Reinforcement Learning

Anna Helena Reali Costa

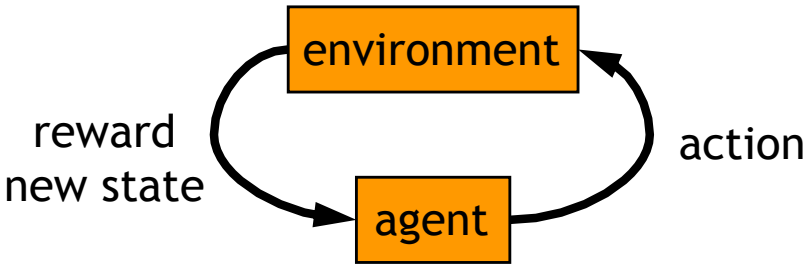
- **Supervised Learning:**
  - **Data**: (x, y). x is data, y is label
  - **Goal**: Learn a function to map $x \rightarrow y$
  - **Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

- **Unsupervised Learning:**
  - **Data**: x. Just data, no labels
  - **Goal**: Learn some underlying hidden structure of the data
  - **Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Reinforcement Learning

- Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

- **Goal**: Learn how to take actions in order to maximize reward



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

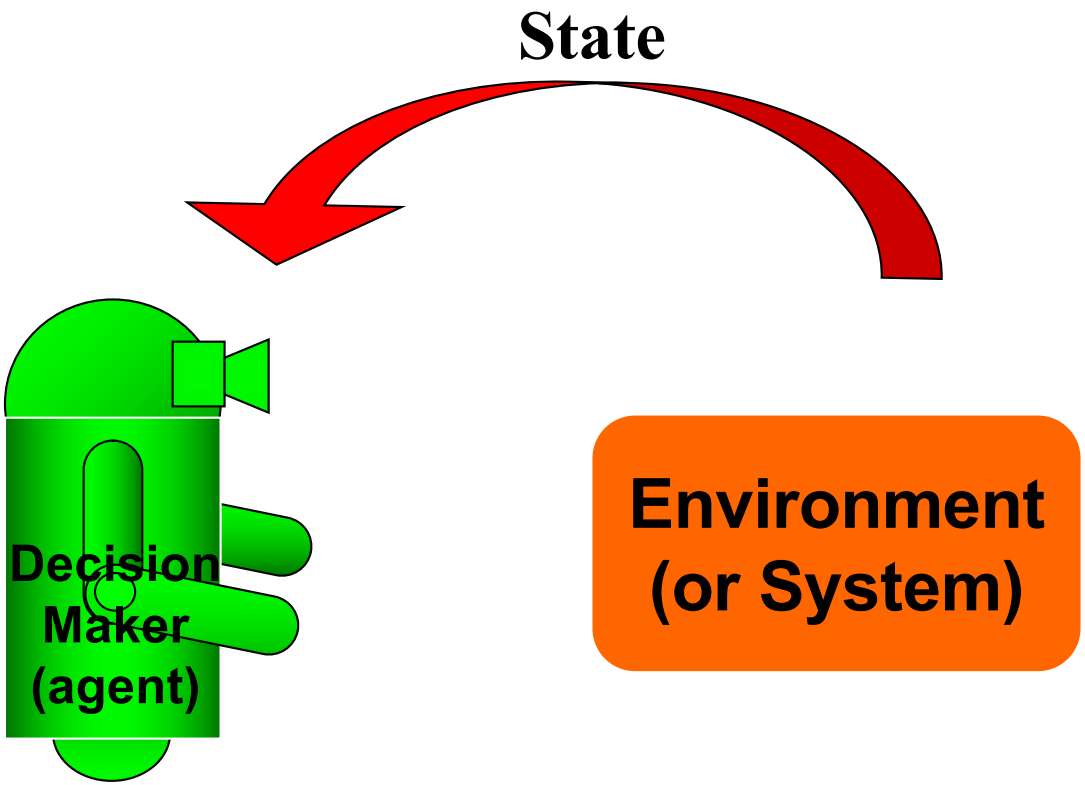
➔ it is a ***sequential decision problem***

Sequential decision problem

At each time step the decision maker:

1. Observes the state of the system;

s_0

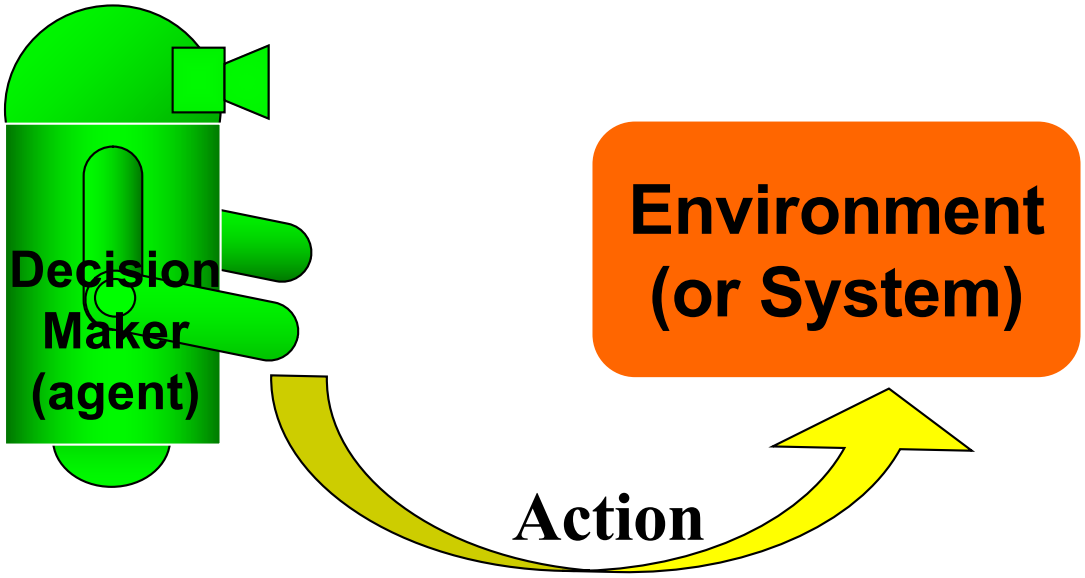


Sequential decision problem

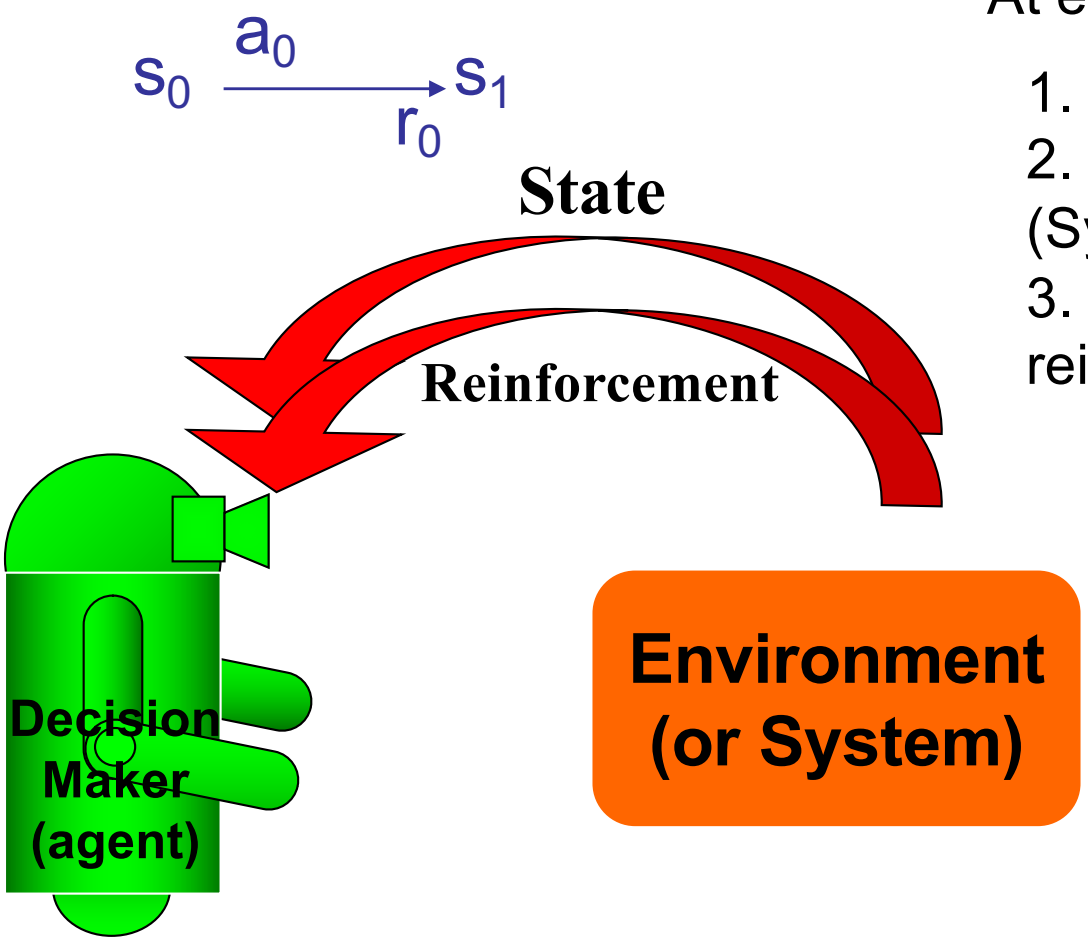
s_0 a_0

At each time step the decision maker:

1. Observes the state of the system;
2. Chooses an action and applies it;



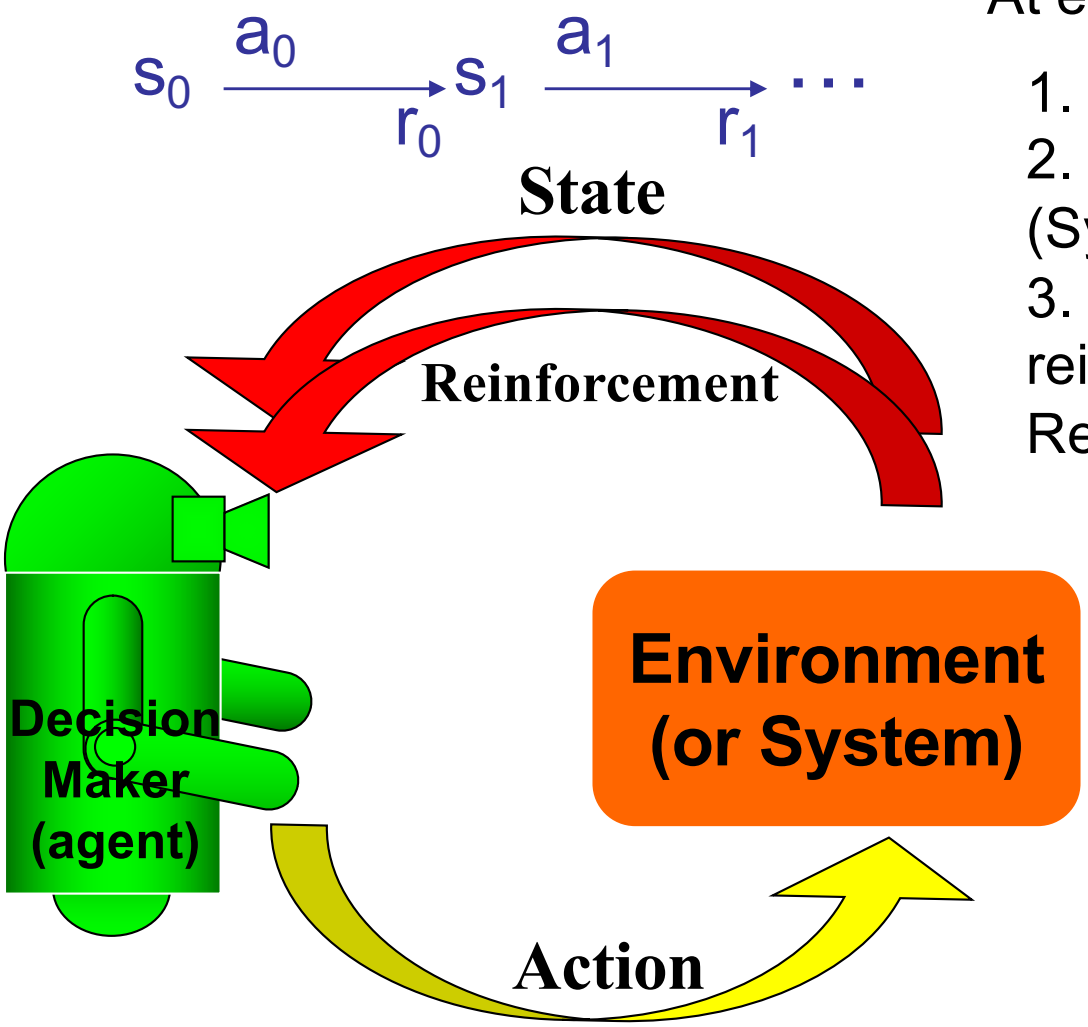
Sequential decision problem



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);

Sequential decision problem

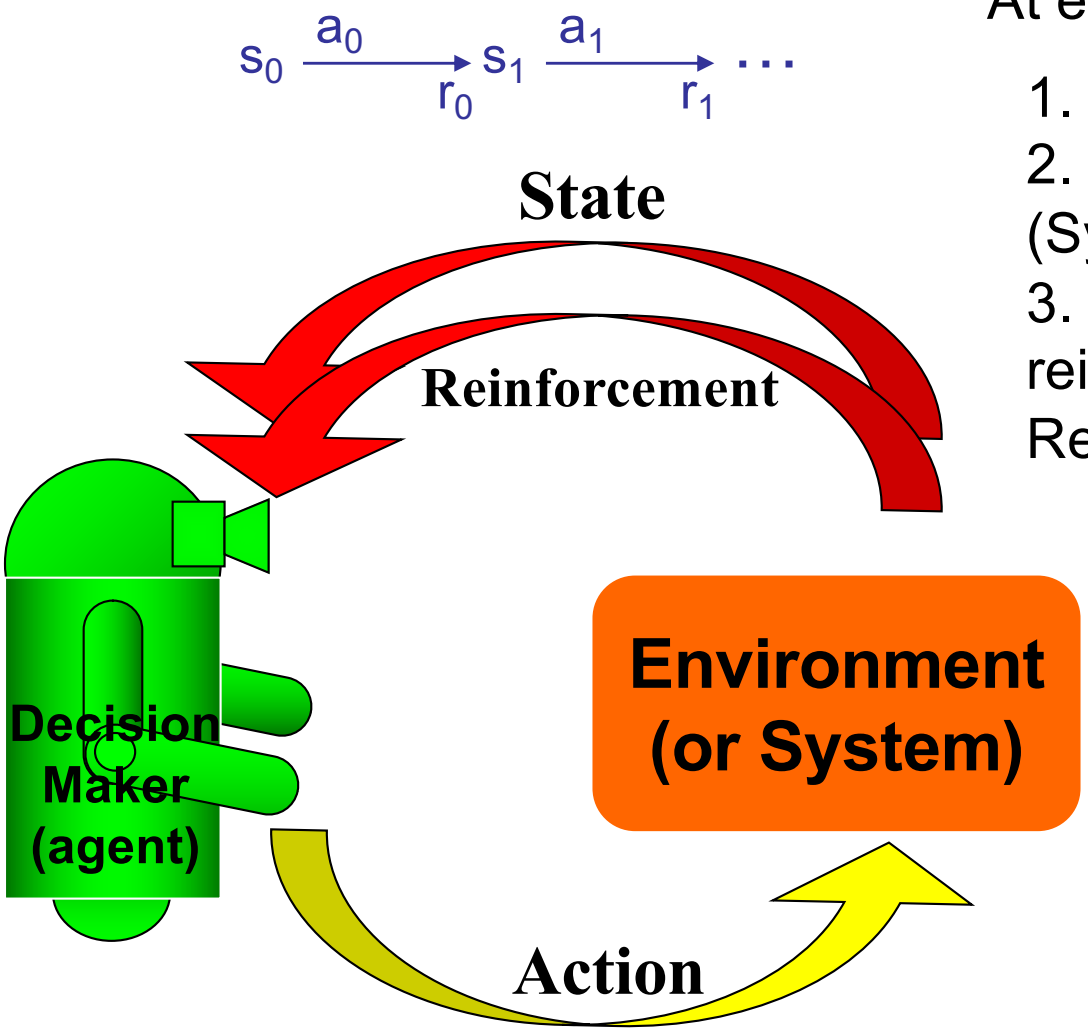


At each time step the decision maker:

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



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**:
$T = \{0,1,2,\ldots,N\}$, $N \leq \infty$.

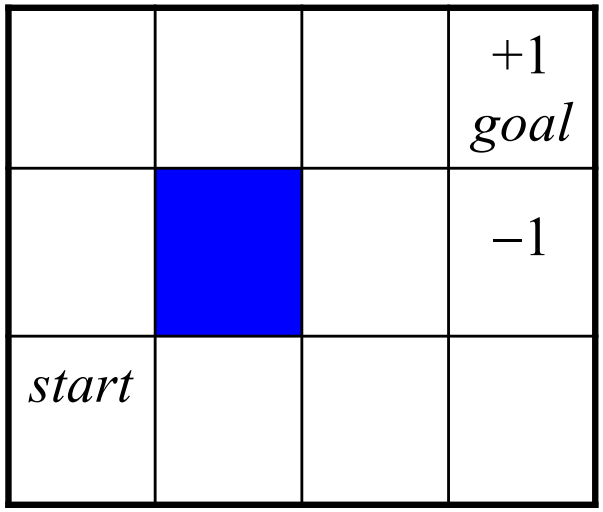
Example

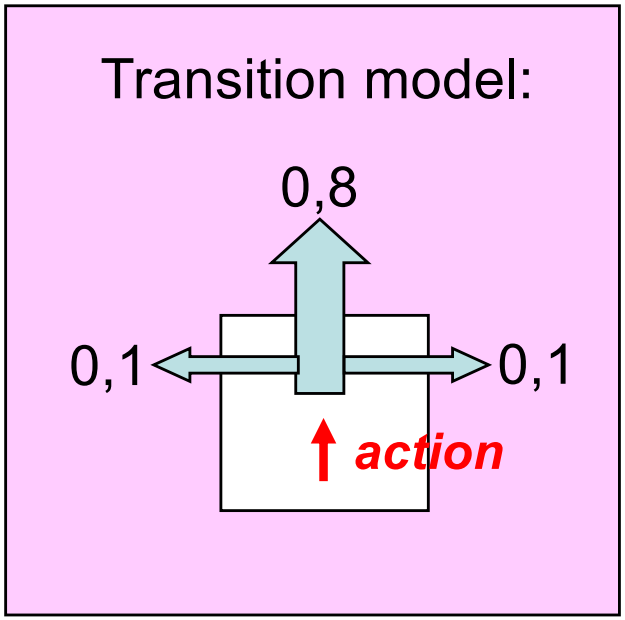
- 4×3 discrete fully-observed environment
- Actions: Up, Down, Left and Right
- Initial state: (1,1)
- Sequence **[U, U, R, R, R]**:

  (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

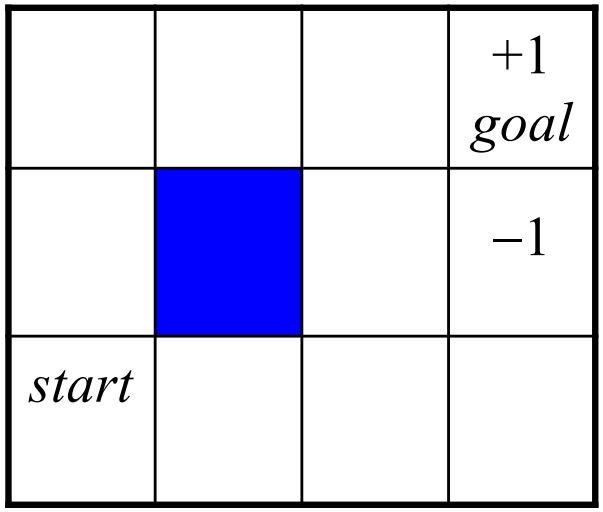
- 4×3 discrete fully-observed environment
- Actions: Up, Down, Left and Right
- Initial state: (1,1)
- Sequence **[U, U, R, R, R]**:

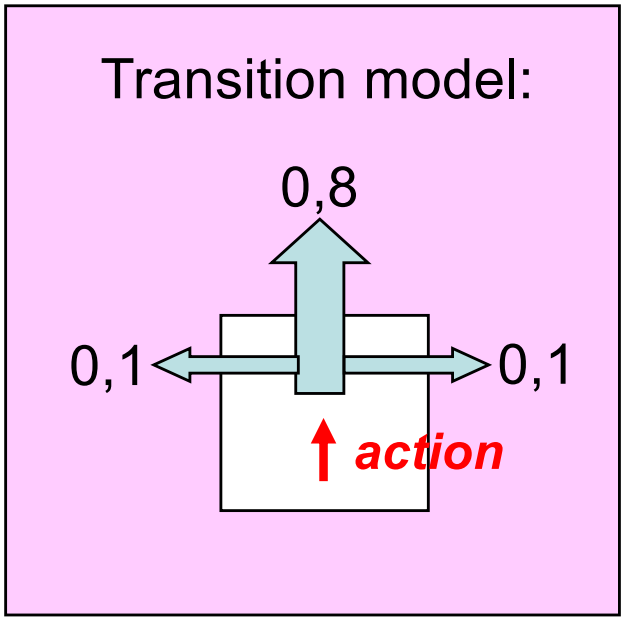
(i) goes up around the barrier and reaches the goal state (3,4) with probability …

$$\mathbf{0{,}8^5 = 0.32768.}$$

(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**

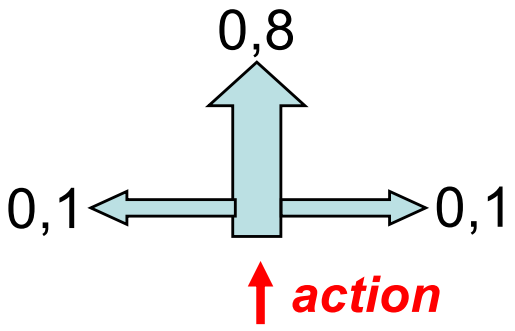
Example

- **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

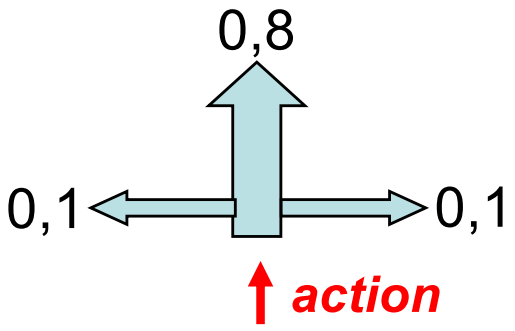
- **Utility function** for the agent depends on a sequence of states (environment ***history***)
- In each state **s**, the agent receives a **reinforcement r(s)**, which may be positive or negative, but must be **bounded.**
- Utility = sum of the rewards received
- Here: **r(s) = –0.04** ∀s except
  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 $\langle S, A, T, R \rangle$:

- **S** is the set of possible system states (arbitrary finite set);
- **A** is the set of allowable actions (arbitrary finite set);
- **T**: $S \times A \times S \rightarrow [0,1]$ is the **transition probability function**; $p(s'|s,a)$
- **R**: $S \times A \rightarrow \Re$ is the **reinforcement function**;

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);

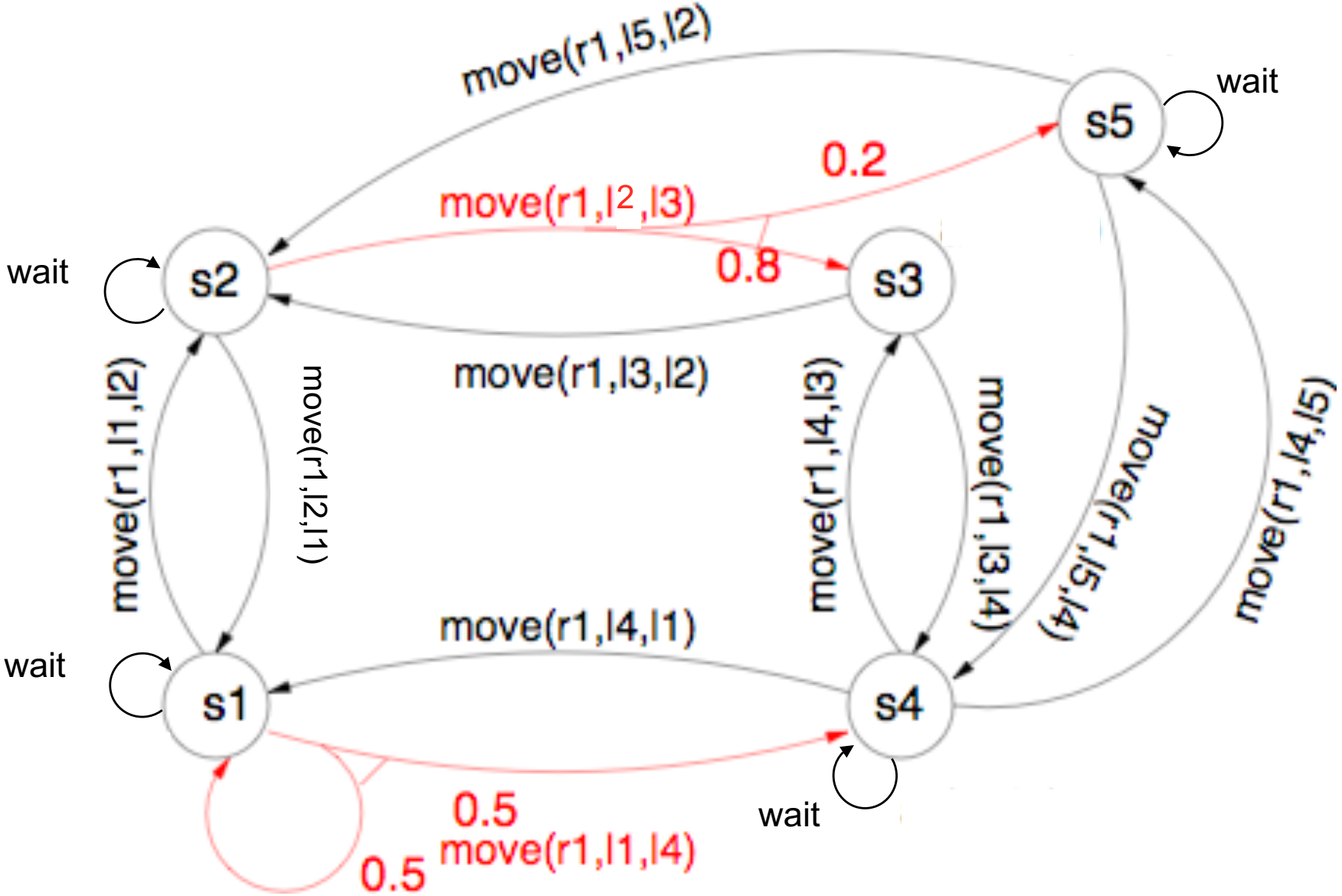
    r(s,a);

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

Why "Markov"?

- The qualifier *Markov* is used because the transition probability function **T** 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: X_t depends on <u>bounded</u> subset of $X_{0:t-1}$**
- <u>First-order Markov process</u>: $\mathbf{p(X_t|X_{0:t-1}) = p(X_t|X_{t-1})}$

Example of an MDP: robot r1



Example of an MDP

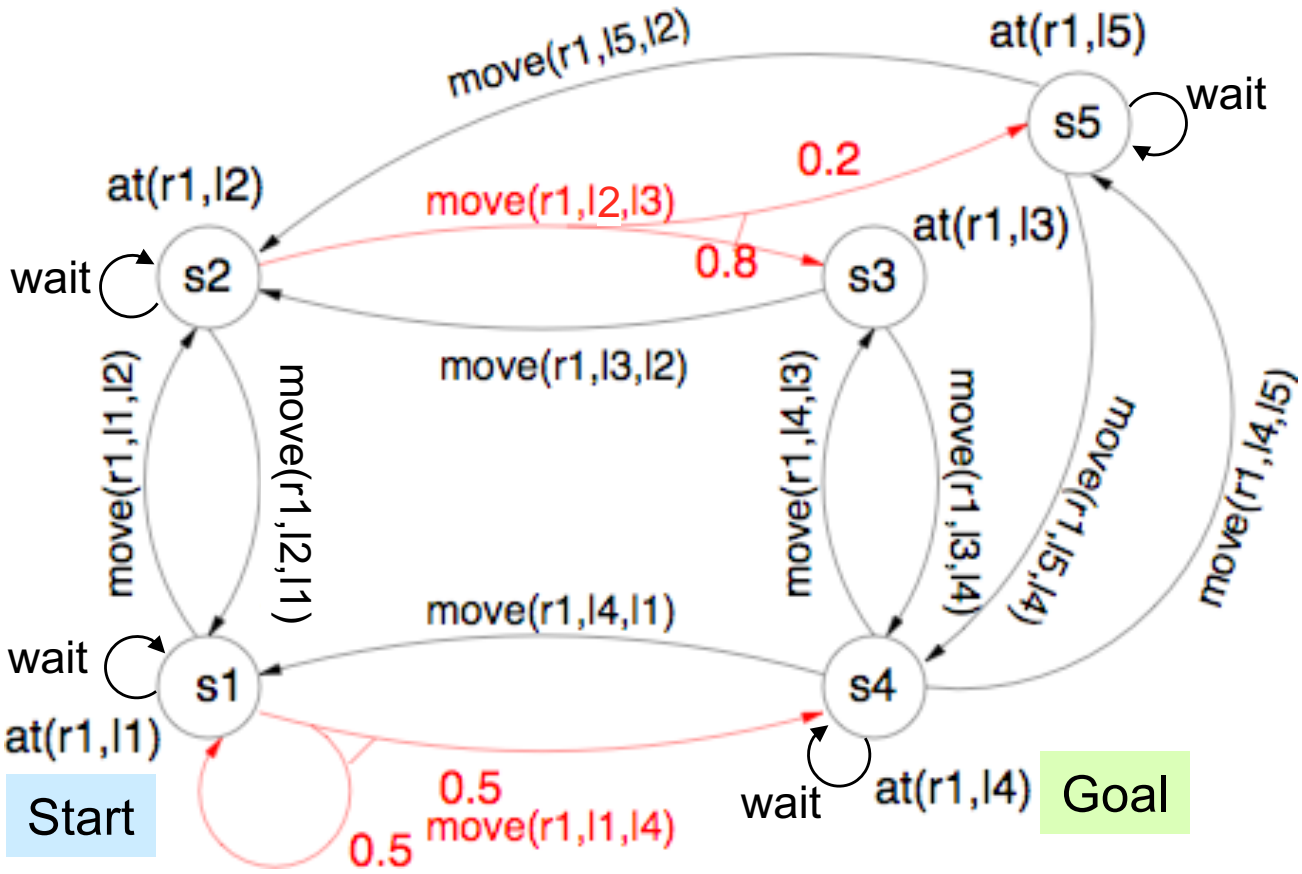
- A = {move(r1,l1,l2), move(r1,l2,l1), move(r1,l4,l1), move(r1,l1,l4), move(r1,l3,l2), move(r1,l2,l3), move(r1,l5,l2), move(r1,l4,l3), move(r1,l3,l4), move(r1,l5,l4), move(r1,l4,l5), wait}
- S = {s1, s2, s3, s4, s5}
- p(s1,move(r1,l1,l4),s4)=0.5; p(s1,move(r1,l1,l4),s1)=0.5; p(s2,move(r1,l2,l3),s3)=0.8; 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!

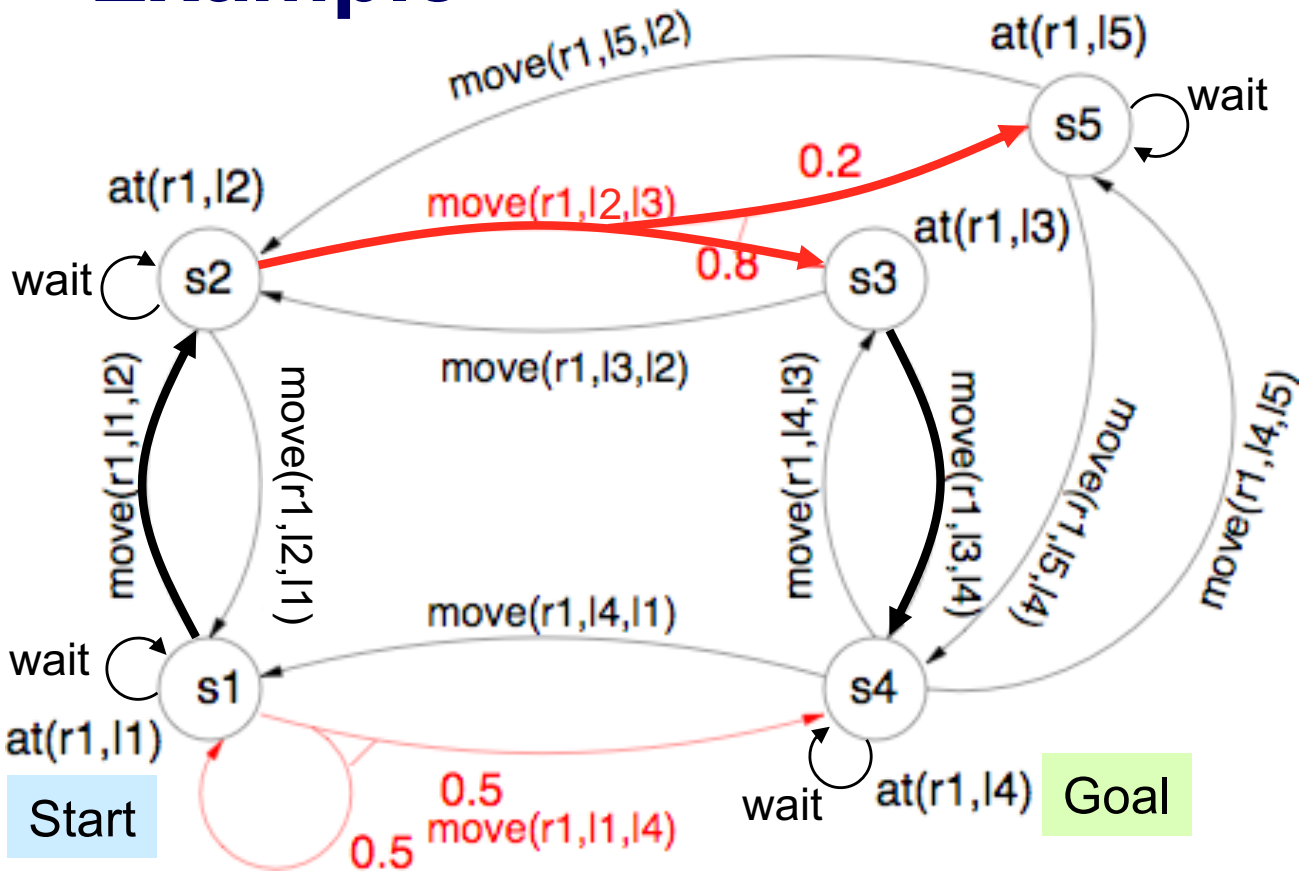
Example

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



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.,

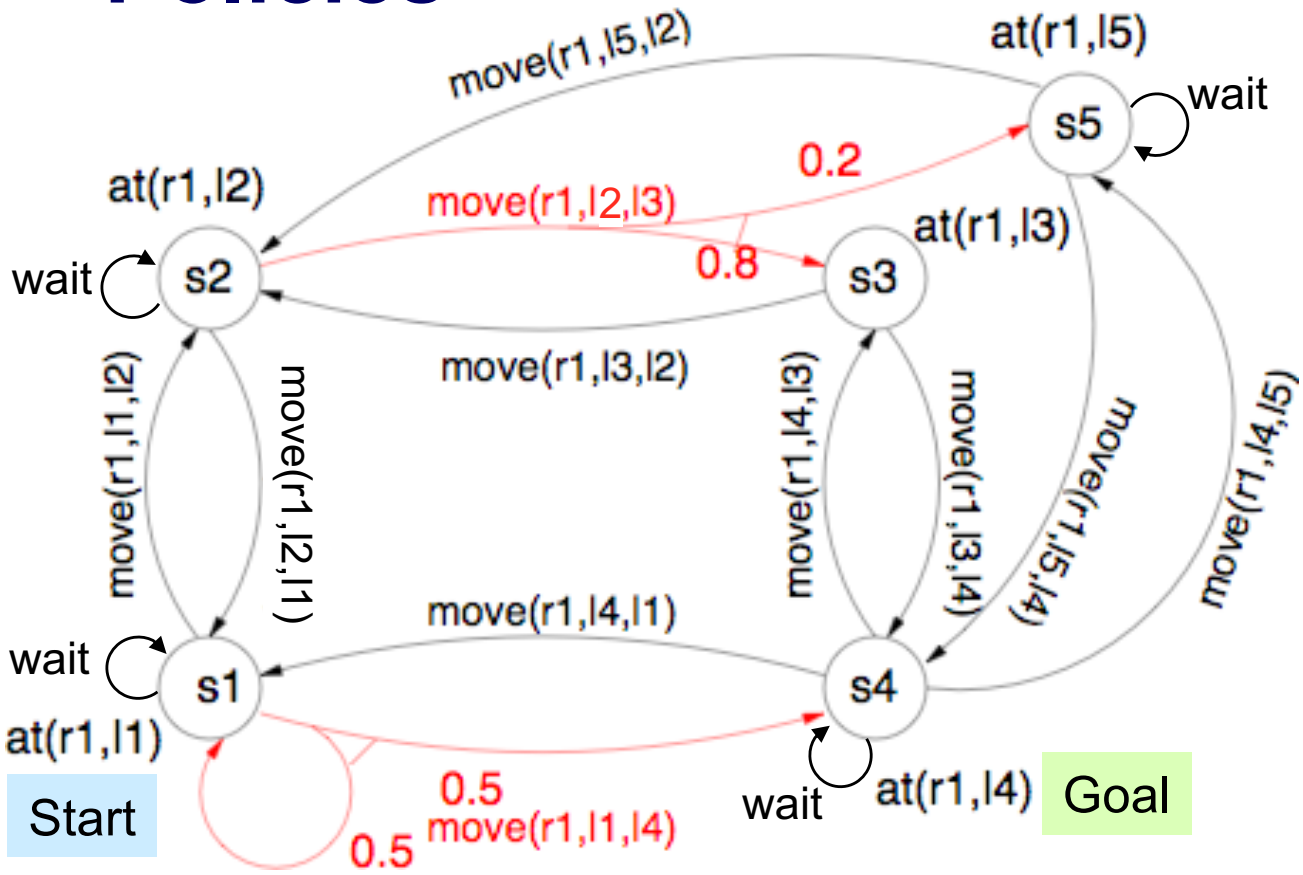
Plan: ⟨move(r1,l1,l2), move(r1,l2,l3), move(r1,l3,l4)⟩

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



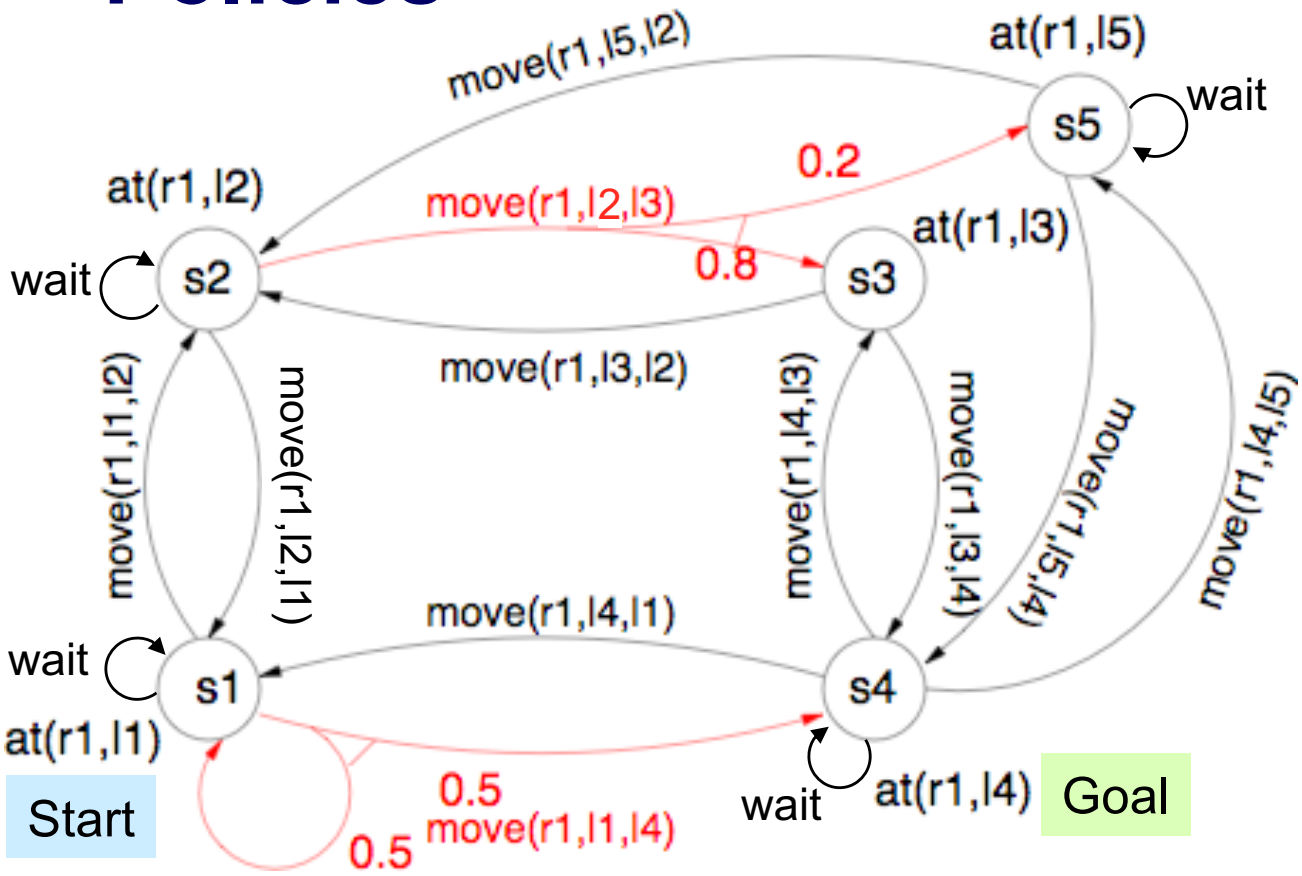
- Policy: a function that maps states into actions
- Write it as a set of state-action pairs

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))}

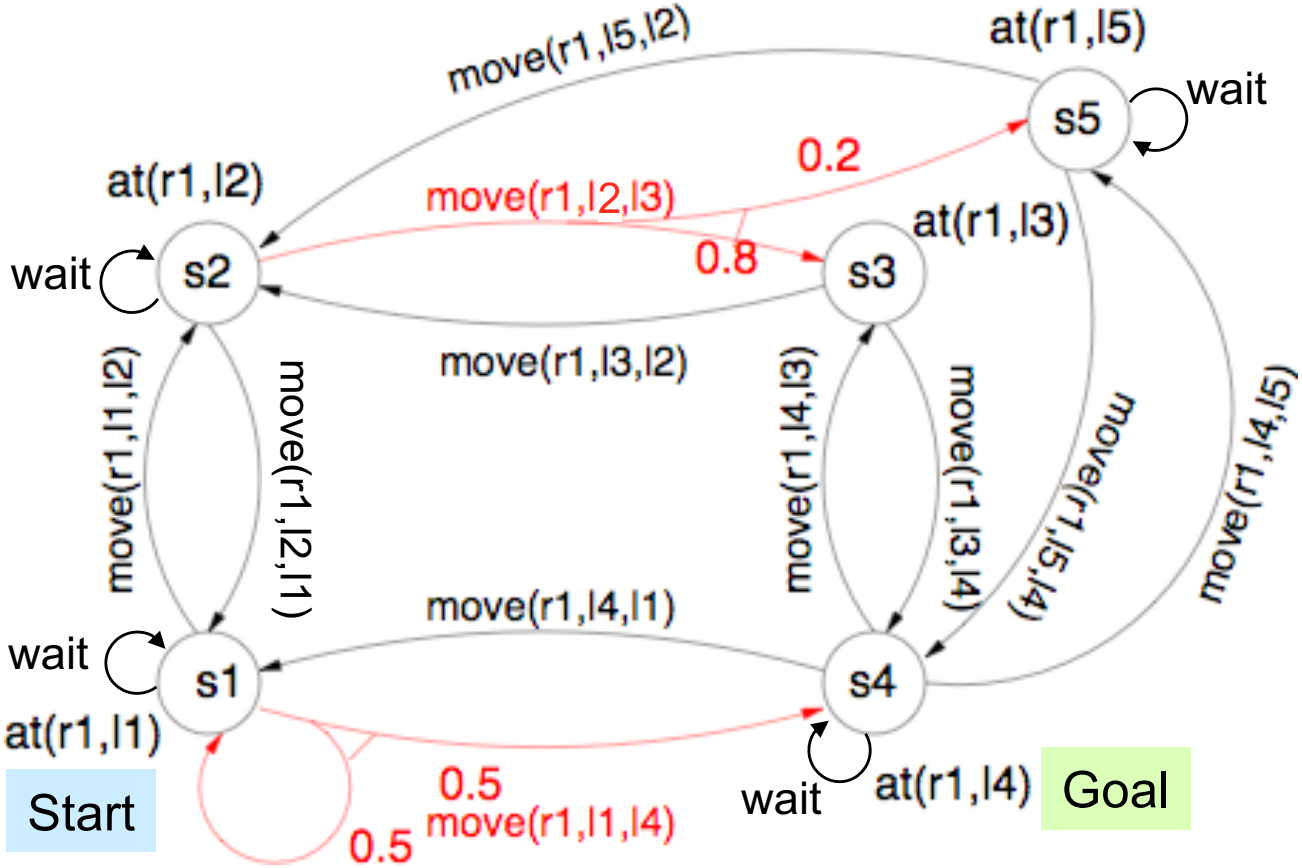
π_3 = {(s1, move(r1,l1,l4)),
(s2, move(r1,l2,l1)),
(s3, move(r1,l3,l4)),
(s4, wait),
(s5, move(r1,l5,l4)}



- 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

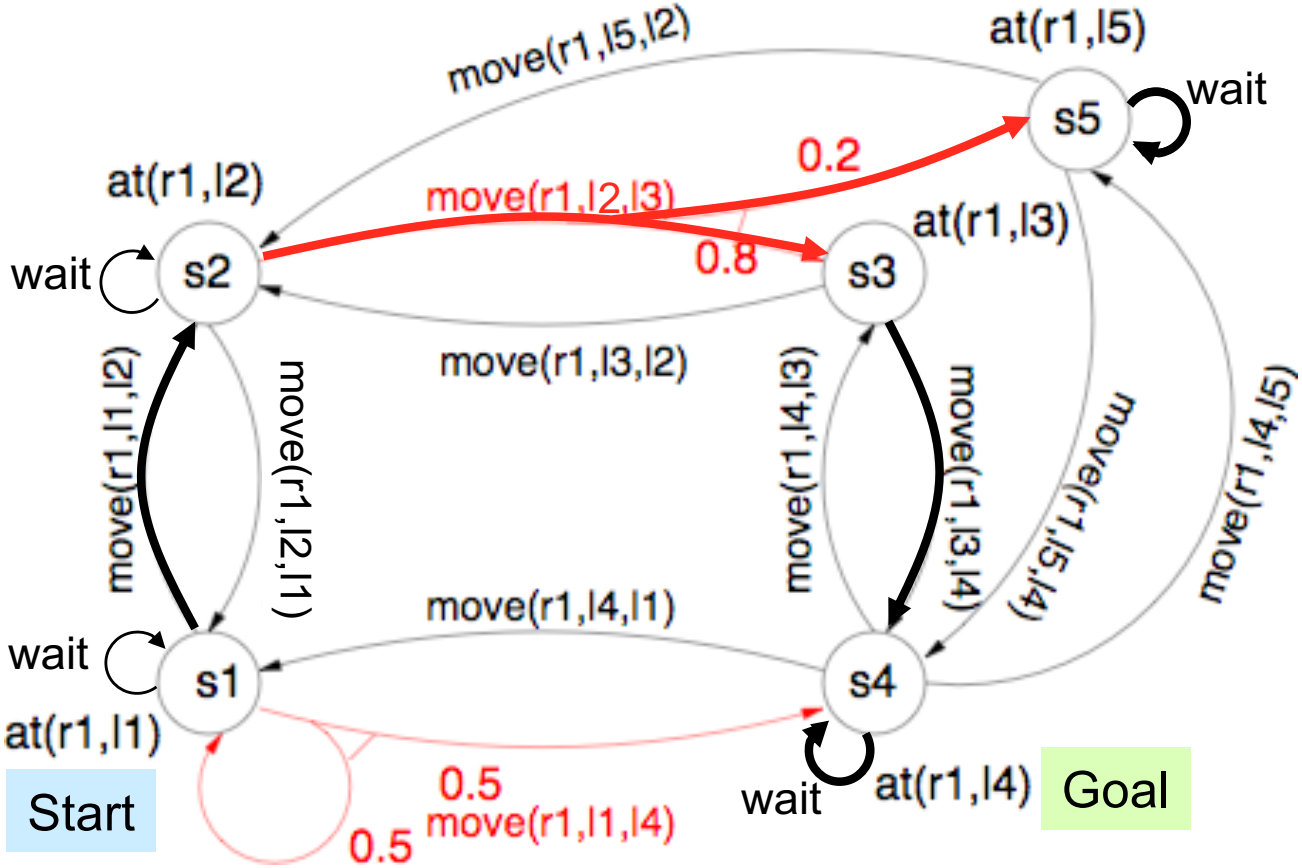
- Each time a given policy is executed starting from the initial state, the stochastic nature of the environment will lead to a different environment history.

- **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

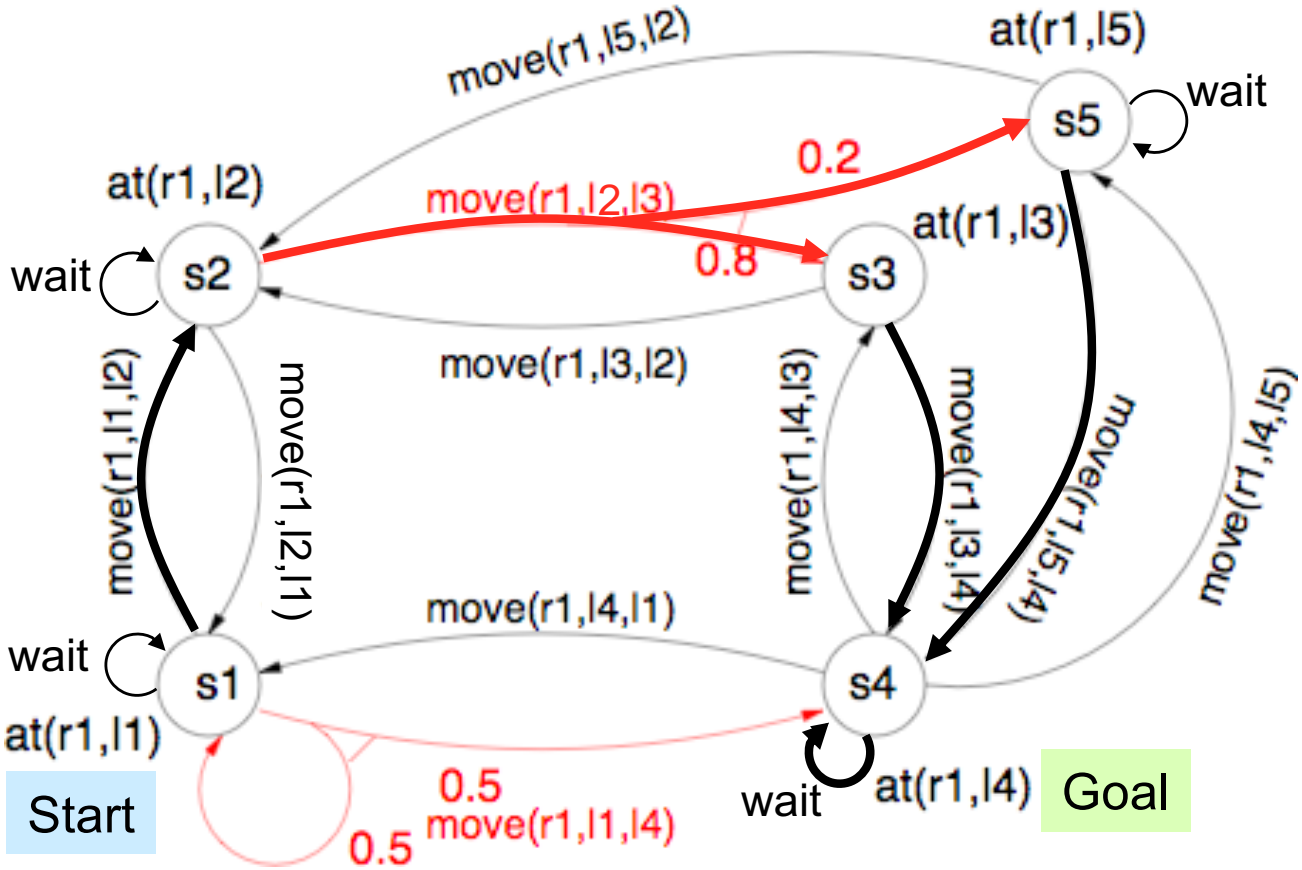
$h_1 = \langle$s1, s2, s3, s4, s4, … $\rangle$

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

$$P(h_1 \mid \pi_1) = 1 \times 1 \times 0.8 \times 1 \times \ldots = 0.8$$
$$P(h_2 \mid \pi_1) = 1 \times 1 \times 0.2 \times 1 \times \ldots = 0.2$$
$$P(h \mid \pi_1) = 0 \text{ for all other } h$$

Example (continued)

$\pi_2 = \{$(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, … $\rangle$

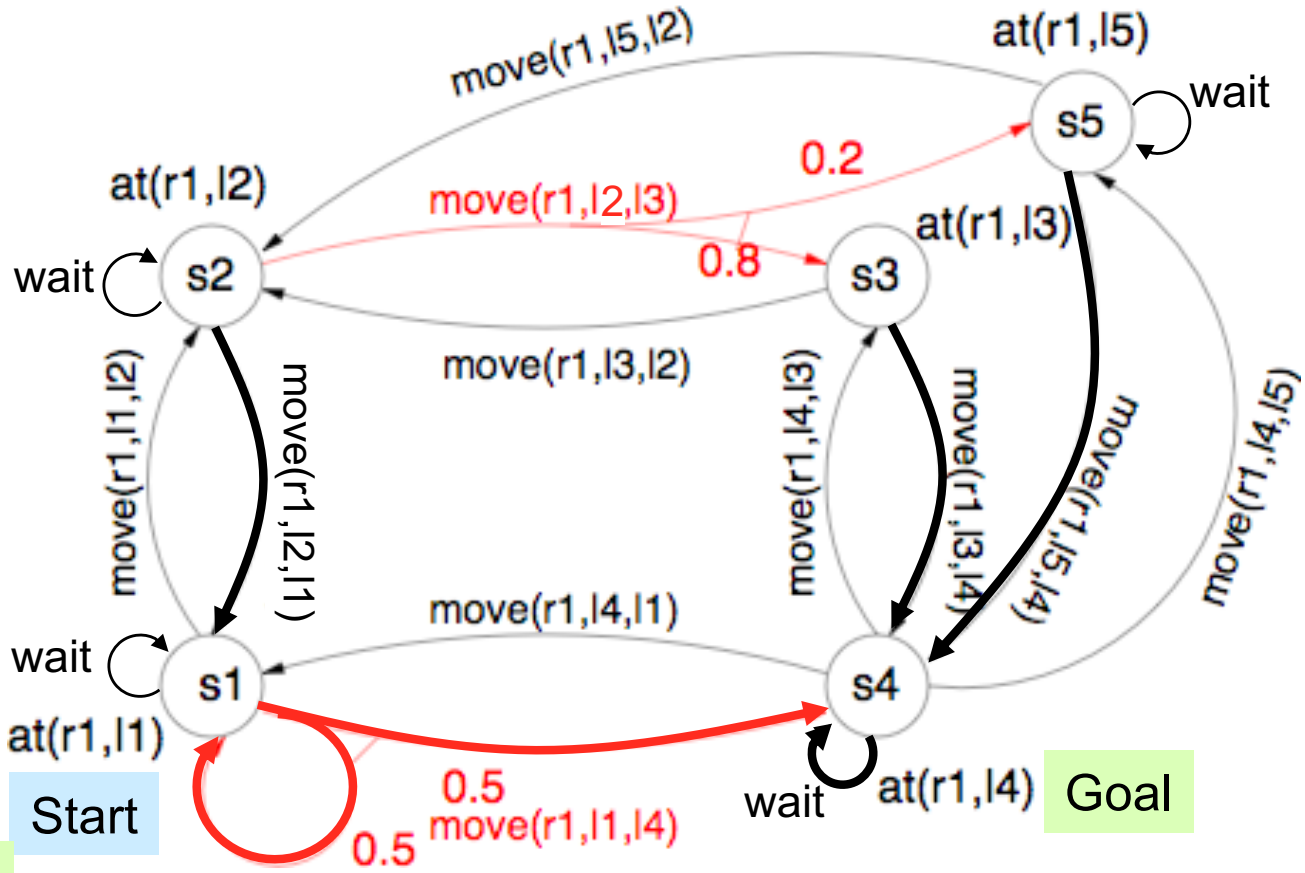
$P(h_1 \mid \pi_2) = 1 \times 0.8 \times 1 \times \ldots = 0.8$

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

$P(h \mid \pi_2) = 0$ 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, … $\rangle$ $P(h_4 \mid \pi_3) = 1 \times 0.5 \times 1 \times 1 \times 1 \times \ \ldots = 0.5$

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

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

…

$h_7 = \langle$s1, s1, s1, s1, s1, s1, … $\rangle$ $P(h_7 \mid \pi_3) = 1 \times 0.5 \times 0.5 \times 0.5 \times 0.5 \times \ldots = 0$

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:

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

Discounted reinforcements:

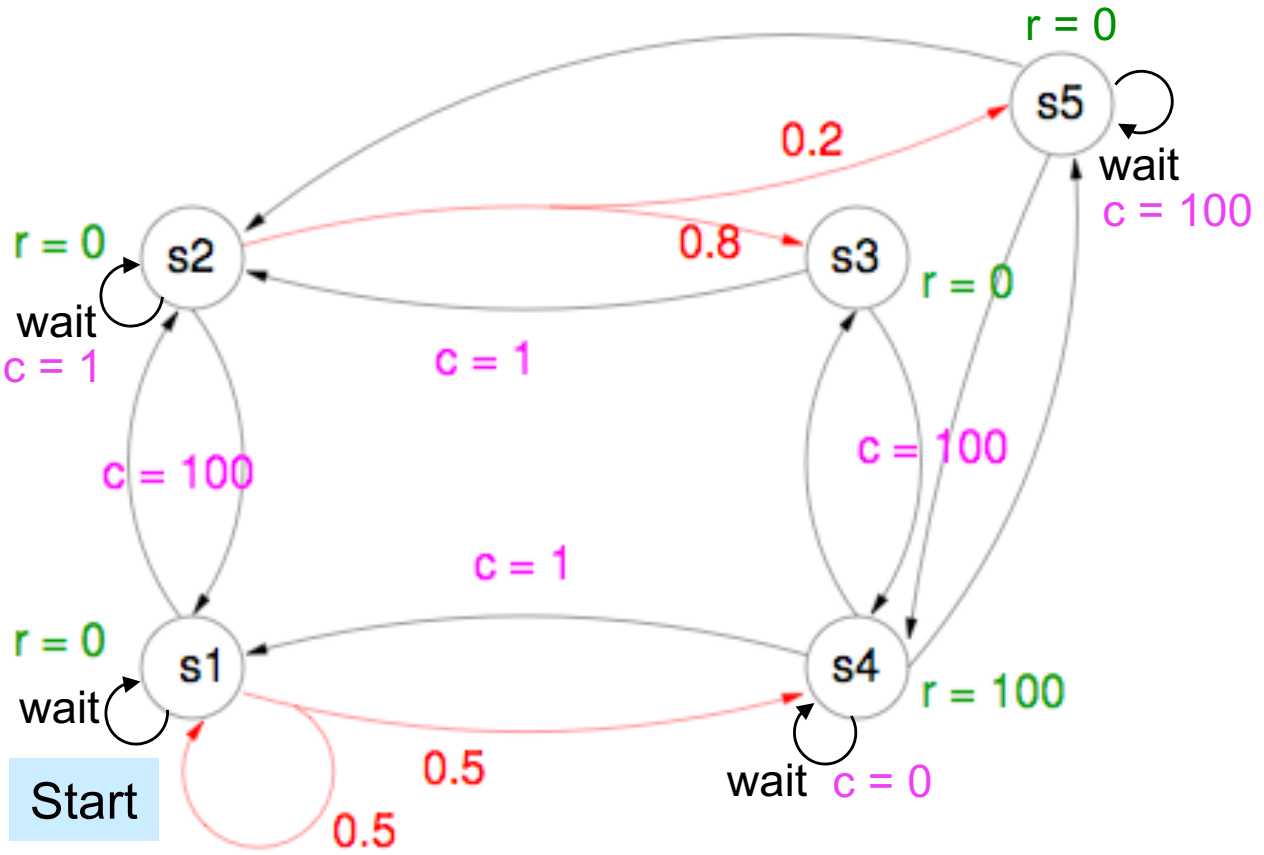
$$V_\pi(h) = \mathrm{r}(s_0) + \gamma \mathrm{r}(s_1) + \gamma^2 \mathrm{r}(s_2) + \gamma^3 \mathrm{r}(s_3) + \ldots$$

- A **discount factor** γ**:** $\quad 0 \le \gamma \le 1$ It determines how much the agent cares about rewards in the distant future relative to those in the immediate future.

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

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



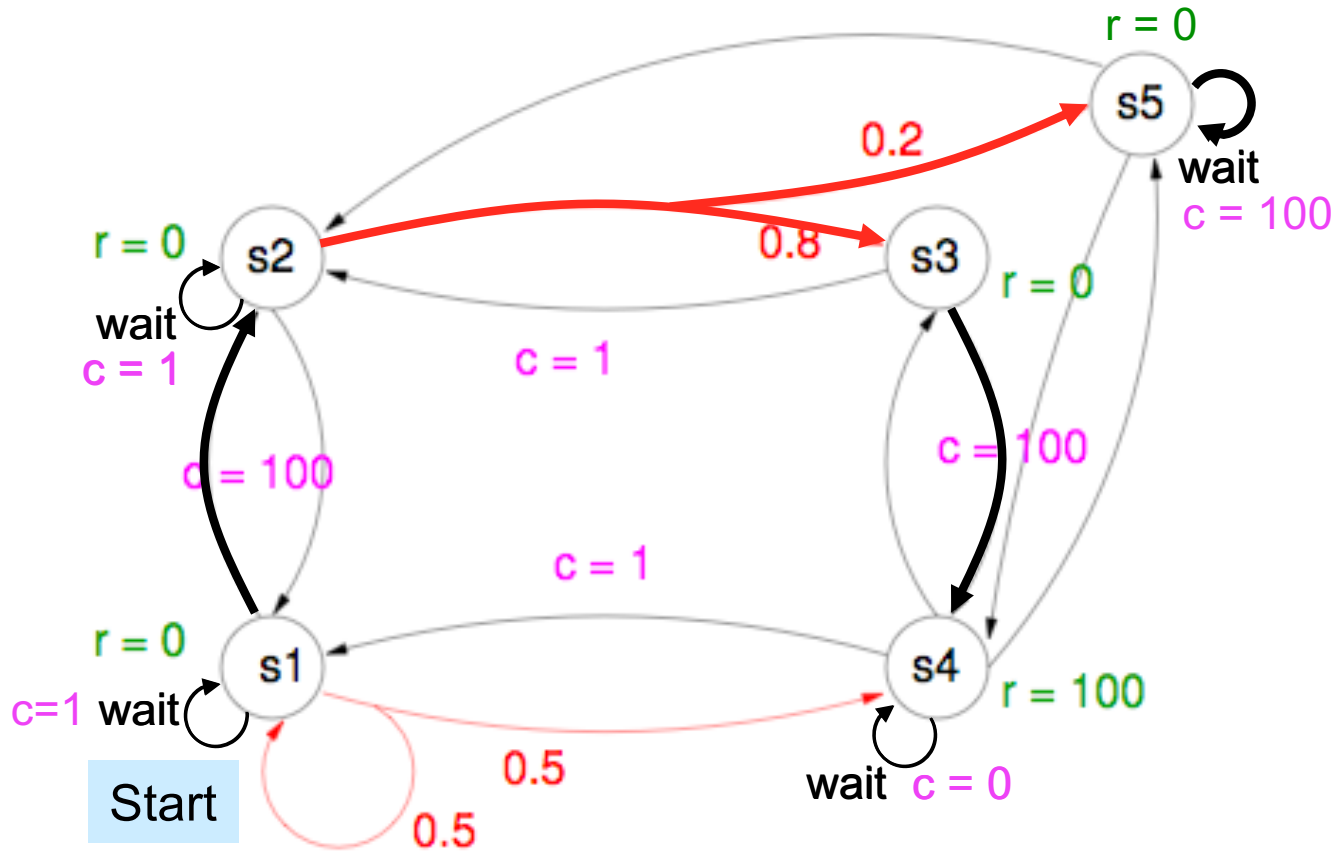
- $C(s_1,wait) = 1$; $C(s_2,wait) = 1$; $C(s_4,wait) = 0$; $C(s_5,wait) = 100$
- R as shown: $r(s_1)=r(s_2)=r(s_3)=r(s_5)= 0$; $r(s_4) = 100$

- Utility function: **generalization of a goal** (additive rewards)
  - If $h = \langle s_0, s_1, \ldots \rangle$, then $V_\pi(h) = \sum_{i \geq 0} \gamma^i (\mathrm{R}(s_i) - \mathrm{C}(s_i, \pi(s_i)))$

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 maximizes the expected utility of the ***subsequent*** state:

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

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

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 **T** when it starts out

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

Estimate a **value function** directly

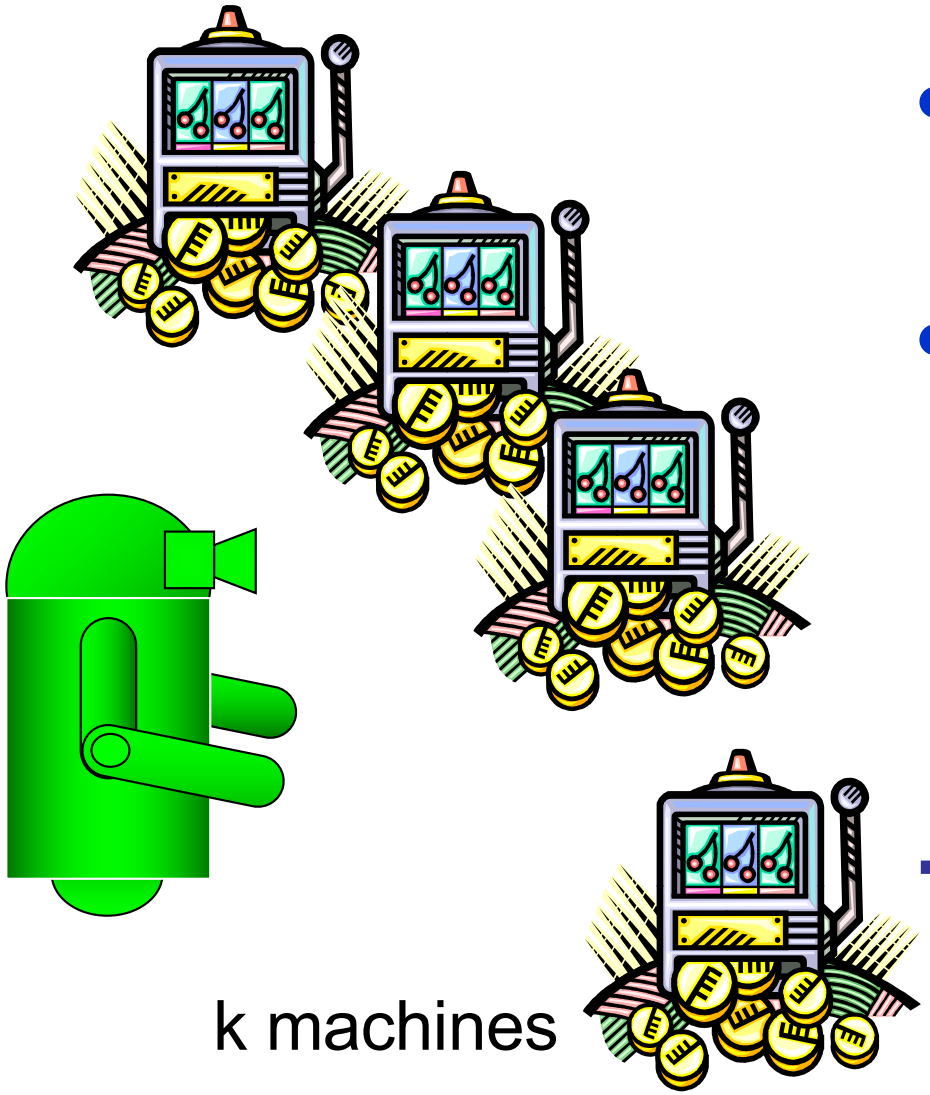
➔ ***We'll investigate an algorithm for doing that.***

RL and the learning agent

- How the agent should choose its actions?
  - There are two, 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** { ε: Exploration; 1– ε: Exploitation

MDP

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

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

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

- Optimal policy:

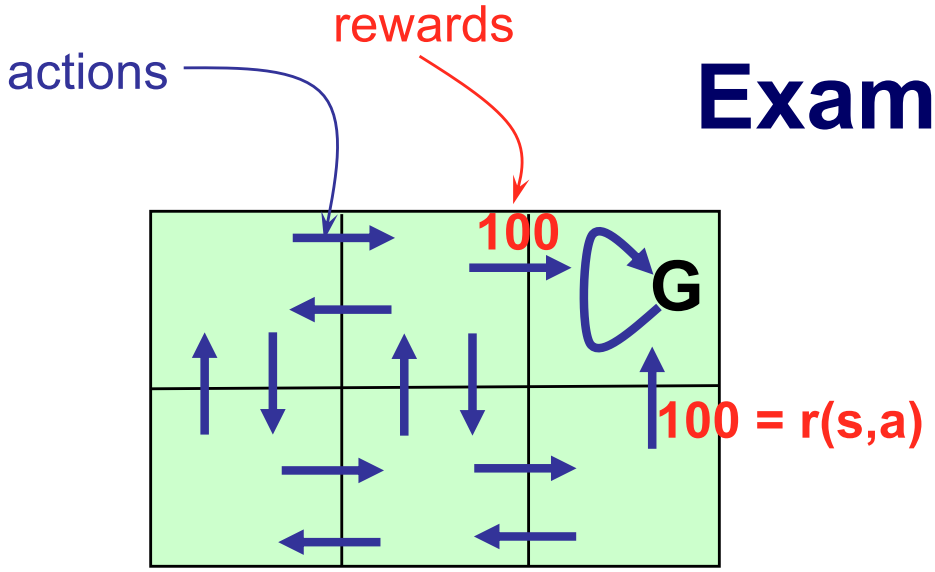
Notation: $V^{\pi^*}(s) = V^*(s)$

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

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

$Q^*(s,a)$

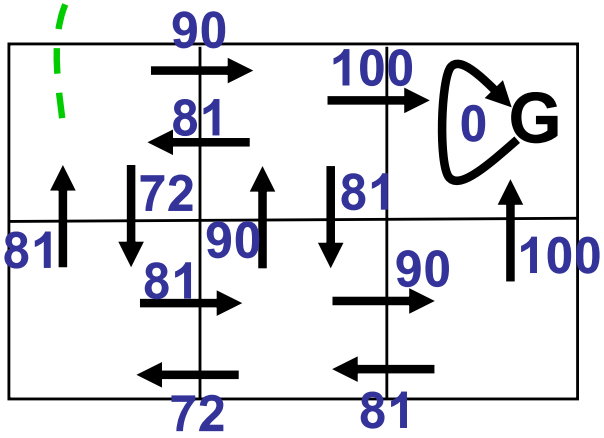
Example



G: absorbing state

$\gamma = 0.9$

r(s,a)=0 otherwise

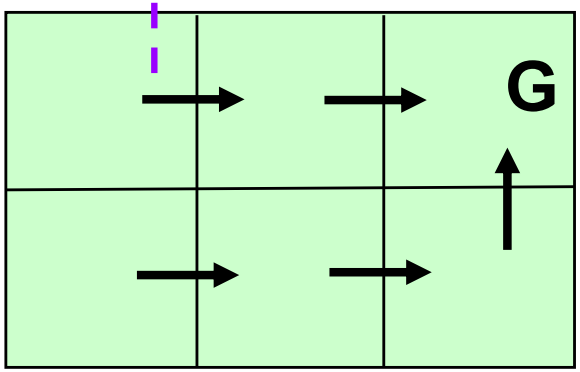


Q(s,a) values

90	100	0
81	90	100

V*(s) values

actions ↓ and →, choose →



One optimal policy

Optimal policy: Right, Up ➔ $0+\gamma 100+ \gamma^2 0+ \gamma^3 0+\ldots=90$

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 optimally.

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

  - 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 $\boldsymbol{s}_t$
do forever
select an action $\boldsymbol{a}_t$ and execute it in $\boldsymbol{s}_t$
receive immediate reward $\boldsymbol{r(s_t,a_t)}$
observe the new state $\boldsymbol{s}_{t+1}$
update Q(s,a) as follows:

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

$\boldsymbol{s}_t \leftarrow s_{t+1}$

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 α is, the longer term the average is. With a small α, 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 α, 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 Algorithm

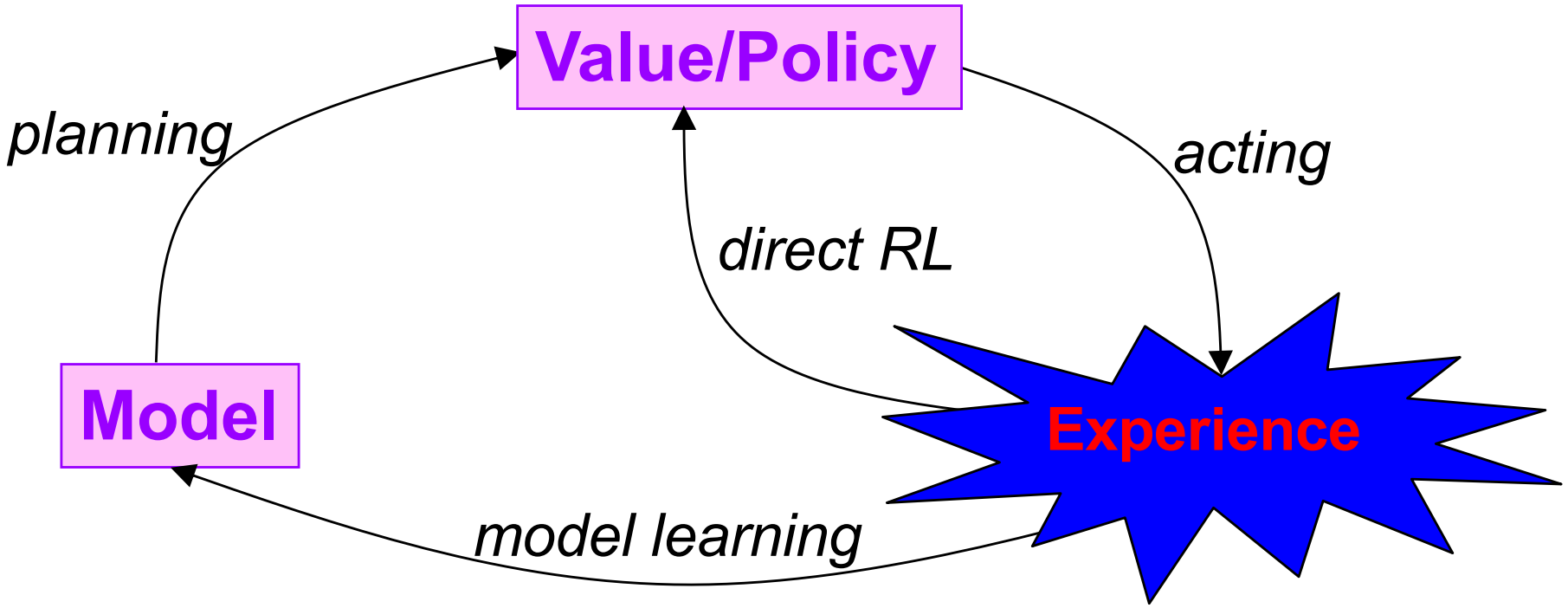
- **Guaranteed to converge to Q***
  - The optimal Q function is achieved **if** the world is really an MDP, **if** we manage the learning rate correctly, and **if** we explore the world in such a way that we never completely ignore some actions and states.

Applications

- **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 Framework for 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:
  - we use the known model to build a simulation.
  - 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!

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.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

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

Chapter 13: Machine Learning, Tom Mitchell