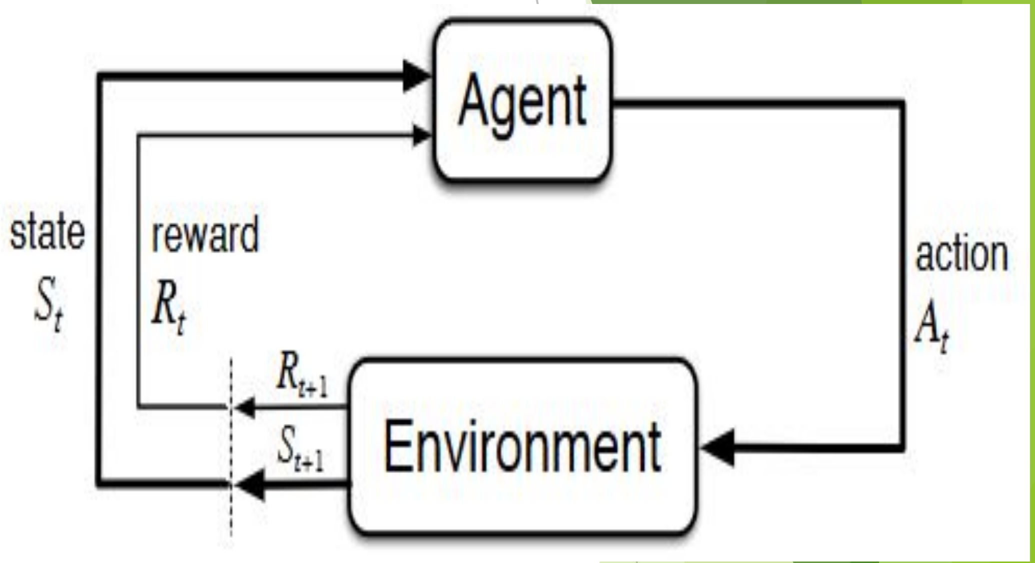
## Markov Decision Processes

- As in all of artificial intelligence, there is a tension between breadth of applicability and mathematical tractability
- This chapter introduce this tension and discuss some of the trade-offs and challenges that it implies
- A Markov decision process (MDP) is a stochastic decision making process that uses a mathematical framework to model the decision making of a dynamic system
- It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time
- MDPs evaluate, which actions the decision maker should take, considering the current state and environment of the system
- MDPs rely on variables such as the environment, agent’s actions and rewards to decide the system’s next optimal action

# The Agent–Environment Interface

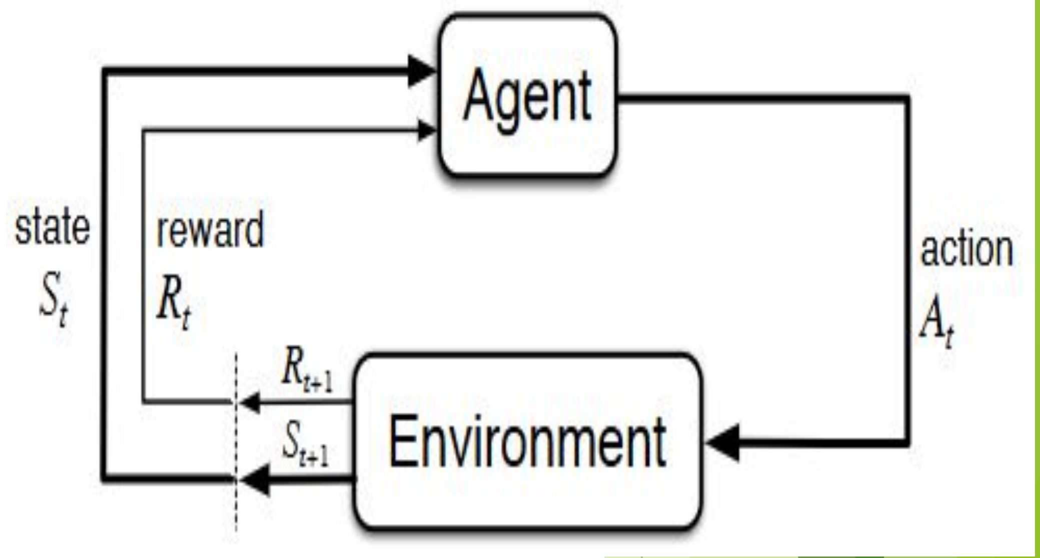
- The **reinforcement learning** problem is framing of the problem of **learning from interaction** to achieve a goal
- The **learner** and **decision-maker** is called the **agent**
- The thing it interacts with, **comprising everything outside the agent**, is called the **environment**
- These interact continually, the agent selecting **actions** and the **environment responding** to those actions and presenting new situations to the **agent rewards**, special numerical values that the agent tries to maximize over time



The agent– environment interaction

## The Agent–Environment Interface

- Markov decision processes describe a fully observable environment for reinforcement learning
- A **complete specification** of an environment defines a **task**
- The agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3, . . .
- At each time step t, the agent receives some representation of the **environment's state**, **St ∈ S**, where S is the set of possible states and on that basis selects an action, **At ∈ A(St)**, where A(St) is the set of actions available in state St
- One time step later, due to its action, the agent receives a numerical reward, Rt+1 ∈ R, and finds itself in a new state, St+1



The agent– environment interaction

## Cont…

- At each time step, the agent implements a mapping from states to probabilities of selecting each possible action
- This mapping is called the agent's policy and is denoted $\pi_t$ , where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$
- Reinforcement learning methods specify how the agent changes its policy as a result of its experience
- The agent's goal is to maximize the total amount of reward it receives over the long run
- Actions can be any decisions to learn how to make, and the states be anything that might be useful in making them
- The boundary between agent and environment is not the same as the physical boundary of an agent (robot's or animal's body)

## Cont…

- For example, the motors and mechanical linkages of a robot and its sensing hardware should be considered parts of the environment rather than parts of the agent
- Rewards, too, are computed inside the physical bodies of natural and artificial learning systems, but are considered external to the agent
- The reinforcement learning framework is a problem of goal-directed learning from interaction that can be achieved through three signals passing back and forth between an agent and its environment:
  - Signal to represent the choices made by the agent (the actions)
  - Signal to represent the basis on which the choices are made (the states)
  - Signal to define the agent’s goal (the rewards)

## Goals and Rewards

- In reinforcement learning, the purpose or goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent
- At each time step, the reward is a simple number, $R_t \in \mathbb{R}$
- The agent's goal is to maximize the total amount of reward it receives, maximizing not immediate reward, but cumulative reward in the long run
- So the goals and purposes can be the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)

## Goals and Rewards

- The use of a reward signal to formalize the idea of a goal
- The reward signal is not the place to impart to the agent prior knowledge about how to achieve but what to do
- For example, a chess-playing agent should be rewarded only for actually winning, not for achieving sub goals such taking its opponent's pieces or gaining control of the center of the board
- The reward signal is the way of communicating to the agent what is to achieve, not how to achieved

## Returns

- The agent's goal is to maximize the cumulative reward it receives in the long run
- If the sequence of rewards received after time step t is denoted Rt+1, Rt+2, Rt+3, . . .  to maximize the expected return, where the return Gt is defined as function of the reward sequence
- The return is the sum of the rewards: $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$

  where T is a final time step
- At T the agent– environment interaction breaks naturally into subsequences, which we call episodes such as plays of a game, trips through a maze, or any repeated interactions

## Cont…

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

- At T the agent– environment interaction breaks naturally into subsequences, which we call episodes such as plays of a game, trips through a maze, or any repeated interactions
- Each episode ends in a special state called the terminal state, followed by a reset to a standard starting state
- Tasks with episodes are called episodic tasks
- In episodic tasks it is required to distinguish the set of all non terminal states, denoted S, from the set of all states plus the terminal state, denoted S+
- In many cases the agent–environment interaction does not break naturally into identifiable episodes, but goes on continually without limit
  - For example, an application to a robot with a long life span, called continuing tasks

## Cont…

- For example, an application to a robot with a long life span, called continuing tasks
- The return formulation is problematic for continuing tasks because the final time step would be T = ∞, and the return, could easily be infinite
- The agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized
- It chooses At to maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate

## Cont…

- The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only $\gamma^{k-1}$ times what it would be worth if it were received immediately
- If $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence {Rk} is bounded
- If $\gamma = 0$, the agent is “myopic” in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose At so as to maximize only Rt+1
- If each of the agent’s actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize by separately maximizing each immediate reward

## Cont…

- But in general, acting to maximize immediate reward can reduce access to future rewards so that the return may actually be reduced
- As $\gamma$ approaches 1, the objective takes future rewards into account more strongly

## The Markov Property

- The property of environments and their state signals is called the Markov property
- The state, given by some preprocessing system, is part of the environment
- The state signal should include immediate sensations, such as sensory measurements, but it can contain much more information
- State representations can be highly processed versions of original sensations, or they can be complex structures built up over time from the sequence of sensations
- For example: Move eyes over a scene, with only a tiny spot corresponding to detail at any one time, build up a rich and detailed representation of a scene

## Cont…

- In this case the state is constructed and maintained on the basis of immediate sensations together with the previous state or some other memory of past sensations
- The state signal should not be expected to inform the agent of everything about the environment, or everything that would be useful to it in making decisions
- For example: If the agent is a paramedic called to a road accident, it is not expect that it know immediately the internal injuries of an unconscious victim
- In this case there is hidden state information in the environment, and that information would be useful if the agent knew it, but the agent cannot know it because it has never received any relevant sensation

## Cont…

- Ideally, a state signal requires to summarizes past sensations compactly so that all relevant information is retained
- A state signal that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property
- For example, a checkers position—the current configuration of all the pieces on the board, serve as a Markov state because it summarizes everything important about the complete sequence of positions
- In the most general, causal case the response may depend on everything that has happened earlier

## Cont…

- In this case the dynamics can be defined only by specifying the complete probability distribution as:

$$\Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$$

for all r, s′, and all possible values of the past events: S0, A0, R1, ..., St−1, At−1, Rt, St, At

- If the state signal has the Markov property, then the environment’s response at t + 1 depends only on the state and action representations at t, in which case the environment’s dynamics can be defined as:

$$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\}$$

for all r, s′, St, and At

## Cont…

- If an environment has the Markov property, then its one-step dynamics enable to predict the next state and expected next reward given the current state and action
- Markov states provide the best possible basis for choosing actions
- The Markov property is important in reinforcement learning because decisions and values are assumed to be a function only of the current state

## Bioreactor

- Suppose reinforcement learning is being applied to determine moment-by-moment temperatures and stirring rates for a bioreactor (a large vat of nutrients and bacteria used to produce useful chemicals)
- The target temperatures and target stirring rates that are passed to lower-level control systems that, in turn, directly activate heating elements and motors to attain the targets
- The thermocouple and other sensors used for readings of temperatures, stirring rates, Chemical concentration, Nutrient levels, Bacterial population density and Target chemical
- Find possible state, action and reward

## Bioreactor

- **Possible state, action and reward**
- Here each state is a list, or vector, of sensor readings and symbolic inputs
- Each action is a vector consisting of a target temperature and a stirring rate
- The rewards might be moment-by-moment measures of the rate at which the useful chemical is produced by the bioreactor and are always single numbers

## Pick-and-Place Robot

- Consider using reinforcement learning to control the motion of a robot arm in a repetitive pick-and-place task
- If we want to learn movements that are fast and smooth, the learning agent will have to control the motors directly and have low-latency information about the current positions and velocities of the mechanical linkages
- Find possible state, action and reward

## Pick-and-Place Robot

- The possible state, action and reward
- The states be the latest readings of joint angles and velocities
- The actions be the voltages applied to each motor at each joint
- The reward might be +1 for each object successfully picked up and placed
- Negative reward for jerkiness
- Penalty for failure to pick/place

## Markov Decision Processes

- A state signal that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property
- A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP. If the state and action spaces are finite, then it is called a finite Markov decision process
- A particular finite MDP is defined by its state and action sets and by the one-step dynamics of the environment
- Given any state and action s and a, the probability of each possible pair of next state and reward, s′, r, is denoted

$$p(s', r | s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}.$$

## Cont…

- The expected rewards for state–action pairs,

$$r(s,a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a).$$

- The state-transition probabilities,

$$p(s'|s,a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a),$$

- The expected rewards for state–action–next-state triples

$$r(s,a,s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r\, p(s', r | s, a)}{p(s'|s,a)}.$$

## Example: Recycling Robot MDP

- A mobile robot has the job of collecting empty soda cans in an office environment
- It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin
- It runs on a rechargeable battery
- The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper
- High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery

## Cont…

- This agent has to decide whether the robot should:
  - (1) actively search for a can for a certain period of time
  - (2) remain stationary and wait for someone to bring it a can
  - (3) head back to its home base to recharge its battery
- Suppose the environment works as follows
- The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not
- Whenever the robot is searching, the possibility exists that its battery will become depleted
- The agent makes its decisions as a function of the energy level of the battery
- It can distinguish two levels, high and low

## Cont…

- The state set is S = {high, low}
- The possible decisions—the agent’s actions— wait, search, and recharge
- The agent’s action sets are:
- A(high) = {search, wait}
- A(low) = {search, wait, recharge}
- If the energy level is high, then active search can be completed without risk of depleting the battery
- A period of searching that begins with a high energy level leaves the energy level high with probability $\alpha$ and reduces it to low with probability $1 - \alpha$

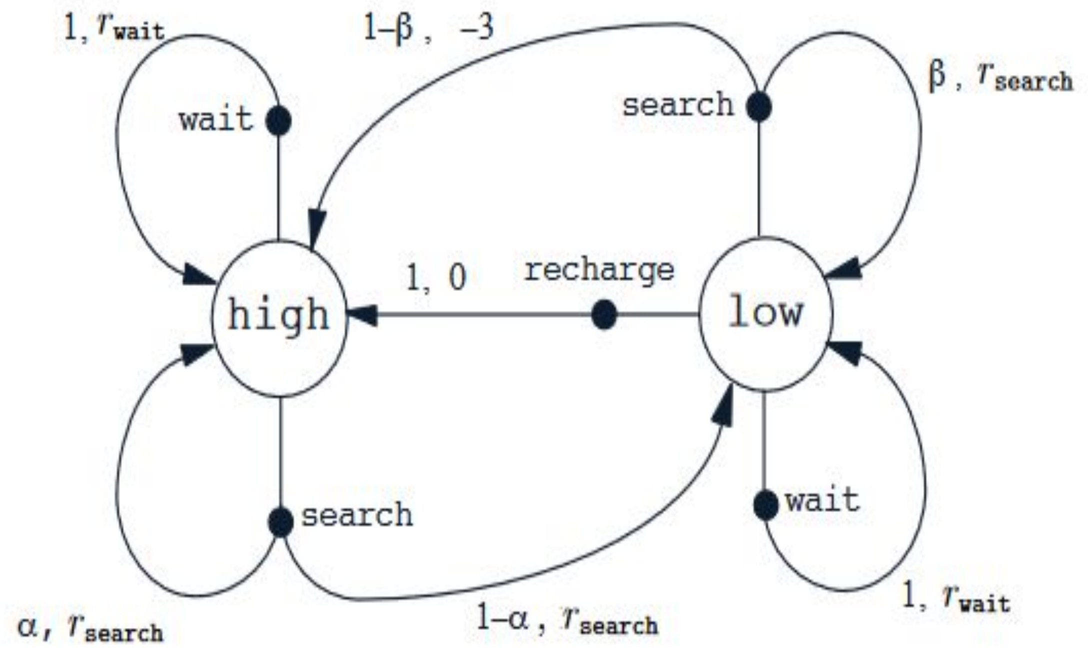
## Cont…

- A period of searching undertaken when the energy level is low leaves it low with probability ß and depletes the battery with probability 1 − ß
- In this case the robot must be rescued, and the battery is then recharged back to high
- Each can collected by the robot counts as a unit reward, whereas a reward of −3 results whenever the robot has to be rescued
- Let ‘r’search and ‘r’wait, with ‘r’search > ‘r’wait, respectively denote the expected number of cans the robot will collect (and hence the expected reward) while searching and while waiting

## Transition Graph

- A transition graph is a useful way to summarize the dynamics of a finite MDP
- There are two kinds of nodes: state nodes and action nodes

| $s$ | $s'$ | $a$ | $p(s'\|s,a)$ | $r(s,a,s')$ |
|---|---|---|---|---|
| high | high | search | $\alpha$ | $r_{\texttt{search}}$ |
| high | low | search | $1-\alpha$ | $r_{\texttt{search}}$ |
| low | high | search | $1-\beta$ | $-3$ |
| low | low | search | $\beta$ | $r_{\texttt{search}}$ |
| high | high | wait | 1 | $r_{\texttt{wait}}$ |
| high | low | wait | 0 | $r_{\texttt{wait}}$ |
| low | high | wait | 0 | $r_{\texttt{wait}}$ |
| low | low | wait | 1 | $r_{\texttt{wait}}$ |
| low | high | recharge | 1 | 0 |
| low | low | recharge | 0 | 0. |



## Value Functions

- Functions of states (or of state–action pairs) that estimate how good it is for the agent to be in a given state
- The value of a state is the expected return starting from that state; depends on the agent’s policy
- The value of a state ‘s’ under a policy ‘π’, denoted Vπ(s), is the expected return when starting in ‘s’ and following ‘π’ thereafter

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

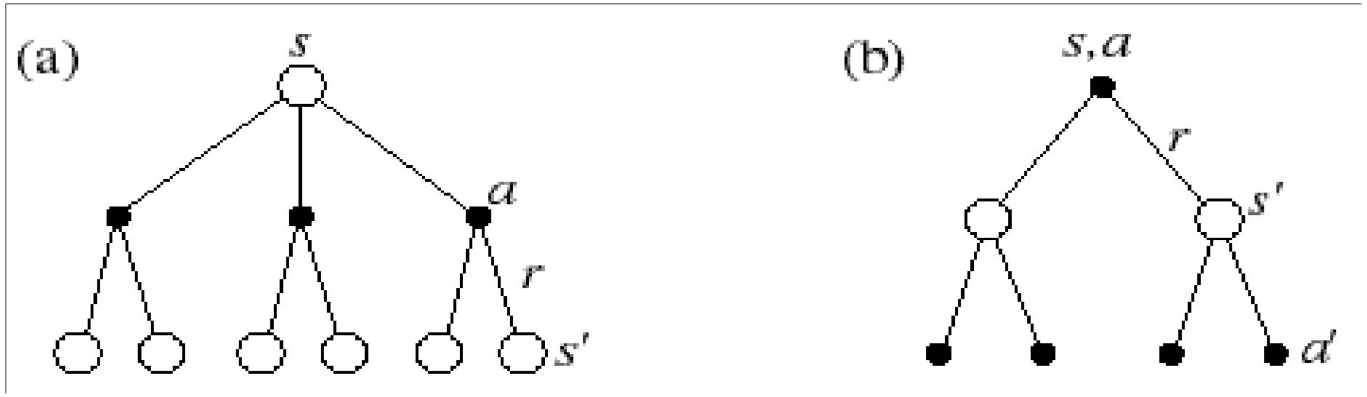
# Bellman Equation for a Policy

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big],
\end{aligned}
$$

## Cont…

- Bellman Equation expresses a relationship between the value of a state and the values of its successor states
- This is a set of linear equations, one for each state
- This reduces the problem of finding the optimal state sequence and action to a graph search:

for $V^{\pi}$



for $Q^{\pi}$

## Action-Value Functions

- The value of taking an action ‘a’ in state ‘s’ under policy ‘π’ is the expected return, denoted by $Q^{\pi}(s, a)$,starting from that state, taking that action and following ‘π’:

**Action - value function for policy** $\pi$ :

$$Q^{\pi}(s, a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

**Note that the value of the terminal state, if any, is always zero**

## Optimal Value Functions

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if } V^{\pi}(s) \geq V^{\pi'}(s) \text{ for all } s \in S$$

- There is always at least one (and possibly many) policy that is better than or equal to all the others. This is an optimal policy, denoted as ‘π*’
- Optimal policies share the same optimal state-value function:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

- Optimal policies also share the same optimal action-value function:

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \text{ for all } s \in S \text{ and } a \in A(s)$$

- This is the expected return for taking action *a* in state *s* and thereafter following an optimal policy

# Bellman Optimality Equation for *V**

- The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$V^*(s) = \max_{a \in A(s)} Q^{\pi^*}(s,a)$$
$$= \max_{a \in A(s)} E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a\right\}$$
$$= \max_{a \in A(s)} \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^*(s')\right]$$

- *V** is the unique solution of this system of nonlinear equations
- The optimal action is again found through the maximization process:

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| s_t = s, a_t = a\right\}$$
$$= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right]$$

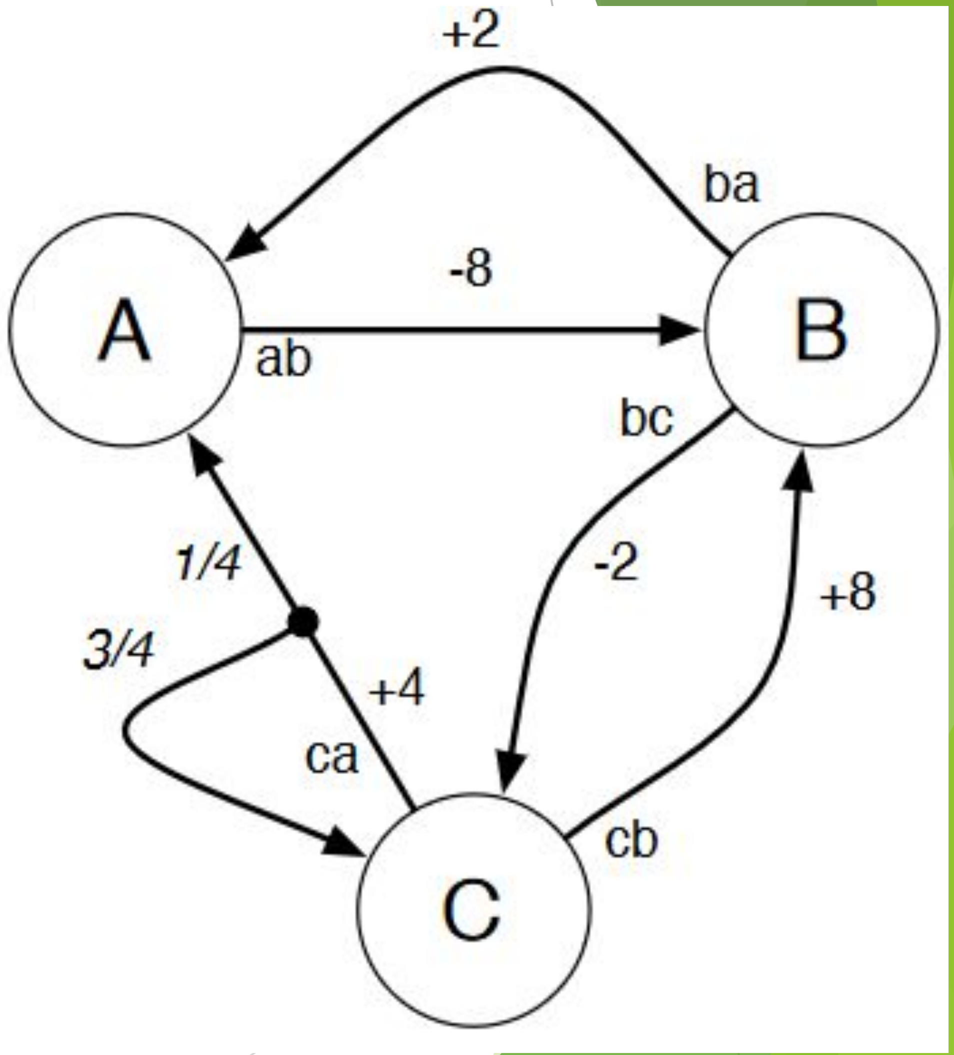
- *Q** is the unique solution of this system of nonlinear equations.

## Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - accurate knowledge of environment dynamics;
  - we have enough space an time to do the computation;
  - the Markov Property
- How much space and time do we need?
  - polynomial in number of states (via dynamic programming methods);
  - BUT, number of states is often huge (e.g., backgammon has about 10**20 states)
- We usually have to settle for approximations
- Many RL methods can be understood as approximately solving the Bellman Optimality Equation

# Example: 6

- Consider the Markov Decision Process (MDP) with discount factor γ = 0.5
- Upper case letters A, B, C represent states; arcs represent state transitions; lower case letters ab, ba, bc, ca, cb represent actions; signed integers represent rewards; and fractions represent transition probabilities
- Consider the uniform random policy π (s,a) that takes all actions from state ‘s’ with equal probability
- The initial value function of V1(A) = V1(B) = V1(C) =2
- Apply two iteration of iterative policy evaluation to compute a new value function V3(s)

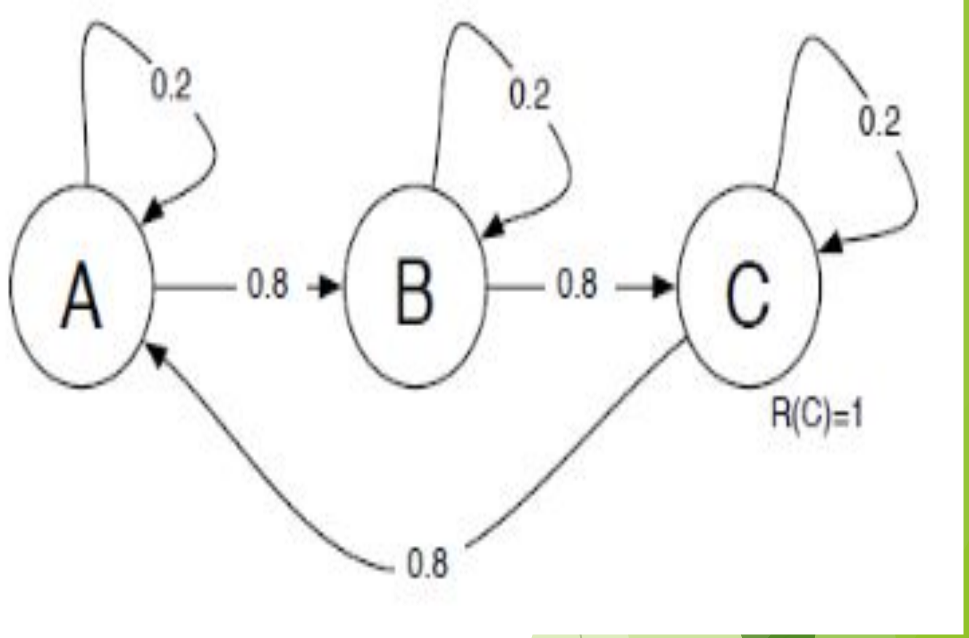


- The initial value function of V1(A) = V1(B) = V1(C) =2

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P_{ss'}^{a} \left[ R_{ss'}^{a} + \gamma V^{\pi}(s') \right]$$

## Example: 7

- Consider an MDP with three states, denoted as A, B, and C, arranged in a loop. The transitions and actions are as follows: In each state, there are two possible actions: "Moves" and "Stays." The probability of stay in same state is 0.2 and probability of move A to B, B to C and C to A is 0.8. A reward of 1 is received when the agent takes the "Moves" action in state C. All other transitions result in a reward of 0. Assume the agent starts in state A, discount factor (γ) = 0.9 and learning rate (α) = 1


A
B
C
0.2
0.2
0.2
0.8
0.8
0.8
R(C)=1