d undefined for s then do
            π_d ← Lookahead(s,θ)
            if π_d = failure then return failure
        perform action π_d(s)
        s ← observe resulting state
```

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 doesn't 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

- *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 by 1 until time runs out

  look for a solution that handles i outcomes per action

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

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

Safely Explorable Domains

- *Safely explorable* domain
  - for every state *s*, at least one goal state is reachable from *s*
- Suppose
  - We use Lookahead-Partial-Plan or FS-Replan in a safely explorable domain
  - Lookahead never returns failure
  - No “unfair” executions
- Then we will eventually reach a goal

- What would happen if we just chose a random action each time?

Online Approaches

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

Assumes each action has cost 1
Can easily be modified to use cost $\neq 1$

- 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

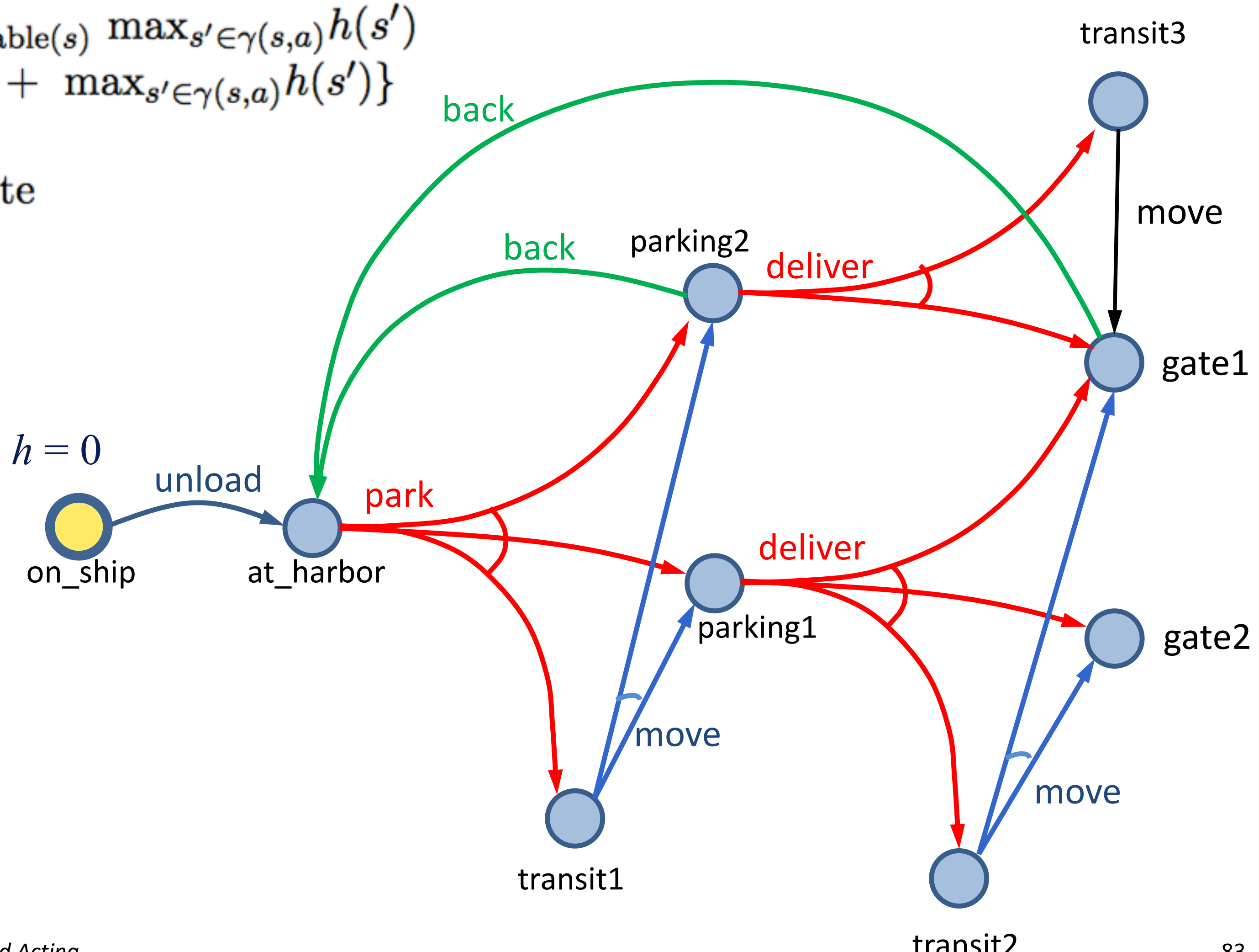
  $a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$

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

  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

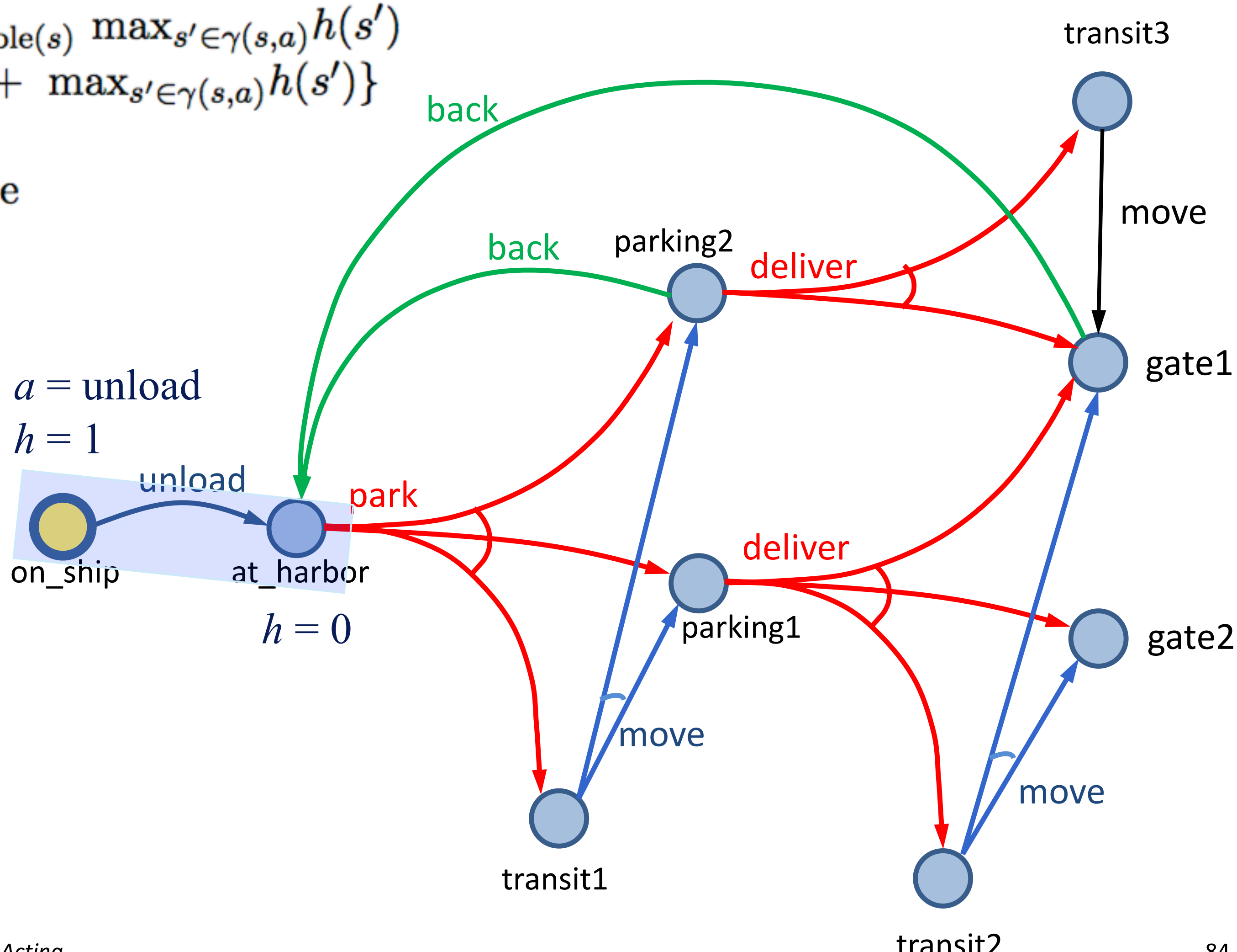
$a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$

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

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)

$s \leftarrow s_0$

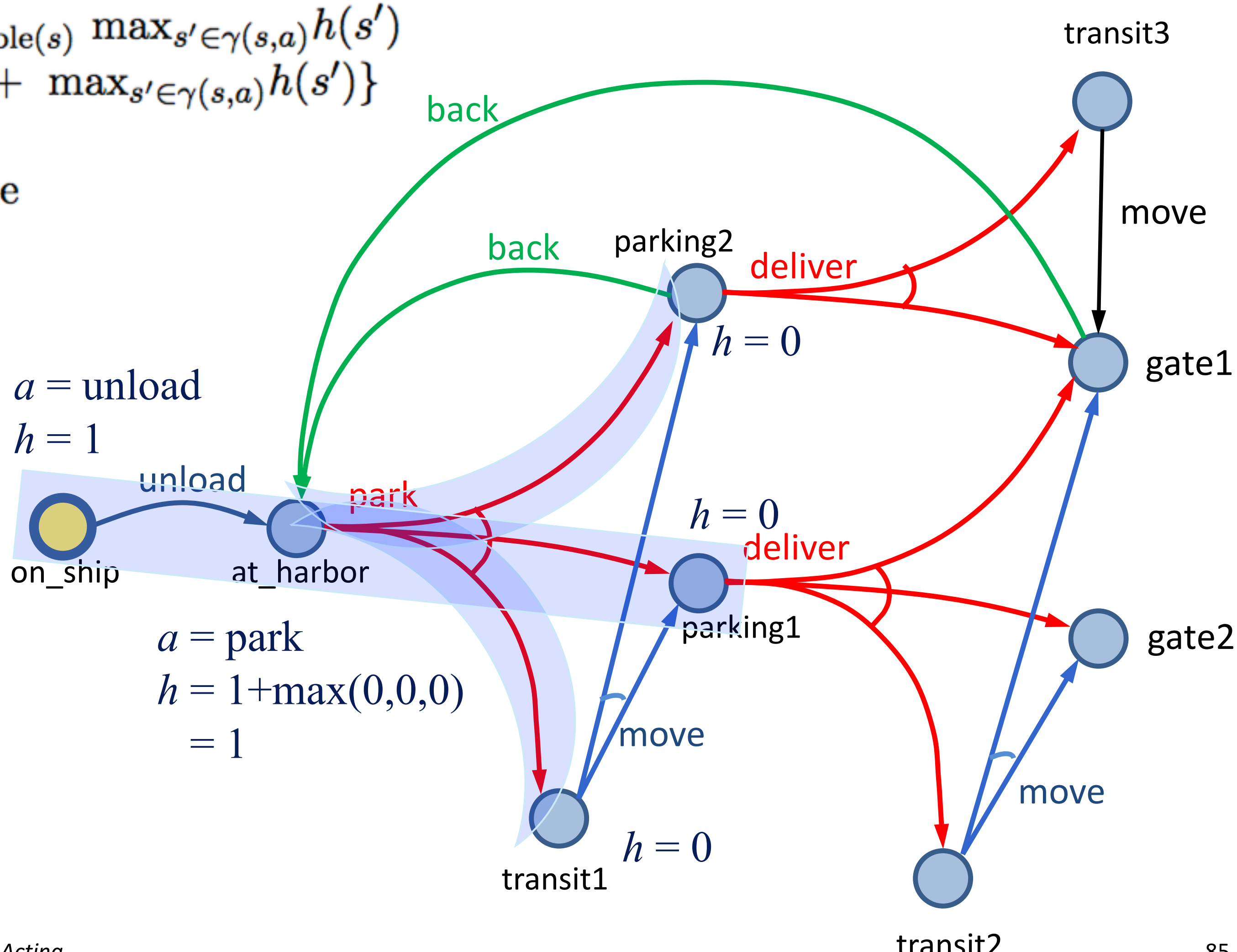
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

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

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

perform action a

$s \leftarrow$ the current state



Example

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

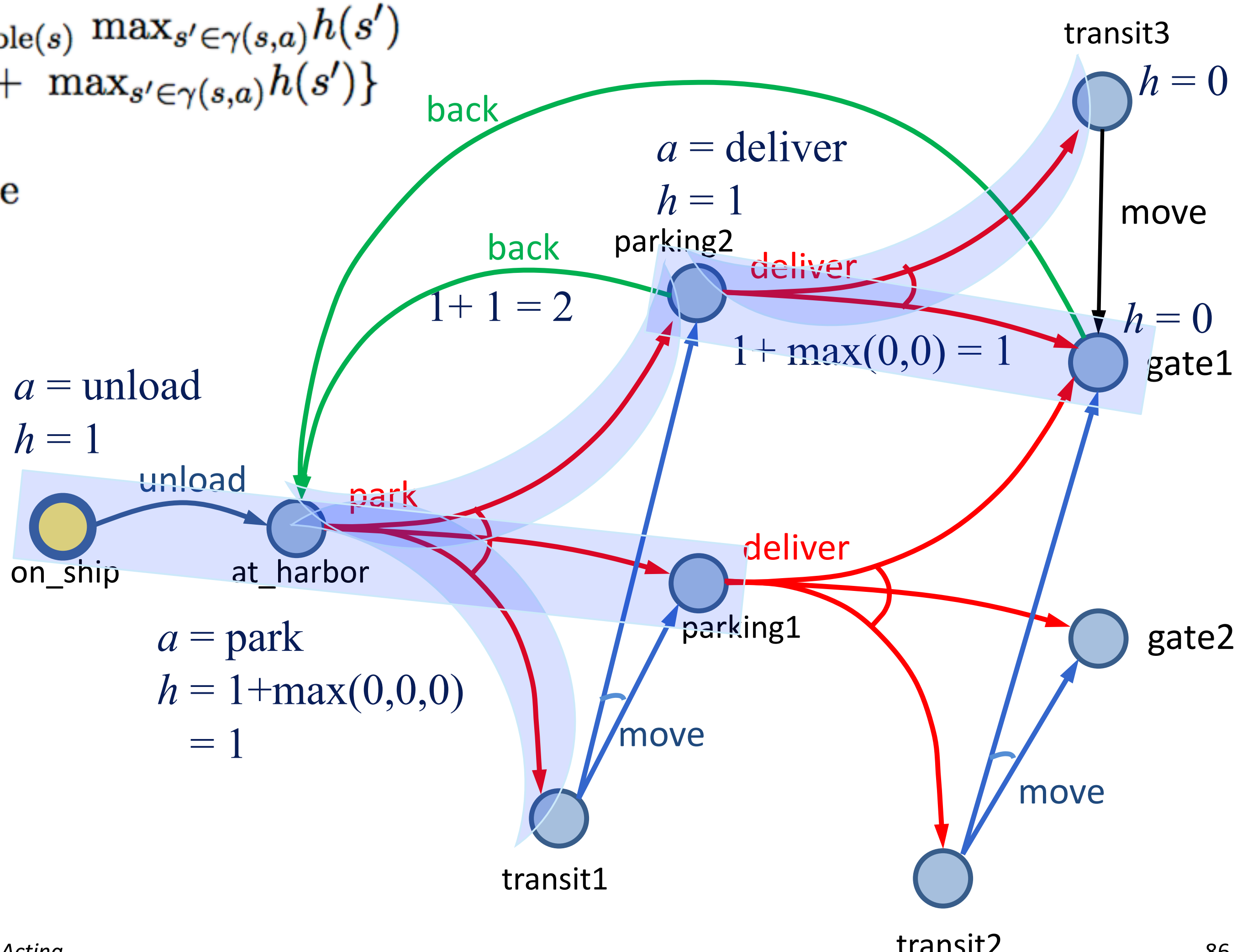
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

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

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

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, cyclic safe, acyclic safe
  - algorithms for each
- 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

- continued on next page

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
  - convergence in safely explorable domains
- min-max-LRTA*

Can also adapt Run-Concurrent-Lookahead

Can put bounds on both depth and breadth