# Markov Decision Processes

## Finite MDPs: States, Actions, Rewards, and Transition Probabilities

Sarwan Ali

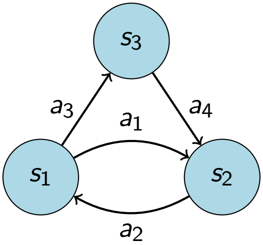
Department of Computer Science
Georgia State University

Understanding Markov Decision Processes

# Today's Learning Journey

# What is a Markov Decision Process?

**Definition:** A mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker.




Agent makes decisions

**Key Components:**

- **States** - Possible situations
- **Actions** - Available choices
- **Rewards** - Immediate feedback
- **Transitions** - State changes

# Why Study MDPs?

**Real-World Applications:**

- Robotics navigation
- Game playing (Chess, Go)
- Resource allocation
- Financial trading
- Medical treatment planning
- Autonomous vehicles

**Mathematical Foundation:**

- Provides formal framework
- Enables optimal decision making
- Handles uncertainty
- Sequential decision problems
- Basis for reinforcement learning

**Key Insight**

MDPs bridge the gap between theoretical mathematics and practical AI applications.

# The Markov Property

## Definition

The **Markov Property** states that the future is independent of the past given the present state.

**Mathematically:**

$$P(S_{t+1} = s' | S_t = s, S_{t-1} = s_{t-1}, \ldots, S_0 = s_0) = P(S_{t+1} = s' | S_t = s)$$

**Markov Process:**

- Current state contains all relevant information
- Past history doesn't matter
- "Memoryless" property

**Non-Markov Process:**

- Future depends on history
- Need to remember past states
- More complex modeling required

# Markov Property Examples

**Markov Examples:**

- Chess position
- Current location in a maze
- Portfolio value
- Weather today (simplified)

**Non-Markov Examples:**

- Stock market trends
- Language modeling
- Medical diagnosis
- Social interactions

**Chess**

Current board position contains all information needed to determine valid next moves and their probabilities.

**Stock Market**

Past price movements often influence future trends, violating the Markov property.

# Formal Definition of Finite MDP

A **finite Markov Decision Process** is a 4-tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$

- $\mathcal{S}$: Finite set of **states**
- $\mathcal{A}$: Finite set of **actions**
- $\mathcal{P}$: **Transition probability** function
- $\mathcal{R}$: **Reward** function

## Transition Dynamics

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

This defines the probability of transitioning to state $s'$ and receiving reward $r$ when taking action $a$ in state $s$.

# MDP Components in Detail

**States ($\mathcal{S}$):**

- Complete description of the world
- Must satisfy Markov property
- Examples: positions, configurations

**Actions ($\mathcal{A}$):**

- Choices available to agent
- May depend on current state
- $\mathcal{A}(s)$ = actions in state $s$

**Rewards ($\mathcal{R}$):**

- Immediate feedback signal
- Scalar values
- Guide learning process

**Transitions ($\mathcal{P}$):**

- Probability distributions
- Model uncertainty
- Sum to 1 for each state-action pair

**Key Constraint**

For finite MDPs: $|\mathcal{S}| < \infty$ and $|\mathcal{A}| < \infty$

# Understanding States

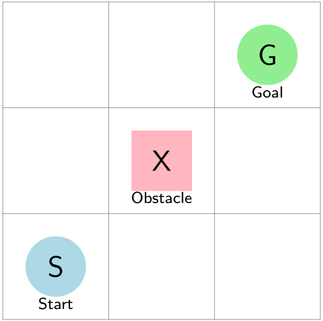
**State** represents all information necessary to make optimal decisions.

**Properties of Good States:**

1. **Completeness:** Contains all relevant information
2. **Markov:** Future independent of past given present
3. **Discriminative:** Different states lead to different optimal actions

**State Space Design:**

- Too small $\Rightarrow$ loses important information
- Too large $\Rightarrow$ computational complexity
- Balance between expressiveness and tractability




Grid World States

# State Representation Examples

**Grid World:**

- State: $(x, y)$ coordinates
- Simple and intuitive
- $\mathcal{S} = \{(i, j) : 0 \leq i < width, 0 \leq j < height\}$

**Tic-Tac-Toe:**

- State: Board configuration
- Each cell: $\{X, O, empty\}$
- $|\mathcal{S}| = 3^9 = 19,683$ (theoretical)

**Robot Navigation:**

- State: $(x, y, \theta)$ position and orientation
- May include velocity information
- Continuous $\Rightarrow$ discretization needed

**Inventory Management:**

- State: Current inventory levels
- Time of year, demand patterns
- Supply chain status

**State Design Principle**

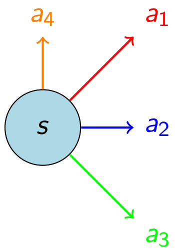
Include **sufficient** information to make optimal decisions, but **minimal** information to keep the problem tractable.

## Understanding Actions

**Actions** represent the choices available to the agent at each state.

### Action Properties:

- May be state-dependent: $\mathcal{A}(s)$
- Discrete or continuous
- Deterministic or stochastic effects




Actions from state $s$

### Action Space Types:

1. **Finite Discrete:** $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$
2. **Continuous:** $\mathcal{A} \subseteq \mathbb{R}^d$
3. **Hybrid:** Mix of discrete and continuous

**Important Note**

In finite MDPs, we focus on **finite action spaces**: $|\mathcal{A}| < \infty$

# Action Examples

**Grid World Navigation:**

$$\mathcal{A} = \{North, South, East, West\}$$

**Game Playing (Chess):**

- Legal moves depend on current position
- $\mathcal{A}(s)$ varies by state
- Complex action space

**Atari Games:**

$$\mathcal{A} = \{Fire, Left, Right, NoOp, \ldots\}$$

**Robot Control:**

- Joint angles/velocities
- Motor commands
- High-level behaviors

**Resource Allocation:**

- How much to invest
- Which resources to allocate
- Binary decisions (yes/no)

**State-Dependent Actions Example**

In a maze, action "move north" is only available if there's no wall to the north of current position.

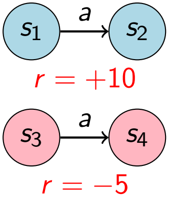
# Understanding Rewards

**Rewards** provide immediate feedback to guide the agent's learning.

**Reward Function:**

$$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$$

Or simplified: $R(s, a)$ or $R(s)$




**Reward Properties:**

- Scalar signal (single number)
- Immediate feedback
- Defines the objective
- Can be positive, negative, or zero

**Reward Hypothesis**

All goals and purposes can be thought of as maximization of expected cumulative reward.

# Reward Design Principles

**Good Reward Design:**

- Reflects true objective
- Provides clear guidance
- Avoids reward hacking
- Considers long-term consequences

**Example - Navigation:**

- Goal reached: $+100$
- Each step: $-1$
- Hit obstacle: $-10$

**Common Pitfalls:**

- Reward hacking
- Sparse rewards
- Misaligned incentives
- Local optima

**Example - Bad Design:**

- Only goal: $+100$
- Everything else: $0$
- $\Rightarrow$ No guidance!

**Key Insight**

Reward engineering is crucial - the agent will optimize exactly what you reward, not necessarily what you want!

# Types of Rewards

**Dense Rewards:**

- Frequent feedback
- Every action gets reward
- Easier learning
- Example: $-1$ per step

**Sparse Rewards:**

- Infrequent feedback
- Most actions get 0 reward
- Harder learning
- Example: Only at goal

**Intrinsic vs Extrinsic:**

- **Extrinsic:** Environment provides
- **Intrinsic:** Agent generates
- Curiosity, exploration bonuses

**Reward Shaping:**

- Additional guidance rewards
- Must preserve optimal policy
- Potential-based shaping

# Transition Probabilities

**Transition probabilities** model the dynamics of the environment.

**Mathematical Definition:**

$$P(s'|s,a) = \Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$$

**Properties:**

- $0 \leq P(s'|s,a) \leq 1$ for all $s', s, a$
- $\sum_{s' \in \mathcal{S}} P(s'|s,a) = 1$ for all $s, a$
- Encodes environment uncertainty
- May be unknown to the agent

**Deterministic:** $P(s'|s,a) \in \{0,1\}$

**Stochastic:** $0 < P(s'|s,a) < 1$

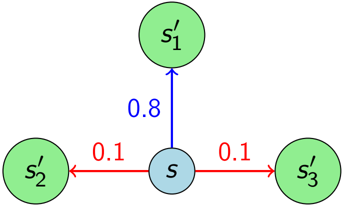
# Transition Probability Examples

## Deterministic Grid World:

- Actions work with 100% certainty
- $P(s'|s,a) = 1$ for intended next state
- $P(s'|s,a) = 0$ for all other states

## Stochastic Grid World:

- Actions may fail
- $P(\text{intended}|s,a) = 0.8$
- $P(\text{left}|s,a) = 0.1$
- $P(\text{right}|s,a) = 0.1$




Action: "Go Up"

### Transition Matrix

For finite MDPs, transitions can be represented as matrices:

$$\mathbf{P}^a_{ss'} = P(s'|s,a)$$

where each row sums to 1.

# Working with Transition Probabilities

**Transition Matrix Example:**

Consider a 3-state MDP with action $a$:

$$\mathbf{P}^a = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$$

Reading the matrix: $\mathbf{P}^a_{ij} = P(s_j | s_i, a)$

**Properties to Check:**

- Each row sums to 1
- All entries $\geq 0$
- Represents valid probability distribution

**Interpretation:**

- From $s_1$: 70% stay, 20% to $s_2$, 10% to $s_3$
- From $s_2$: 10% to $s_1$, 80% stay, 10% to $s_3$
- From $s_3$: 20% to $s_1$, 30% to $s_2$, 50% stay

# MDP Dynamics

**Complete MDP Specification:**

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

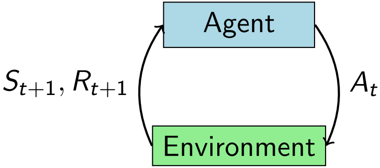
**Decomposition:**

$$P(s'|s, a) = \sum_{r \in \mathcal{R}} P(s', r|s, a) \tag{1}$$

$$R(s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s, a) \tag{2}$$

**The Agent-Environment Interface:**

- Agent observes state $S_t$
- Agent takes action $A_t$
- Environment returns $S_{t+1}$ and $R_{t+1}$
- Process repeats



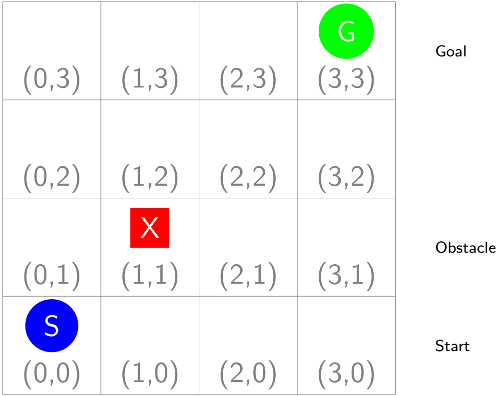
# A Complete MDP Example: Grid World

**Problem Setup:**

- $4 \times 4$ grid
- Start at $(0, 0)$
- Goal at $(3, 3)$
- Obstacle at $(1, 1)$

**States:** $\mathcal{S} = \{(i, j) : 0 \leq i, j \leq 3\} \setminus \{(1, 1)\}$
**Actions:** $\mathcal{A} = \{N, S, E, W\}$
**Rewards:**

- Goal: $+10$
- Each step: $-1$
- Invalid move: $-1$ (stay in place)




Grid World MDP

**Transition Probabilities:**

- Deterministic: Actions succeed with probability 1
- Invalid actions (into walls/obstacles) keep agent in same state

# MDP Example: Transition Details

**Example Transitions from State $(2, 1)$:**

| Action | Next State | Probability | Reward |
|---|---|---|---|
| North | $(2, 2)$ | 1.0 | $-1$ |
| South | $(2, 0)$ | 1.0 | $-1$ |
| East | $(3, 1)$ | 1.0 | $-1$ |
| West | $(2, 1)$ | 1.0 | $-1$ |

**Special Cases:**

- From goal state $(3, 3)$: All actions lead back to goal with reward 0
- Actions leading into obstacles: Agent stays in current state
- Actions leading outside grid: Agent stays in current state

**Key Observation**

This MDP is **deterministic** - each state-action pair has exactly one possible outcome.

# Policies

**Policy** $\pi$: A strategy for choosing actions.

**Deterministic Policy:** $\pi : \mathcal{S} \to \mathcal{A}$

$$a = \pi(s)$$

**Stochastic Policy:** $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$

$$\pi(a|s) = \Pr\{A_t = a | S_t = s\}$$

**Properties:**

- Maps states to actions
- Can be deterministic or stochastic
- Defines agent's behavior
- Goal: Find optimal policy

**Examples:**

- Random policy: $\pi(a|s) = \frac{1}{|\mathcal{A}|}$
- Greedy policy: Always best action
- $\epsilon$-greedy: Mostly greedy, sometimes random

# Value Functions

**Value functions** measure how good it is to be in a state or take an action.

**State Value Function:**

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s\right]$$

**Action Value Function (Q-function):**

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s, A_0 = a\right]$$

- $\gamma \in [0, 1]$: **discount factor**
- $\gamma = 0$: Only immediate rewards matter
- $\gamma = 1$: All future rewards equally important
- $\gamma < 1$: Ensures convergence for infinite horizons

# Bellman Equations

**Bellman Equations** provide recursive relationships for value functions.

**Bellman Equation for $V^\pi$:**

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \gamma V^\pi(s')]$$

**Bellman Equation for $Q^\pi$:**

$$Q^\pi(s, a) = \sum_{s',r} P(s', r|s, a) \left[ r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right]$$

**Relationship between V and Q:**

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s, a) \tag{3}$$

$$Q^\pi(s, a) = \sum_{s',r} P(s', r|s, a)[r + \gamma V^\pi(s')] \tag{4}$$

# Optimal Policies and Value Functions

**Goal:** Find the best possible policy.

**Optimal State Value Function:**

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S}$$

**Optimal Action Value Function:**

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

**Optimal Policy:**

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

**Key Theorem**

For finite MDPs, there exists at least one optimal policy $\pi^*$ that is better than or equal to all other policies.

# Bellman Optimality Equations

**Bellman Optimality Equation for $V^*$:**

$$V^*(s) = \max_a \sum_{s',r} P(s', r|s, a)[r + \gamma V^*(s')]$$

**Bellman Optimality Equation for $Q^*$:**

$$Q^*(s, a) = \sum_{s',r} P(s', r|s, a) \left[ r + \gamma \max_{a'} Q^*(s', a') \right]$$

**Solving MDPs:**

- **Value Iteration:** Iteratively update value function
- **Policy Iteration:** Alternate between policy evaluation and improvement
- **Linear Programming:** Formulate as optimization problem

### Computational Complexity

For finite MDPs with $|\mathcal{S}|$ states and $|\mathcal{A}|$ actions, solving requires $O(|\mathcal{S}|^2|\mathcal{A}|)$ operations per iteration.

# Key Takeaways

**Markov Decision Processes provide a mathematical framework for sequential decision making under uncertainty.**

**Essential Components:**

1. **States** - Complete description of the situation
2. **Actions** - Available choices for the agent
3. **Rewards** - Immediate feedback signal
4. **Transition Probabilities** - Environment dynamics

**Key Concepts:**

- **Markov Property:** Future depends only on present state
- **Policies:** Strategies for action selection
- **Value Functions:** Measure goodness of states/actions
- **Bellman Equations:** Recursive relationships for optimal solutions

# Applications and Extensions

**Real Applications:**

- Autonomous vehicles
- Game playing (AlphaGo, Chess)
- Resource management
- Medical treatment planning
- Financial trading
- Robotics control

**Extensions:**

- Partially Observable MDPs (POMDPs)
- Continuous state/action spaces
- Multi-agent MDPs
- Hierarchical MDPs
- Constrained MDPs
- Infinite horizon problems

### Connection to Machine Learning

MDPs form the foundation of **Reinforcement Learning**, where agents learn optimal policies through interaction with the environment.

# Next Steps

**Building on MDPs:**

1. **Dynamic Programming:** Value iteration, policy iteration
2. **Monte Carlo Methods:** Learning from episodes
3. **Temporal Difference Learning:** Q-learning, SARSA
4. **Function Approximation:** Handling large state spaces
5. **Deep Reinforcement Learning:** Neural networks + RL

**Practical Considerations:**

- State space design
- Reward engineering
- Exploration vs exploitation
- Sample efficiency
- Computational complexity

# Thank You!

Questions & Discussion

**sali85@student.gsu.edu**