Lecture slides for

Automated Planning: Theory and Practice

Chapter 16
Planning Based on Markov Decision Processes

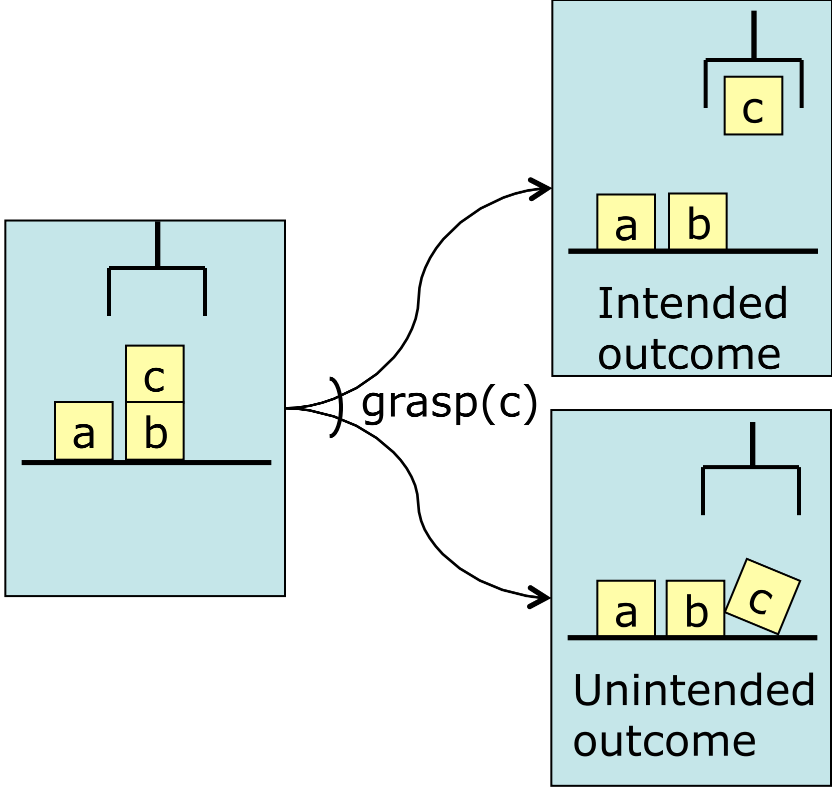
Dana S. Nau

University of Maryland

12:48 PM   February 29, 2012

Motivation

- Until now, we've assumed that each action has only one possible outcome
  - But often that's unrealistic
- In many situations, actions may have more than one possible outcome
  - Action failures
    - » e.g., gripper drops its load
  - Exogenous events
    - » e.g., road closed
- Would like to be able to plan in such situations
- One approach: Markov Decision Processes



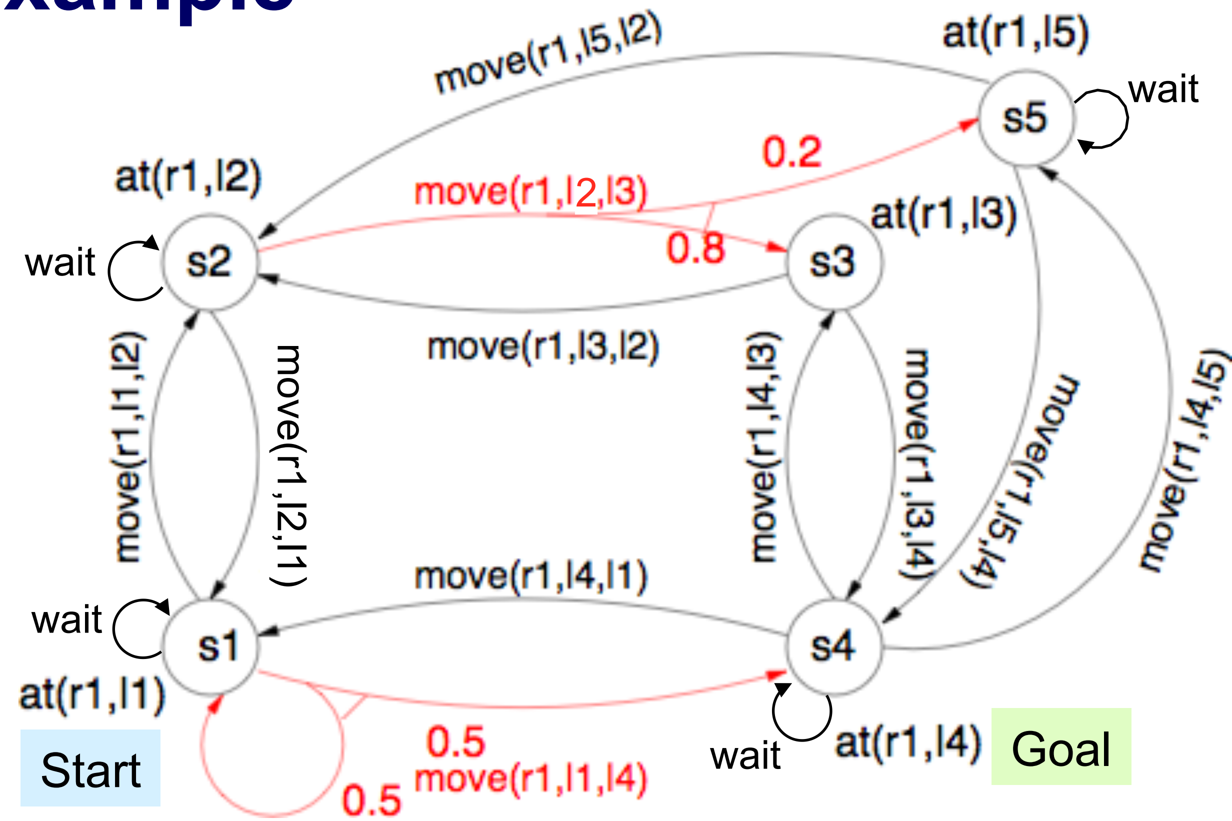
Stochastic Systems

- *Stochastic system*: a triple $\Sigma = (S, A, P)$
  - S = finite set of states
  - A = finite set of actions
  - $P_a(s' \mid s)$ = probability of going to s' if we execute a in s
  - $\sum_{s' \in S} P_a(s' \mid s) = 1$

- Several different possible action representations
  - e.g., Bayes networks, probabilistic operators
- The book does not commit to any particular representation
  - It only deals with the underlying semantics
  - Explicit enumeration of each $P_a(s' \mid s)$

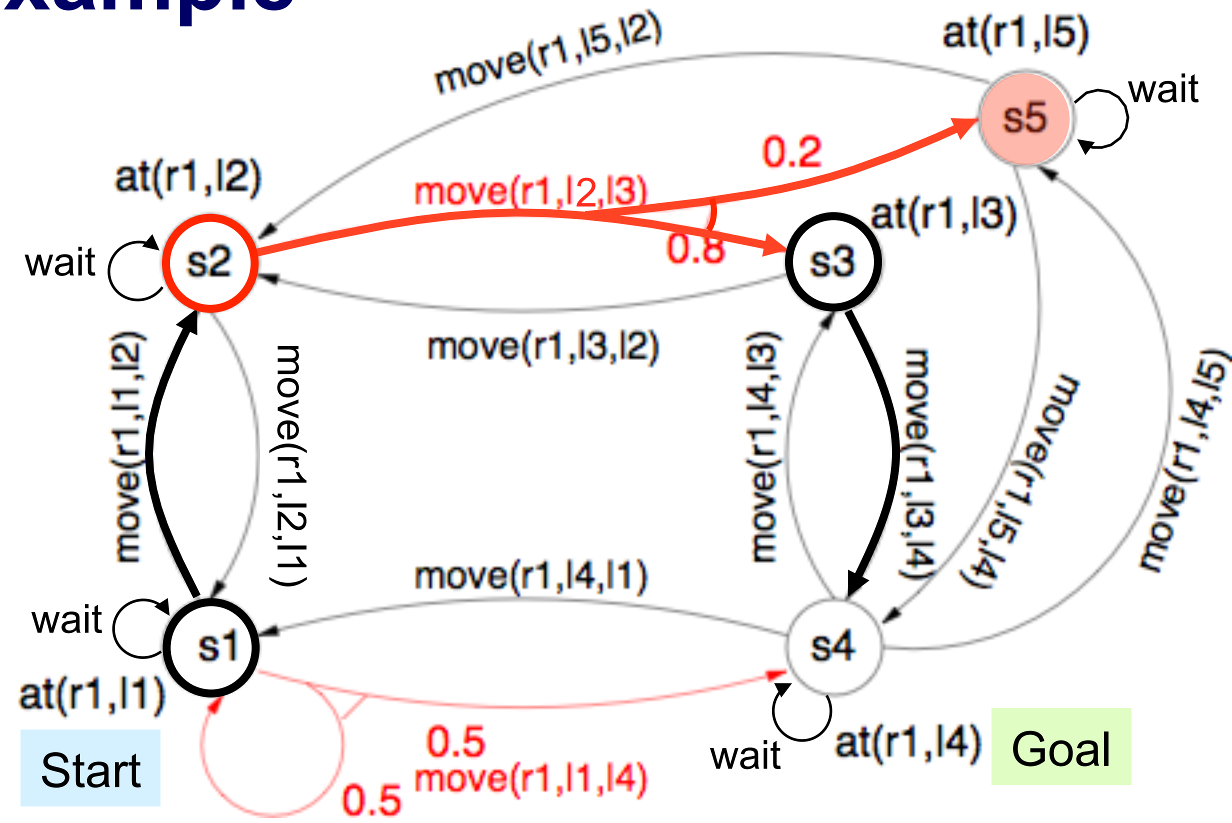
Example

- Robot r1 starts at location l1
  - State s1 in the diagram
- Objective is to get r1 to location l4
  - State s4 in the diagram



Example

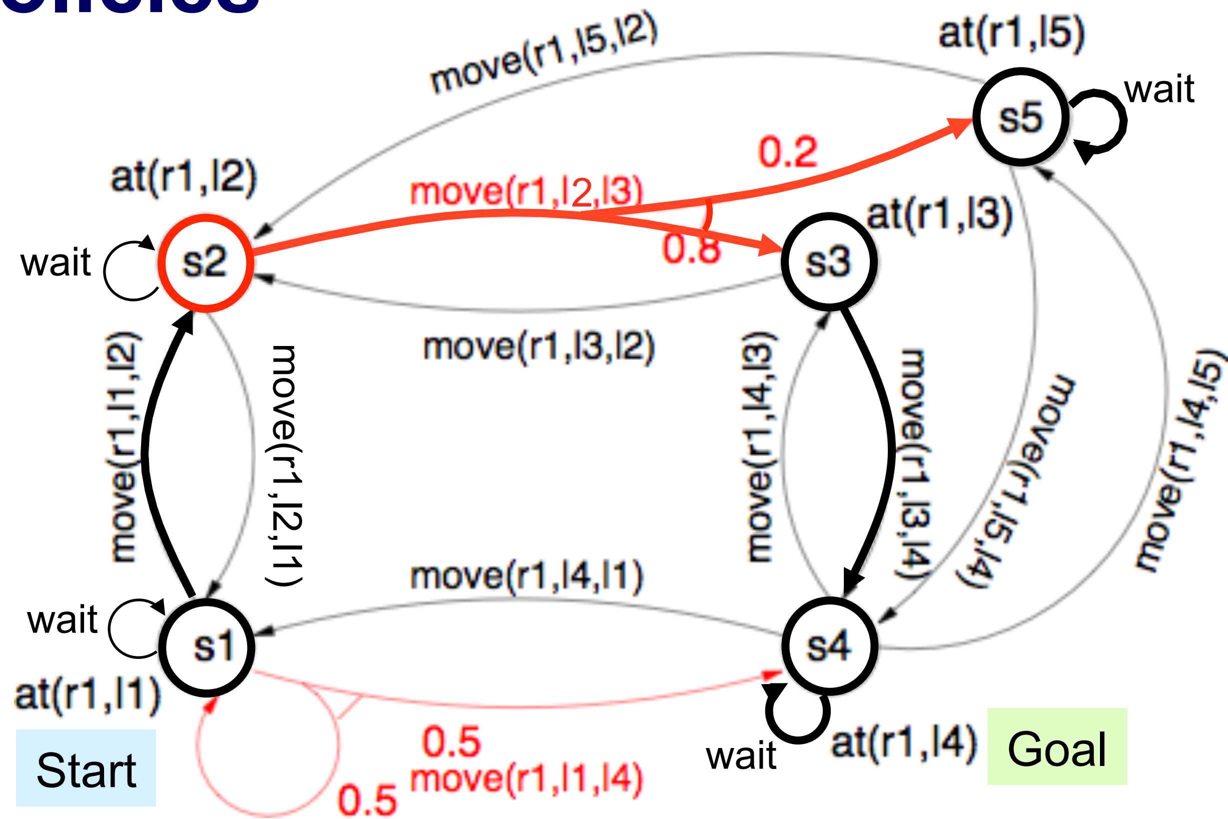
- Robot r1 starts at location l1
  - State s1 in the diagram
- Objective is to get r1 to location l4
  - State s4 in the diagram



- No classical plan (sequence of actions) can be a solution, because we can't guarantee we'll be in a state where the next action is applicable

  $\pi = \langle$move(r1,l1,l2), move(r1,l2,l3), move(r1,l3,l4)$\rangle$

Policies

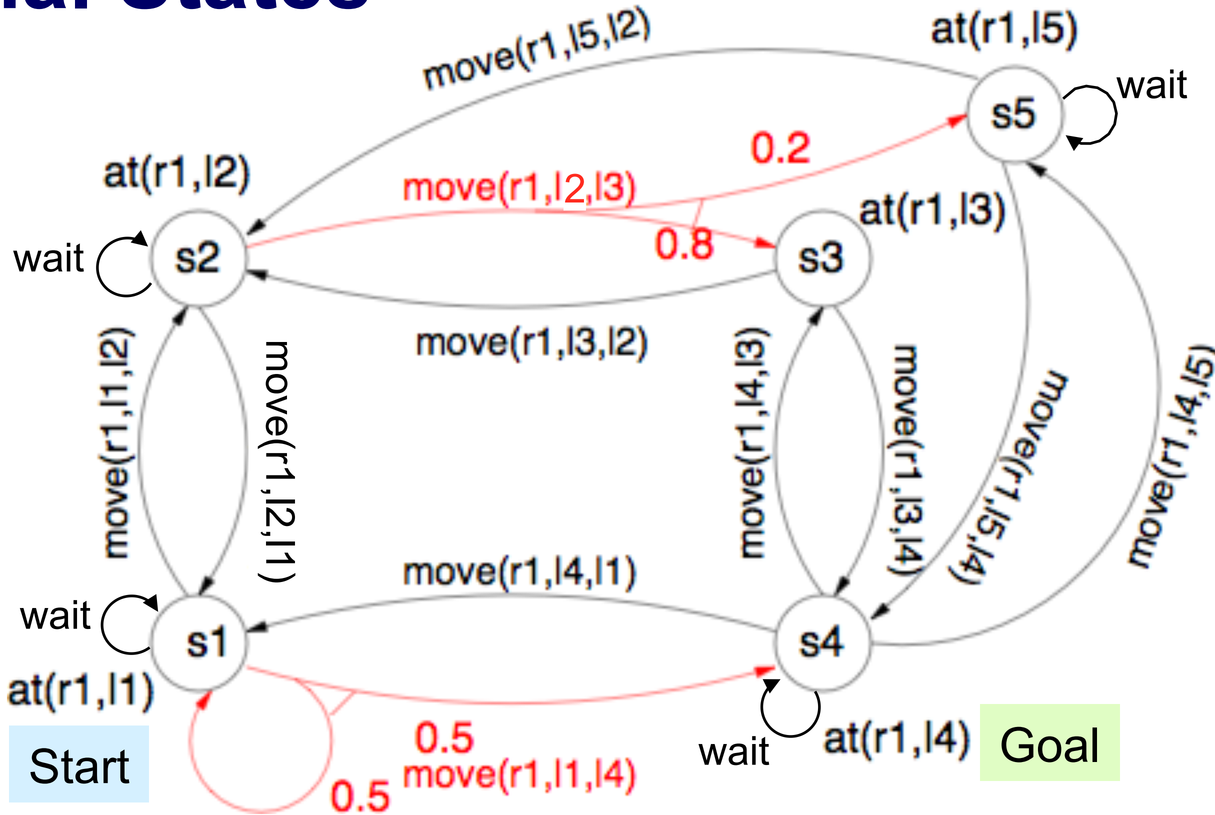


π_1 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)}

- *Policy*: a function that maps states into actions
  - Write it as a set of state-action pairs

Initial States

- For every state s, there will be a probability $P(s)$ that the system starts in s
- The book assumes there's a unique state s_0 such that the system always starts in s_0
- In the example, s_0 = s1
  - $P(s1) = 1$
  - $P(s) = 0$ for all $s \neq s1$

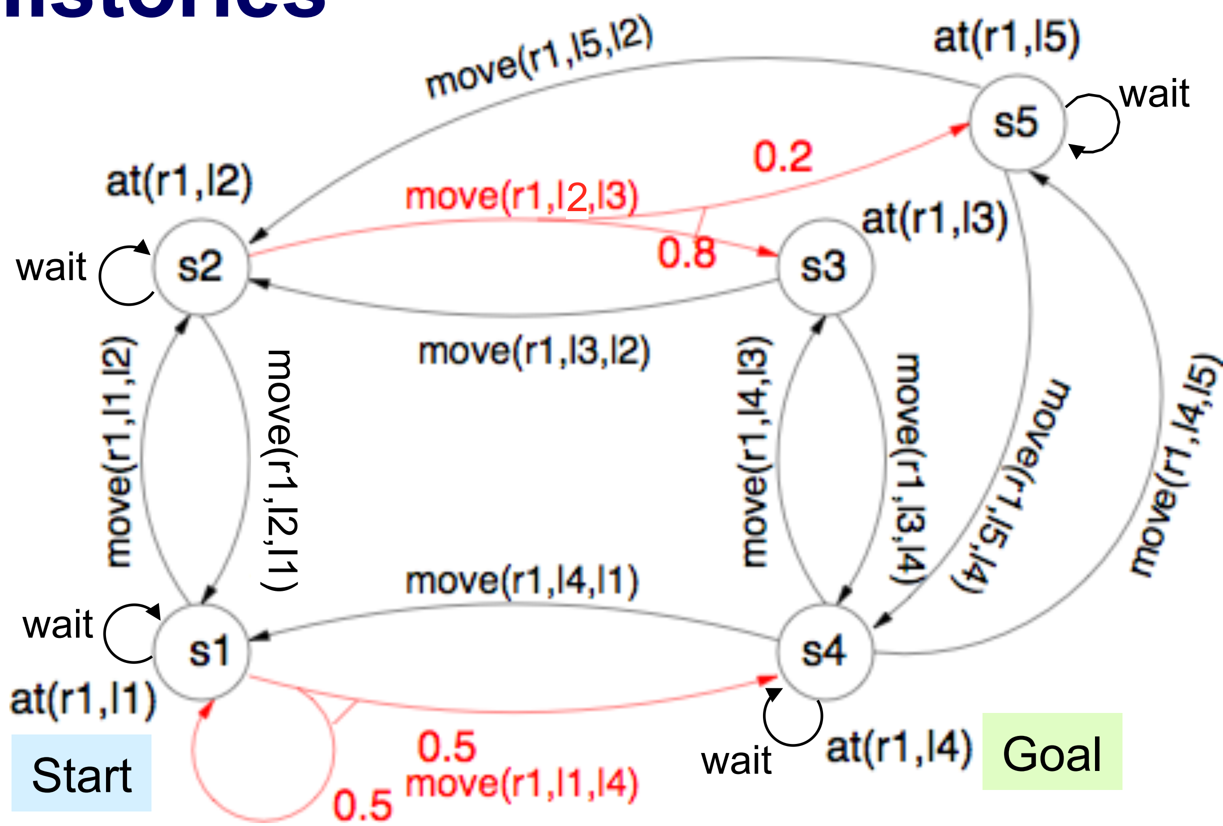


Histories

- *History*: a sequence of system states

$h = \langle s_0, s_1, s_2, s_3, s_4, \ldots \rangle$

$h_0 = \langle$s1, s3, s1, s3, s1, … $\rangle$
$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$
$h_2 = \langle$s1, s2, s5, s5, s5, … $\rangle$
$h_3 = \langle$s1, s2, s5, s4, s4, … $\rangle$
$h_4 = \langle$s1, s4, s4, s4, s4, … $\rangle$
$h_5 = \langle$s1, s1, s4, s4, s4, … $\rangle$
$h_6 = \langle$s1, s1, s1, s4, s4, … $\rangle$
$h_7 = \langle$s1, s1, s1, s1, s1, … $\rangle$

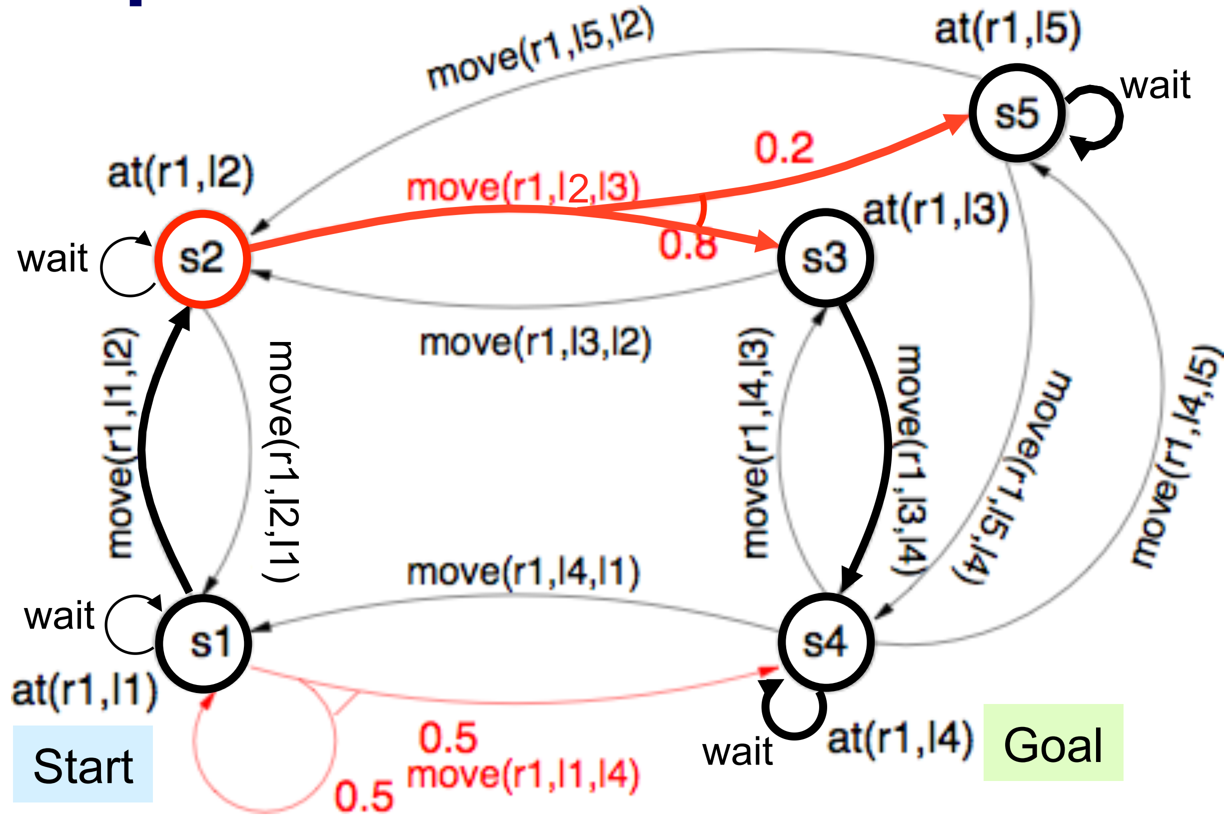


- Each policy induces a probability distribution over histories
  - If $h = \langle s_0, s_1, \ldots \rangle$ then $P(h|\pi) = P(s_0) \prod_{i \geq 0} P_{\pi(s_i)}(s_{i+1} \mid s_i)$

The book omits this because it assumes a unique starting state

Example

π_1 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)}



$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$ goal

$h_2 = \langle$s1, s2, s5, s5 … $\rangle$

$P(h_1 \mid \pi_1) = 1 \times 1 \times .8 \times 1 \times \ldots = 0.8$

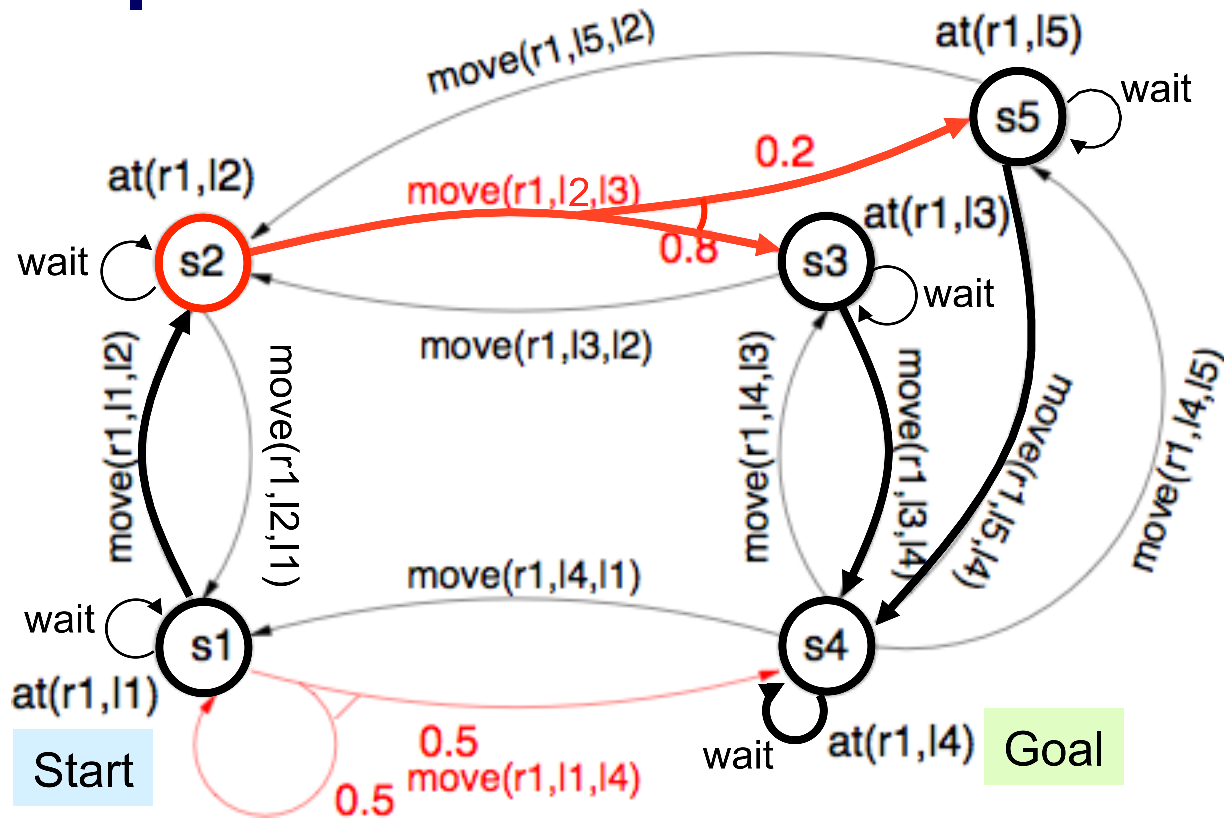
$P(h_2 \mid \pi_1) = 1 \times 1 \times .2 \times 1 \times \ldots = 0.2$

$P(h \mid \pi_1) = 0$ for all other h

so π_1 reaches the goal with probability 0.8

Example

$\pi_2 = \{$(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4))$\}$



$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$ $\quad P(h_1 \mid \pi_2) = 1 \times 0.8 \times 1 \times 1 \times \ldots = 0.8$

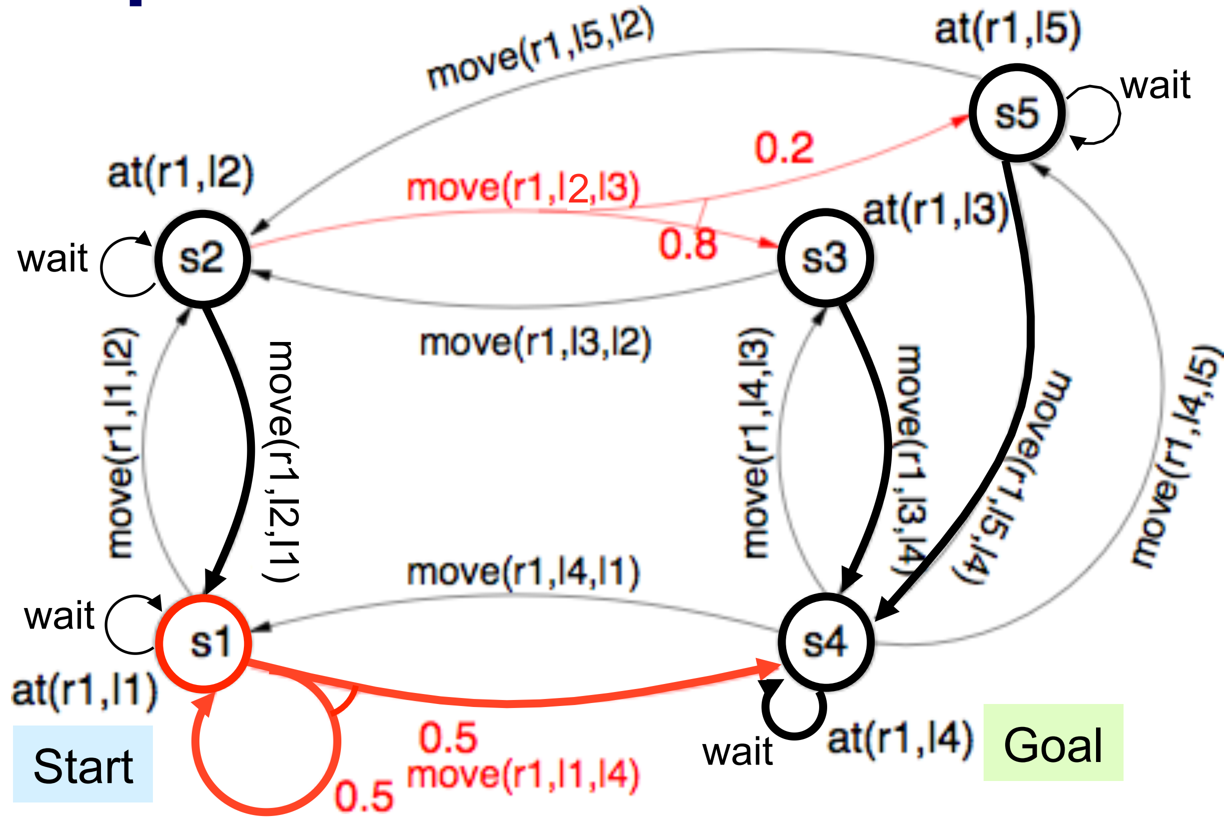
$h_3 = \langle$s1, s2, s5, s4, s4, … $\rangle$ $\quad P(h_3 \mid \pi_2) = 1 \times 0.2 \times 1 \times 1 \times \ldots = 0.2$

goal

$P(h \mid \pi_1) = 0$ for all other h

so π_2 reaches the goal with probability 1

Example



$\pi_3 = \{$(s1, move(r1,l1,l4)),
(s2, move(r1,l2,l1)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4)$\}$

π_3 reaches the goal with probability 1.0

goal

$h_4 = \langle$s1, s4, s4, s4, … $\rangle$ $\qquad P(h_4 \mid \pi_3) = 0.5 \times 1 \times 1 \times 1 \times 1 \times \ldots = 0.5$

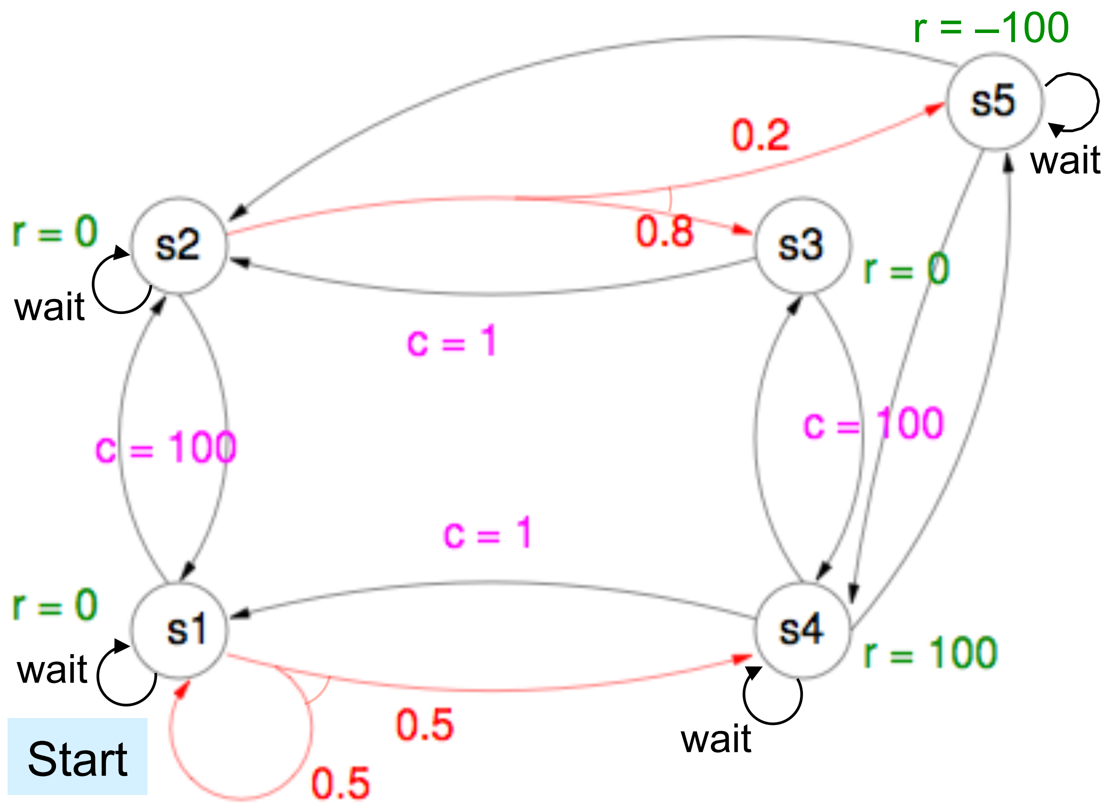
$h_5 = \langle$s1, s1, s4, s4, s4, … $\rangle$ $\qquad P(h_5 \mid \pi_3) = 0.5 \times 0.5 \times 1 \times 1 \times 1 \times \ldots = 0.25$

$h_6 = \langle$s1, s1, s1, s4, s4, … $\rangle$ $\qquad P(h_6 \mid \pi_3) = 0.5 \times 0.5 \times 0.5 \times 1 \times 1 \times \ldots = 0.125$

…

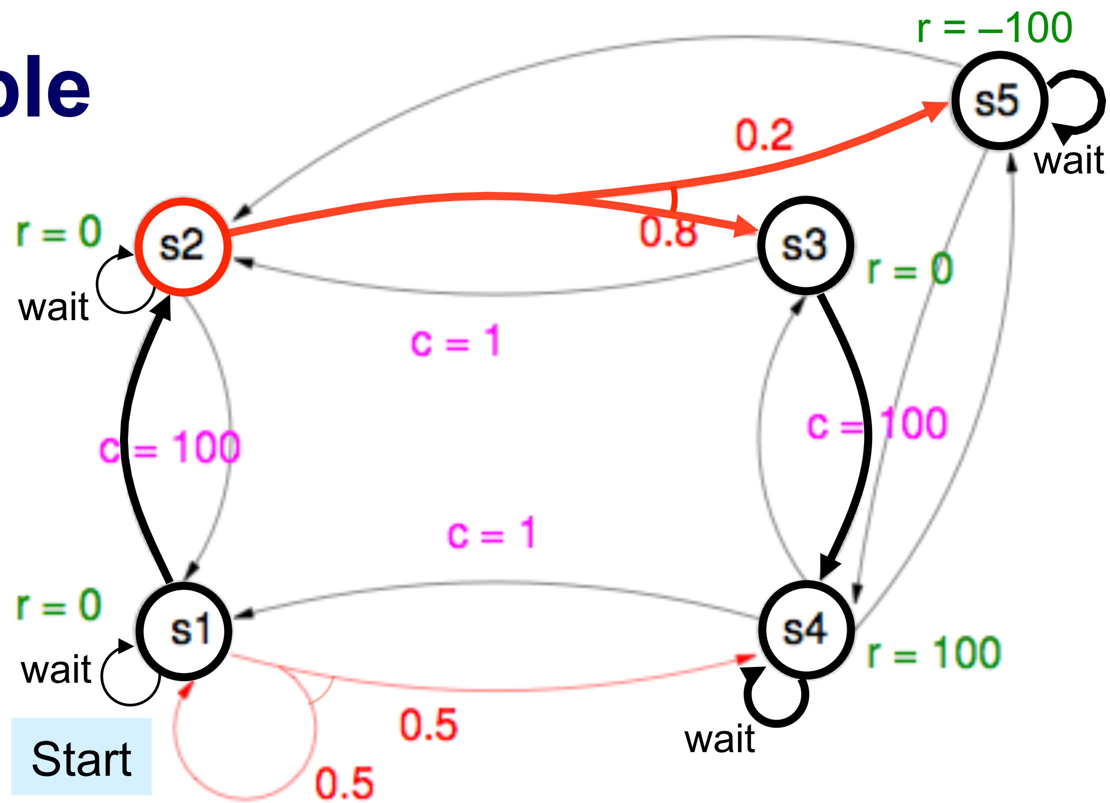
$h_7 = \langle$s1, s1, s1, s1, s1, s1, … $\rangle$ $\qquad P(h_7 \mid \pi_3) = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \times \ldots = 0$

Utility



- Numeric *cost* $C(s,a)$ for each state s and action a
- Numeric *reward* $R(s)$ for each state s
- No explicit goals any more
  - Desirable states have high rewards
- Example:
  - $C(s,\text{wait}) = 0$ at every state except s3
  - $C(s,a) = 1$ for each "horizontal" action
  - $C(s,a) = 100$ for each "vertical" action
  - R as shown
- Utility of a history:
  - If $h = \langle s_0, s_1, \ldots \rangle$, then $V(h \mid \pi) = \sum_{i \geq 0} [R(s_i) - C(s_i,\pi(s_i))]$

Example



π_1 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)}

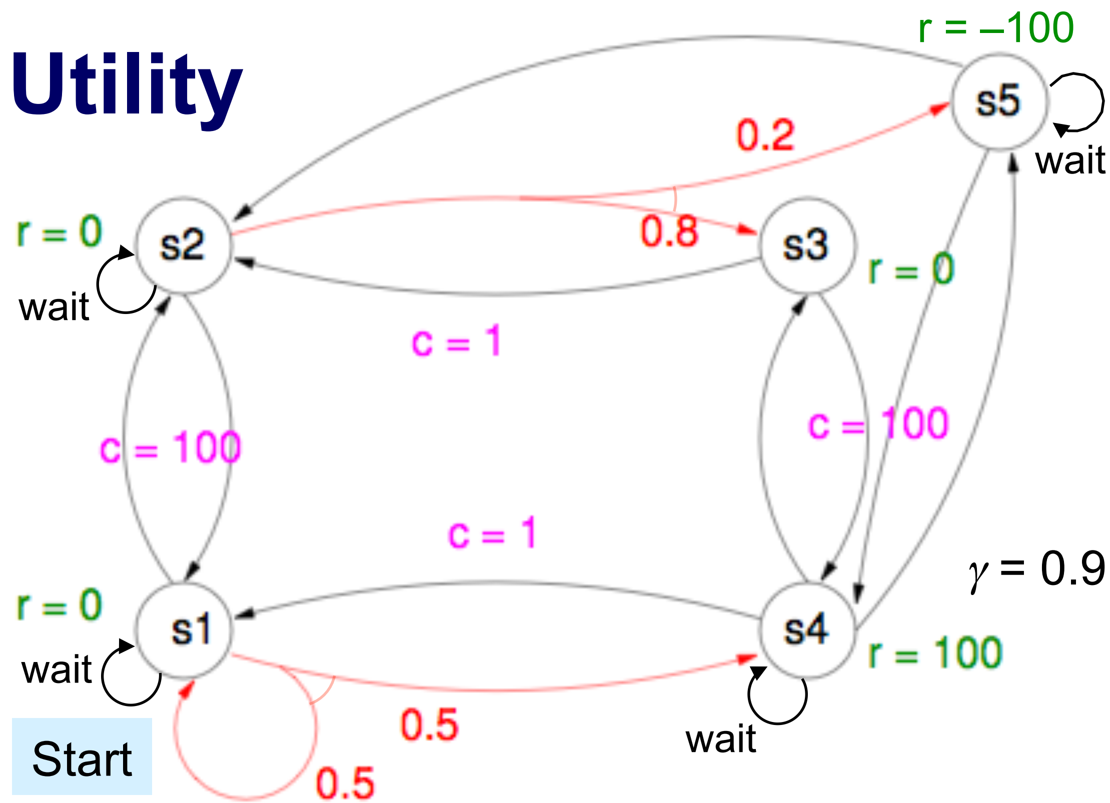
$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$

$h_2 = \langle$s1, s2, s5, s5 … $\rangle$

$$V(h_1|\pi_1) = [R(\text{s1}) - C(\text{s1},\pi_1(\text{s1}))] + [R(\text{s2}) - C(\text{s2},\pi_1(\text{s2}))] + [R(\text{s3}) - C(\text{s3},\pi_1(\text{s3}))] + [R(\text{s4}) - C(\text{s4},\pi_1(\text{s4}))] + [R(\text{s4}) - C(\text{s4},\pi_1(\text{s4}))] + \ldots$$
$$= [0 - 100] + [0 - 1] + [0 - 100] + [100 - 0] + [100 - 0] + \ldots = \infty$$

$$V(h_2|\pi_1) = [0 - 100] + [0 - 1] + [-100 - 0] + [-100 - 0] + [-100 - 0] + \ldots = -\infty$$

Discounted Utility

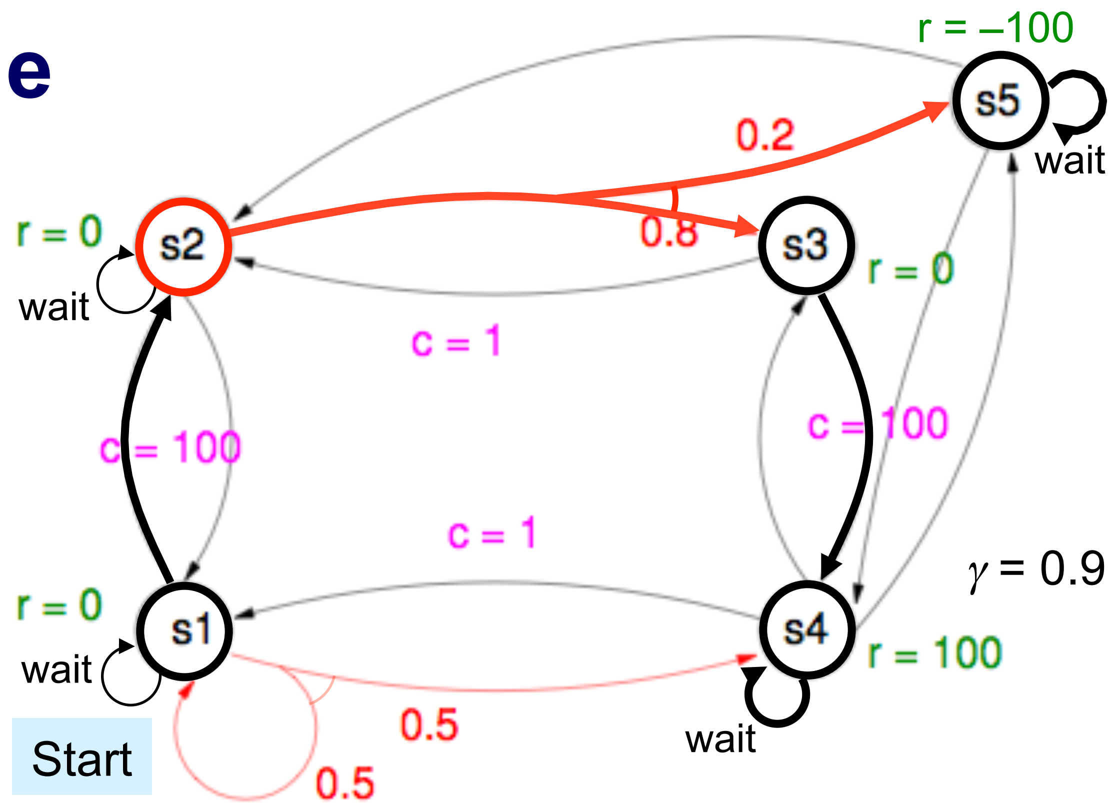


- We often need to use a **discount factor**, γ
  - $0 \leq \gamma \leq 1$
- Discounted utility of a history:

$$V(h \mid \pi) = \sum_{i \geq 0} \gamma^i \left[R(s_i) - C(s_i, \pi(s_i))\right]$$

  - Distant rewards/costs have less influence
  - Convergence is guaranteed if $0 \leq \gamma < 1$
- Expected utility of a policy:
  - $E(\pi) = \sum_h P(h|\pi)\, V(h|\pi)$

Example



π_1 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)}

$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$
$h_2 = \langle$s1, s2, s5, s5 … $\rangle$

$$V(h_1|\pi_1) = .9^0[0 - 100] + .9^1[0 - 1] + .9^2[0 - 100] + .9^3[100 - 0] + .9^4[100 - 0] + \ldots = 547.9$$

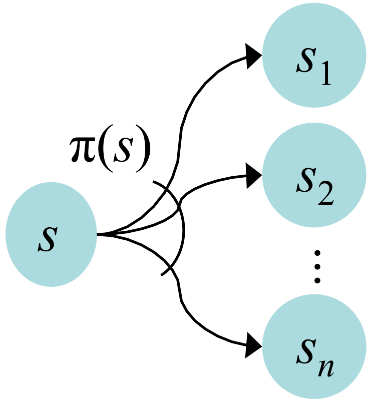
$$V(h_2|\pi_1) = .9^0[0 - 100] + .9^1[0 - 1] + .9^2[-100 - 0] + .9^3[-100 - 0] + \ldots = -910.1$$

$$E(\pi_1) = 0.8\, V(h_1|\pi_1) + 0.2\, V(h_2|\pi_1) = 0.8(547.9) + 0.2(-910.1) = 256.3$$

Planning as Optimization

- For the rest of this chapter, a special case:
  - Start at state s_0
  - All rewards are 0
  - Consider *cost* rather than *utility*
    - » the negative of what we had before
- This makes the equations slightly simpler
  - Can easily generalize everything to the case of nonzero rewards
- Discounted cost of a history h:
  - $C(h \mid \pi) = \sum_{i \geq 0} \gamma^i C(s_i, \pi(s_i))$
- Expected cost of a policy π:
  - $E(\pi) = \sum_h P(h \mid \pi) C(h \mid \pi)$
- A policy π is *optimal* if for every π', $E(\pi) \leq E(\pi')$
- A policy π is *everywhere optimal* if for every s and every π', $E_\pi(s) \leq E_{\pi'}(s)$
  - where $E_\pi(s)$ is the expected utility if we start at s rather than s_0

Bellman's Theorem



- If π is any policy, then for every s,
  - $E_\pi(s) = C(s, \pi(s)) + \gamma \sum_{s \in S} P_{\pi(s)}(s' \mid s)\, E_\pi(s')$
- Let $Q_\pi(s,a)$ be the expected cost in a state s if we start by executing the action a, and use the policy π from then onward
  - $Q_\pi(s,a) = C(s,a) + \gamma \sum_{s' \in S} P_a(s' \mid s)\, E_\pi(s')$
- **Bellman's theorem:** Suppose π^* is everywhere optimal. Then for every s, $E_{\pi^*}(s) = \min_{a \in A(s)} Q_{\pi^*}(s,a)$.
- **Intuition:**
  - If we use π^* everywhere else, then the set of optimal actions at s is $\arg\min_{a \in A(s)} Q_{\pi^*}(s,a)$
  - If π^* is optimal, then at each state it should pick one of those actions
  - Otherwise we can construct a better policy by using an action in $\arg\min_{a \in A(s)} Q_{\pi^*}(s,a)$, instead of the action that π^* uses
- From Bellman's theorem it follows that for all s,
  - $E_{\pi^*}(s) = \min_{a \in A(s)} \{C(s,a) + \gamma \sum_{s' \in S} P_a(s' \mid s)\, E_{\pi^*}(s')\}$

Policy Iteration

- Policy iteration is a way to find π^*
  - Suppose there are n states $s_1, \ldots, s_n$
  - Start with an arbitrary initial policy π_1
  - For $i = 1, 2, \ldots$
    - » Compute π_i's expected costs by solving n equations with n unknowns
      - n instances of the first equation on the previous slide

$$E_{\pi_i}(s_1) = C(s,\pi_i(s_1)) + \gamma \sum_{k=1}^{n} P_{\pi_i(s_1)}(s_k \mid s_1)\; E_{\pi_i}(s_k)$$
$$\vdots$$
$$E_{\pi_i}(s_n) = C(s,\pi_i(s_n)) + \gamma \sum_{k=1}^{n} P_{\pi_i(s_n)}(s_k \mid s_n)\; E_{\pi_i}(s_k)$$

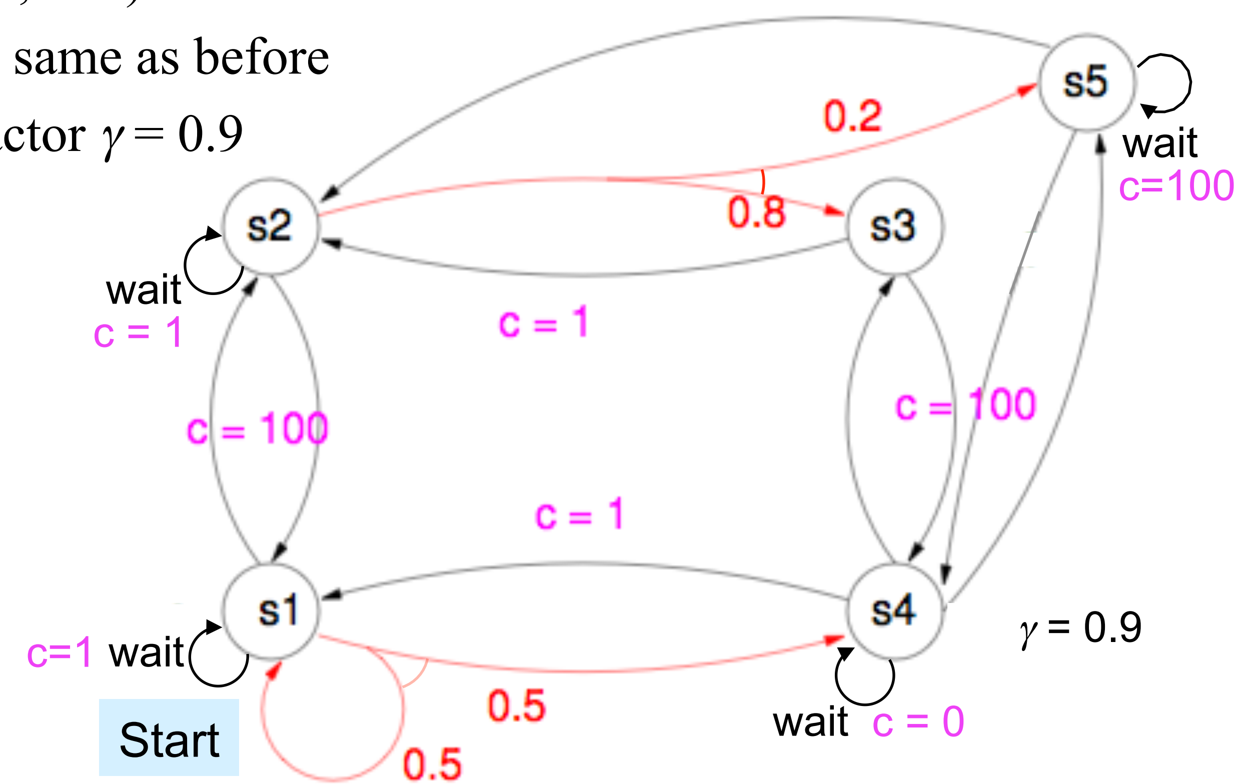
    - » For every s_j,

$$\pi_{i+1}(s_j) = \arg\min_{a \in A} Q_{\pi_i}(s_j,a)$$
$$= \arg\min_{a \in A} C(s_j,a) + \gamma \sum_{k=1}^{n} P_a(s_k \mid s_j)\; E_{\pi_i}(s_k)$$

    - » If $\pi_{i+1} = \pi_i$ then exit
- Converges in a finite number of iterations

Example

- Modification of the previous example
  - To get rid of the rewards but still make s5 undesirable:
    - » C(s5, wait) = 100
  - To provide incentive to leave non-goal states:
    - » C(s1,wait) = C(s2,wait) = 1
  - All other costs are the same as before
  - As before, discount factor $\gamma = 0.9$

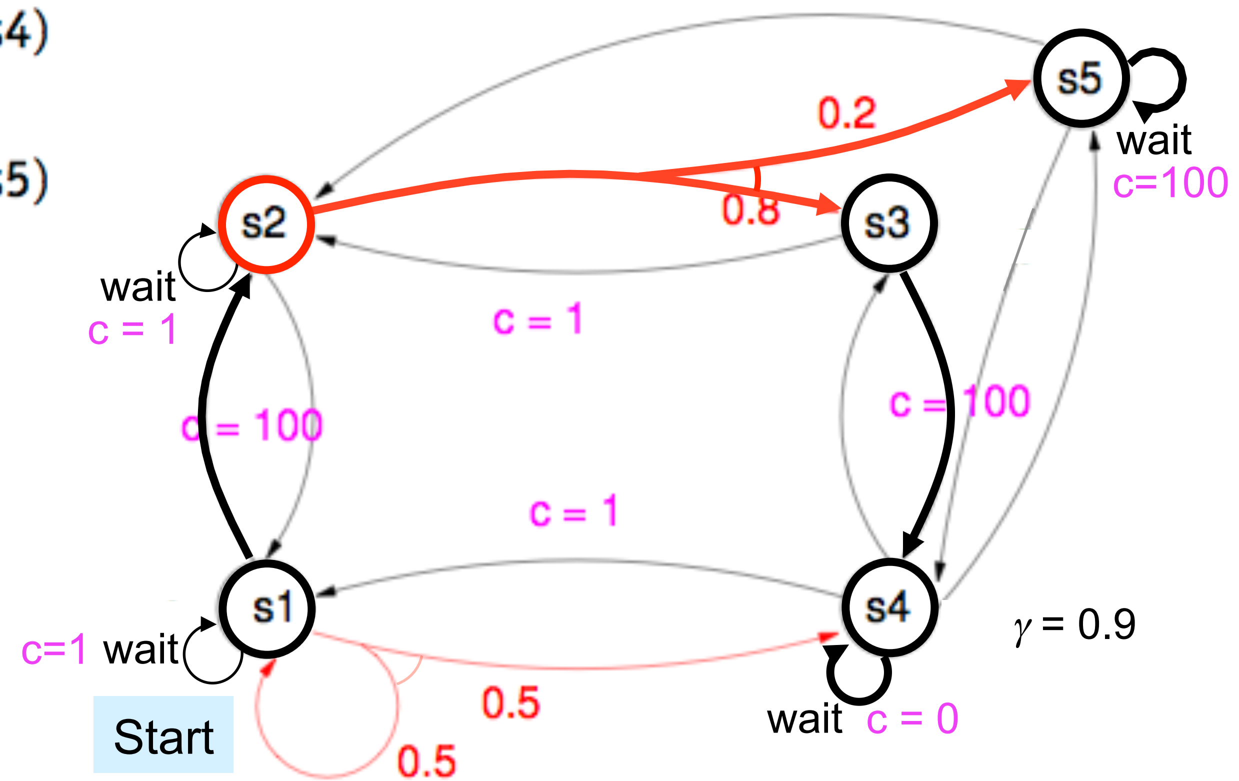


$$E_{\pi_1}(s1) = C(s1, \text{move}(r1, l1, l2)) + \gamma E_{\pi_1}(s2)$$
$$E_{\pi_1}(s2) = C(s2, \text{move}(r1, l2, l3)) + \gamma (0.8\, E_{\pi_1}(s3) + 0.2\, E_{\pi_1}(s5))$$
$$E_{\pi_1}(s3) = C(s4, \text{move}(r1, l3, l4)) + \gamma E_{\pi_1}(s4)$$
$$E_{\pi_1}(s4) = C(s4, \text{wait}) + \gamma E_{\pi_1}(s4)$$
$$E_{\pi_1}(s5) = C(s5, \text{wait}) + \gamma E_{\pi_1}(s5)$$

$\pi_1 = \{$(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)$\}$

$$E_{\pi_1}(s1) = 100 + (0.9)\, E_{\pi_1}(s2)$$
$$E_{\pi_1}(s2) = 1 + (0.9)(0.8\, E_{\pi_1}(s3) + 0.2\, E_{\pi_1}(s5))$$
$$E_{\pi_1}(s3) = 100 + (0.9)\, E_{\pi_1}(s4)$$
$$E_{\pi_1}(s4) = 0 + (0.9)\, E_{\pi_1}(s4)$$
$$E_{\pi_1}(s5) = 100 + (0.9)\, E_{\pi_1}(s5)$$

$$E_{\pi_1}(s1) = 181.9$$
$$E_{\pi_1}(s2) = 91$$
$$E_{\pi_1}(s3) = 100$$
$$E_{\pi_1}(s4) = 0$$
$$E_{\pi_1}(s5) = 1000$$



$\gamma = 0.9$

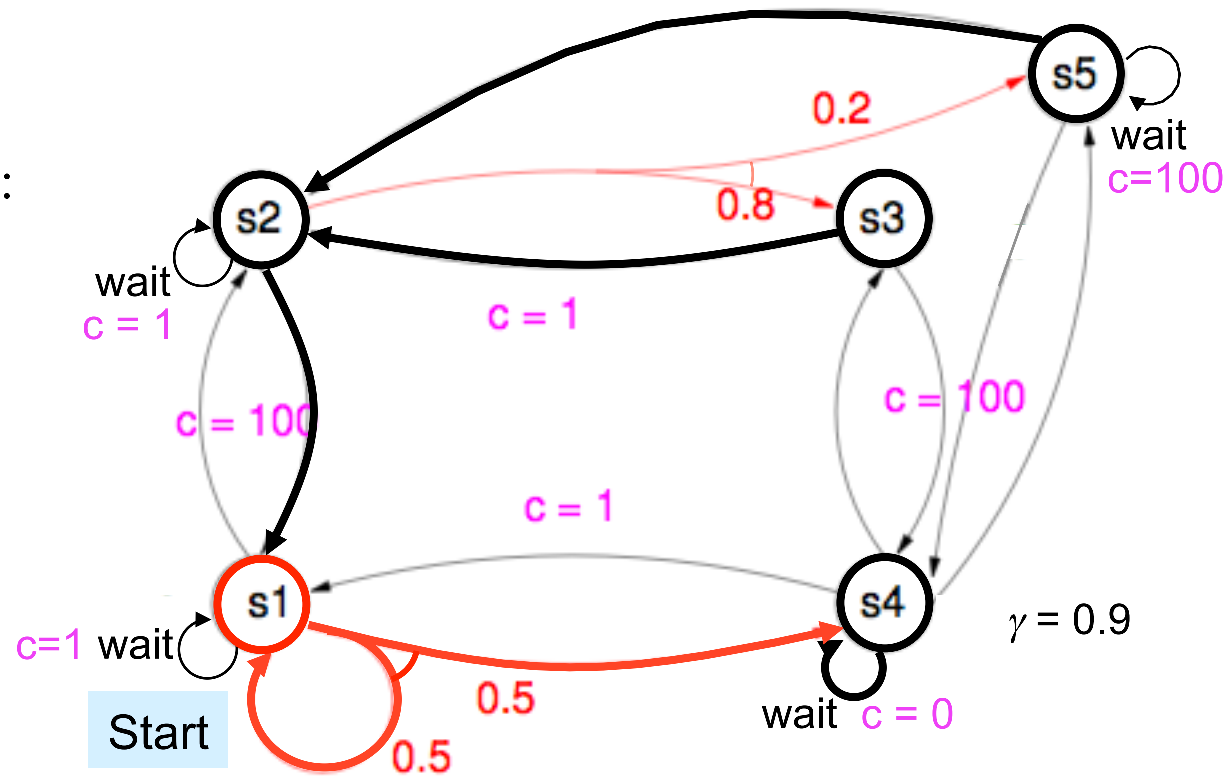
Example (Continued)

$$E_{\pi_1}(\text{s1}) = 181.9$$
$$E_{\pi_1}(\text{s2}) = 91$$
$$E_{\pi_1}(\text{s3}) = 100$$
$$E_{\pi_1}(\text{s4}) = 0$$
$$E_{\pi_1}(\text{s5}) = 1000$$

$\pi_1 = \{$(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, wait)$\}$

- At each state s, let $\pi_2(s) = \arg\min_{a \in A(s)} Q_{\pi_1}(s,a)$:

- $\pi_2 = \{$(s1, move(r1,l1,l4)),
(s2, move(r1,l2,l1)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4)$\}$



Value Iteration

- Start with an arbitrary cost $E_0(s)$ for each s and a small $\varepsilon > 0$
- For $i = 1, 2, \ldots$
  - for every s in S and a in A,
    - $Q_i(s,a) := C(s,a) + \gamma \sum_{s' \in S} P_a(s' \mid s)\, E_{i-1}(s')$
    - » $E_i(s) = \min_{a \in A(s)} Q_i(s,a)$
    - » $\pi_i(s) = \arg\min_{a \in A(s)} Q_i(s,a)$
  - If $\max_{s \in S} |E_i(s) - E_{i-1}(s)| < \varepsilon$ for every s then exit

- π_i converges to π^* after finitely many iterations, but how to tell it has converged?
  - In Policy Iteration, we checked whether π_i stopped changing
  - In Value Iteration, that doesn't work
- In general, $E_i \neq E_{\pi_i}$
  - When π_i doesn't change, E_i may still change
  - The changes in E_i may make π_i start changing again

Value Iteration

- Start with an arbitrary cost $E_0(s)$ for each s and a small $\varepsilon > 0$
- For $i = 1, 2, \ldots$
  - for each s in S do
    - » for each a in A do
      - $Q(s,a) := C(s,a) + \gamma \sum_{s' \in S} P_a(s' \mid s)\, E_{i-1}(s')$
    - » $E_i(s) = \min_{a \in A(s)} Q(s,a)$
    - » $\pi_i(s) = \arg\min_{a \in A(s)} Q(s,a)$
  - If $\max_{s \in S} |E_i(s) - E_{i-1}(s)| < \varepsilon$ for every s then exit

- If E_i changes by $< \varepsilon$ and if ε is small enough, then π_i will no longer change
  - In this case π_i has converged to π^*

- How small is small enough?

Example

- Let a_{ij} be the action that moves from s_i to s_j
  - e.g., a_{11}= wait and a_{12} = move(r1,l1,l2))
- Start with $E_0(s) = 0$ for all s, and $\varepsilon = 1$

$Q(\text{s1}, a_{11}) = 1 + .9\times 0 = 1$
$Q(\text{s1}, a_{12}) = 100 + .9\times 0 = 100$
$Q(\text{s1}, a_{14}) = 1 + .9(.5\times 0 + .5\times 0) = 1$

$Q(\text{s2}, a_{21}) = 100 + .9\times 0 = 100$
$Q(\text{s2}, a_{22}) = 1 + .9\times 0 = 1$
$Q(\text{s2}, a_{23}) = 1 + .9(.5\times 0 + .5\times 0) = 1$

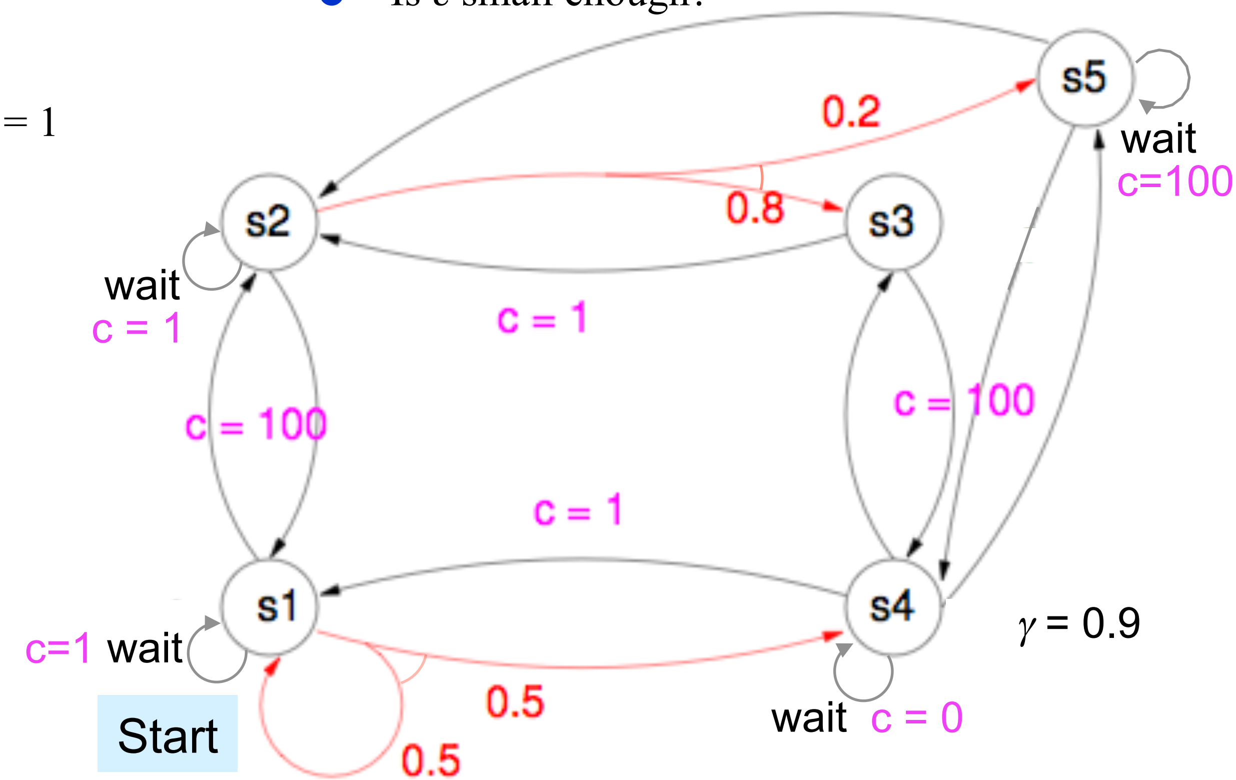
$Q(\text{s3}, a_{32}) = 1 + .9\times 0 = 1$
$Q(\text{s3}, a_{34}) = 100 + .9\times 0 = 100$

$Q(\text{s4}, a_{41}) = 1 + .9\times 0 = 1$
$Q(\text{s4}, a_{43}) = 100 + .9\times 0 = 1$
$Q(\text{s4}, a_{44}) = 0 + .9\times 0 = 0$
$Q(\text{s4}, a_{45}) = 100 + .9\times 0 = 100$

$Q(\text{s5}, a_{52}) = 1 + .9\times 0 = 1$
$Q(\text{s5}, a_{54}) = 100 + .9\times 0 = 100$
$Q(\text{s5}, a_{55}) = 100 + .9\times 0 = 100$

$E_1(\text{s1}) = 1;\quad \pi_1(\text{s1}) = a_{11} = \text{wait}$
$E_1(\text{s2}) = 1;\quad \pi_1(\text{s2}) = a_{22} = \text{wait}$
$E_1(\text{s3}) = 1;\quad \pi(\text{s3}) = a_{32} = \text{move(r1,l3,l2)}$
$E_1(\text{s4}) = 0;\quad \pi_1(\text{s4}) = a_{44} = \text{wait}$
$E_1(\text{s5}) = 1;\quad \pi_1(\text{s3}) = a_{52} = \text{move(r1,l5,l2)}$

- What other actions could we have chosen?
- Is ε small enough?



Discussion

- Policy iteration computes an entire policy in each iteration, and computes values based on that policy
  - More work per iteration, because it needs to solve a set of simultaneous equations
  - Usually converges in a smaller number of iterations
- Value iteration computes new values in each iteration, and chooses a policy based on those values
  - In general, the values are not the values that one would get from the chosen policy or any other policy
  - Less work per iteration, because it doesn’t need to solve a set of equations
  - Usually takes more iterations to converge

Discussion (Continued)

- For both, the number of iterations is polynomial *in the number of states*
  - But the number of states is usually quite large
  - Need to examine the entire state space in each iteration
- Thus, these algorithms can take huge amounts of time and space

- To do a complexity analysis, we need to get explicit about the syntax of the planning problem
  - Can define probabilistic versions of set-theoretic, classical, and state-variable planning problems
  - I will do this for set-theoretic planning

Probabilistic Set-Theoretic Planning

- The statement of a probabilistic set-theoretic planning problem is $P = (S_0, g, A)$
  - $S_0 = \{(s_1, p_1), (s_2, p_2), \ldots, (s_j, p_j)\}$
    - » Every state that has nonzero probability of being the starting state
  - g is the usual set-theoretic goal formula - a set of propositions
  - A is a set of probabilistic set-theoretic actions
    - » Like ordinary set-theoretic actions, but multiple possible outcomes, with a probability for each outcome
    - » a = (name(a), precond(a),

$$\begin{aligned}
&\text{effects}_1^+(a),\ \text{effects}_1^-(a),\ p_1(a),\\
&\text{effects}_2^+(a),\ \text{effects}_2^-(a),\ p_2(a),\\
&\ldots,\\
&\text{effects}_k^+(a),\ \text{effects}_k^-(a),\ p_k(a))
\end{aligned}$$

Probabilistic Set-Theoretic Planning

- Probabilistic set-theoretic planning is EXPTIME-complete
  - Much harder than ordinary set-theoretic planning, which was only PSPACE-complete
- Worst case requires exponential time
- Unknown whether worst case requires exponential space
  - PSPACE ⊆ EXPTIME ⊆ NEXPTIME ⊆ EXPSPACE

- What does this say about the complexity of solving an MDP?

- Value Iteration and Policy Iteration take exponential amounts of time *and* space because they iterate over all states in every iteration
  - In some cases we can do better

Real-Time Value Iteration

- A class of algorithms that work roughly as follows

- loop
  - Forward search from the initial state(s), following the current policy π
    - Each time you visit a new state s, use a heuristic function to estimate its expected cost $E(s)$
    - For every state s along the path followed
      - Update π to choose the action a that minimizes $Q(s,a)$
      - Update $E(s)$ accordingly

- Best-known example: Real-Time Dynamic Programming

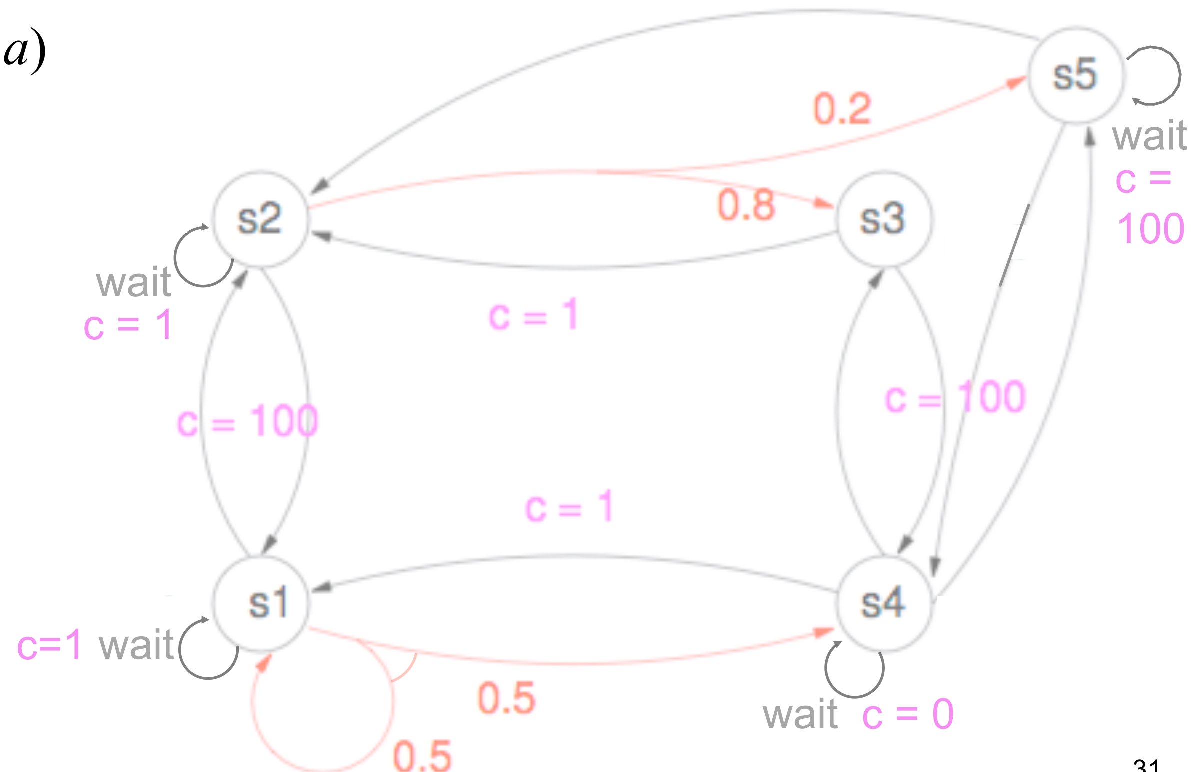
Real-Time Dynamic Programming

- Need explicit goal states
  - If s is a goal, then actions at s have no cost and produce no change
- For each state s, maintain a value $V(s)$ that gets updated as the algorithm proceeds
  - Initially $V(s) = h(s)$, where h is a heuristic function
- **Greedy policy**: $\pi(s) = \arg\min_{a \in A(s)} Q(s,a)$
  - where $Q(s,a) = C(s,a) + \gamma \sum_{s' \in S} P_a(s'|s)\, V(s')$
- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Real-Time Dynamic Programming

- procedure RTDP(s) *(the outer loop on the previous slide)*
  - loop until *termination condition*
    - » RTDP-trial(s)

- procedure RTDP-trial(s) *(the forward search on the previous slide)*
  - while s is not a goal state
    - » a := arg $\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$



Real-Time Dynamic Programming

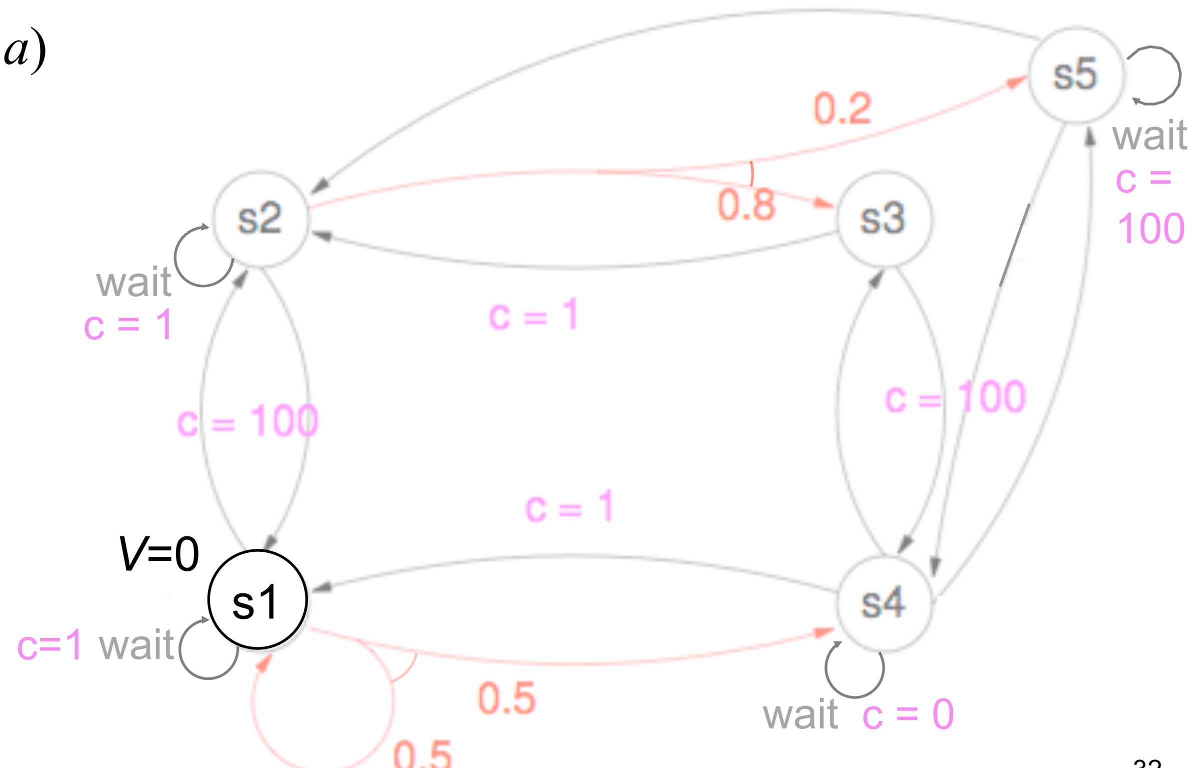
- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)

- procedure RTDP-trial(s)
  - while **s is not a goal state**
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:

$\gamma = 0.9$

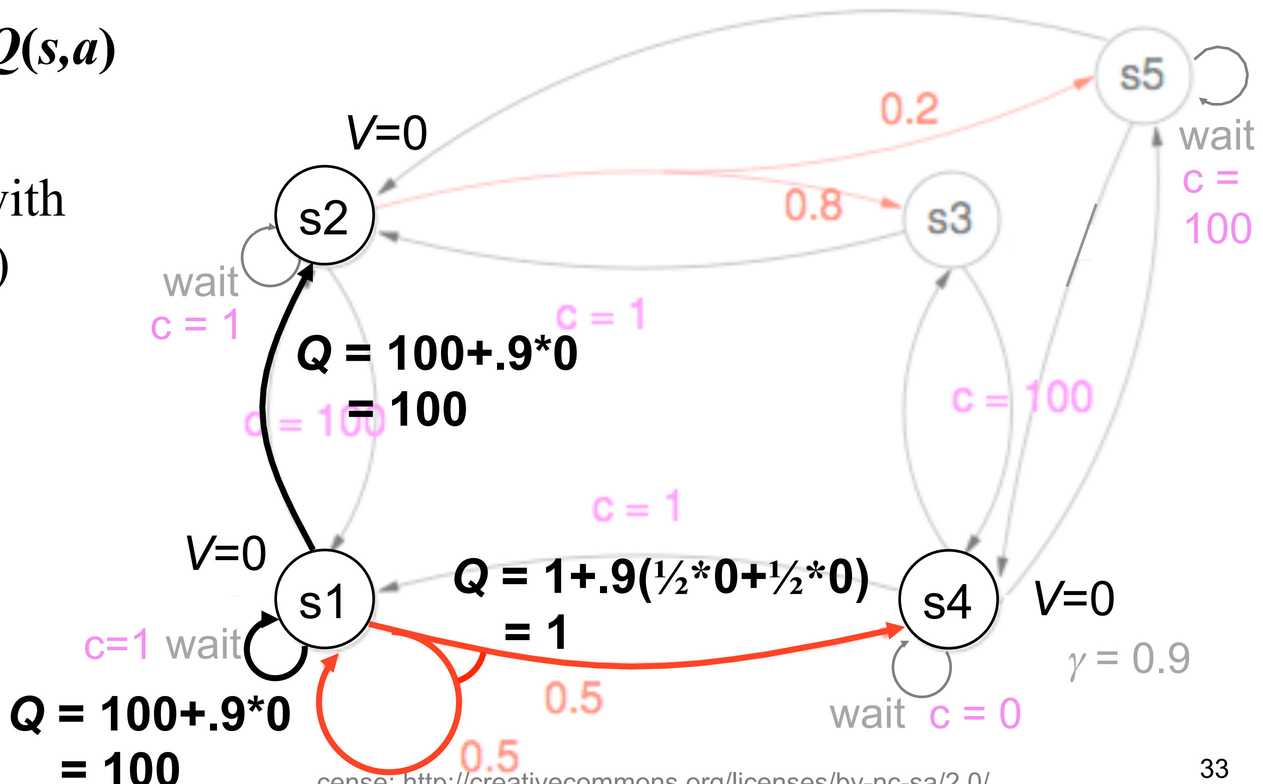
$h(s) = 0$ for all s



Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $a := \arg\min_{a \in A(s)} \boldsymbol{Q(s,a)}$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:
$\gamma = 0.9$
$h(s) = 0$ for all s



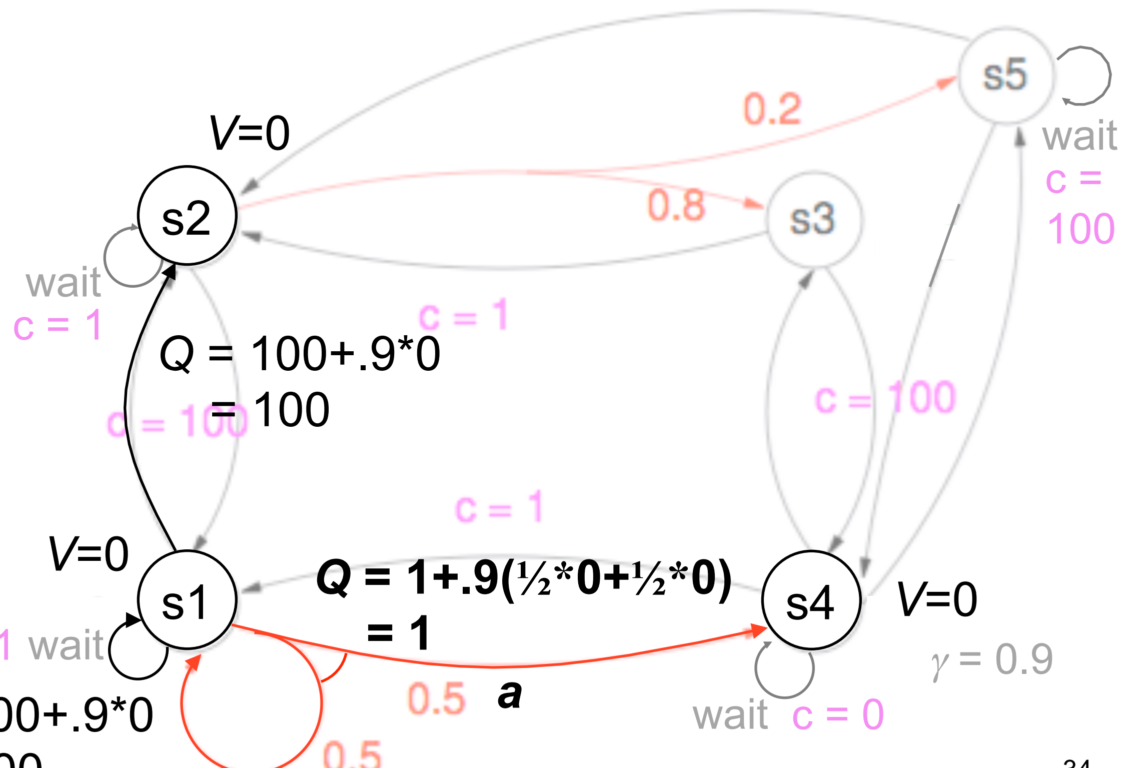
Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $\boldsymbol{a}$ **:= arg min**$_{a \in A(s)}$ $Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:

$\gamma = 0.9$

$h(s) = 0$ for all s



Real-Time Dynamic Programming

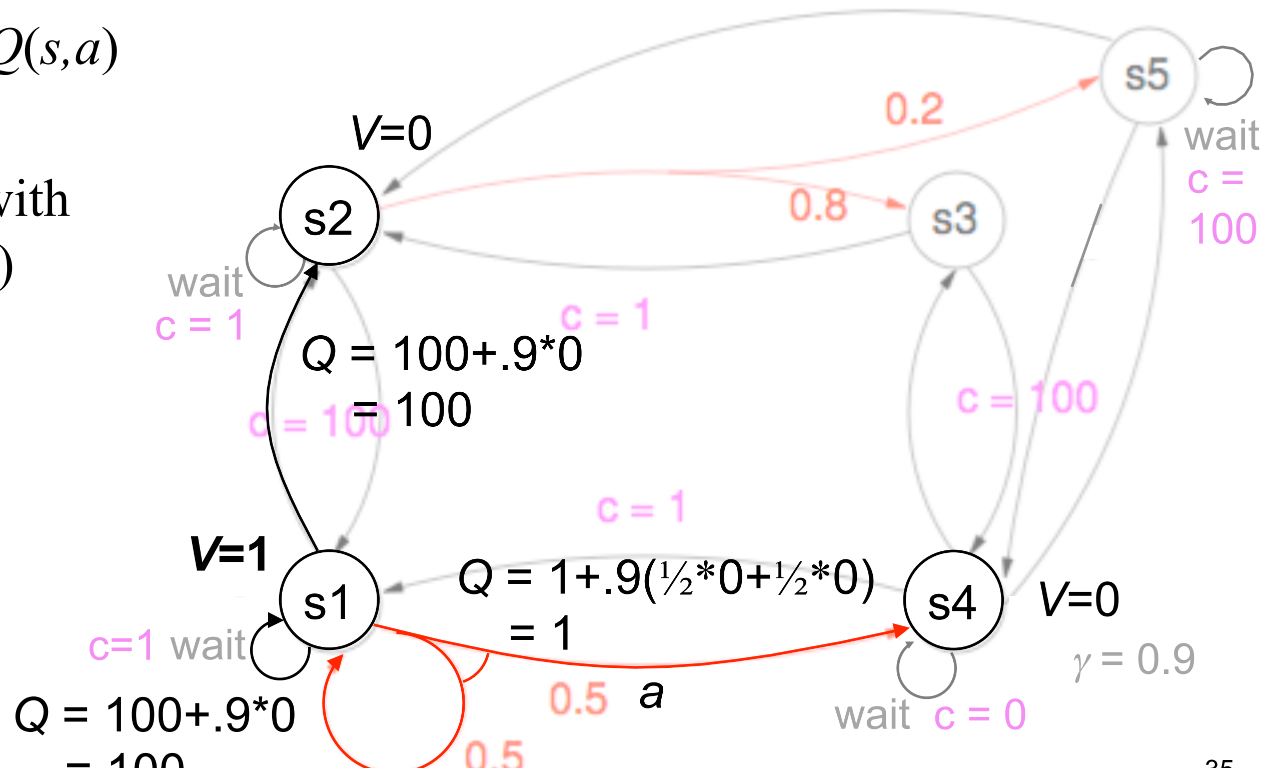
- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)

Example:

γ = 0.9

$h(s)$ = 0 for all s

- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $\mathbf{V(s) := Q(s,a)}$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$



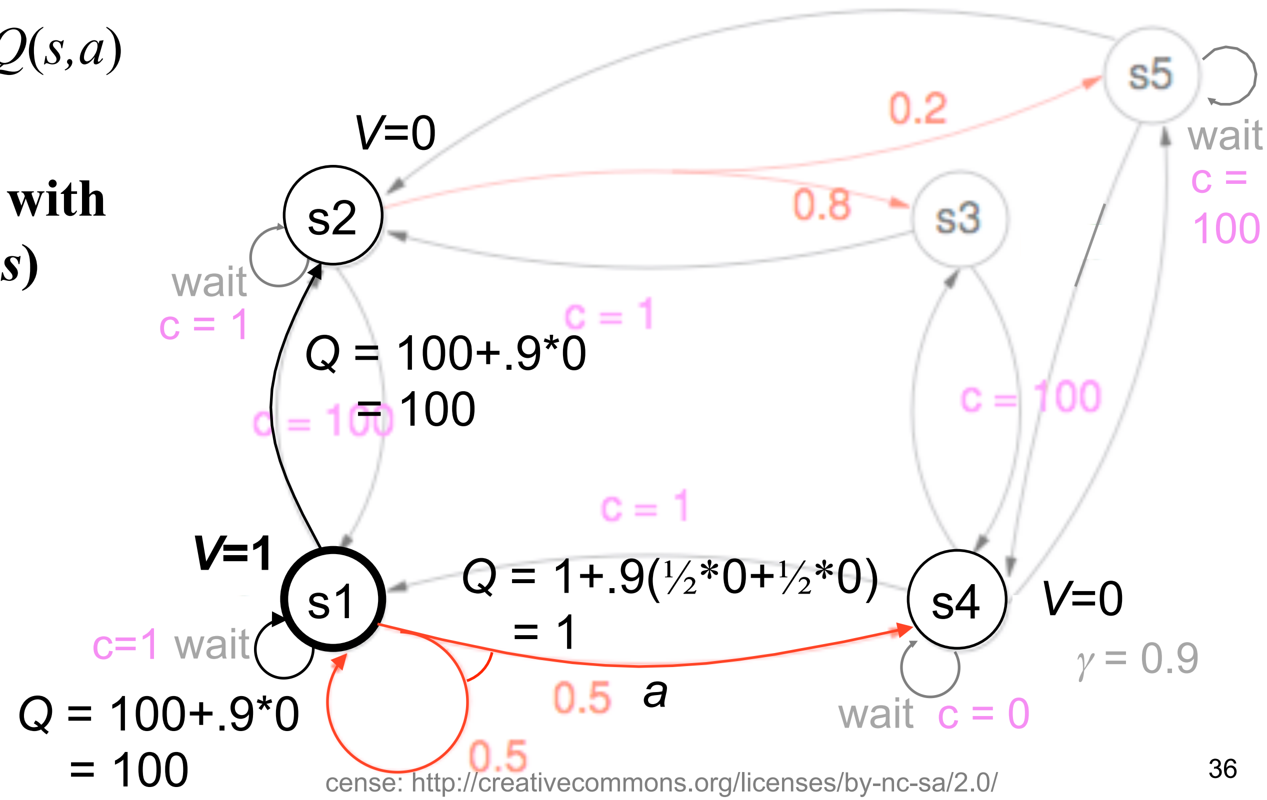
Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while s is not a goal state
    - » a := arg $\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » **randomly pick s' with probability $P_a(s'|s)$**
    - » $s := s'$

Example:

$\gamma = 0.9$

$h(s) = 0$ for all s



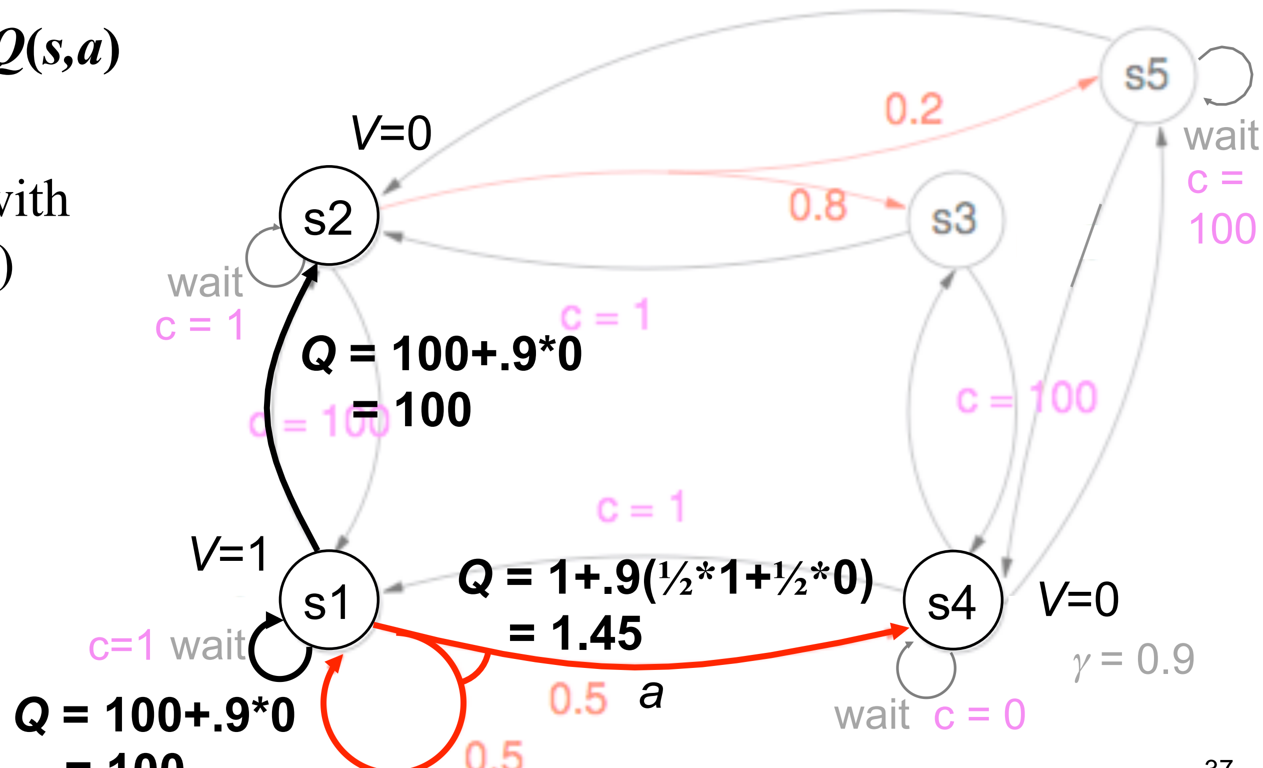
Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while **s is not a goal state**
    - » a := arg min$_{a \in A(s)}$ $\boldsymbol{Q(s,a)}$
    - » $V(s)$:= $Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » s := s'

Example:

γ = 0.9

$h(s)$ = 0 for all s

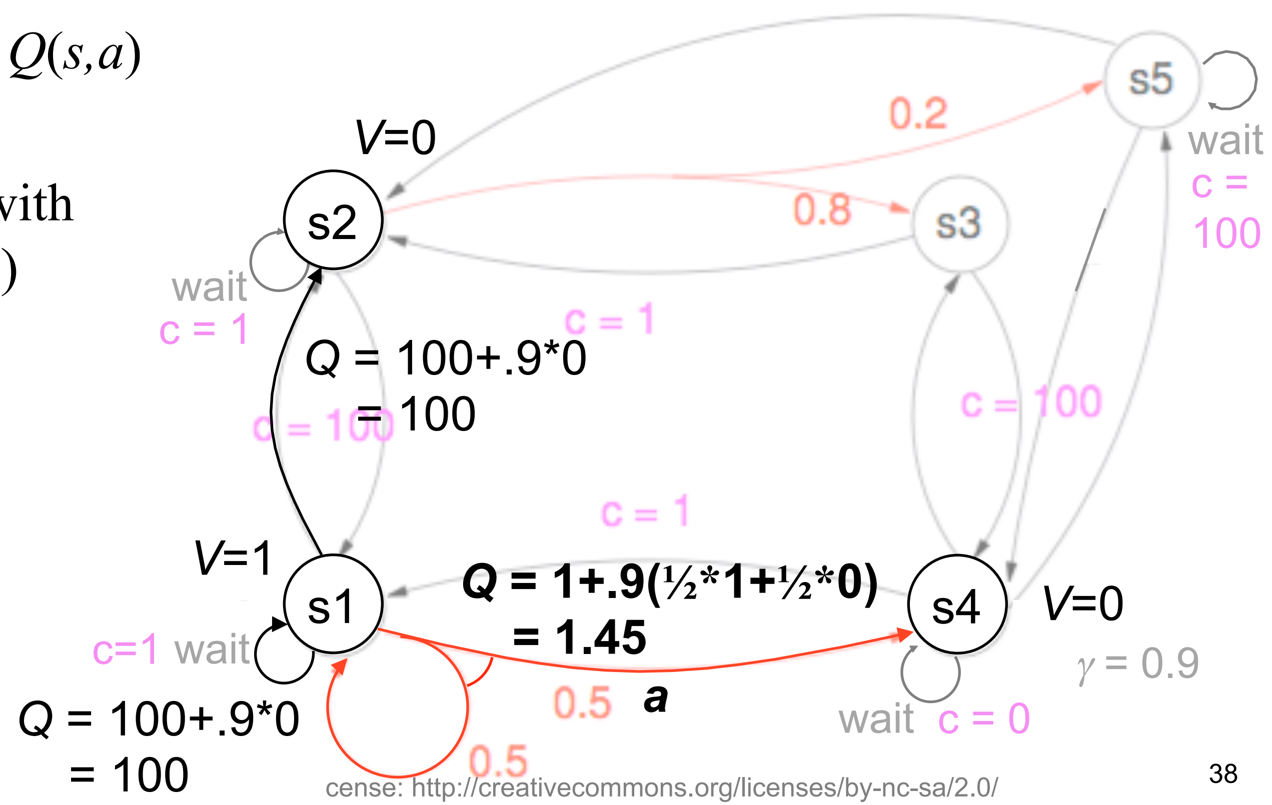


Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)

- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $\boldsymbol{a} := \textbf{arg min}_{a \in A(s)}\, Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:
$\gamma = 0.9$
$h(s) = 0$ for all s

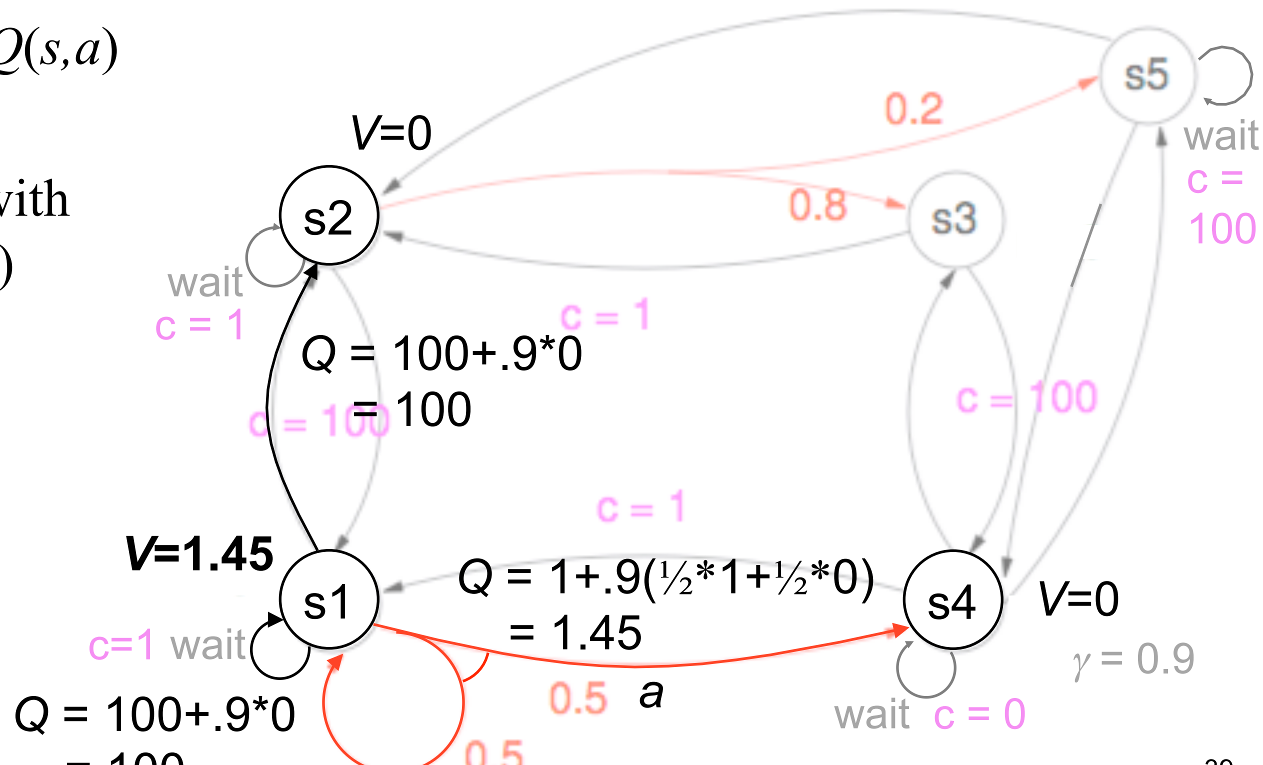


Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)

- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $\boldsymbol{V(s) := Q(s,a)}$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:
$\gamma = 0.9$
$h(s) = 0$ for all s



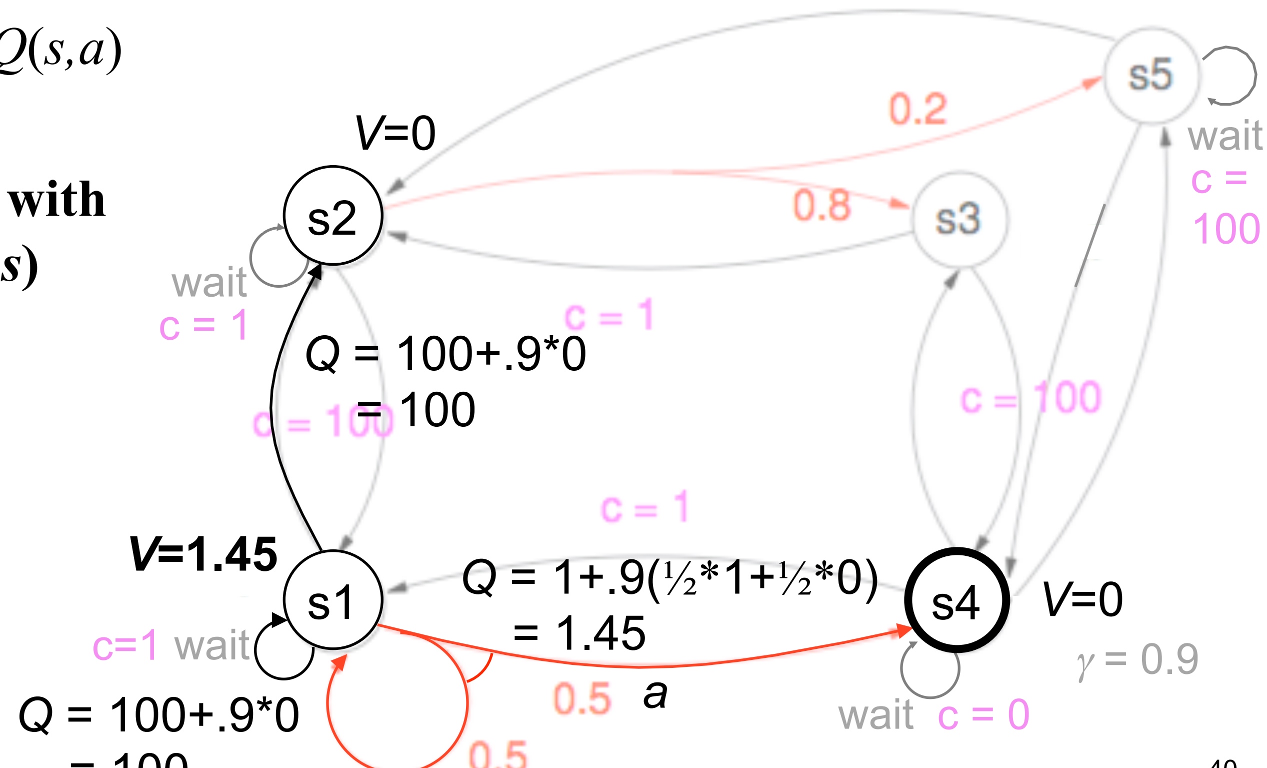
Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » **randomly pick s' with probability $P_a(s'|s)$**
    - » $s := s'$

Example:

$\gamma = 0.9$

$h(s) = 0$ for all s



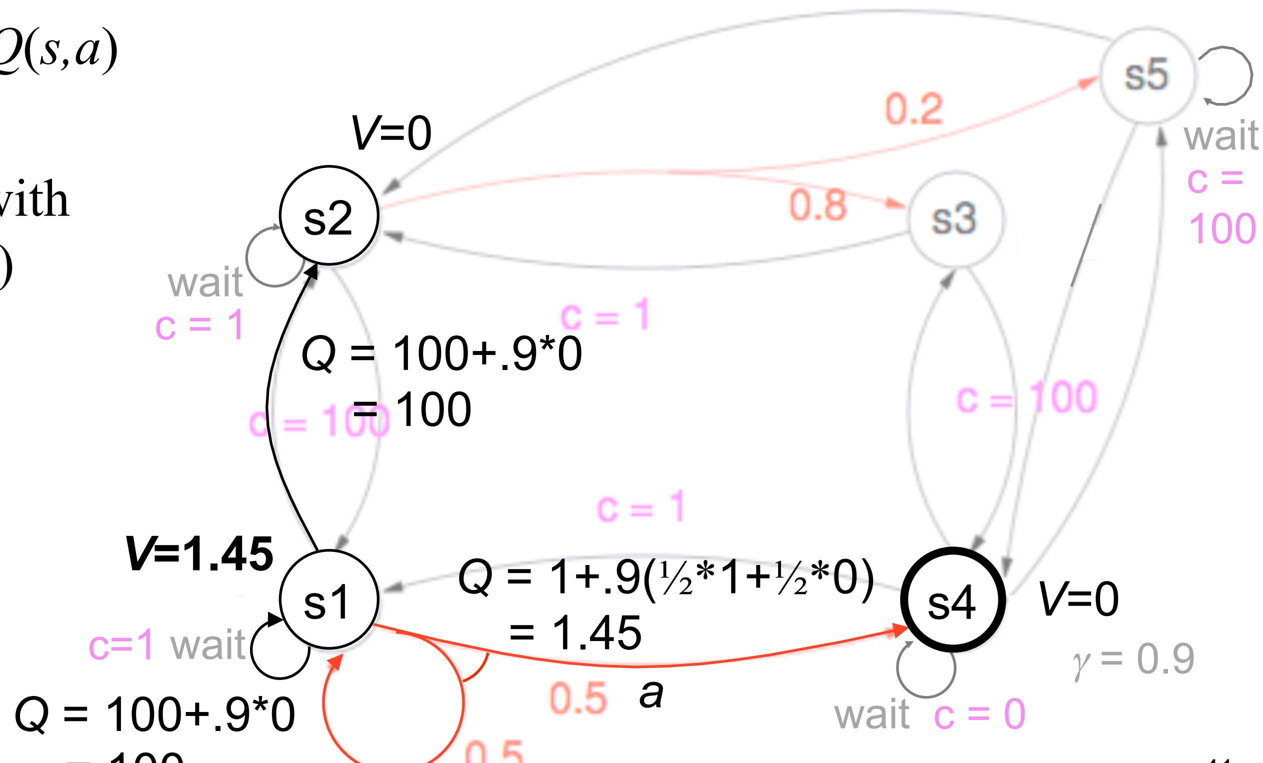
Real-Time Dynamic Programming

- procedure RTDP(s)
  - loop until *termination condition*
    - » RTDP-trial(s)
- procedure RTDP-trial(s)
  - while **s is ~~not~~ a goal state**
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:

$\gamma = 0.9$

$h(s) = 0$ for all s



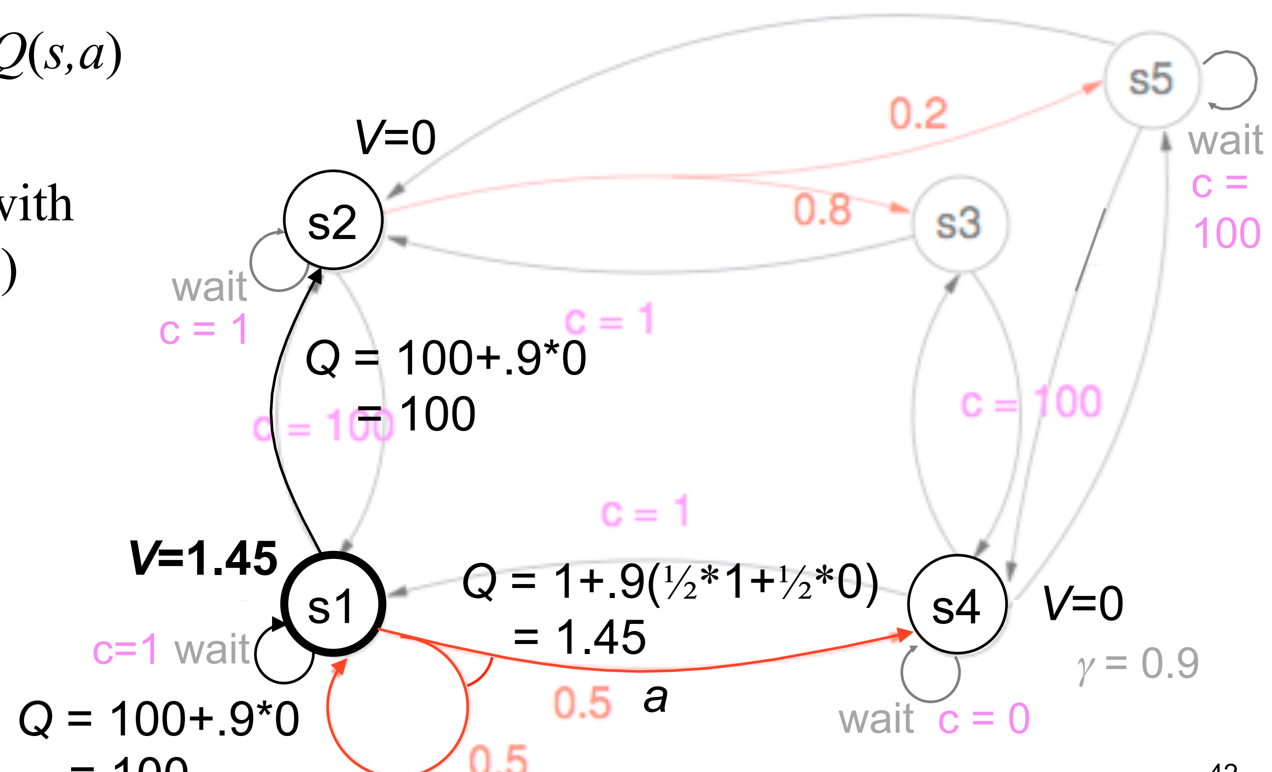
Real-Time Dynamic Programming

- procedure RTDP(s)
  - **loop until *termination condition***
    - » **RTDP-trial(s)**
- procedure RTDP-trial(s)
  - while s is not a goal state
    - » $a := \arg\min_{a \in A(s)} Q(s,a)$
    - » $V(s) := Q(s,a)$
    - » randomly pick s' with probability $P_a(s'|s)$
    - » $s := s'$

Example:

$\gamma = 0.9$

$h(s) = 0$ for all s



Real-Time Dynamic Programming

- In practice, it can solve much larger problems than policy iteration and value iteration
- Won't always find an optimal solution, won't always terminate
  - If h doesn't overestimate, and if a goal is reachable (with positive probability) at every state
    - Then it will terminate
  - If in addition to the above, there is a positive-probability path between every pair of states
    - Then it will find an optimal solution

POMDPs

- *Partially observable Markov Decision Process (POMDP)*:
  - a stochastic system $\Sigma = (S, A, P)$ as defined earlier
  - A finite set O of *observations*
    - » $P_a(o|s)$ = probability of observation o after executing action a in state s
  - Require that for each a and s, $\sum_{o \in O} P_a(o|s) = 1$
- O models partial observability
  - The controller can't observe s directly; it can only do a then observe o
  - The same observation o can occur in more than one state
- Why do the observations depend on the action a?
  - » Why do we have $P_a(o|s)$ rather than $P(o|s)$?

POMDPs

- *Partially observable Markov Decision Process (POMDP)*:
  - a stochastic system $\Sigma = (S, A, P)$ as defined earlier
    - » $P_a(s'|s)$ = probability of being in state s' after executing action a in state s
  - A finite set O of *observations*
    - » $P_a(o|s)$ = probability of observation o after executing action a in state s
  - Require that for each a and s, $\sum_{o \in O} P_a(o|s) = 1$

- O models partial observability
  - The controller can't observe s directly; it can only do a then observe o
  - The same observation o can occur in more than one state

- Why do the observations depend on the action a?
    - » Why do we have $P_a(o|s)$ rather than $P(o|s)$?
  - This is a way to model *sensing actions*
    - » e.g., a is the action of obtaining observation o from a sensor

More about Sensing Actions

- Suppose a is an action that never changes the state
  - $P_a(s|s) = 1$ for all s
- Suppose there are a state s and an observation o such that a gives us observation o iff we're in state s
  - $P_a(o|s) = 0$ for all $s' \neq s$
  - $P_a(o|s) = 1$
- Then to tell if you're in state s, just perform action a and see whether you observe o

- Two states s and s' are *indistinguishable* if for every o and a, $P_a(o|s) = P_a(o|s')$

Belief States

- At each point we will have a probability distribution $b(s)$ over the states in S
  - b is called a *belief state*
  - Our current belief about what state we're in
- Basic properties:
  - $0 \leq b(s) \leq 1$ for every s in S
  - $\sum_{s \in S} b(s) = 1$
- Definitions:
  - $b_a =$ the belief state after doing action a in belief state b
    - » $b_a(s) = P(\text{we're in } s \text{ after doing } a \text{ in } b) = \sum_{s' \in S} P_a(s|s')\, b(s')$
  - $b_a(o) = P(\text{observe } o \text{ after doing } a \text{ in } b) = \sum_{s' \in S} P_a(o|s')\, b(s')$
  - $b_a^o(s) = P(\text{we're in } s \mid \text{we observe } o \text{ after doing } a \text{ in } b)$

Belief States (Continued)

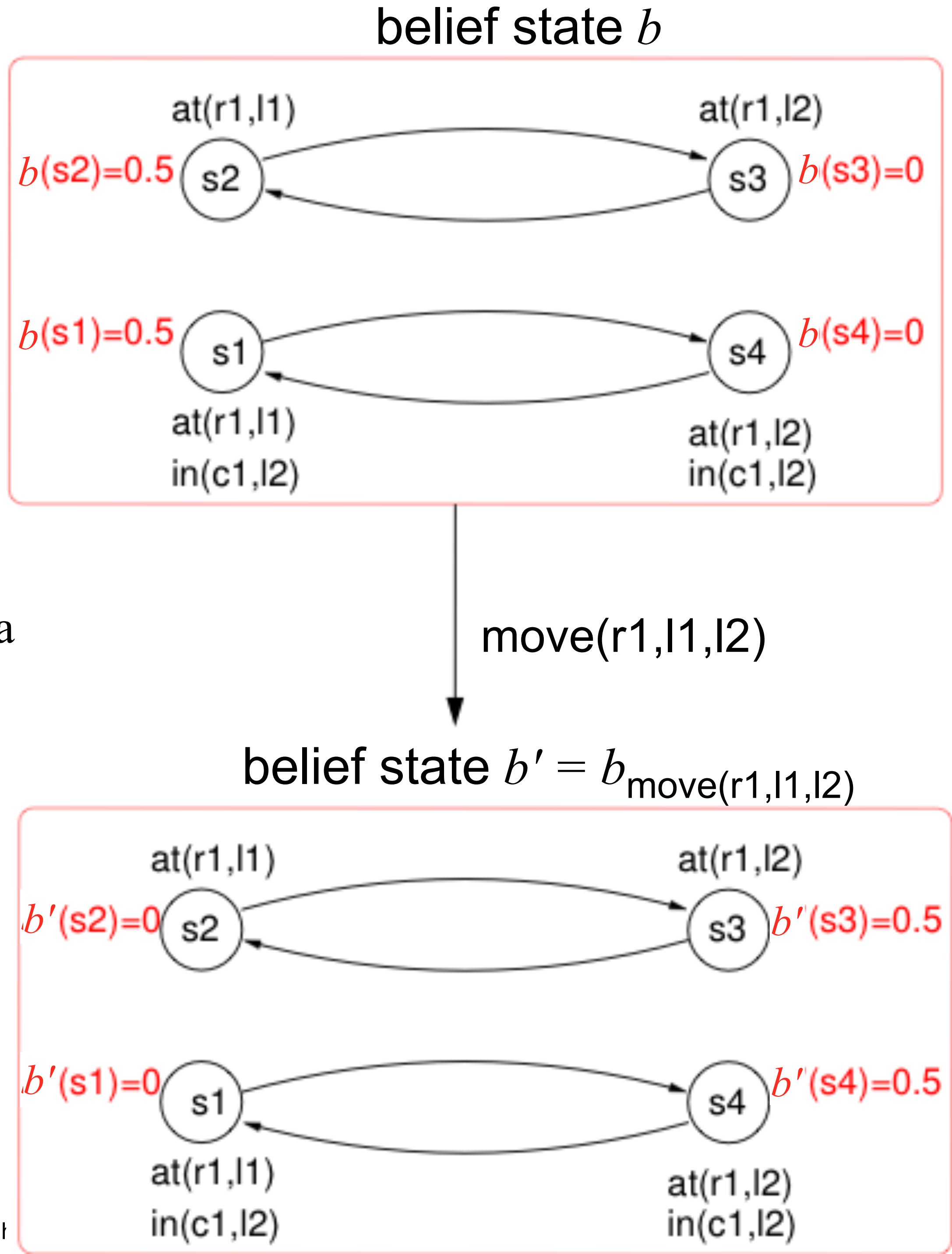
- According to the book,
  - $b_a^o(s) = P_a(o|s)\ b_a(s)\ /\ b_a(o)$ (16.14)

- I'm not completely sure whether that formula is correct
- But using it (possibly with corrections) to distinguish states that would otherwise be indistinguishable
  - Example on next page

Example

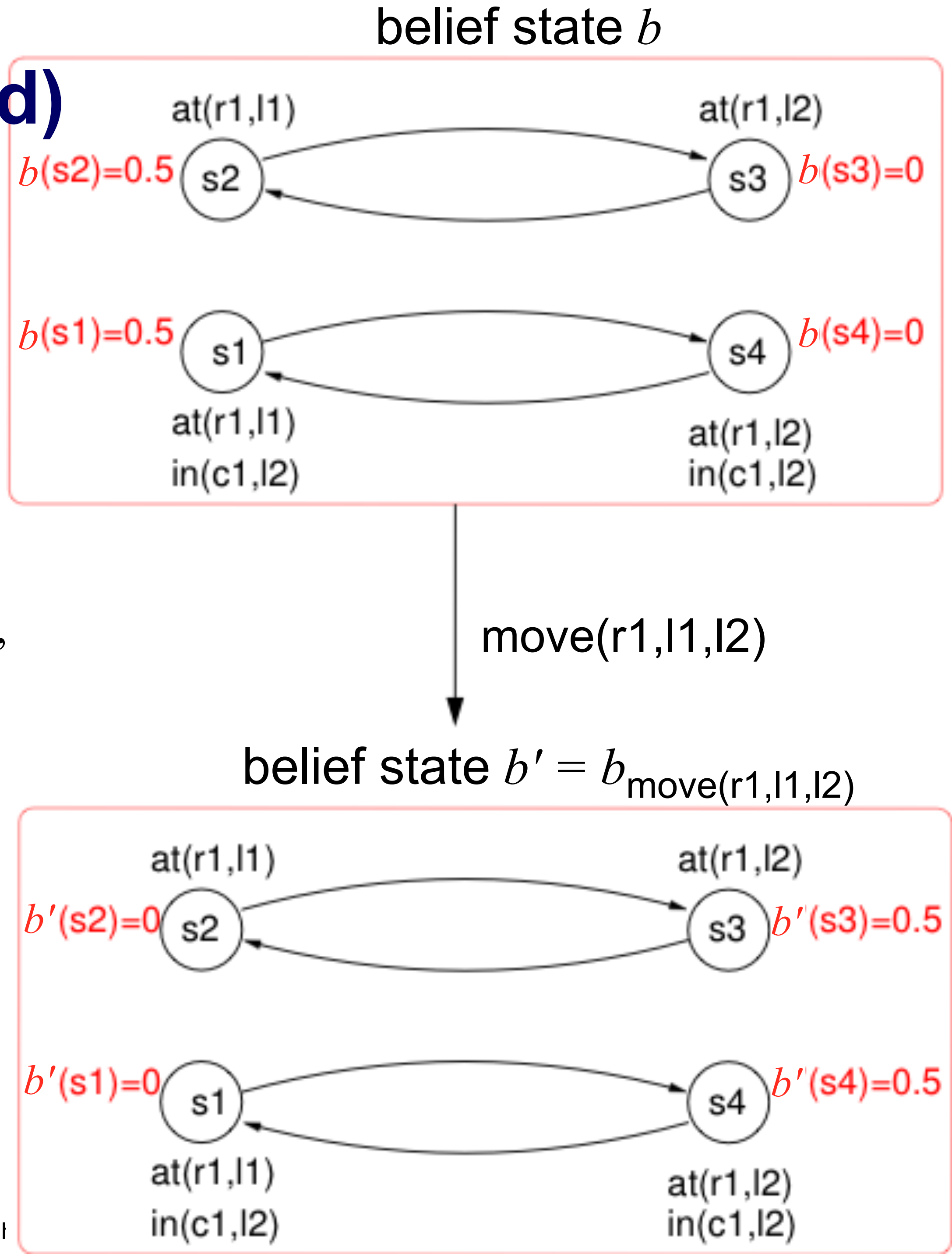
- Modified version of DWR
- Robot r1 can move between l1 and l2
    - » move(r1,l1,l2)
    - » move(r1,l2,l1)

- ◆ With probability 0.5, there's a container c1 in location l2
    - » in(c1,l2)
- ◆ O = {full, empty}
    - » full: c1 is present
    - » empty: c1 is absent
    - » abbreviate full as f, and empty as e



Example (Continued)

- move doesn't return a useful observation
- For every state s and for move action a,
  - $P_a(\mathsf{f}|s) = P_a(\mathsf{e}|s) =$ $P_a(\mathsf{f}|s) = P_a(\mathsf{e}|s) = 0.5$
- Thus if there are no other actions, then
  - s1 and s2 are indistinguishable
  - s3 and s4 are indistinguishable



Example (Continued)

- Suppose there's a sensing action see that works perfectly in location l2

  $P_{\text{see}}(\text{f}|\text{s4}) = P_{\text{see}}(\text{e}|\text{s3}) = 1$

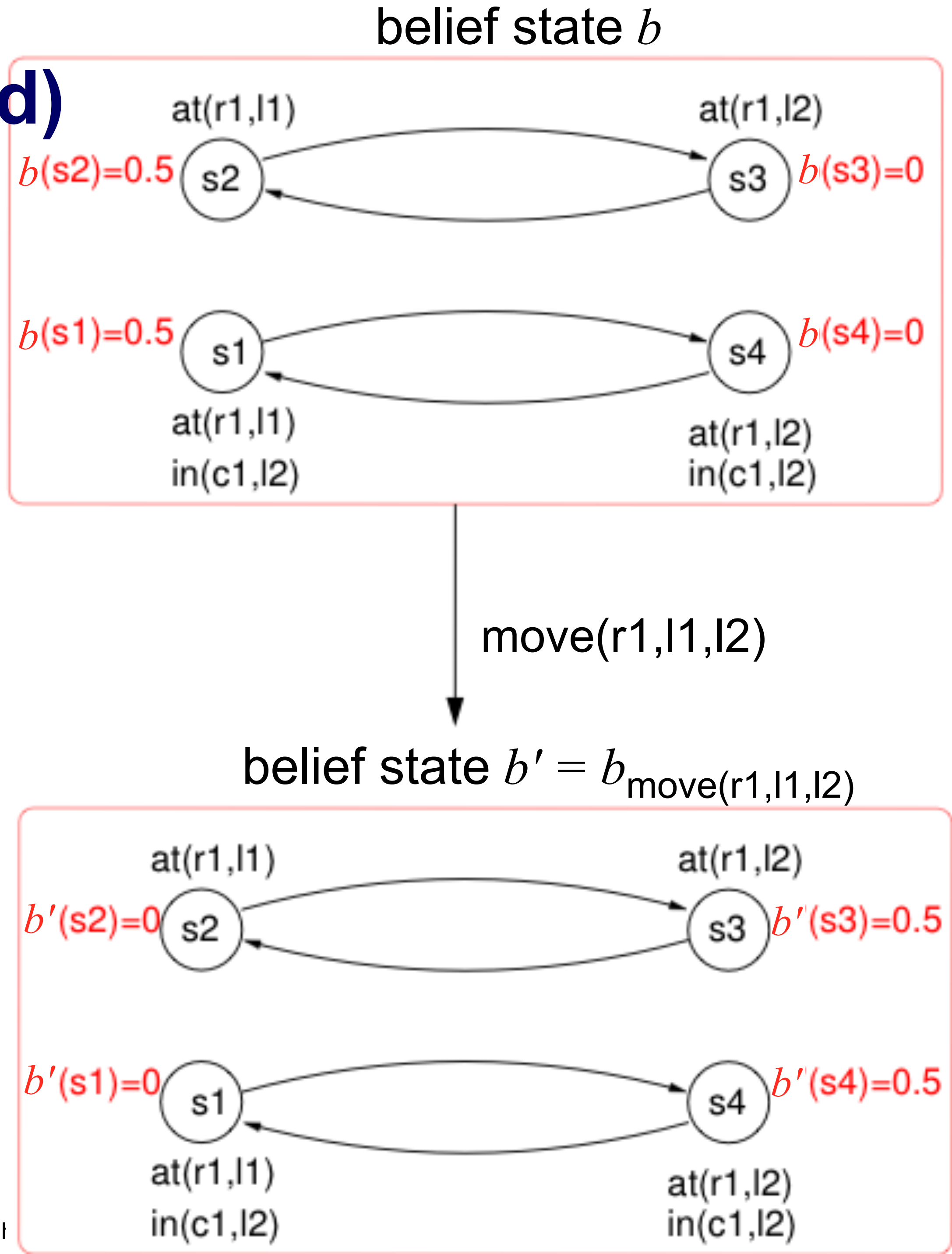
  $P_{\text{see}}(\text{f}|\text{s3}) = P_{\text{see}}(\text{e}|\text{s4}) = 0$

  - Then s3 and s4 are distinguishable

  - Suppose see doesn't work elsewhere

  $P_{\text{see}}(\text{f}|\text{s1}) = P_{\text{see}}(\text{e}|\text{s1}) = 0.5$

  $P_{\text{see}}(\text{f}|\text{s2}) = P_{\text{see}}(\text{e}|\text{s2}) = 0.5$



Example (Continued)

- In b, see doesn't help us any

$b_{see}^{e}(s1)$

$= P_{see}(e|s1)\ b_{see}(s1) / b_{see}(e)$

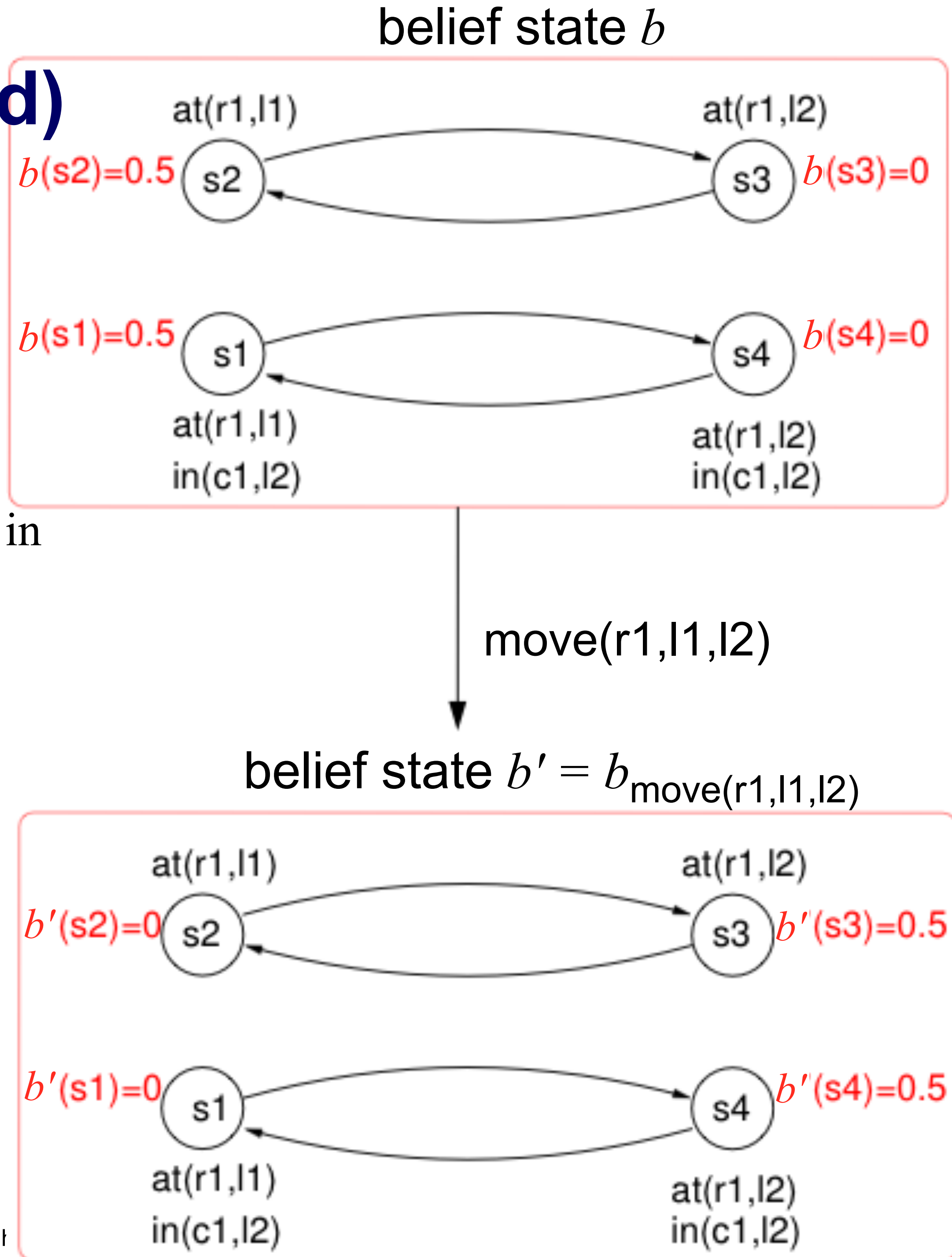
$= 0.5 \cdot 0.5 / 0.5 = 0.5$

- In b', see tells us what state we're in

$b'^{e}_{see}(s3)$

$= P_{see}(e|s3)\ \ b'_{see}(s3) / b'_{see}(e)$

$= 1 \cdot 0.5 / 0.5 = 1$

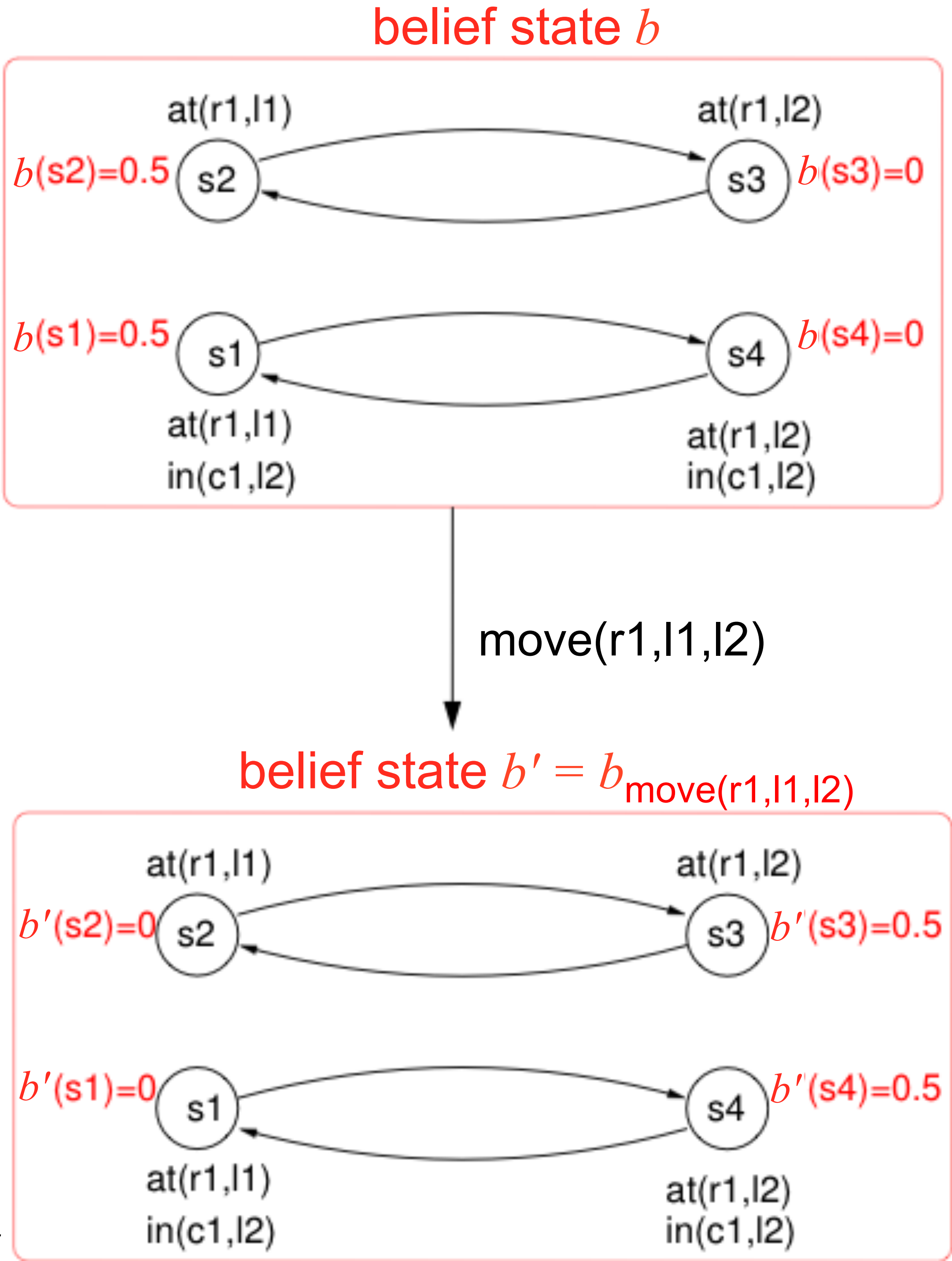


Policies on Belief States

- In a fully observable MDP, a policy is a partial function from S into A
- In a partially observable MDP, a policy is a partial function from B into A
  - where B is the set of all belief states
- S was finite, but B is infinite and continuous
  - A policy may be either finite or infinite

Example

- Suppose we know the initial belief state is b
- Policy to tell if there's a container in l2:
  - $\pi = \{(b,$ move(r1,l1,l2)), $(b',$ see)$\}$



Planning Algorithms

- POMDPs are very hard to solve
- The book says very little about it
- I'll say even less!

Reachability and Extended Goals

- The usual definition of MDPs does not contain explicit goals
  - Can get the same effect by using *absorbing* states
- Can also handle problems where there the objective is more general, such as maintaining some state rather than just reaching it
- DWR example: whenever a ship delivers cargo to l1, move it to l2
  - Encode ship's deliveries as nondeterministic outcomes of the robot's actions