# A Guided Tour of Chapter 2: Markov Decision Process and Bellman Equations

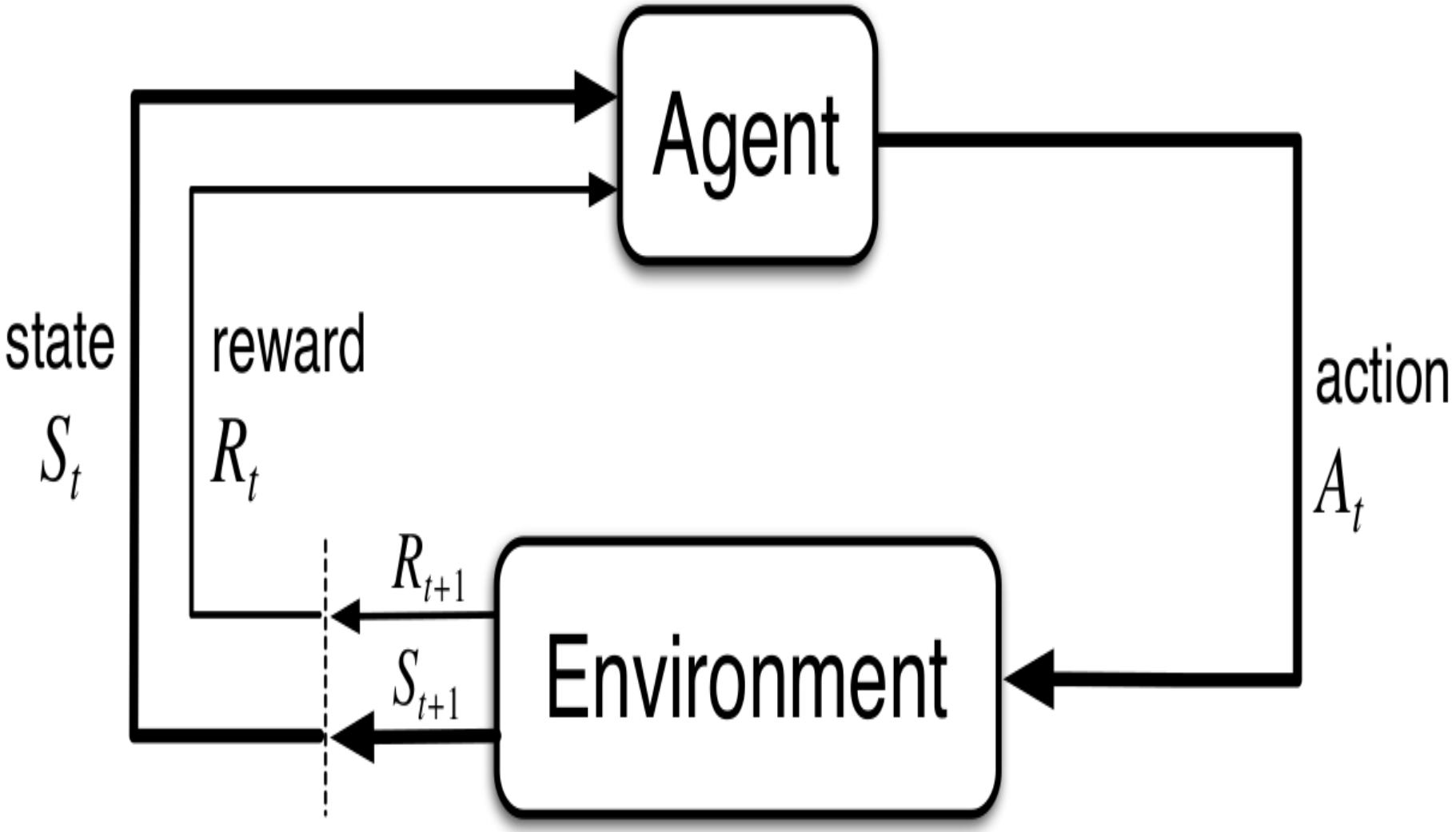
Ashwin Rao

ICME, Stanford University

# Developing Intuition on Optimal Sequential Decisioning

- Chapter 1 covered "Sequential Uncertainty" and notion of "Rewards"
- Here we extend the framework to include "Sequential Decisioning"
- Developing intuition by revisiting the Inventory example
- Over-ordering risks "holding costs" of overnight inventory
- Under-ordering risks "stockout costs" (empty shelves more damaging)
- Orders influence future inventory levels, and consequent future orders
- Also need to deal with delayed costs and demand uncertainty
- Intuition on how challenging it is to determine *Optimal Actions*
- Cyclic interplay between the *Agent* and *Environment*
- Unlike supervised learning, there's no "teacher" here (only *Rewards*)

# Cyclic Interplay between *Agent* and *Environment*



# MDP Definition for Discrete Time, Countable States

## Definition

A *Markov Decision Process (MDP)* comprises of:

- A countable set of states $\mathcal{S}$ (State Space), a set $\mathcal{T} \subseteq \mathcal{S}$ (known as the set of Terminal States), and a countable set of actions $\mathcal{A}$
- A time-indexed sequence of *environment-generated* pairs of random states $S_t \in \mathcal{S}$ and random rewards $R_t \in \mathcal{D}$ (a countable subset of $\mathbb{R}$), alternating with *agent-controllable* actions $A_t \in \mathcal{A}$ for time steps $t = 0, 1, 2, \ldots$
- Markov Property: $\mathbb{P}[(R_{t+1}, S_{t+1})|(S_t, A_t, S_{t-1}, A_{t-1}, \ldots, S_0, A_0)] = \mathbb{P}[(R_{t+1}, S_{t+1})|(S_t, A_t)]$ for all $t \geq 0$
- Termination: If an outcome for $S_T$ (for some time step $T$) is a state in the set $\mathcal{T}$, then this sequence outcome terminates at time step $T$.

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$$

# Stationarity, Transition Function and Reward Functions

- Stationary (by default): $\mathbb{P}[(R_{t+1}, S_{t+1})|(S_t, A_t)]$ independent of $t$
- $\Rightarrow$ *Transition Probability Function* $\mathcal{P}_R : \mathcal{N} \times \mathcal{A} \times \mathcal{D} \times \mathcal{S} \to [0, 1]$

$$\mathcal{P}_R(s, a, r, s') = \mathbb{P}[(R_{t+1} = r, S_{t+1} = s')|S_t = s, A_t = a]$$

- State Transition Probability Function $\mathcal{P} : \mathcal{N} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$:

$$\mathcal{P}(s, a, s') = \sum_{r \in \mathcal{D}} \mathcal{P}_R(s, a, r, s')$$

- Reward Transition Function $\mathcal{R}_T : \mathcal{N} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ defined as:

$$\mathcal{R}_T(s, a, s') = \mathbb{E}[R_{t+1}|(S_{t+1} = s', S_t = s, A_t = a)]$$

$$= \sum_{r \in \mathcal{D}} \frac{\mathcal{P}_R(s, a, r, s')}{\mathcal{P}(s, a, s')} \cdot r = \sum_{r \in \mathcal{D}} \frac{\mathcal{P}_R(s, a, r, s')}{\sum_{r \in \mathcal{D}} \mathcal{P}_R(s, a, r, s')} \cdot r$$

- Reward Function $\mathcal{R} : \mathcal{N} \times \mathcal{A} \to \mathbb{R}$ defined as:

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

# Policy: Function defining the Behavior of the Agent

- A Policy is an *Agent-controlled* function $\pi : \mathcal{N} \times \mathcal{A} \to [0, 1]$

$$\pi(s, a) = \mathbb{P}[A_t = a | S_t = s] \text{ for all time steps } t = 0, 1, 2, \ldots$$

- Above definition assumes Policy is Markovian and Stationary
- If not stationary, we can include time in *State* to make it stationary
- We denote a deterministic policy as a function $\pi_D : \mathcal{N} \to \mathcal{A}$

$$\pi(s, \pi_D(s)) = 1 \text{ and } \pi(s, a) = 0 \text{ for all } a \in \mathcal{A} \text{ with } a \neq \pi_D(s)$$

```
class Policy(ABC, Generic[S, A]):
    @abstractmethod
    def act(self, state: S) -> \
            Optional[Distribution[A]]:
        pass
```

# [MDP, Policy] := MRP

$$\mathcal{P}_R^\pi(s, r, s') = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{P}_R(s, a, r, s')$$

$$\mathcal{P}^\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{P}(s, a, s')$$

$$\mathcal{R}_T^\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{R}_T(s, a, s')$$

$$\mathcal{R}^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{R}(s, a)$$

# @abstractclass MarkovDecisionProcess

```
class MarkovDecisionProcess(ABC, Generic[S, A]):
    @abstractmethod
    def actions(self, state: S) -> Iterable[A]:
        pass

    @abstractmethod
    def step(self, state: S, action: A) -> \
            Optional[Distribution[Tuple[S, float]]]:
        pass

    def is_terminal(self, state: S) -> bool:
        try:
            next(iter(self.actions(state)))
            return False
        except StopIteration:
            return True
```

# @abstractclass MarkovDecisionProcess

```
def apply_policy(self, policy: Policy[S, A]) \
        -> MarkovRewardProcess[S]:
    mdp = self

    class RewardProcess(MarkovRewardProcess[S]):
        def transition_reward(self, state: S) -> \
                Optional[Distribution[Tuple[S, float]]]:
            if policy.act(state) is None:
                return None
            return policy.act(state).apply(
                lambda a: mdp.step(state, a)
            )

    return RewardProcess()
```

# Finite Markov Decision Process

- Finite State Space $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, $|\mathcal{N}| = m \leq n$
- Action Space $\mathcal{A}(s)$ is finite for each $s \in \mathcal{N}$
- Finite set of (next state, reward) transitions
- We'd like a *sparse representation* for $\mathcal{P}_R$
- Conceptualize $\mathcal{P}_R : \mathcal{N} \times \mathcal{A} \times \mathcal{D} \times \mathcal{S} \to [0, 1]$ as:

$$\mathcal{N} \to (\mathcal{A} \to (\mathcal{S} \times \mathcal{D} \to [0, 1]))$$

```
StateReward = FiniteDistribution[Tuple[S, float]]
ActionMapping = Mapping[A, StateReward[S]]
StateActionMapping = Mapping[S, Optional[ActionMapping
```

# class FiniteMarkovDecisionProcess

```
class FiniteMarkovDecisionProcess(
        MarkovDecisionProcess[S, A]
):
    mapping: StateActionMapping[S, A]
    non_terminal_states: Sequence[S]

    def __init__(self, mapping: StateActionMapping[S,
        self.mapping = mapping
        self.non_terminal_states = [s for s, v in mapp
                                    if v is not None]

    def step(self, state: S, action: A) \
            -> Optional[StateReward]:
        if self.mapping[state] is None:
            return None
        return self.mapping[state][action]
```

# class FinitePolicy

```
class FinitePolicy(Policy[S, A]):
    policy_map: Mapping[S, Optional[FiniteDistribution

    def __init__(
        self,
        policy_map: Mapping[S, Optional[FiniteDistribu
    ):
        self.policy_map = policy_map

    def act(self, state: S) -> Optional[FiniteDistribu
        return self.policy_map[state]
```

With this, we can write a method for FiniteMarkovDecisionProcess that takes a FinitePolicy and produces a FiniteMarkovRewardProcess

# Inventory MDP

- $\alpha :=$ On-Hand Inventory, $\beta :=$ On-Order Inventory
- $h :=$ Holding Cost (per unit of overnight inventory)
- $p :=$ Stockout Cost (per unit of missed demand)
- $C :=$ Shelf Capacity (number of inventory units shelf can hold)
- $\mathcal{S} = \{(\alpha, \beta) : 0 \leq \alpha + \beta \leq C\}$
- $\mathcal{A}((\alpha, \beta)) = \{\theta : 0 \leq \theta \leq C - (\alpha + \beta)\}$
- $f(\cdot) :=$ PMF of demand, $F(\cdot) :=$ CMF of demand

$$\mathcal{R}_T((\alpha, \beta), \theta, (\alpha + \beta - i, \theta)) = -h\alpha \text{ for } 0 \leq i \leq \alpha + \beta - 1$$

$$\mathcal{R}_T((\alpha, \beta), \theta, (0, \theta)) = -h\alpha - p(\sum_{j=\alpha+\beta+1}^{\infty} f(j) \cdot (j - (\alpha + \beta)))$$

$$= -h\alpha - p(\lambda(1 - F(\alpha + \beta - 1)) - (\alpha + \beta)(1 - F(\alpha + \beta)))$$

# State-Value Function of an MDP for a Fixed Policy

- Define the *Return* $G_t$ from state $S_t$ as:

$$G_t = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} \cdot R_i = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots$$

- $\gamma \in [0, 1]$ is the discount factor
- *State-Value Function* (for policy $\pi$) $V^\pi : \mathcal{N} \to \mathbb{R}$ defined as:

$$V^\pi(s) = \mathbb{E}_{\pi, \mathcal{P}_R}[G_t | S_t = s] \text{ for all } s \in \mathcal{N}, \text{ for all } t = 0, 1, 2, \ldots$$

- $V^\pi$ is Value Function of $\pi$-implied MRP, satisfying MRP Bellman Eqn

$$V^\pi(s) = \mathcal{R}^\pi(s) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}^\pi(s, s') \cdot V^\pi(s')$$

- This yields the *MDP (State-Value Function) Bellman Policy Equation*

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot (\mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot V^\pi(s')) \quad (1)$$

# Action-Value Function of an MDP for a Fixed Policy

- *Action-Value Function* (for policy $\pi$) $Q^\pi : \mathcal{N} \times \mathcal{A} \to \mathbb{R}$ defined as:

$$Q^\pi(s, a) = \mathbb{E}_{\pi, \mathcal{P}_R}[G_t | (S_t = s, A_t = a)] \text{ for all } s \in \mathcal{N}, a \in \mathcal{A}$$

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot Q^\pi(s, a) \tag{2}$$

- Combining Equation (1) and Equation (2) yields:

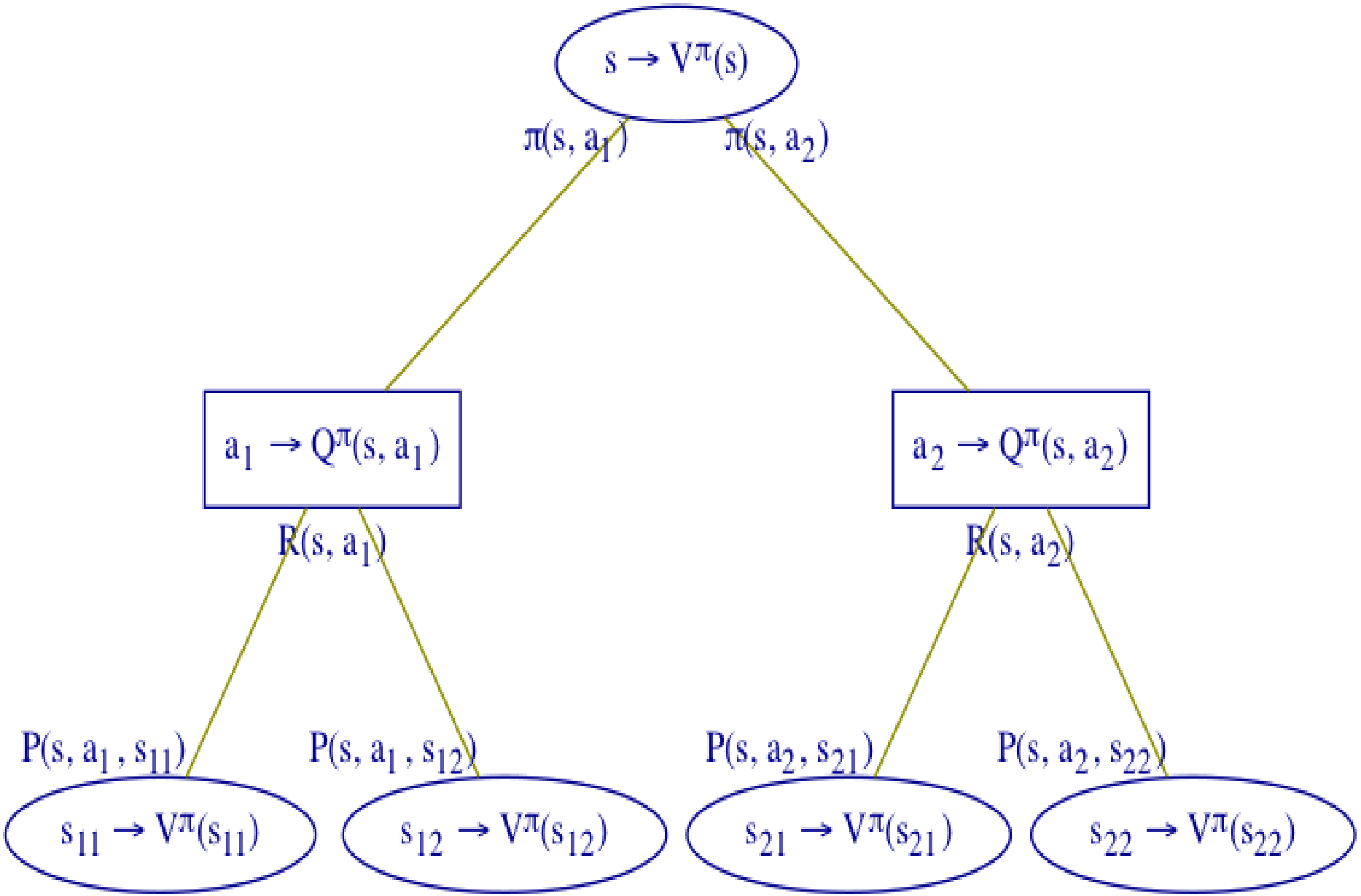
$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot V^\pi(s') \tag{3}$$

- Combining Equation (3) and Equation (2) yields:

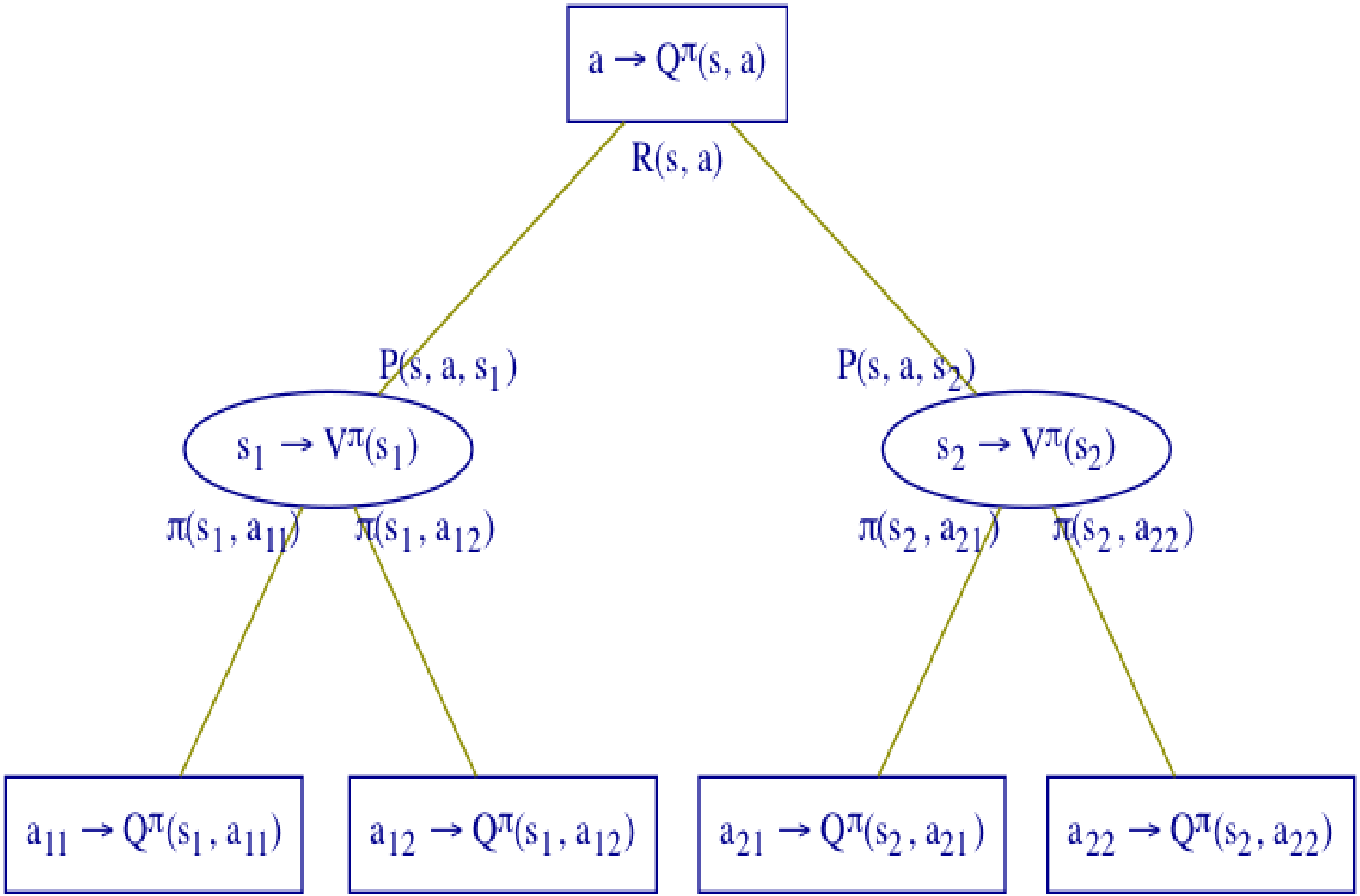
$$Q^\pi(s, a) = \mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \sum_{a' \in \mathcal{A}} \pi(s', a') \cdot Q^\pi(s', a') \tag{4}$$

**MDP Prediction Problem:** Evaluating $V^\pi(\cdot)$ and $Q^\pi(\cdot)$ for fixed policy $\pi$

# MDP State-Value Function Bellman Policy Equation



# MDP Action-Value Function Bellman Policy Equation



# Optimal Value Functions

- *Optimal State-Value Function* $V^*: \mathcal{N} \to \mathbb{R}$ defined as:

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s) \text{ for all } s \in \mathcal{N}$$

  where $\Pi$ is the space of all stationary (stochastic) policies
- *For each* $s$, maximize $V^\pi(s)$ across choices of $\pi \in \Pi$
- Does this mean we could have different maximizing $\pi$ for different $s$?
- We'll answer this question later
- *Optimal Action-Value Function* $Q^*: \mathcal{N} \times \mathcal{A} \to \mathbb{R}$ defined as:

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

# Bellman Optimality Equations

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{5}$$

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot V^*(s') \tag{6}$$

These yield the *MDP State-Value Function Bellman Optimality Equation*

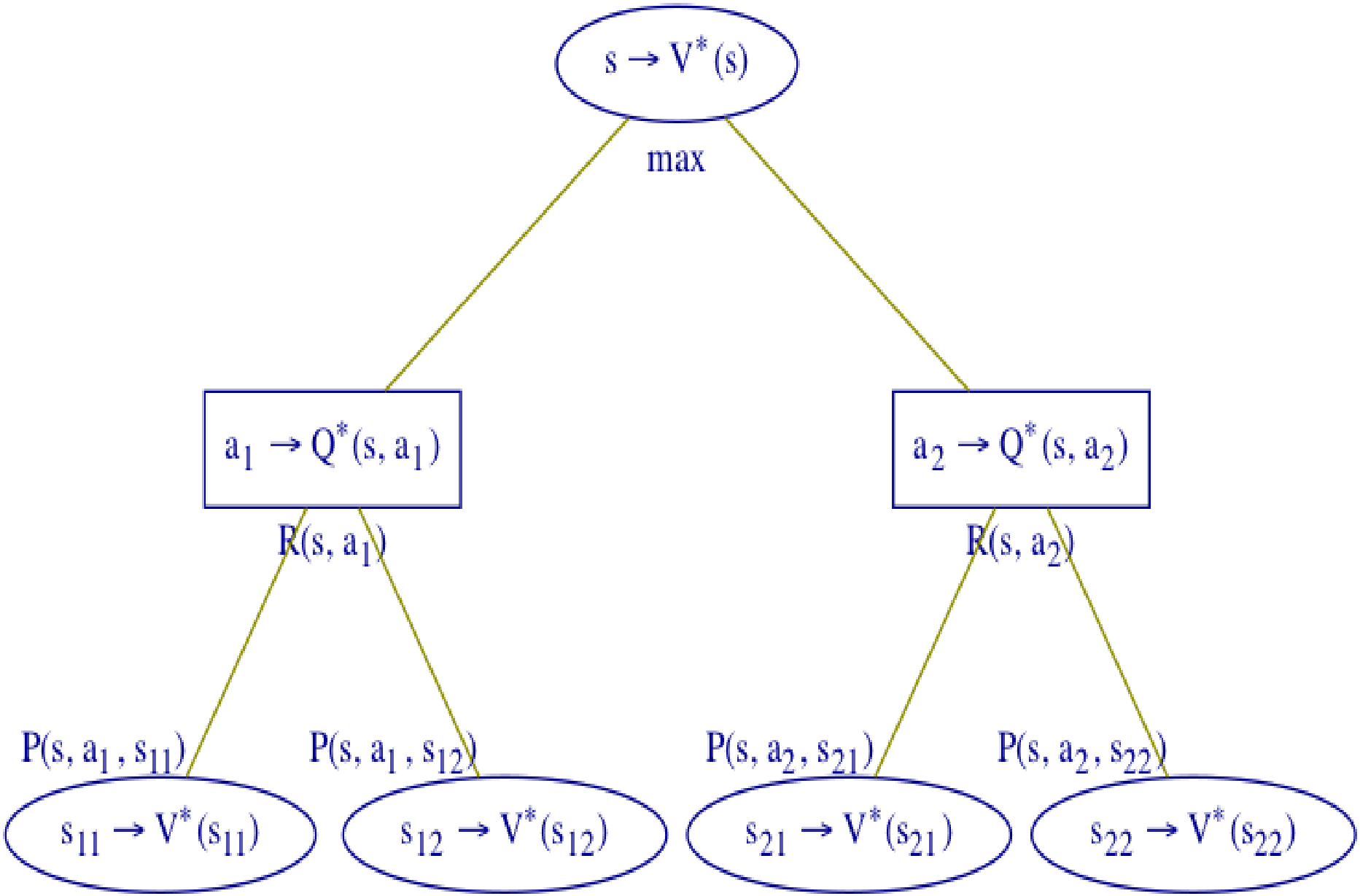
$$V^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot V^*(s')\} \tag{7}$$

and the *MDP Action-Value Function Bellman Optimality Equation*

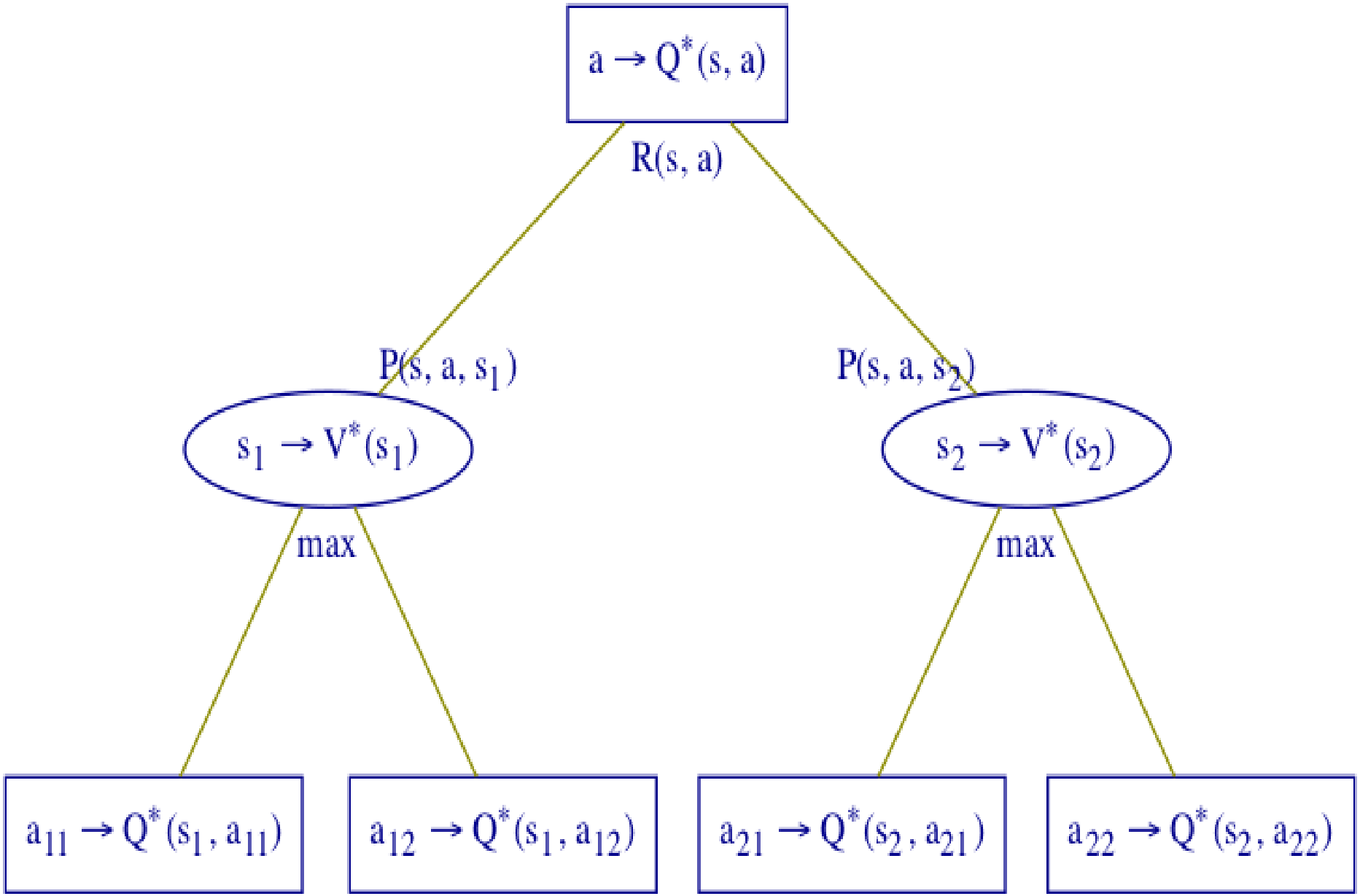
$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \max_{a' \in \mathcal{A}} Q^*(s', a') \tag{8}$$

**MDP Control Problem:** Computing $V^*(\cdot)$ and $Q^*(\cdot)$

# MDP State-Value Function Bellman Optimality Equation



# MDP Action-Value Function Bellman Optimality Equation



# Optimal Policy

- Bellman Optimality Equations don't directly solve *Control*
- Because (unlike Bellman Policy Equations), these are non-linear
- But these equations form the foundations of DP/RL algos for Control
- But will solving Control give us the *Optimal Policy*?
- What does *Optimal Policy* mean anyway?
- What if different $\pi$ maximize $V^{\pi}(s)$ for different $s$?
- So define an *Optimal Policy* $\pi^*$ as one that "dominates" all other $\pi$:

  $\pi^* \in \Pi$ is an Optimal Policy if $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $\pi$ *and* for all $s$

- Is there an Optimal Policy $\pi^*$ such that $V^*(s) = V^{\pi^*}(s)$ for all $s$?

# Optimal Policy achieves Optimal Value Function

## Theorem

*For any (discrete-time, countable-spaces, stationary) MDP:*

- *There exists an Optimal Policy* $\pi^* \in \Pi$, *i.e., there exists a Policy* $\pi^* \in \Pi$ *such that* $V^{\pi^*}(s) \geq V^{\pi}(s)$ *for all policies* $\pi \in \Pi$ *and for all states* $s \in \mathcal{N}$
- *All Optimal Policies achieve the Optimal Value Function, i.e.* $V^{\pi^*}(s) = V^*(s)$ *for all* $s \in \mathcal{N}$, *for all Optimal Policies* $\pi^*$
- *All Optimal Policies achieve the Optimal Action-Value Function, i.e.* $Q^{\pi^*}(s, a) = Q^*(s, a)$ *for all* $s \in \mathcal{N}$, *for all* $a \in \mathcal{A}$, *for all Optimal Policies* $\pi^*$

# Proof Outline

- For any Optimal Policies $\pi_1^*$ and $\pi_2^*$, $V^{\pi_1^*}(s) = V^{\pi_2^*}(s)$ for all $s \in \mathcal{N}$
- Construct a candidate Optimal (Deterministic) Policy $\pi_D^* : \mathcal{N} \to \mathcal{A}$:

$$\pi_D^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a) \text{ for all } s \in \mathcal{N}$$

- $\pi_D^*$ achieves the Optimal Value Functions $V^*$ and $Q^*$:

$$V^*(s) = Q^*(s, \pi_D^*(s)) \text{ for all } s \in \mathcal{N}$$

$$V^{\pi_D^*}(s) = V^*(s) \text{ for all } s \in \mathcal{N}$$

$$Q^{\pi_D^*}(s, a) = Q^*(s, a) \text{ for all } s \in \mathcal{N}, \text{ for all } a \in \mathcal{A}$$

- $\pi_D^*$ is an Optimal Policy:

$$V^{\pi_D^*}(s) \geq V^{\pi}(s) \text{ for all policies } \pi \in \Pi \text{ and for all states } s \in \mathcal{N}$$

# State Space Size and Transitions Complexity

- Tabular Algorithms for State Spaces that are not too large
- In real-world, state spaces are very large/infinite/continuous
- *Curse of Dimensionality*: Size Explosion as a function of dimensions
- *Curse of Modeling*: Transition Probabilities hard to model/estimate
- Dimension-Reduction techniques, Unsupervised ML methods
- Function Approximation of the Value Function (in ADP and RL)
- Sampling, Sampling, Sampling ... (in ADP and RL)

# Action Space Sizes

- Large Action Spaces: Hard to represent, estimate and evaluate:
  - Policy $\pi$
  - Action-Value Function for a policy $Q^{\pi}$
  - Optimal Action-Value Function $Q^*$
- Large Actions Space makes it hard to calculate $\text{argmax}_a\, Q(s, a)$
- Optimization over Action Space for each non-terminal state
- Policy Gradient a technique to deal with large action spaces

# Time-Steps Variants and Continuity

- Time-Steps: terminating (*episodic*) or non-terminating (*continuing*)
- Discounted or Undiscounted MDPs, Average-Reward MDPs
- Continuous-time MDPs: Stochastic Processes and Stochastic Calculus
- When States/Actions/Time all continuous, Hamilton-Jacobi-Bellman

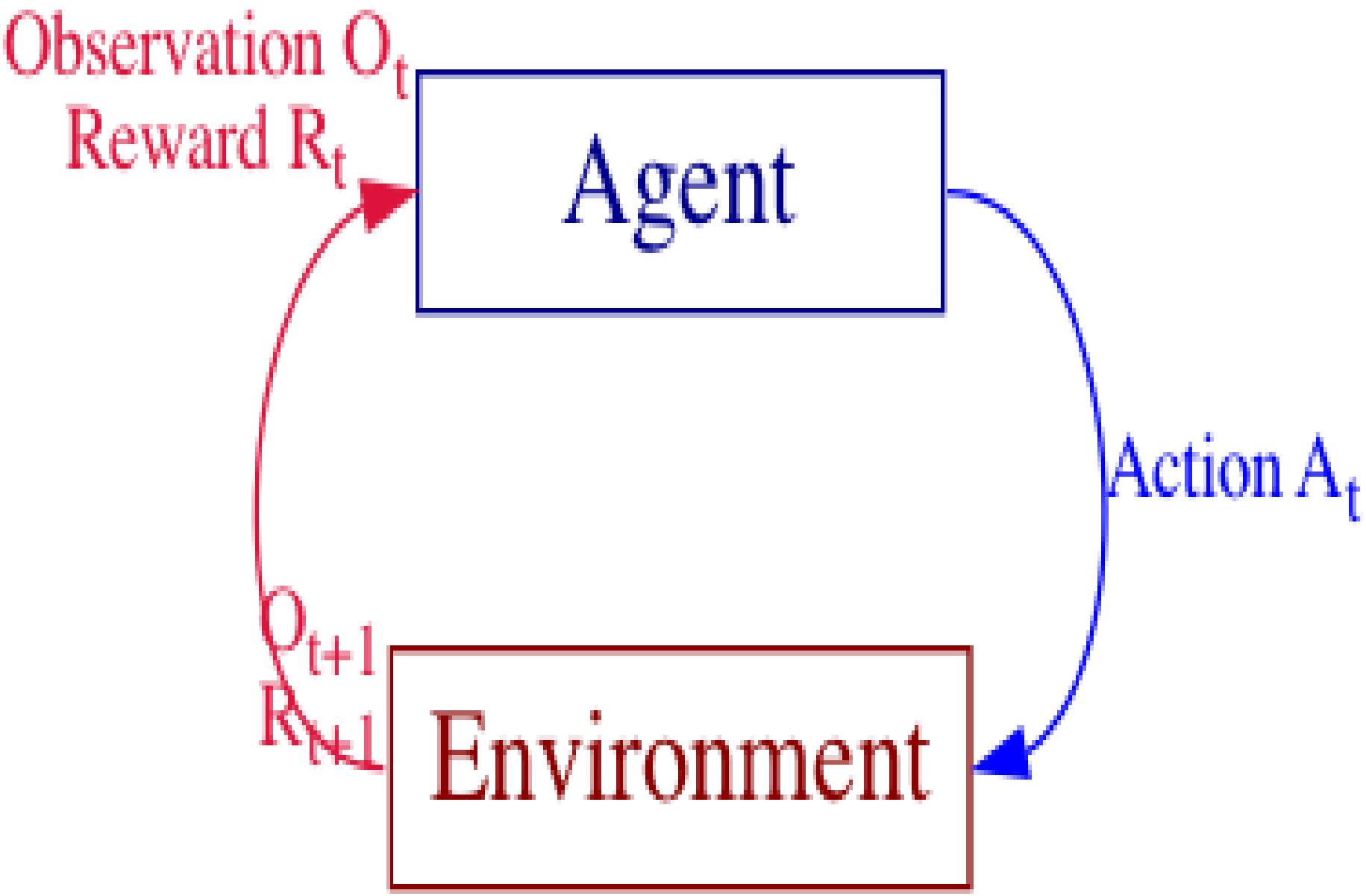
# Partially-Observable Markov Decision Process (POMDP)

- Two different notions of *State*:
  - Internal representation of the environment at each time step $t$ ($S_t^{(e)}$)
  - The agent's state at each time step $t$ (let's call it $S_t^{(a)}$)
- We assumed $S_t^{(e)} = S_t^{(a)} (= S_t)$ and that $S_t$ is *fully observable*
- A more general framework assumes agent sees *Observations* $O_t$
- Agent cannot see (or infer) $S_t^{(e)}$ from history of observations
- This more general framework is called *POMDP*
- POMDP is specified with *Observation Space* $\mathcal{O}$ and observation probability function $\mathcal{Z} : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$ defined as:

$$\mathcal{Z}(s', a, o) = \mathbb{P}[O_{t+1} = o | (S_{t+1} = s', A_t = a)]$$

- Along with the usual transition probabilities specification $\mathcal{P}_R$
- MDP is a special case of POMDP with $O_t = S_t^{(e)} = S_t^{(a)} = S_t$

# POMDP Framework



# Belief States, Tractability and Modeling

- Agent doesn't have knowledge of $S_t$, only of $O_t$
- So Agent has to "guess" $S_t$ by maintaining *Belief States*

$$b(h)_t = (\mathbb{P}[S_t = s_1 | H_t = h], \mathbb{P}[S_t = s_2 | H_t = h], \ldots)$$

where history $H_t$ is all data known to agent by time $t$:

$$H_t := (O_0, R_0, A_0, O_1, R_1, A_1, \ldots, O_t, R_t)$$

- $H_t$ satisfies Markov Property $\Rightarrow$ $b(h)_t$ satisfies Markov Property
- POMDP yields (huge) MDP whose states are POMDP's belief states
- Real-world: Model as accurate POMDP or approx as tractable MDP?

# Key Takeaways from this Chapter

- MDP Bellman Policy Equations
- MDP Bellman Optimality Equations
- Existence of an Optimal Policy, and of each Optimal Policy achieving the Optimal Value Function