Markov Decision Process, Value Iteration, and Reinforcement Learning

Anna Helena Reali Costa

Lecture slides partially extracted from:
http://aima.eecs.berkeley.edu/instructors.html
http://ocw.mit.edu/ (course 6.825)
http://www.laas.fr/planning/
Chapter 13: Machine Learning, Tom Mitchell

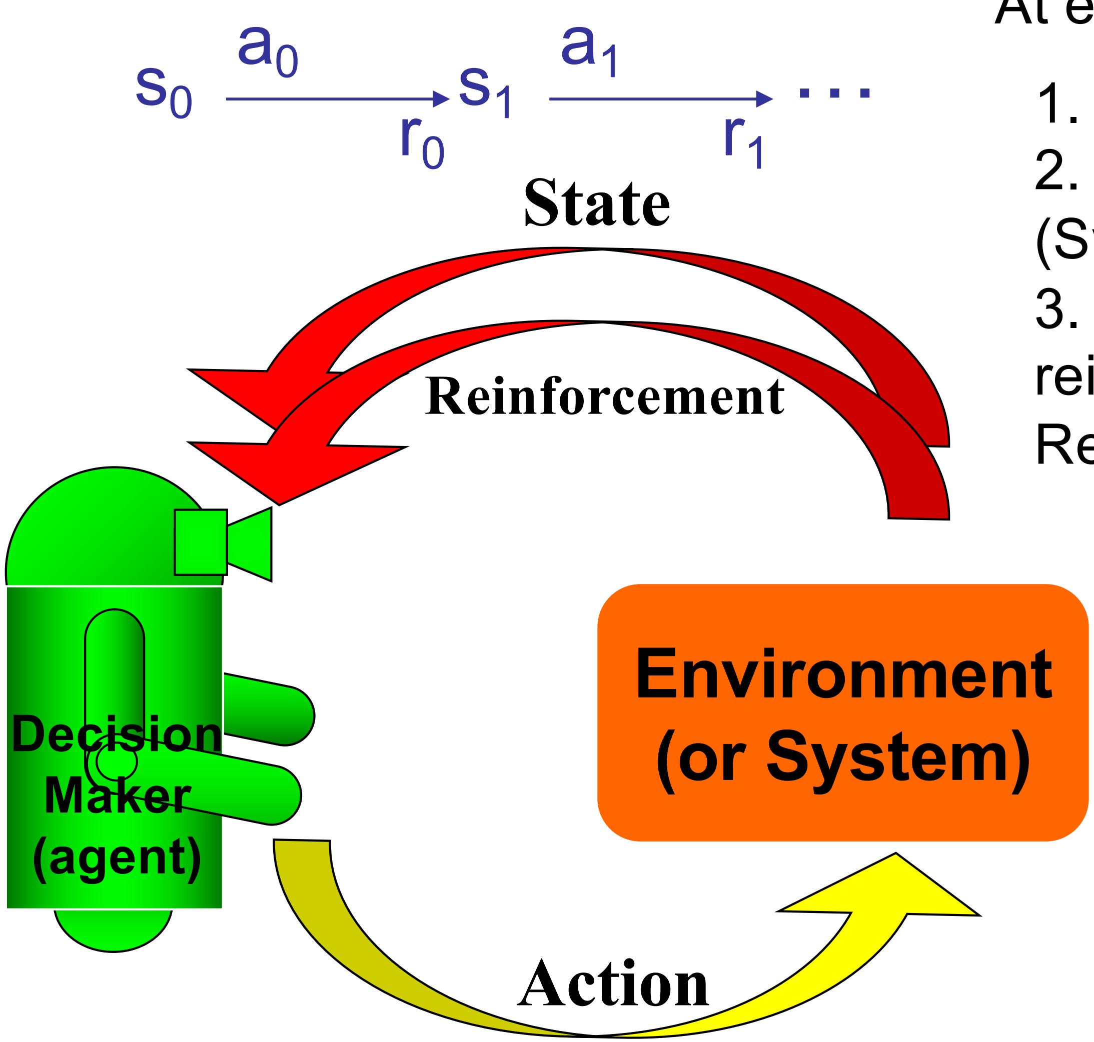
Characteristics of the RL problem

- The agent has a set of **sensors** to observe the ***state*** of its environment
- The agent has a set of ***actions*** it can perform to alter this state
- The agent perceives a **reward** (or penalty) to indicate the desirability of the resulting state

❖ The task of the agent is to learn from this indirect, delayed reward, to choose **sequences of actions** that produce the greatest cumulative reward

➔ it is a ***sequential decision problem***

Sequential decision problem

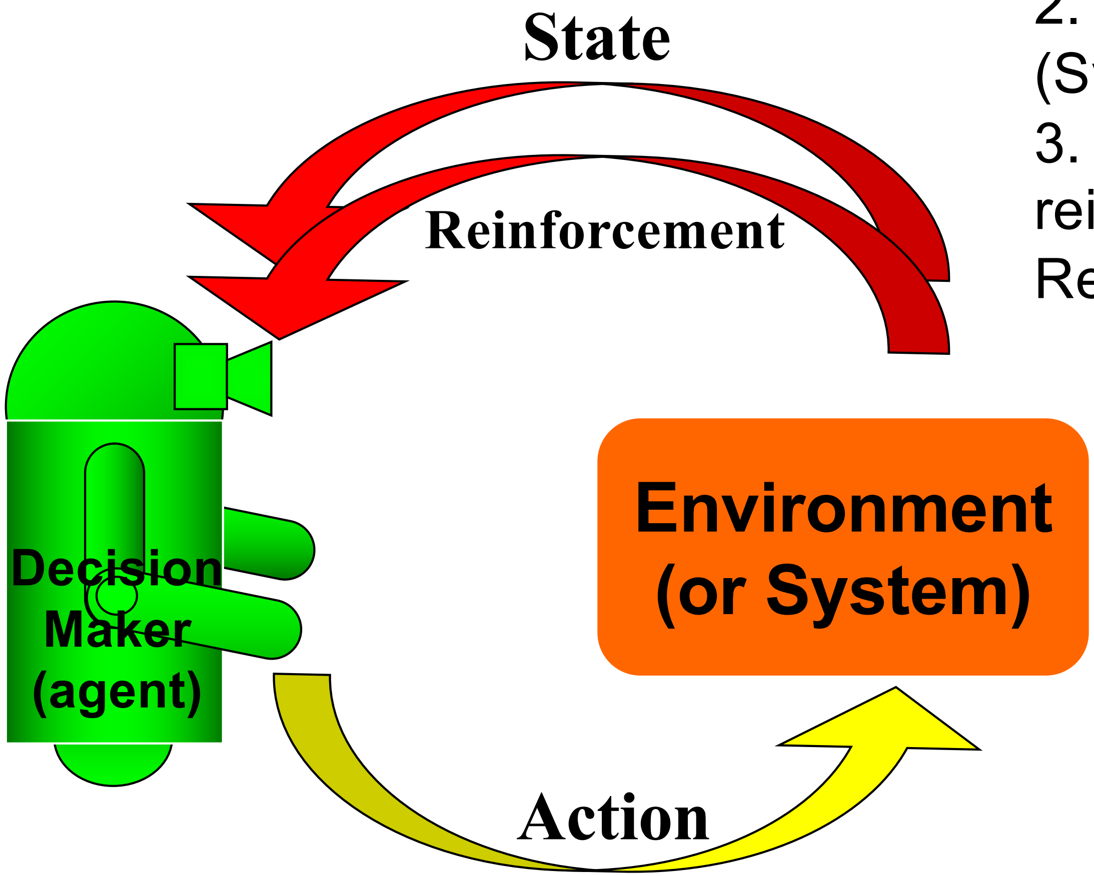


At each time step the decision maker:

1. Observes the state of the system;
2. Chooses an action and applies it;
(System evolves to a new state)
3. Observes an immediate reinforcement (reward or penalty);
Repeat 1 – 3

Sequential decision problem

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} \cdots$$



At each time step the decision maker:

1. Observes the state of the system;
2. Chooses an action and applies it;
(System evolves to a new state)
3. Observes an immediate reinforcement;
Repeat 1 – 3

This assumes discrete time.

Decisions are made at points of time referred to as ***decision epochs.***

The set of decision epochs can be **finite or infinite**: $T = \{0,1,2,\ldots,N\}$, $N \leq \infty$.

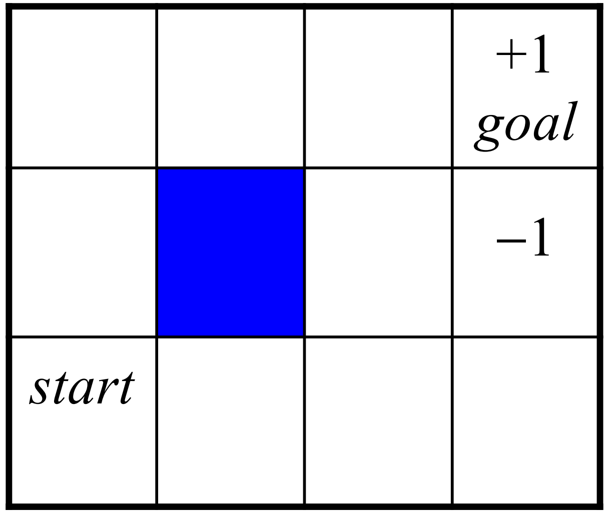
Example

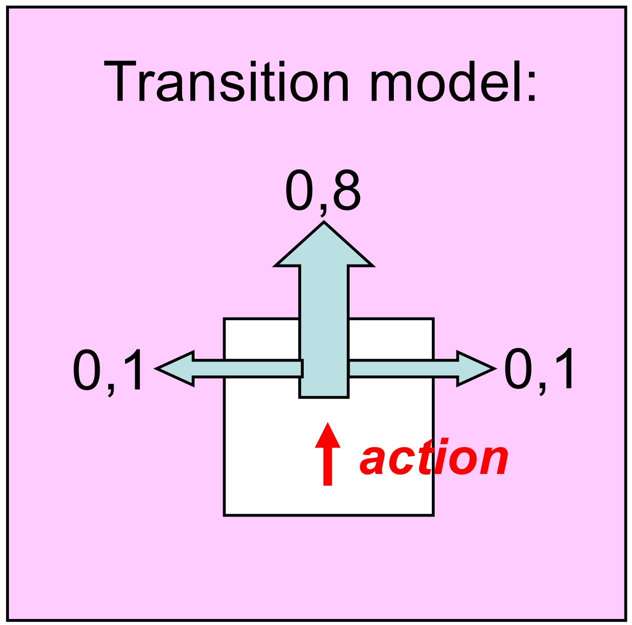
- 4×3 discrete fully-observed environment
- Actions: Up, Down, Left and Right
- Initial state: (1,1)
- Sequence **[U, U, R, R, R]**:

  (i) goes up around the barrier and reaches the goal state (4,3) with probability

  ………

  (ii) there is also a chance of accidentally reaching the goal by going **(1,1)** → **(2,1)** → **(3,1)** → **(3,2)** → **(3,3)** → **(4,3)** with probability ………





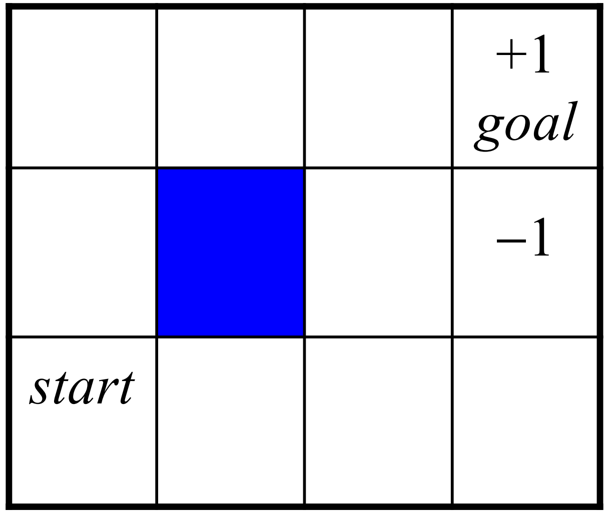
Example

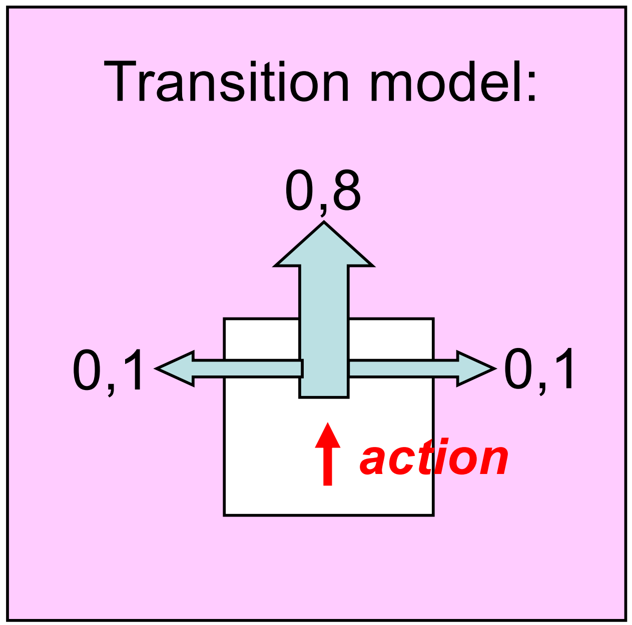
- 4×3 discrete fully-observed environment
- Actions: Up, Down, Left and Right
- Initial state: (1,1)
- Sequence **[U, U, R, R, R]**:

(i) goes up around the barrier and reaches the goal state (3,4) with probability … $\mathbf{0{,}8^5 = 0.32768.}$

(ii) there is also a chance of accidentally reaching the goal by going **(1,1) → (2,1) → (3,1) → (3,2) → (3,3) → (4,3)** with probability

$$\mathbf{0{,}1^4 \times 0{,}8 = 0.00008}$$





Utility function

- **Utility function** for the agent depends on a sequence of states (environment ***history***)
- In each state **s**, the agent receives a **reinforcement r(s)**, which may be positive or negative, but must be **bounded.**
- Utility = sum of the rewards received

Example

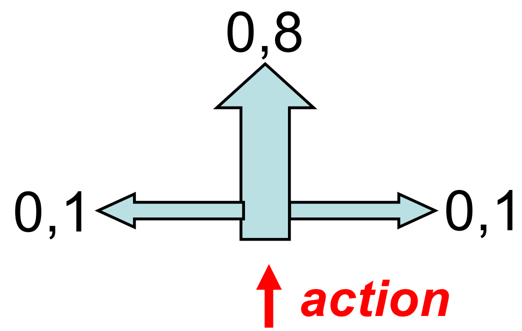
- **Utility function** for the agent depends on a sequence of states (environment ***history***)
- In each state **s**, the agent receives a **reinforcement r(s)**, which may be positive or negative, but must be **bounded.**
- Utility = sum of the rewards received
- Here: **r(s) = –0.04** ∀s except
  r(4,3) = +1 and r(4,2) = −1

➔ reach goal after 10 steps (avoiding (4,2)):

utility = ……………………………

-0,04	-0,04	-0,04	**+1** *goal*
-0,04		-0,04	**–1**
-0,04 *start*	-0,04	-0,04	-0,04

Transition model:



Ex: (1,1),U,(1,2),D,(1,1),U,(1,2),U,(1,3),L,(1,3),R,(2,3),R,(3,3),L,(2,3),R,(3,3),R,(4,3)

Example

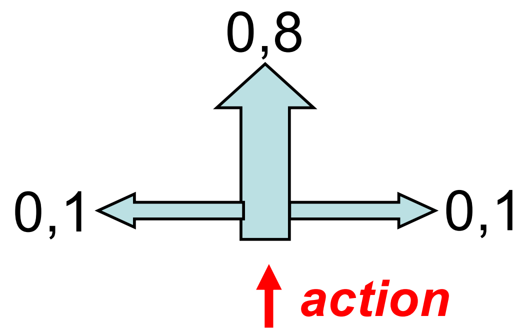
- **Utility function** for the agent depends on a sequence of states (environment ***history***)
- In each state **s**, the agent receives a **reinforcement r(s)**, which may be positive or negative, but must be **bounded.**
- Utility = sum of the rewards received
- Here: **r(s) = –0.04** $\forall$s except
  r(4,3) = +1 and r(4,2) = −1

➔ reach goal after 10 steps (avoiding (4,2)):

utility = **10 × −0,04 + 1 = 0,6**

-0,04	-0,04	-0,04	**+1** *goal*
-0,04		-0,04	**–1**
-0,04 *start*	-0,04	-0,04	-0,04

Transition model:



Ex: (1,1),U,(1,2),D,(1,1),U,(1,2),U,(1,3),L,(1,3),R,(2,3),R,(3,3),L,(2,3),R,(3,3),R,(4,3)

Modeling sequential decision problems as Markov Decision Processes

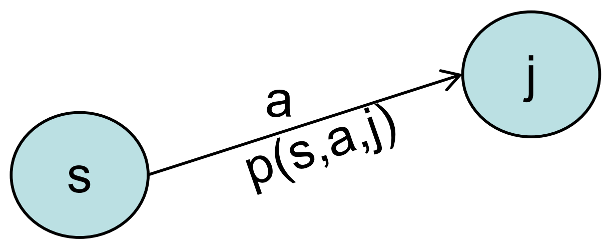
MDP – Model Formulation

An MDP is defined as $\langle S, A, p, r \rangle$:

- **S** is the set of possible system states (arbitrary finite set);
- **A** is the set of allowable actions (arbitrary finite set);
- **p**: $S \times A \times S \rightarrow [0,1]$ is the **transition probability function**;
- **r**: $S \times A \rightarrow \Re$ is the **reinforcement function**;

MDP

- The set **A** of allowable actions:
  - $A = \cup_{s \in S} A_s$ where A_s is the set of allowable actions in state $s \in S$
  - Or we might restrict the model: $A = A_s$ for all $s \in S$
- The transition probability function **p**:
  - $p(j \mid s, a)$ --- or $p(s, a, j)$ --- denotes the probability that the system is in state $j \in S$ at time $t+1$, when the decision maker performs action $a \in A_s$ in state $s \in S$ at time t.
  - $\sum_{j \in S} p(j \mid s, a) = 1$



MDP

- The reinforcement function **r:**
  - r(s,a), denotes the value of the reward (or reinforcement or cost) received when performing $a \in A_s$ in $s \in S$ at time t.
  - When positive, r(s,a) is an *income*, and when negative it is a ***cost***.
  - Can be:

    r(s);

    r(s,a);

    r(s,a,j) with $r(s,a) = \sum_{j \in S} r(s,a,j)\, p(j \mid s, a)$

Why "Markov"?

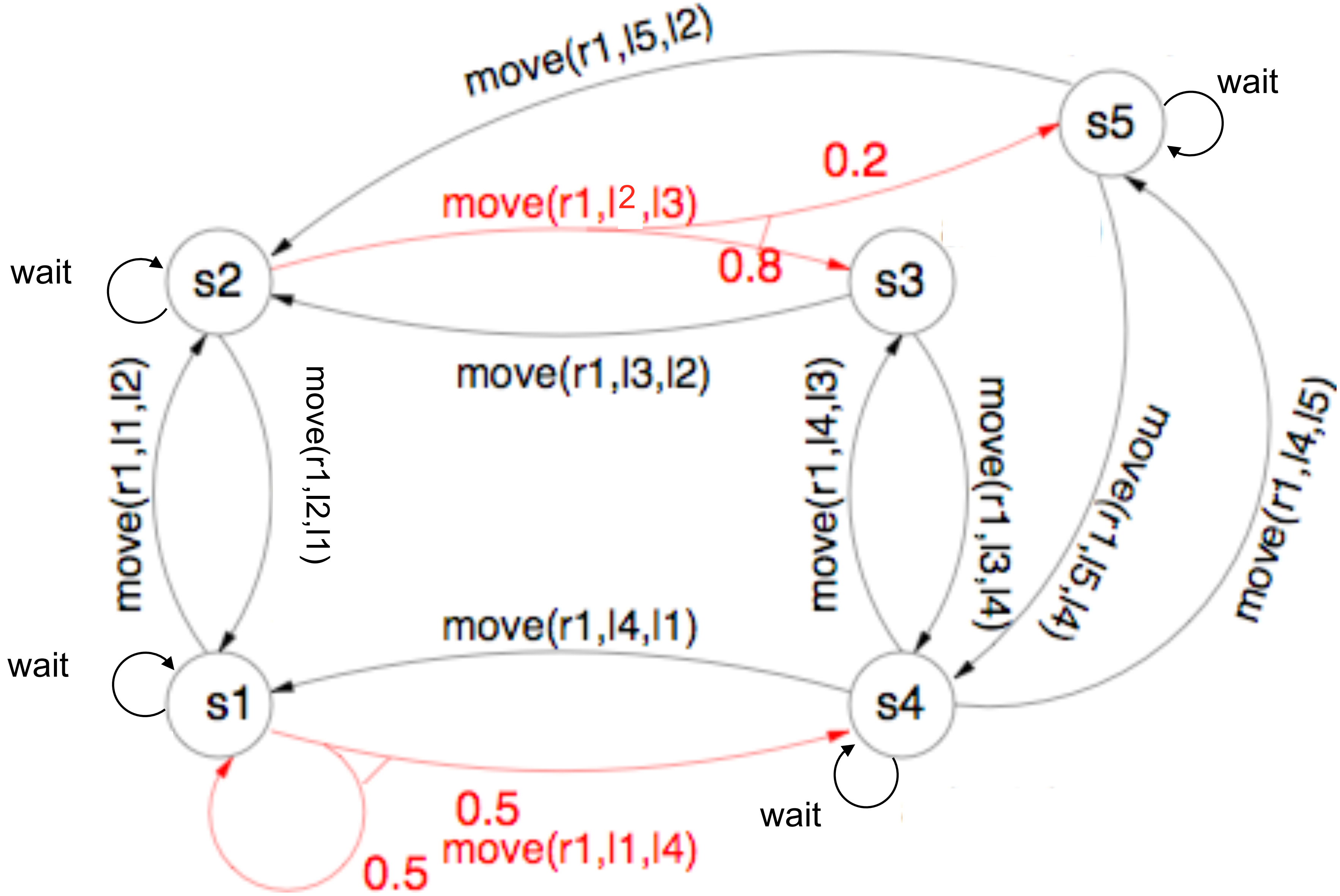
- The qualifier *Markov* is used because the transition probability function **p** and the reinforcement function **r** depend on the past through the current state of the system and the action selected by the decision maker in that state.

Notation: $X_{a:b} = X_a, X_{a+1}, \ldots, X_{b-1}, X_b$

Markov assumption: $\mathbf{X_t}$ depends on <u>bounded</u> subset of $\mathbf{X_{0:t-1}}$

- <u>First-order Markov process</u>: $\mathbf{p(X_t|X_{0:t-1}) = p(X_t|X_{t-1})}$

Example of an MDP



Example of an MDP

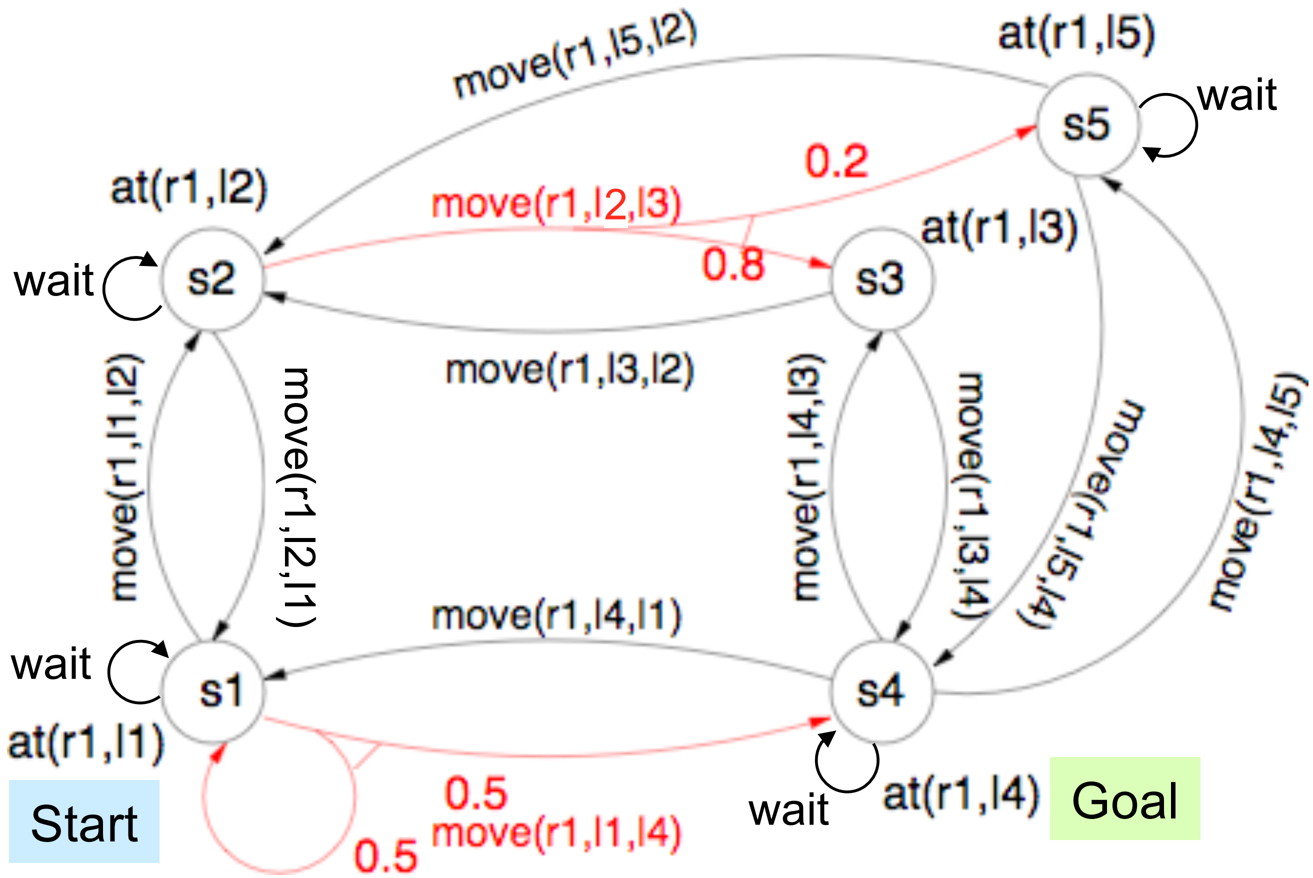
- A = {move(r1,l1,l2), move(r1,l2,l1), move(r1,l4,l1), move(r1,l1,l4), move(r1,l3,l2), move(r1,l2,l3), move(r1,l5,l2), move(r1,l4,l3), move(r1,l3,l4), move(r1,l5,l4), move(r1,l4,l5), wait}
- S = {s1, s2, s3, s4, s5}
- p(s1,move(r1,l1,l4),s4)=0.5; p(s1,move(r1,l1,l4),s1)=0.5; p(s2,move(r1,l2,l3),s3)=0.8; p(s2,move(r1,l2,l3),s5)=0.2; All others p(.) have a value of 1.
- r(s1,wait) = r(s2,wait) = -1; r(s4,wait)=0; r(s5,wait)= -100; r(s1,move(r1,l1,l2))=r(s2,move(r1,l2,l1))= -100; r(s3,move(r1,l3,l4))=r(s4,move(r1,l4,l3))= -100; r(s4,move(r1,l4,l5))=r(s5,move(r1,l5,l4))= -100; r(s1,move(r1,l1,l4))=r(s4,move(r1,l4,l1))= -1; r(s2,move(r1,l2,l3))=r(s3,move(r1,l3,l2))= -1; r(s5,move(r1,l5,l2))= -1; r(s1)=r(s2)=r(s3)=r(s5)=0; r(s4)=100

What is a Solution?

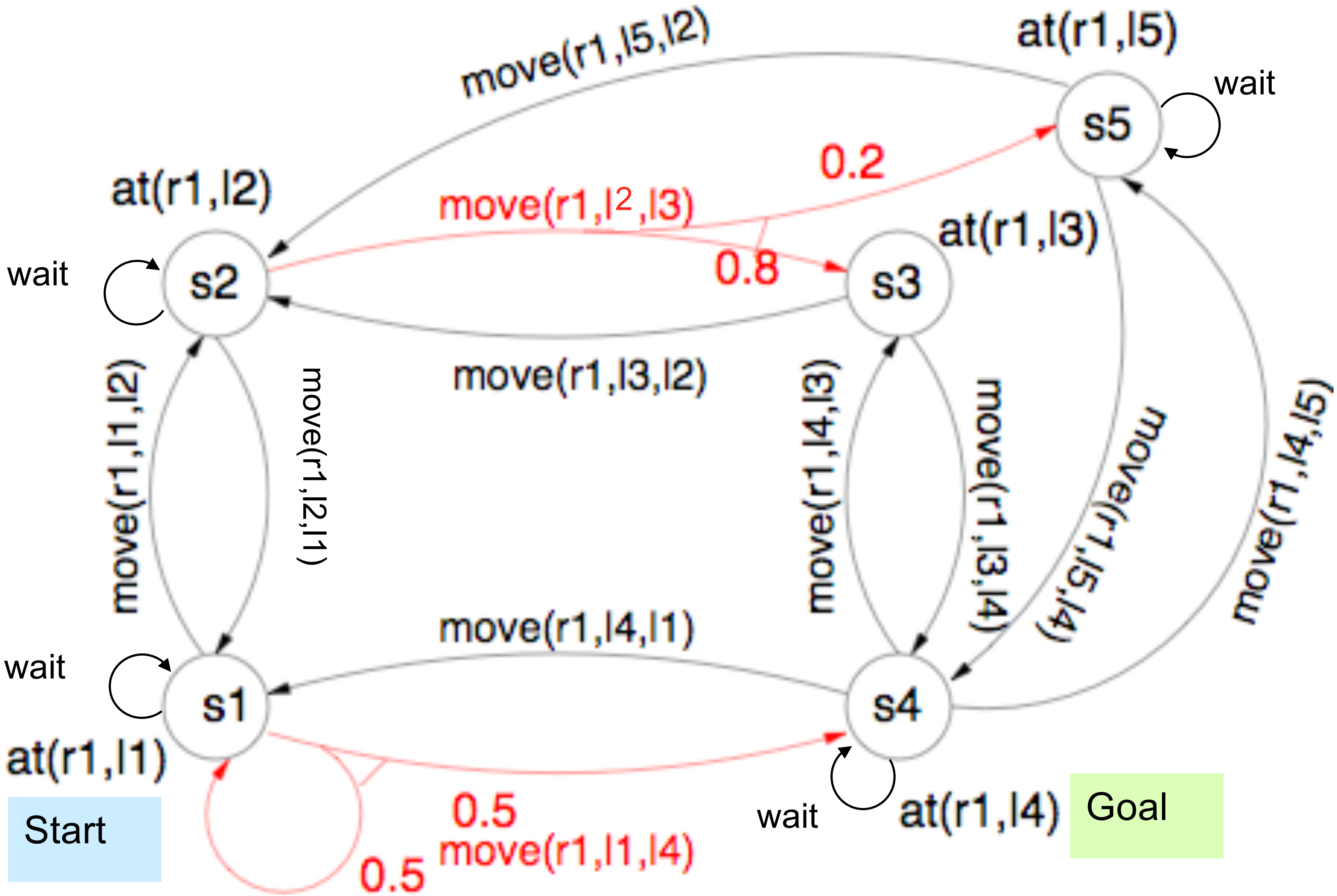
- What does a solution to the problem look like?
  - Any fixed action sequence will **not** solve the problem!

Example

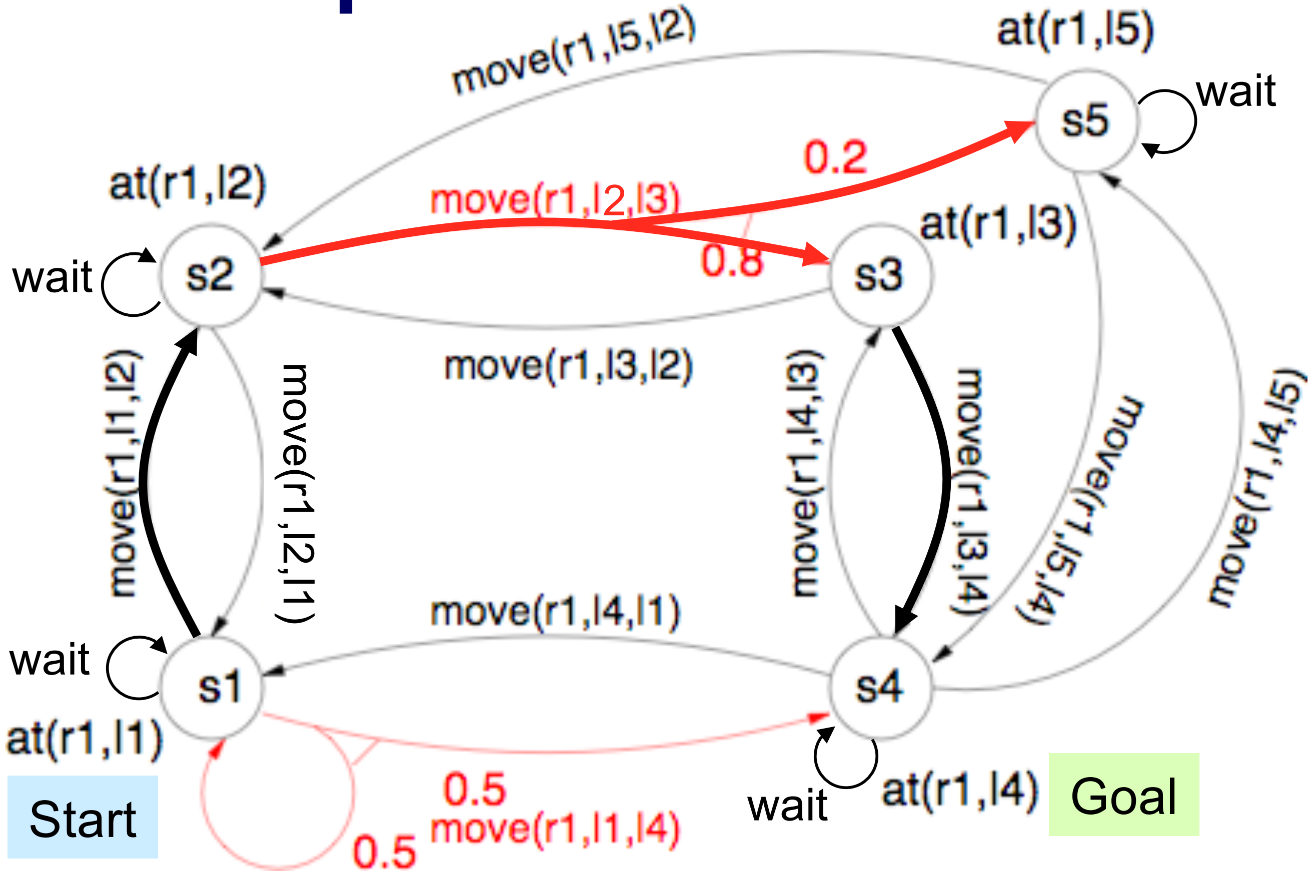
- Robot r1 starts at location l1
  - State s1 in the diagram
- Objective is to get r1 to location l4
  - State s4 in the diagram



Is there a plan that will guarantee the solution?



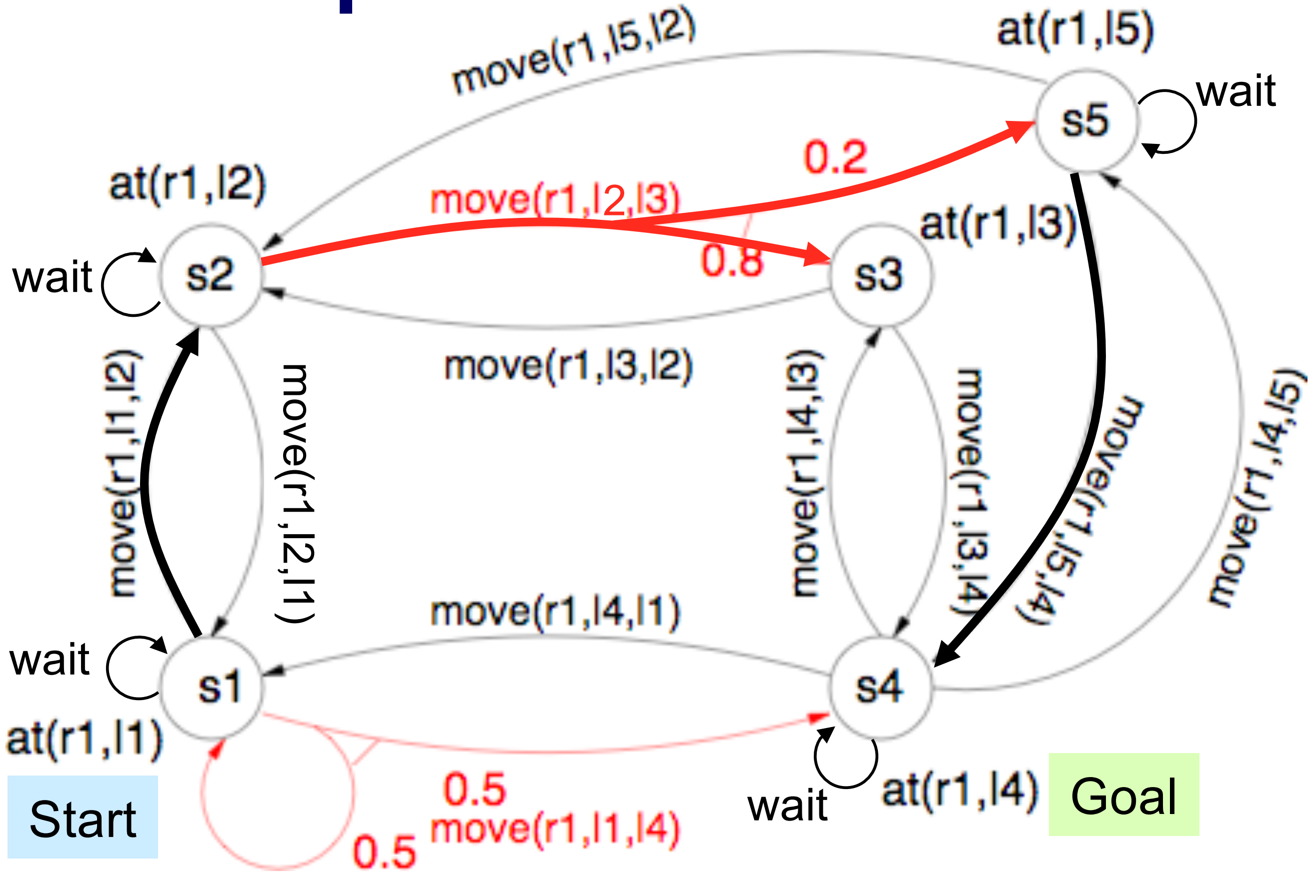
Example



- 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 fixed sequence of actions can be a solution, because we can not guarantee we will be in a state where the next action is applicable
  - e.g.,

Plan 1:⟨move(r1,l1,l2), move(r1,l2,l3), move(r1,l3,l4)⟩

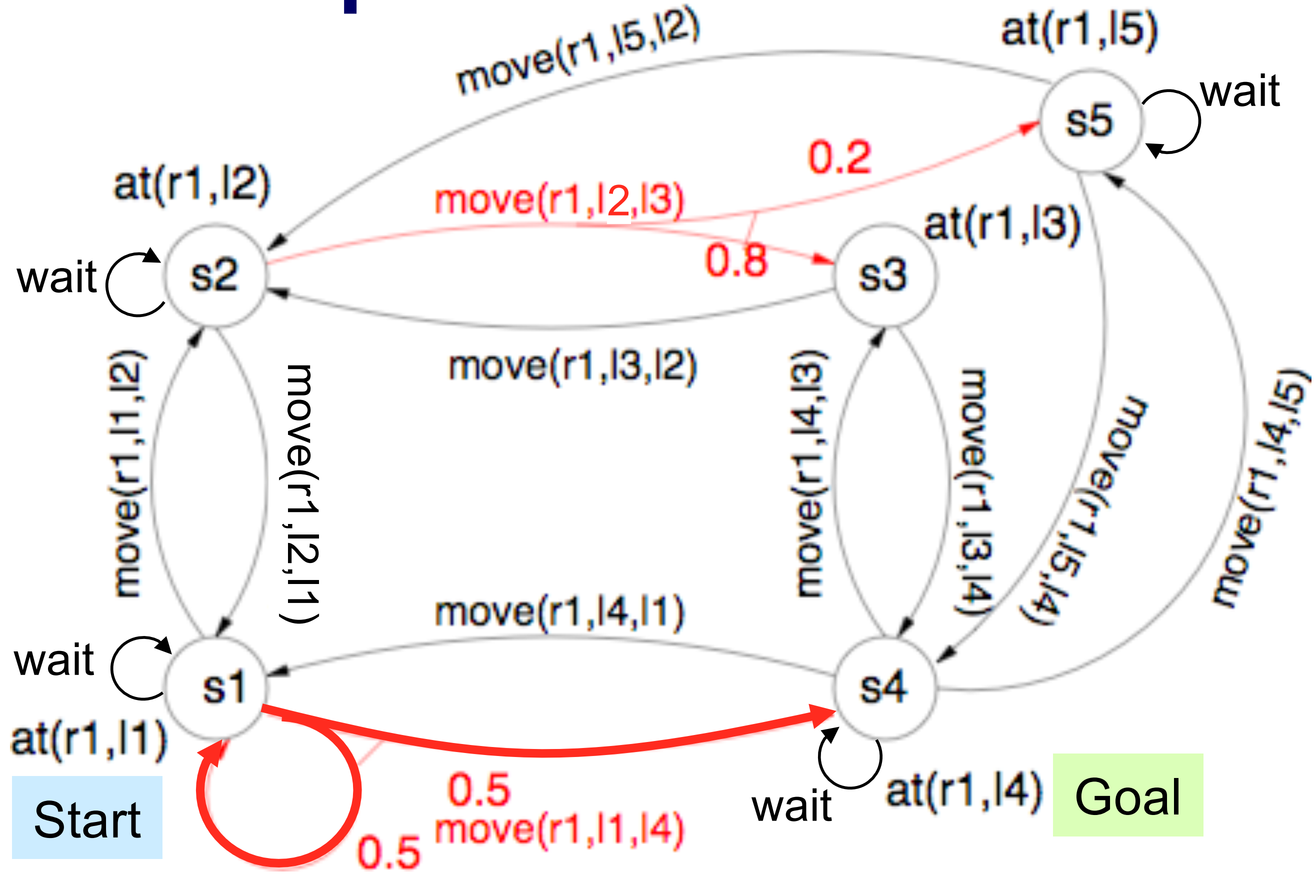
Example



- 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 fixed sequence of actions can be a solution, because we can not guarantee we will be in a state where the next action is applicable
  - e.g.,

Plan 2: ⟨move(r1,l1,l2), move(r1,l2,l3), move(r1,l5,l4)⟩

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 fixed sequence of actions can be a solution, because we can not guarantee we will be in a state where the next action is applicable
  - e.g.,

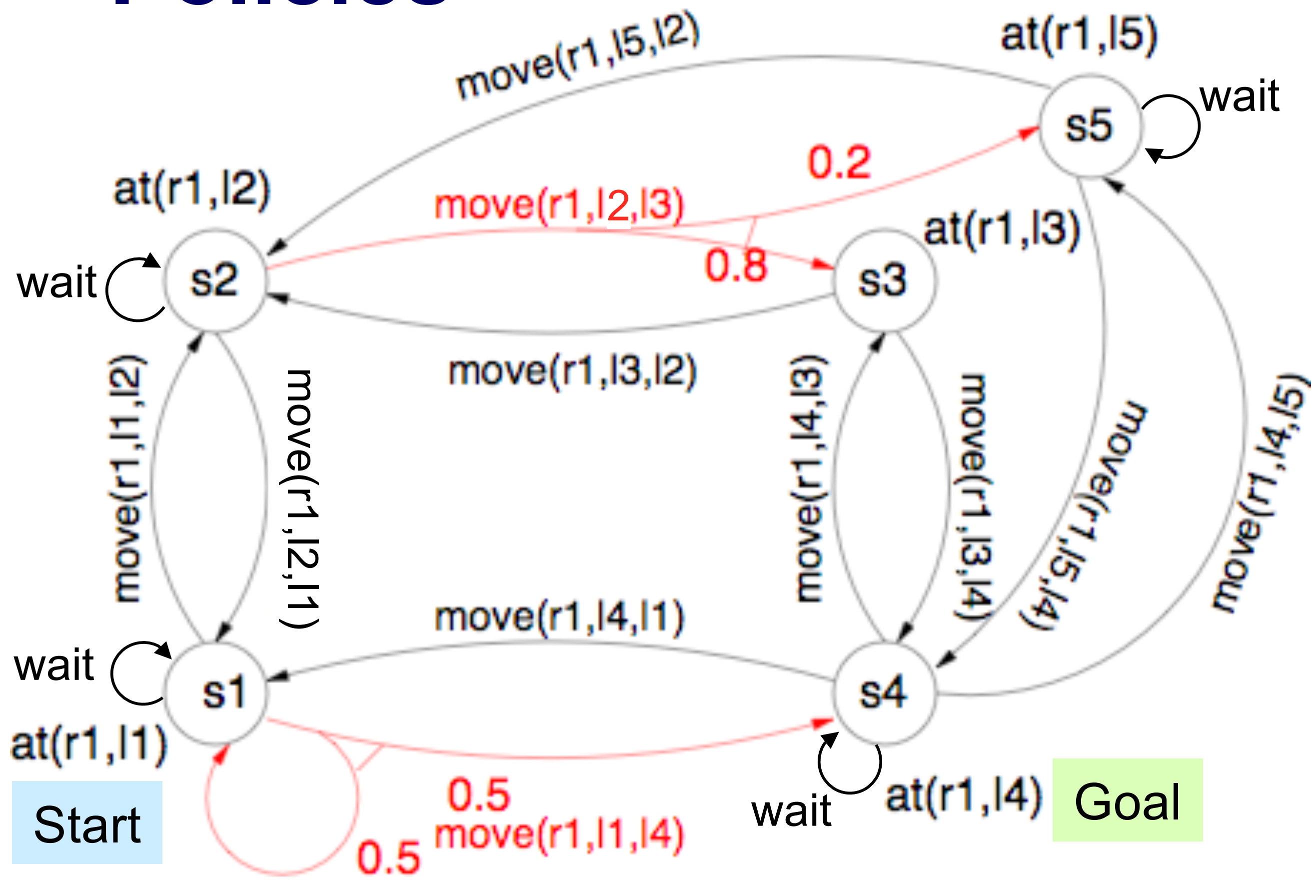
Plan 3: ⟨move(r1,l1,l4)⟩

What is a Solution?

- A solution must specify what the agent should do for ***any*** state that the agent might reach
  ➔ ***policy*** π

$$\Pi: S \rightarrow A, \quad \pi(s) = a \in A$$

Policies



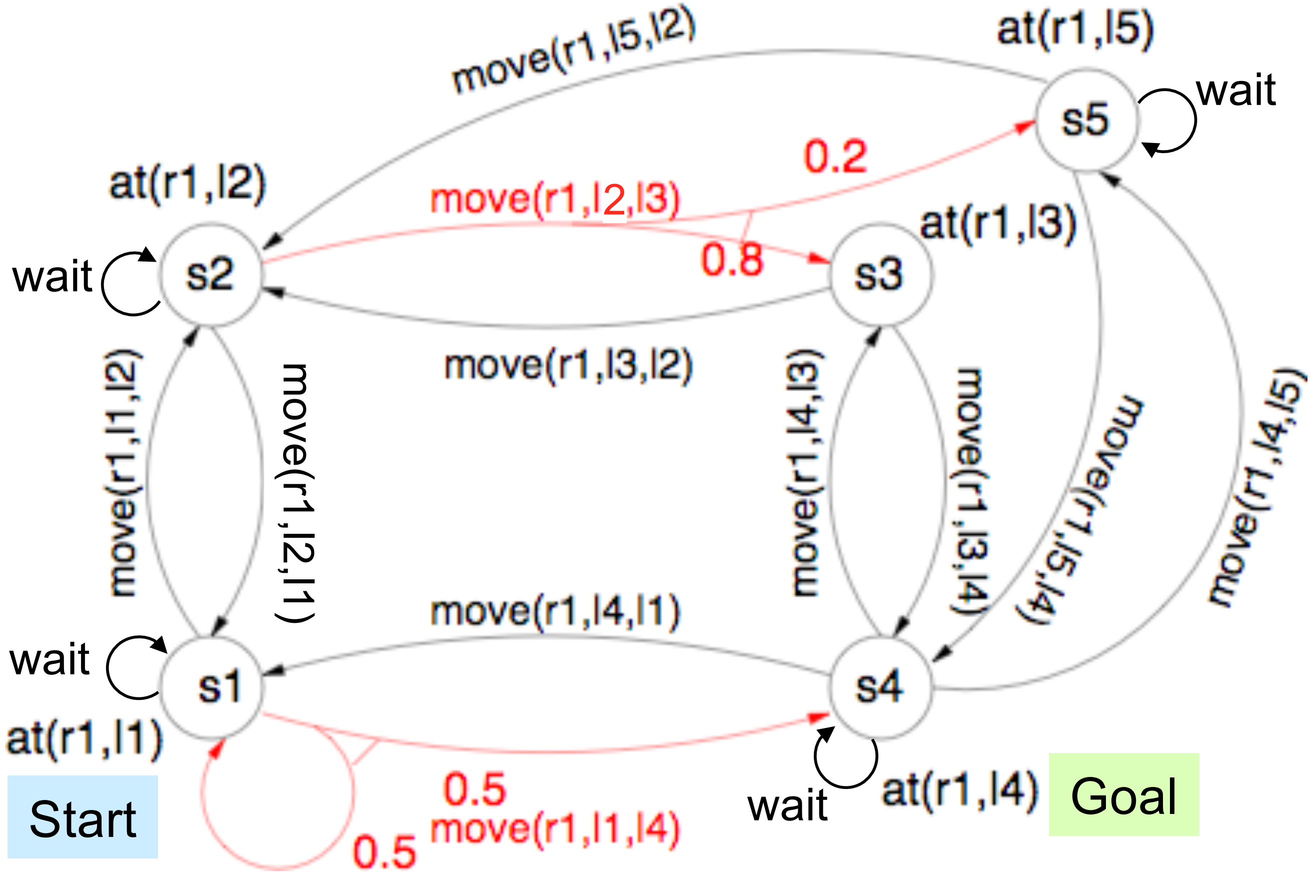
- Policy: a function that maps states into actions
- Write it as a set of state-action pairs

Policies

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

π_2 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4))}

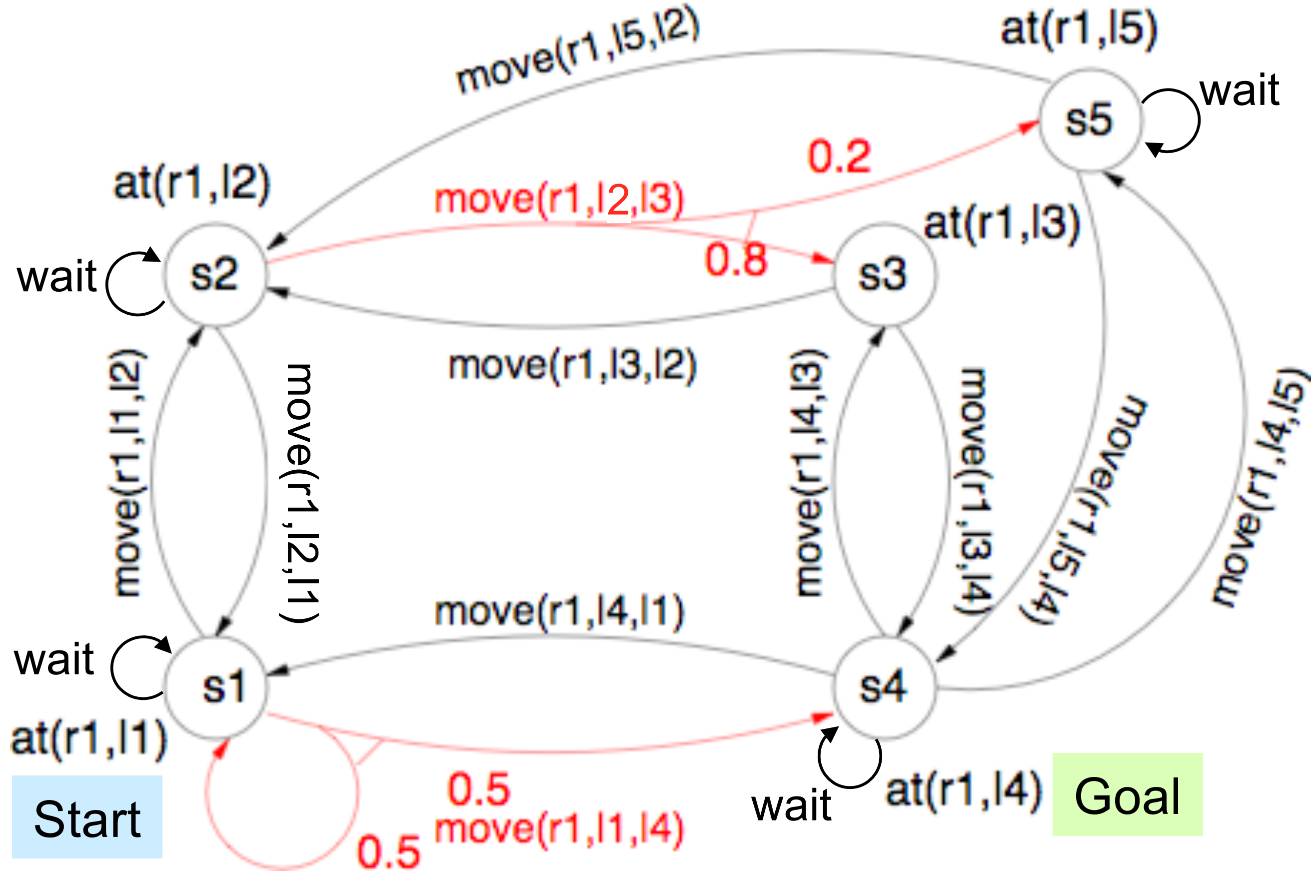
π_3 = {(s1, move(r1,l1,l4)),
(s2, move(r1,l2,l1)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4)}



- 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 begins in the state s
  - We assume the system starts in a unique initial state s_0
    - » $P(s_0) = 1$
    - » $P(s_i) = 0$ for $i \neq 0$



- In the example, $P(s_1) = 1$, and $P(s) = 0$ for all other states

Histories

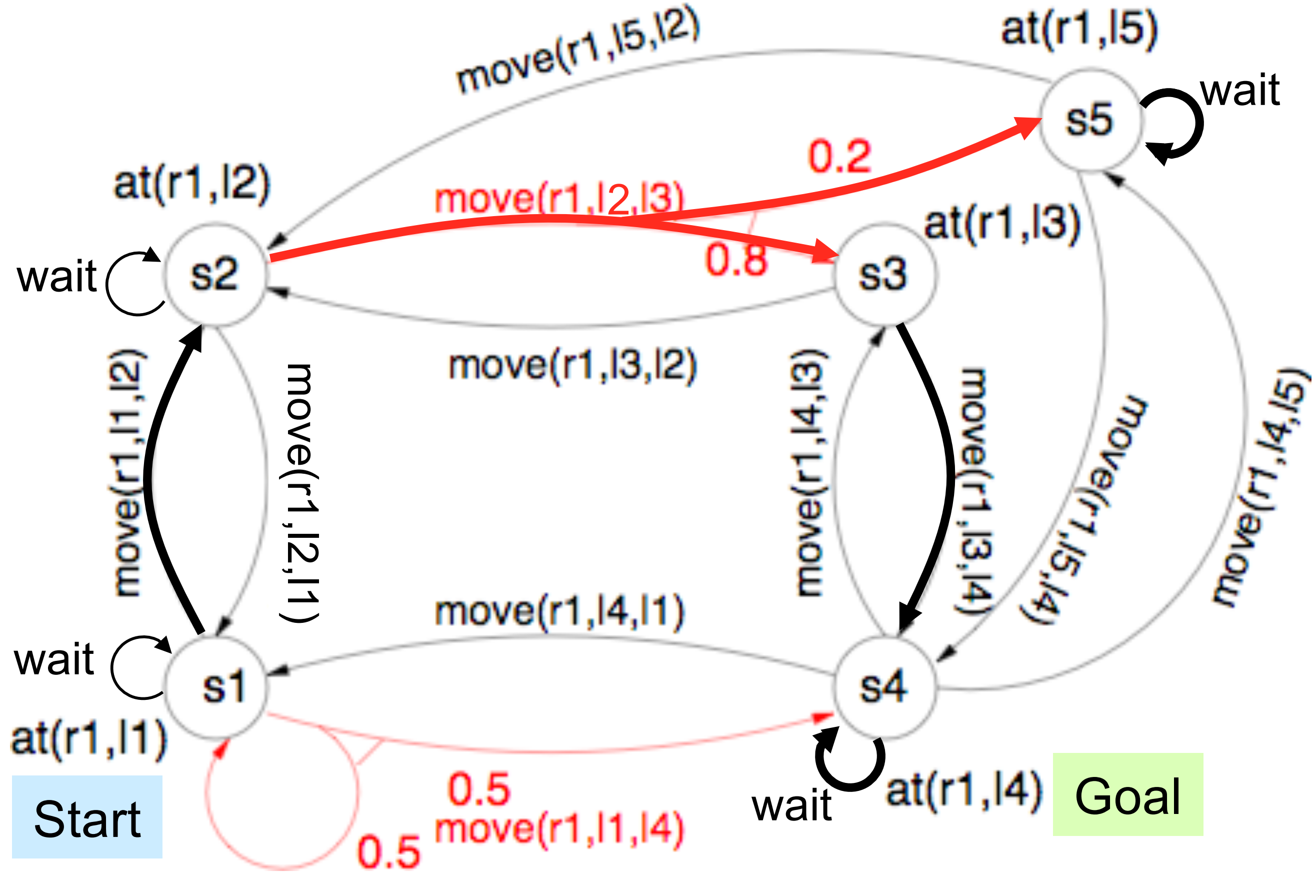
- Each time a given policy is executed starting from the initial state, the stochastic nature of the environment will lead to a different environment history.

- **Each policy induces a probability distribution over histories**
  - If $h = \langle s_0, s_1, \ldots \rangle$ then

$$P(h \mid \pi) = P(s_0) \prod_{i \geq 0} p_{\pi(s_i)}(s_{i+1} \mid s_i, a_i)$$

Example

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



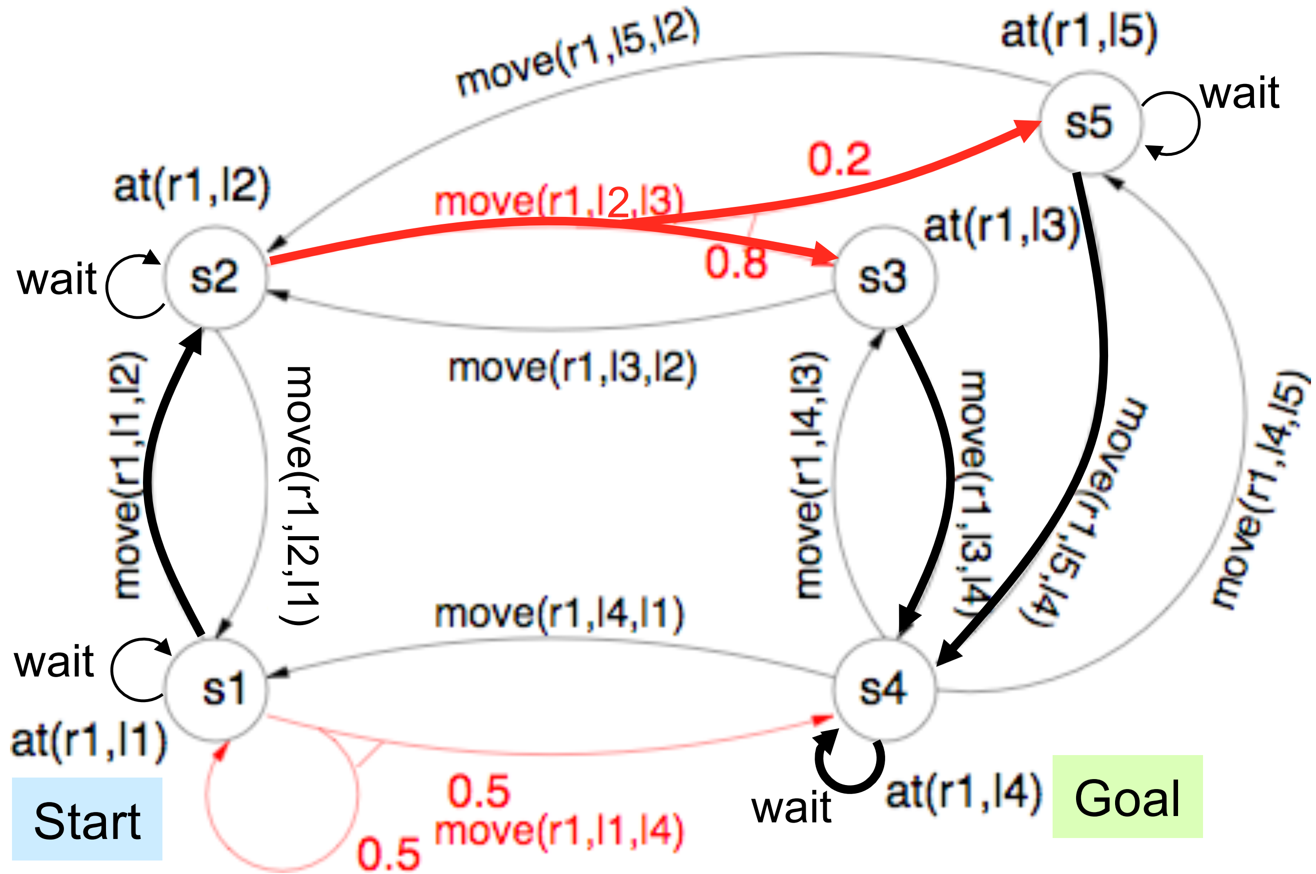
goal

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

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

$$\left\{\begin{array}{l} P(h_1 \mid \pi_1) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8 \\ P(h_2 \mid \pi_1) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2 \\ P(h \mid \pi_1) = 0 \text{ for all other } h \end{array}\right.$$

Example (continued)



π_2 = {(s1, move(r1,l1,l2)),
(s2, move(r1,l2,l3)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4))}

goal

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

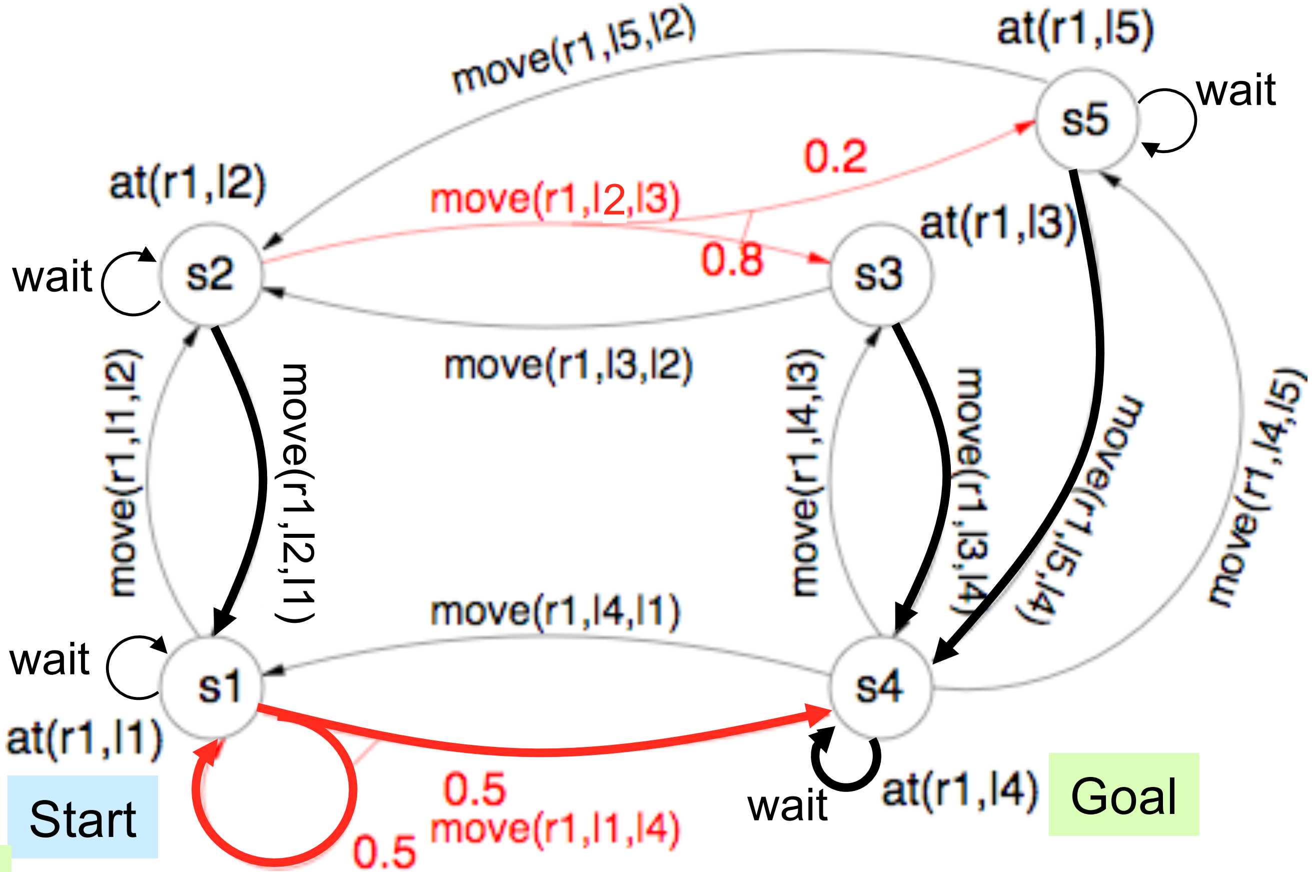
$h_3 = \langle$s1, s2, s5, s4, s4, … $\rangle$

$P(h_1 \mid \pi_2) = 1 \times 0.8 \times 1 \times \ldots = 0.8$

$P(h_3 \mid \pi_2) = 1 \times 0.2 \times 1 \times \ldots = 0.2$

$P(h \mid \pi_2) = 0$ for all other h

Example (continued)



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

goal

$h_4 = \langle$s1, s4, s4, … $\rangle$

$h_5 = \langle$s1, s1, s4, s4, … $\rangle$

$h_6 = \langle$s1, s1, s1, s4, s4, … $\rangle$

…

$h_7 = \langle$s1, s1, s1, s1, s1, s1, … $\rangle$

$$\left\{\begin{array}{l} P(h_4 \mid \pi_3) = 1 \times 0.5 \times 1 \times 1 \times 1 \times \ \ldots = 0.5 \\ P(h_5 \mid \pi_3) = 1 \times 0.5 \times 0.5 \times 1 \times 1 \times \ \ldots = 0.25 \\ P(h_6 \mid \pi_3) = 1 \times 0.5 \times 0.5 \times 0.5 \times 1 \times \ldots = 0.125 \\ \\ P(h_7 \mid \pi_3) = 1 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \times \ldots = 0 \end{array}\right.$$

Quality of a policy

- The quality of the policy is measured by the ***expected utility (***or ***value)*** of the possible environment histories generated by that policy:

$$E[V_\pi(h)] = \sum_h P(h|\, \pi)\, V_\pi(h)$$

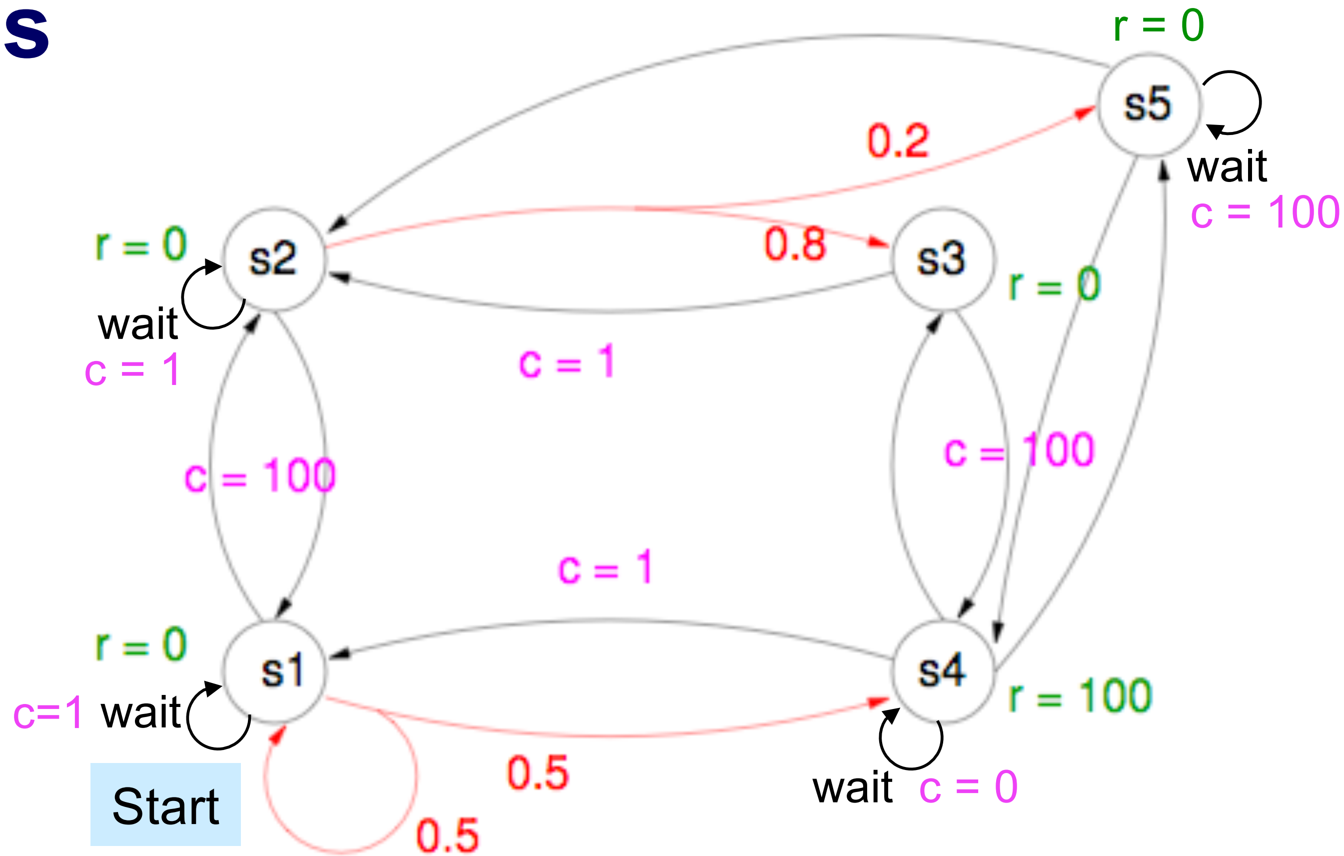
- An **optimal policy π*** is a policy that yields the **highest expected utility**.

Discounted reinforcements:

$$V_\pi(h) = \mathrm{r}(s_0) + \gamma\,\mathrm{r}(s_1) + \gamma^2\mathrm{r}(s_2) + \gamma^3\mathrm{r}(s_3) + \ldots$$

- A **discount factor** $\boldsymbol{\gamma}$**:** $0 \leq \gamma \leq 1$

Utility Functions

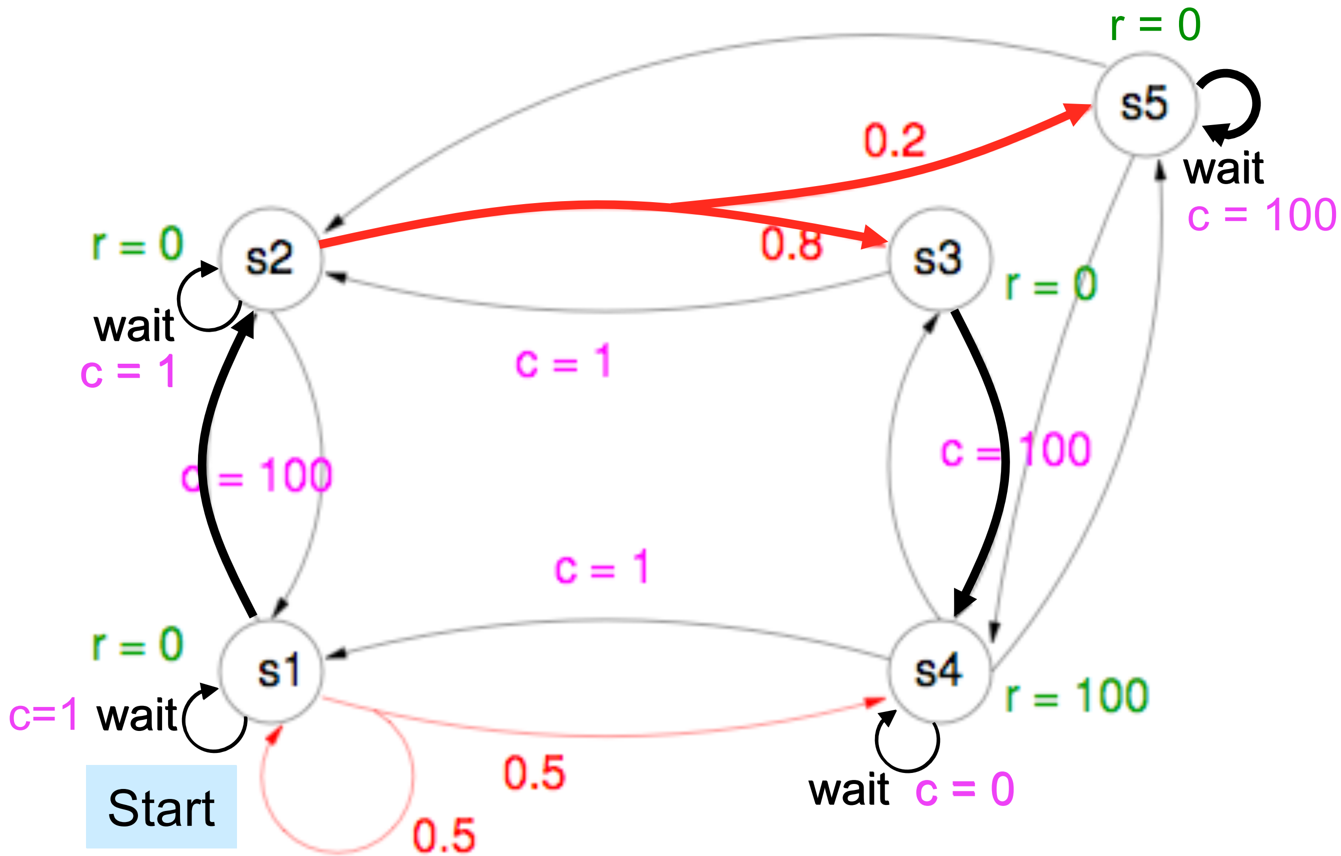


- Reinforcement $r(s)$ for each state s:
  - – : *cost C(s,a)*
  - + : *reward R(s)*
- Example:
  - $C(s,a) = 1$ for each "horizontal" action
  - $C(s,a) = 100$ for each "vertical" action
  - $C(s_1,wait) = 1$; $C(s_2,wait) = 1$; $C(s_4,wait) = 0$; $C(s_5,wait) = 100$
  - R as shown: $r(s_1)=r(s_2)=r(s_3)=r(s_5)= 0$; $r(s_4) = 100$
- Utility function: **generalization of a goal** (additive rewards)
  - If $h = \langle s_0, s_1, \ldots \rangle$, then $V_\pi(h) = \sum_{i \geq 0} \gamma^i (R(s_i) - C(s_i, \pi(s_i)))$

Example

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

$\gamma = 0.9$



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

$V_{\pi 1}(h_1) = .9^0(0 - 100) + .9^1(0 - 1) + .9^2(0 - 100) + .9^3\, 100 + .9^4\, 100 + \ldots = 547.9$

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

$V_{\pi 1}(h_2) = .9^0(0 - 100) + .9^1(0 - 1) + .9^2(-100) + .9^3(-100) + \ldots = -910.1$

$E[V_{\pi 1}(\mathrm{h})] = 0.8 \times 547.9 + 0.2 \times (-910.1) = 256.3$

Optimal Policy

- Utility of a state – defined in terms of the utility of state sequences:

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \,\middle|\, \pi, s_0 = s\right]$$

The true utility of a state, V(s), is just $V_{\pi^*}(s)$, which allows the agent to choose the action that <u>maximizes</u> the expected utility of the ***subsequent*** state:

$$\boxed{\pi^*(s) = \arg\max_a \sum_{s'} p(s' \mid s, a) V^*(s')}$$

If we know V*, then it's easy to find the optimal policy.

Computing V* Approaches

- **Value iteration**
- Policy iteration
- Linear programming

Value Iteration

1. Initialize $V_0(s) = 0$, for all s.
2. Loop until a stop criterion is met:
   - Loop for all s:

$$V^{t+1}(s) \leftarrow r(s) + \max_a \gamma \sum_{s'} p(s' \mid s,a) V^t(s')$$

This algorithm is guaranteed to converge to V*.
The influence of **r** and **p**, which we know, drives the successive Vs to get closer and closer to V*.

Função Valor: exemplo

- Ambiente discreto 4x4, com obstáculos.
- Agente deve alcançar posição destino D a partir de qualquer lugar do ambiente.
- D é um estado absorvente: $V^*(D) = 0$
- Ações que o agente pode realizar: N, S, L, O
- Penalidade por executar uma ação (qualquer) = −1
  - Melhor política => caminho mais curto
- Exemplo para $\gamma = 1$ e MDP determinístico (p=1)

Exemplo: algoritmo de iteração de valor para MDP determinístico

- Cálculo iterativo da função valor ótima.

$$V(s) \leftarrow r_{s,a} + \max_a (V(s'))$$

Repetir até V(s) estabilizar.

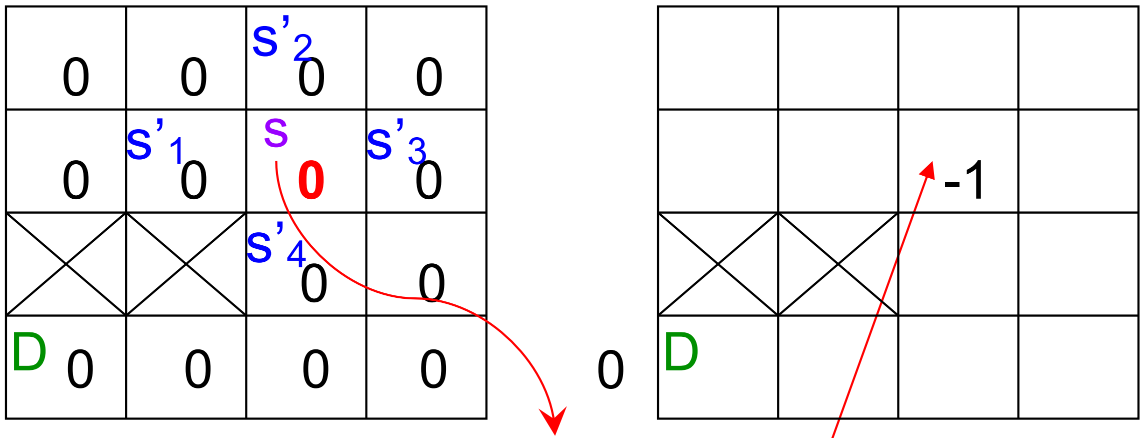
Sendo:

s – estado atual, s’ – próximo estado,

$r_{s,a}$ – reforço recebido por executar a em s

V(.) – valor do estado

Exemplo de cálculo de V(s)

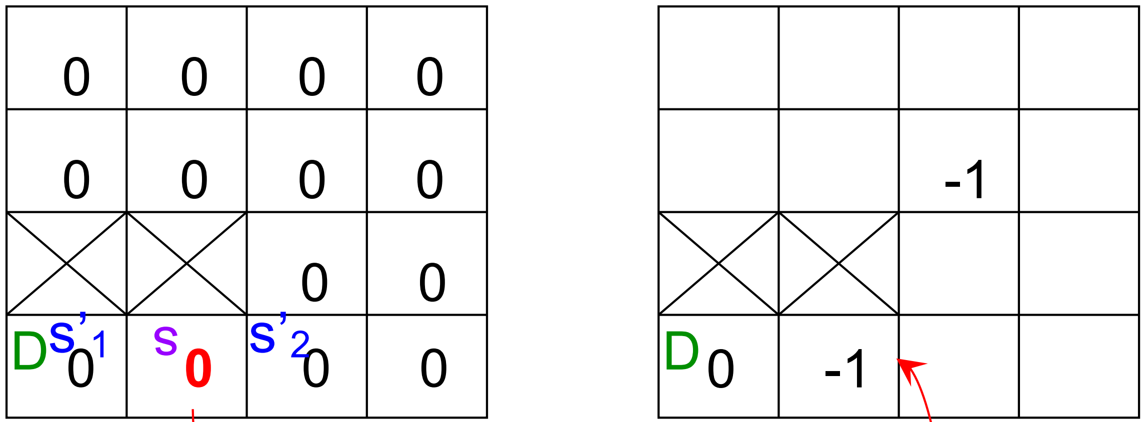


$$
\begin{aligned}
V(s) = \max_a \big(& (r(s, O) + V(s'_1)), \\
& (r(s, N) + V(s'_2)), \\
& (r(s, L) + V(s'_3)), \\
& (r(s, S) + V(s'_4)) \big)
\end{aligned}
$$

$$= \max_a ((-1+0), (-1+0), (-1+0), (-1+0))$$

$$= -1$$

Exemplo de cálculo de V(s)



$$V(s) = \max_a \left((r(s, O) + V(s'_1)), \; (r(s, L) + V(s'_3)) \right)$$

$$= \max_a ((-1+0), (-1+0)) = -1$$

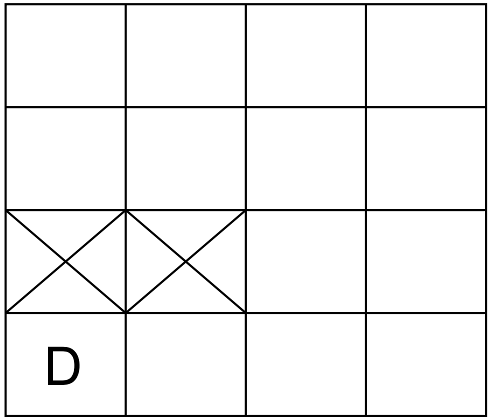
Exemplo de cálculo de V(s)

0	0	0	0
0	0	0	0
		0	0
D 0	0	0	0

-1	-1	-1	-1
-1	-1	-1	-1
		-1	-1
D 0	-1	-1	-1

Ao final da 1a. iteração do “Loop for all s:” do algoritmo VI

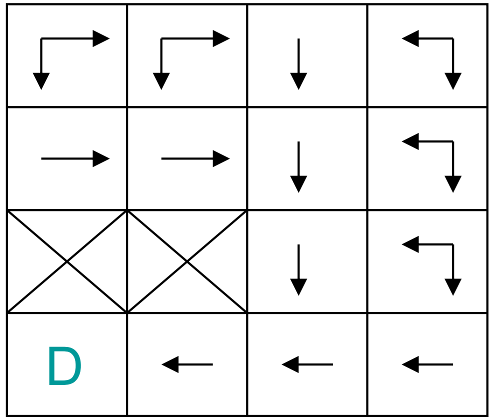
Função valor: exemplo



Ambiente exemplo

-7	-6	-5	-6
-6	-5	-4	-5
		-3	-4
D 0	-1	-2	-3

Função valor ótima: indica as penalidades esperadas até atingir o destino, seguindo uma política ótima.



Políticas ótimas obtidas a partir da função valor ótima (melhores ações, para cada estado).

VI: Discussion

- VI computes new values in each iteration, and chooses a policy based on those values
- This algorithm converges in a polynomial number of iterations
  - But the variable in the polynomial is the number of states
  - The number of states is usually huge
- Need to examine the **entire state space** in each iteration
  - Thus, this algorithm takes huge amounts of time and space

MDP and RL

In ***Reinforcement Learning*** (RL), we would like an agent to **learn** to behave well in an **MDP** world, but **without knowing** anything about **r** or **p** when it starts out

What do you do when you do not know how the world works?

Estimate a **value function** directly

➔ ***We'll investigate an algorithm for doing that.***

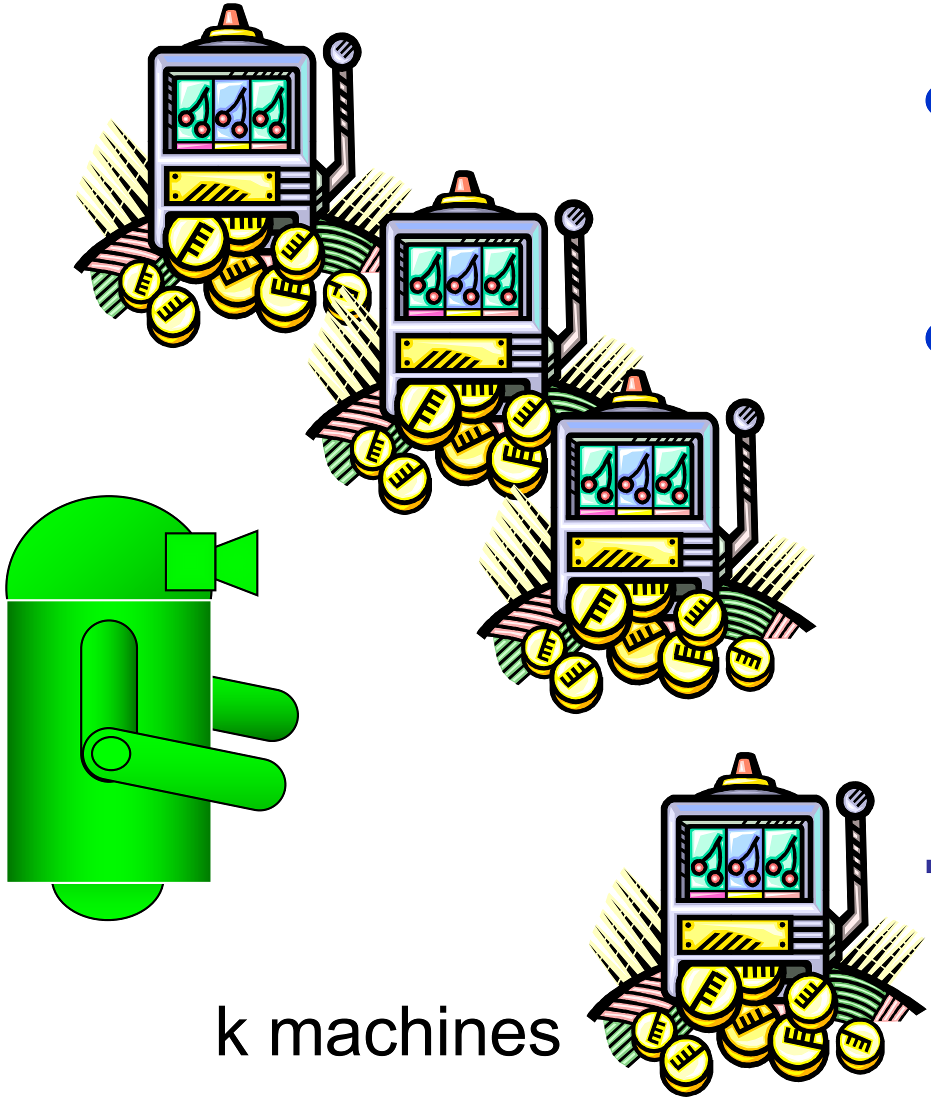
RL and the learning agent

- How the agent should choose its actions?
  - There are <u>two</u>, possibly opposing reasons for the agent to choose an action:

1. because it thinks the action will have a good result in the world (***exploitation***), or
2. because it thinks the action will give it more information about how the world works (***exploration***).

→ an example: *k-armed bandit*

k-Armed Bandit



- Every time you pull an arm on a machine, it either pays off a dollar or nothing.
- Assume that each machine has a **hidden probability of paying off**, and that whenever you pull an arm, the outcome is independent of previous outcomes and is determined by the hidden payoff probability

→ ***What should you do to make as much money as possible during a given time?***

k-Armed Bandit: Strategies

- **Behave at random**
- **Switch on a loser**: pick one arm; as long as it keeps paying off, you should keep pulling it; as soon as it loses, go to the next arm, and so on. ➔ better than random, but it's not optimal!
- **Always choose the apparent best**: keep estimates of the payoff probabilities of each arm (by counting). Then, always choose the arm with the highest estimated probability of paying off ➔ greedy
- **Combined**: choose the apparent best 90% of the time; choose randomly the other 10% … etc….

k-Armed Bandit: Strategies

- Ultimately, the best strategies spend
  - some time **exploring**: trying all the arms to see what their probabilities are like, and
  - some time **exploiting**: doing the apparently best action to try to get reward.

In general, the longer you expect to live, the more time you should devote to exploration.

➔ **ε-greedy strategy** $\begin{cases} \varepsilon\text{: Exploration} \\ 1-\varepsilon\text{: Exploitation} \end{cases}$

MDP

- Cumulative value $V^{\pi}(s_t)$ achieved by following an arbitrary policy π from an arbitrary initial state s_t

$$V^{\pi}(s_t) = r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \gamma^3 r(s_{t+3}) + \ldots$$

$$= \sum_{i=0 \,..\, \infty} \gamma^i r(s_{t+i})$$

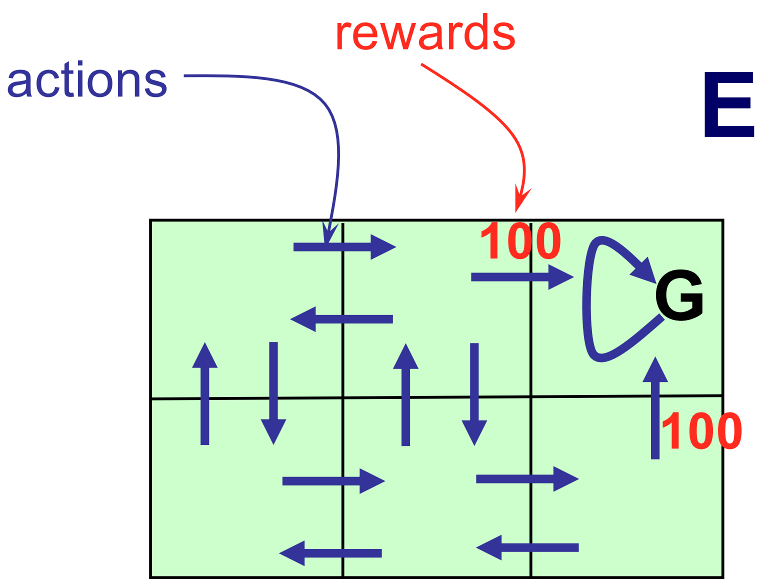
- Optimal policy:

Notation: $V^{\pi^*}(s) = V^*(s)$

$$\pi = \arg\max_{\pi} V^{\pi}(s)\ ,\ \forall s \in S$$

$$\pi^* = \arg\max_a \underbrace{[r(s,a) + \gamma V^*(s')]}_{Q^*(s,a)}\ ,\ \forall s, s' \in S,\ \forall a \in A$$

Example



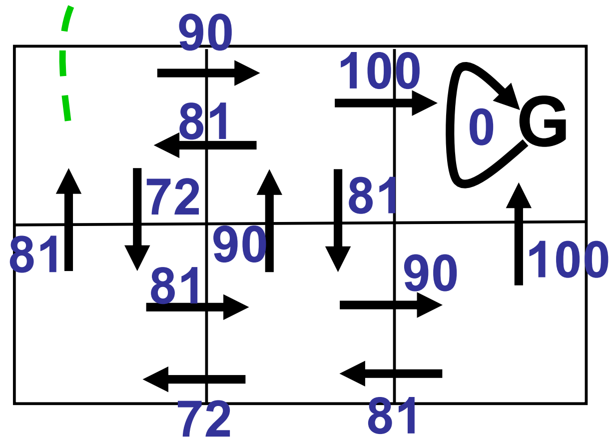
G: absorbing state

$\gamma = 0.9$

r(s,a)=0 otherwise

max(72,90)

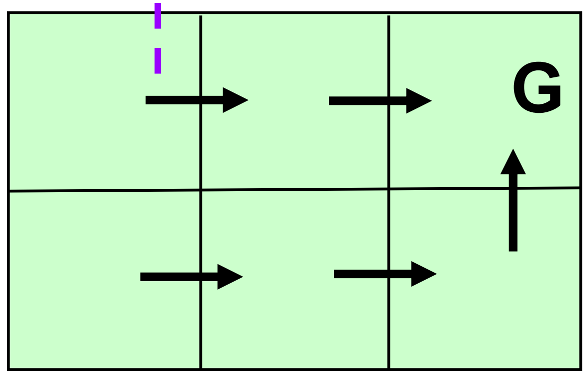
actions ↓ and →, choose →



Q(s,a) values

90	100	0
81	90	100

V*(s) values



One optimal policy

Optimal policy: Right, Up => 0+□100+□^{2}0+□^{3}0+…=90

Q values

- Q(s,a) is the expected discounted future reward for starting in state ***s***, taking ***a*** as our first action, and then continuing <u>optimally</u>.

$$\pi^* = arg\ max_a\ Q^*(s,a)$$

  - The agent only needs to consider each available action *a* in its current state *s* and chooses the action that maximizes Q(s,a).
  - Q(s,a) summarizes all the information needed to determine the discounted cumulative reward that will be gained in the future if *a* is selected in *s*.

Requirements for an RL algorithm

We need:

- Decision on what constitutes an **internal state** (e.g. Q values, etc)
- Decision on what constitutes a **world state**
- **Sensing** of a world state
- Action-choice mechanism (**policy**) based usually on an **evaluation function** (of internal and world state)
- A means of **executing** the action
- A way of **updating** the internal state

Q-learning Algorithm

- Estimates the Q* function directly, without estimating the transition probabilities

Initialize Q(s,a) arbitrarily

Observe the current state s_t

do forever

select an action a_t and execute it in s_t

receive immediate reward $r(s_t,a_t)$

observe the new state s_{t+1}

update Q(s,a) as follows:

$$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)\, Q_t(s_t,a_t) + \alpha\, [r(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a)]$$

$s_t \leftarrow s_{t+1}$

α: *learning rate*

Learning rate

- The basic form of the update looks like this:

$$X_{t+1} \leftarrow (1 - \alpha)\, X_t + \alpha\, New_t$$

α is a learning rate; usually it is something like 0.1 or 0.2.

  - So, we are updating our estimate of X to be mostly like our old value of X, but adding in a new term *New*
    - » This kind of update is essentially a **running average** of the new terms received on each step.
  - The smaller alpha is, the longer term the average is. With a small alpha, the system will be slow to converge, but the estimates will not fluctuate very much.
  - **It is quite typical (and, in fact, required for convergence), to start with a large alpha, and then decrease it over time.**

Q-learning Algorithm

- There are **two iterative processes** going on:
  1. One is the usual kind of **averaging** we do, when we collect a lot of samples and try to estimate their mean (using the **learning rate**)
  2. The other is the **dynamic programming iteration** done by value iteration, updating the value of a state based on the estimated values of its successors.

Q-learning – notes

- **Session:** at the beginning of each session, start the Q-table with zeros or random values
- **Episode:** each learning restart, without restarting the Q-table (it stops when reaching the target or timeout).
  - Typically, one presents the result as the average of sessions (each session comprising a number of episodes) in different iterations.

Q-learning Algorithm

- **Guaranteed to converge to Q***
  - The optimal Q function is achieved **if** the world is really an MDP, **if** we manage the learning rate correctly, and **if** we explore the world in such a way that we never completely ignore some actions and states.

☺

Applications

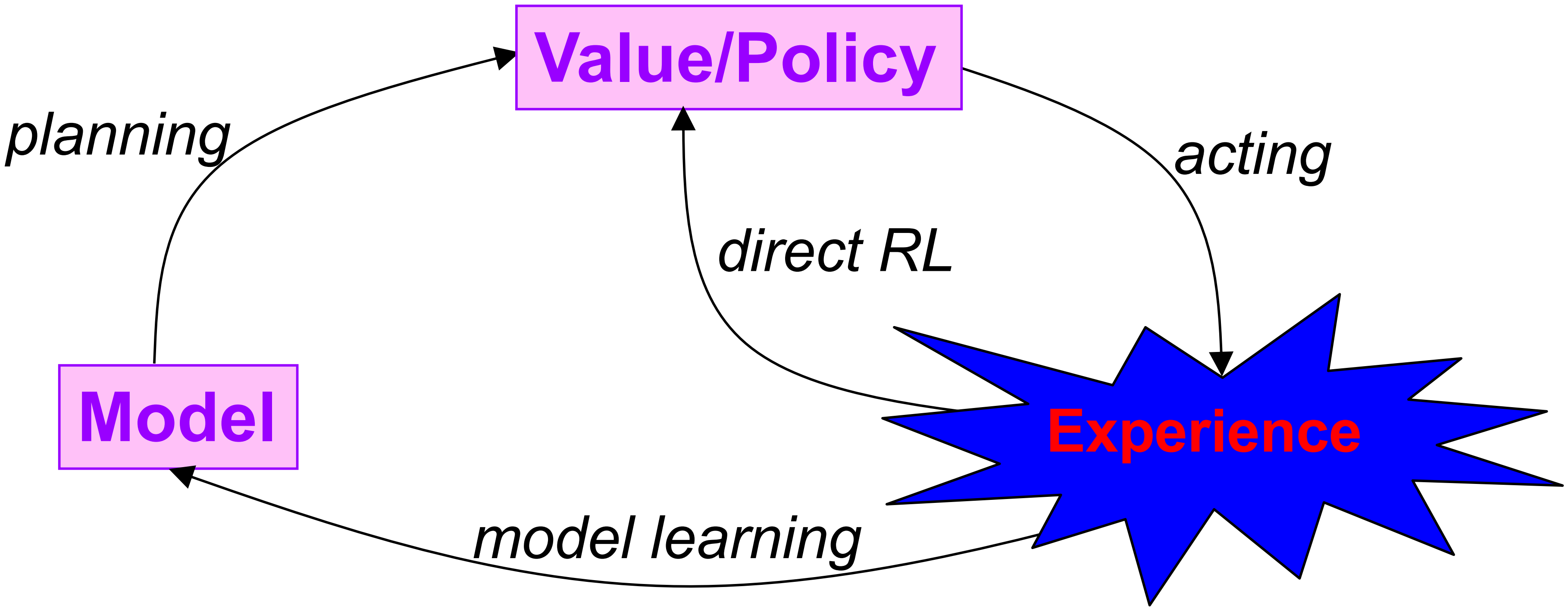
- **TD Gammon:** starts out not knowing anything about backgammon. It plays more than 2 **million** games of backgammon against itself. It can now draw the human world-champion backgammon player.
- **Elevator scheduling**: in a building with many floors and many elevators, there is a serious control problem in deciding which elevators to send to which floors next. The input to the system is the locations of the elevators and the set of all buttons that have been pressed. The output is a direction for each elevator so that the throughput of people could be maximized. The learned policies are considerably more effective that the ones that are standardly built in by the elevator companies.
- **Atari**, **AlphaGo**

Applications

- **Resources management in computer clusters**
  - H.Mao, Alizadeh, M. Alizadeh, Menache, I.Menache, and S.Kandula. Resource Management With deep Reinforcement Learning. In ACM Workshop on Hot Topics in Networks, 2016.
- **Traffic Light Control**
  - I. Arel, C. Liu, T. Urbanik, and A. Kohls, "Reinforcement learning-basedmulti-agent system for network traffic signal control,"IET IntelligentTransport Systems, 2010.
- **Robotics**
  - J. Kober, J. A. D. Bagnell, J. Peters. Reinforcement Learning in Robotics: A survey. *Int. J. Robot. Res.* Jul. 2013.
- **Personalized Recommendations**
  - G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, Ni. J. Yuan, X. Xie, and Z. Li. DRN: A Deep Reinforcement Learning Frameworkfor News Recommendation. 2018.
- **etc, etc…**

RL and Planning

- Planning vs. Learning: planning uses a **model**
- Planning in RL interleaves cycles of learning based on experience in the world and experience gained via using the model to predict what will happen.



Q-learning Algorithm: Problems

- **Large or continuous state spaces**
  - It requires that S and A be drawn from a small enough set that we can store the Q function in a table.
  - Possible solutions: use of function approximations

    E.g. neural network (DL), regression trees, factored representations (represent p(s’|s,a) using Bayes net), etc to store the Q function.

    Such approaches are no longer theoretically guaranteed to work, and they can be a bit tricky, but sometimes they work very well.

Q-learning Algorithm: Problems

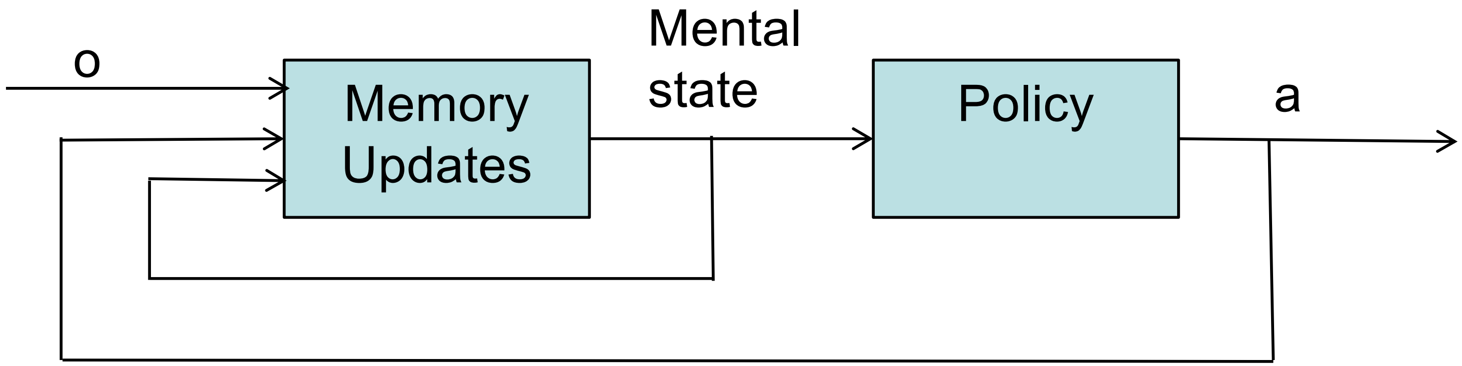
- **Slow convergence**
  - Because of this, most of the applications of Q learning have been in very large domains for which we actually know a model:

  1. we use the known model to build a simulation.
  2. then, using Q learning plus a function approximation technique, we learn to behave in the simulated environment, which yields a good control policy for the original problem

  - **Hot topics**: batch-RL, transfer learning!

Q-learning Algorithm: Problems

- MDPs assume **complete observability** (can always tell what state the agent is in)
  - **POMDP** (Partially Observable MDP)
  - Observation: Pr(O|s,a) [O is observation]



Optimal solution: intractable computationally.
Current research topic: how to approximately solve POMDPs

Conclusion

- Reinforcement learning is a promising technology!
- There are a lot of possible refinements that are being made to have a truly widespread application of RL.

- References:

http://www.cs.ualberta.ca/~sutton/book/ebook/index.html

http://ocw.mit.edu/ (course 6.825)

Chapter 13: Machine Learning, Tom Mitchell

Q-learning: exercício

- Ambiente 2x2, MDP determinístico
- Meta: estado 0.
- Ações: A={ N, S, L, O}
- Estados: S={0, 1, 2, 3}
- Reforços: +10 – atingir meta

  -10 – bater nas bordas

  0 – outros casos

0	1
2	3

Considerar $\alpha = 1$, $\gamma = 0,9$. Zerar a tabela Q no início da sessão. Efetuar 2 episódios:

1. $s_0 = 3$, ações: <SNNO>
2. $s_0 = 2$, ações: <LNNLO>

Fornecer a tabela Q após o término do 1º. Episódio e a política aprendida até então.