Mathematical Foundations of Reinforcement Learning

Shiyu Zhao

Contents

Preface

This book aims to provide a *mathematical* but *friendly* introduction to the fundamental concepts, basic problems, and classic algorithms in reinforcement learning. Some essential features of this book are highlighted as follows.

- ◇ The book introduces reinforcement learning from a mathematical point of view. Hopefully, readers will not only know the procedure of an algorithm but also understand why the algorithm was designed in the first place and why it works effectively.
- ◇ The depth of the mathematics is carefully controlled to an adequate level. The mathematics is also presented in a carefully designed manner to ensure that the book is friendly to read. Readers can read the materials presented in gray boxes selectively according to their interests.
- ◇ Many illustrative examples are given to help readers better understand the topics. All the examples in this book are based on a grid world task, which is easy to understand and helpful for illustrating concepts and algorithms.
- ◇ When introducing an algorithm, the book aims to separate its core idea from complications that may be distracting. In this way, readers can better grasp the core idea of an algorithm.
- ◇ The contents of the book are coherently organized. Each chapter is built based on the preceding chapter and lays a necessary foundation for the subsequent one.

This book is designed for senior undergraduate students, graduate students, researchers, and practitioners who are interested in reinforcement learning. It does not require readers to have any background in reinforcement learning because it starts by introducing the most basic concepts. If the reader already has some background in reinforcement learning, I believe the book can help them understand some topics more deeply or provide different perspectives. This book, however, requires the reader to have some knowledge of probability theory and linear algebra. Some basics of the required mathematics are also included in the appendix of this book.

I have been teaching a graduate-level course on reinforcement learning since 2019. I want to thank the students in my class for their feedback on my teaching. I put the draft of this book online in August 2022. Up to now, I have received valuable feedback from many readers. I want to express my gratitude to these readers. Moreover, I would like

to thank my research assistant, Jialing Lv, for her excellent support in editing the book and my lecture videos; my teaching assistants, Jianan Li and Yize Mi, for their help in my teaching; my Ph.D. student Canlun Zheng for his help in the design of a picture in the book; and my family for their wonderful support. Finally, I would like to thank the editors of this book, Dr. Lanlan Chang and Mr. Sai Guo from Springer Nature Press and Tsinghua University Press, for their great support.

Please note that I have created an open course based on this textbook. Both the slides of the open course and the PDF of this textbook are available online for free download. For more information, you can visit the homepage of the textbook: `https://github.com/MathFoundationRL/Book-Mathematical-Foundation-of-Reinforcement-Learning`

I sincerely hope this book can help readers smoothly enter the exciting field of reinforcement learning.

Shiyu Zhao

Overview of this Book

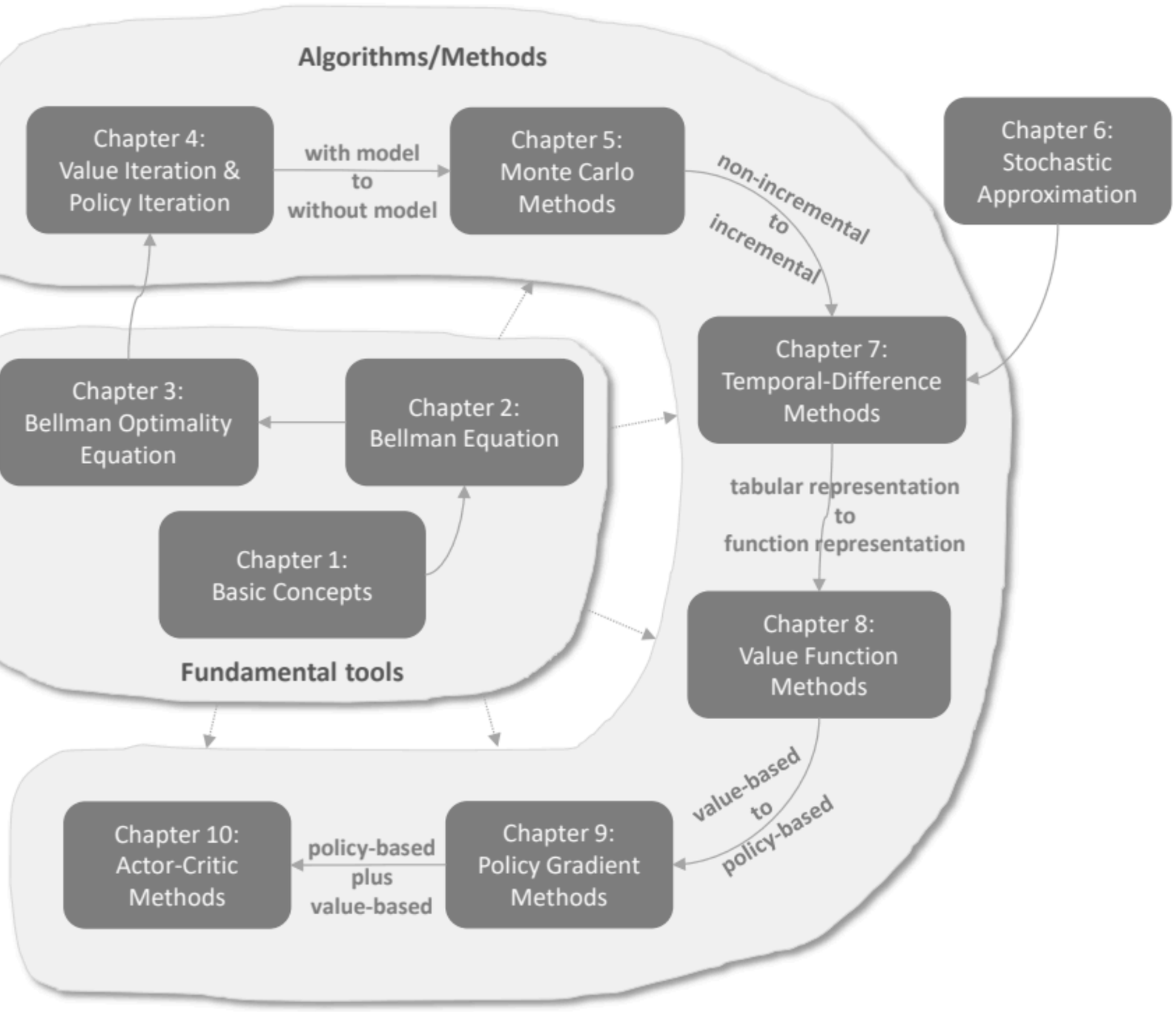


Figure 1: The map of this book.

Before we start the journey, it is important to look at the "map" of the book shown in Figure 1. This book contains ten chapters, which can be classified into two parts: the first part is about basic tools, and the second part is about algorithms. The ten chapters are highly correlated. In general, it is necessary to study the earlier chapters first before the later ones.

Next, please follow me on a quick tour through the ten chapters. Two aspects of each chapter will be covered. The first aspect is the contents introduced in each chapter, and the second aspect is its relationships with the previous and subsequent chapters. A heads up for you to read this overview is as follows. The purpose of this overview is to give you an impression of the contents and structure of this book. It is all right if you encounter many concepts you do not understand. Hopefully, you can make a proper study plan

that is suitable for you after reading this overview.

- ◇ Chapter 1 introduces the basic concepts such as states, actions, rewards, returns, and policies, which are widely used in the subsequent chapters. These concepts are first introduced based on a grid world example, where a robot aims to reach a prespecified target. Then, the concepts are introduced in a more formal manner based on the framework of Markov decision processes.

- ◇ Chapter 2 introduces two key elements. The first is a key concept, and the second is a key tool. The *key concept* is the *state value*, which is defined as the expected return that an agent can obtain when starting from a state if it follows a given policy. The greater the state value is, the better the corresponding policy is. Thus, state values can be used to evaluate whether a policy is good or not.

  The *key tool* is the *Bellman equation*, which can be used to analyze state values. In a nutshell, the Bellman equation describes the relationship between the values of all states. By solving the Bellman equation, we can obtain the state values. Such a process is called *policy evaluation*, which is a fundamental concept in reinforcement learning. Finally, this chapter introduces the concept of action values.

- ◇ Chapter 3 also introduces two key elements. The first is a key concept, and the second is a key tool. The *key concept* is the *optimal policy*. An optimal policy has the greatest state values compared to other policies. The *key tool* is the *Bellman optimality equation*. As its name suggests, the Bellman optimality equation is a special Bellman equation.

  Here is a fundamental question: what is the ultimate goal of reinforcement learning? The answer is to obtain optimal policies. The Bellman optimality equation is important because it can be used to obtain optimal policies. We will see that the Bellman optimality equation is elegant and can help us thoroughly understand many fundamental problems.

The first three chapters constitute the first part of this book. This part lays the necessary foundations for the subsequent chapters. Starting in Chapter 4, the book introduces algorithms for learning optimal policies.

- ◇ Chapter 4 introduces three algorithms: value iteration, policy iteration, and truncated policy iteration. The three algorithms have close relationships with each other. First, the value iteration algorithm is exactly the algorithm introduced in Chapter 3 for solving the Bellman optimality equation. Second, the policy iteration algorithm is an extension of the value iteration algorithm. It is also the foundation for Monte Carlo (MC) algorithms introduced in Chapter 5. Third, the truncated policy iteration algorithm is a unified version that includes the value iteration and policy iteration algorithms as special cases.

The three algorithms share the same structure. That is, every iteration has two steps. One step is to update the value, and the other step is to update the policy. The idea of the interaction between value and policy updates widely exists in reinforcement learning algorithms. This idea is also known as *generalized policy iteration*. In addition, the algorithms introduced in this chapter are actually *dynamic programming* algorithms, which require system models. By contrast, all the algorithms introduced in the subsequent chapters do not require models. It is important to well understand the contents of this chapter before proceeding to the subsequent ones.

- Starting in Chapter 5, we introduce *model-free* reinforcement learning algorithms that do not require system models. While this is the first time we introduce model-free algorithms in this book, we must fill a knowledge gap: how to find optimal policies without models? The philosophy is simple. If we do not have a model, we must have some data. If we do not have data, we must have a model. If we have neither, then we can do nothing. The "data" in reinforcement learning refer to the experience samples generated when the agent interacts with the environment.

  This chapter introduces three algorithms based on MC estimation that can learn optimal policies from experience samples. The first and simplest algorithm is MC Basic, which can be readily obtained by extending the policy iteration algorithm introduced in Chapter 4. Understanding the MC Basic algorithm is important for grasping the fundamental idea of MC-based reinforcement learning. By extending this algorithm, we further introduce two more complicated but more efficient MC-based algorithms. The fundamental trade-off between *exploration* and *exploitation* is also elaborated in this chapter.

Up to this point, the reader may have noticed that the contents of these chapters are highly correlated. For example, if we want to study the MC algorithms (Chapter 5), we must first understand the policy iteration algorithm (Chapter 4). To study the policy iteration algorithm, we must first know the value iteration algorithm (Chapter 4). To comprehend the value iteration algorithm, we first need to understand the Bellman optimality equation (Chapter 3). To understand the Bellman optimality equation, we need to study the Bellman equation (Chapter 2) first. Therefore, it is highly recommended to study the chapters one by one. Otherwise, it may be difficult to understand the contents in the later chapters.

- There is a knowledge gap when we move from Chapter 5 to Chapter 7: the algorithms in Chapter 7 are *incremental*, but the algorithms in Chapter 5 are *non-incremental*. Chapter 6 is designed to fill this knowledge gap by introducing the stochastic approximation theory. Stochastic approximation refers to a broad class of stochastic iterative algorithms for solving root-finding or optimization problems. The classic Robbins-Monro and stochastic gradient descent algorithms are special stochastic approximation algorithms. Although this chapter does not introduce any reinforcement

learning algorithms, it is important because it lays the necessary foundations for studying Chapter 7.

- ◇ Chapter 7 introduces the classic temporal-difference (TD) algorithms. With the preparation in Chapter 6, I believe the reader will not be surprised when seeing the TD algorithms. From a mathematical point of view, TD algorithms can be viewed as stochastic approximation algorithms for solving the Bellman or Bellman optimality equations. Like Monte Carlo learning, TD learning is also model-free, but it has some advantages due to its incremental form. For example, it can learn in an online manner: it can update the value estimate every time an experience sample is received. This chapter introduces quite a few TD algorithms such as Sarsa and Q-learning. The important concepts of on-policy and off-policy are also introduced.

- ◇ Chapter 8 introduces the value function approximation method. In fact, this chapter continues to introduce TD algorithms, but it uses a different way to represent state/action values. In the preceding chapters, state/action values are represented by *tables*. The tabular method is straightforward to understand, but it is inefficient for handling large state or action spaces. To solve this problem, we can employ the value function approximation method. The key to understanding this method is to understand the three steps in its optimization formulation. The first step is to select an objective function for defining optimal policies. The second step is to derive the gradient of the objective function. The third step is to apply a gradient-based algorithm to solve the optimization problem. This method is important because it has become the standard technique to represent values. It is also the location in which *artificial neural networks* are incorporated into reinforcement learning as function approximators. The famous deep Q-learning algorithm is also introduced in this chapter.

- ◇ Chapter 9 introduces the policy gradient method, which is the foundation of many modern reinforcement learning algorithms. The policy gradient method is *policy-based*. It is a large step forward in this book because all the methods in the previous chapters are *value-based*. The basic idea of the policy gradient method is simple: it selects an appropriate scalar metric and then optimizes it via a gradient-ascent algorithm. Chapter 9 has an intimate relationship with Chapter 8 because they both rely on the idea of function approximation. The advantages of the policy gradient method are numerous. For example, it is more efficient for handling large state/action spaces. It has stronger generalization abilities and is more efficient in sample usage.

- ◇ Chapter 10 introduces actor-critic methods. From one point of view, actor-critic refers to a structure that incorporates both policy-based and value-based methods. From another point of view, actor-critic methods are not new since they still fall into the scope of the policy gradient method. Specifically, they can be obtained by extending the policy gradient algorithm introduced in Chapter 9. It is necessary for the reader to properly understand the contents in Chapters 8 and 9 before studying Chapter 10.

Chapter 1

Basic Concepts

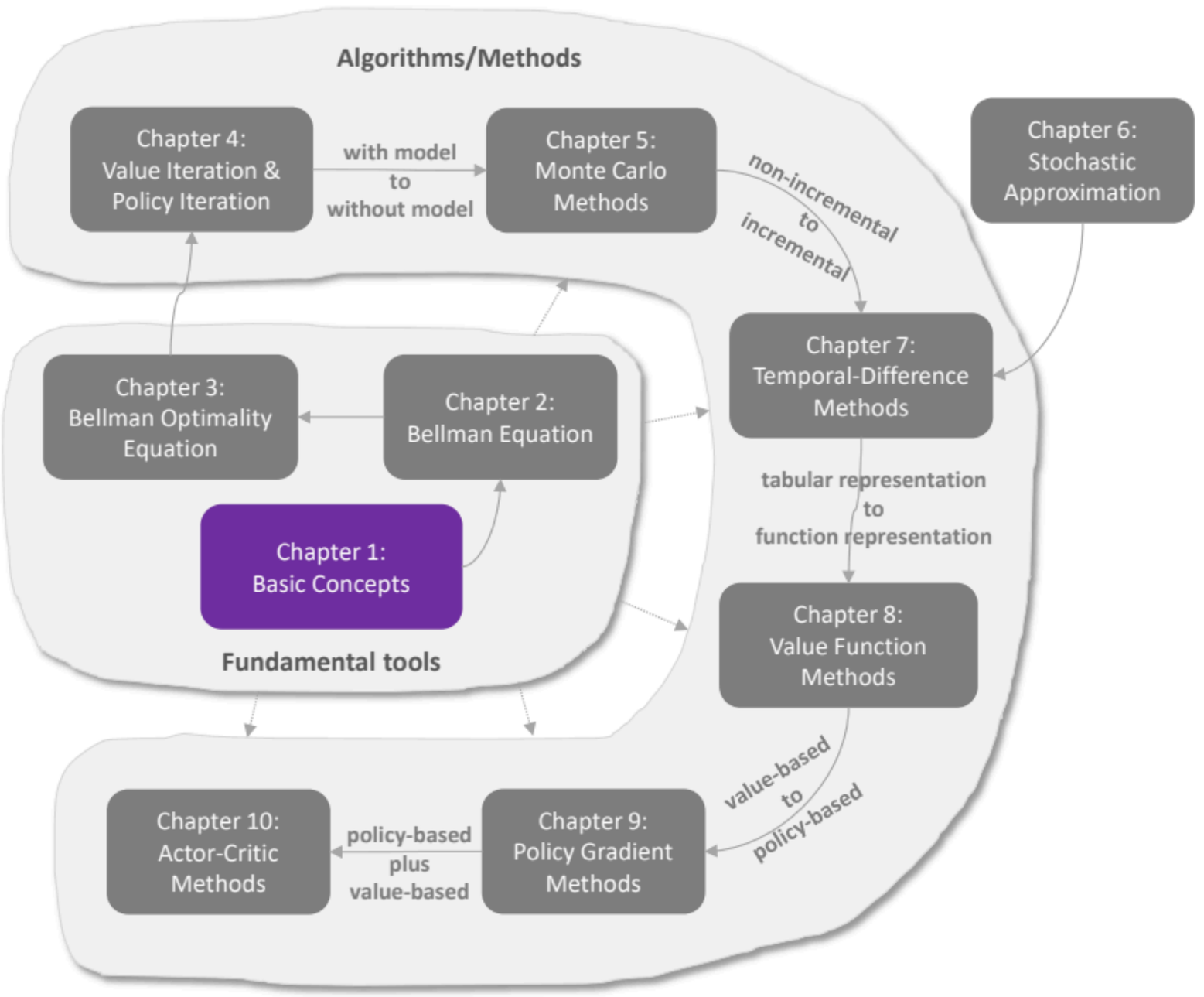


Figure 1.1: Where we are in this book.

This chapter introduces the basic concepts of reinforcement learning. These concepts are important because they will be widely used in this book. We first introduce these concepts using examples and then formalize them in the framework of Markov decision processes.

1.1 A grid world example

Consider an example as shown in Figure 1.2, where a robot moves in a grid world. The robot, called *agent*, can move across adjacent cells in the grid. At each time step, it can

only occupy a single cell. The white cells are *accessible* for entry, and the orange cells are *forbidden*. There is a *target* cell that the robot would like to reach. We will use such grid world examples throughout this book since they are intuitive for illustrating new concepts and algorithms.

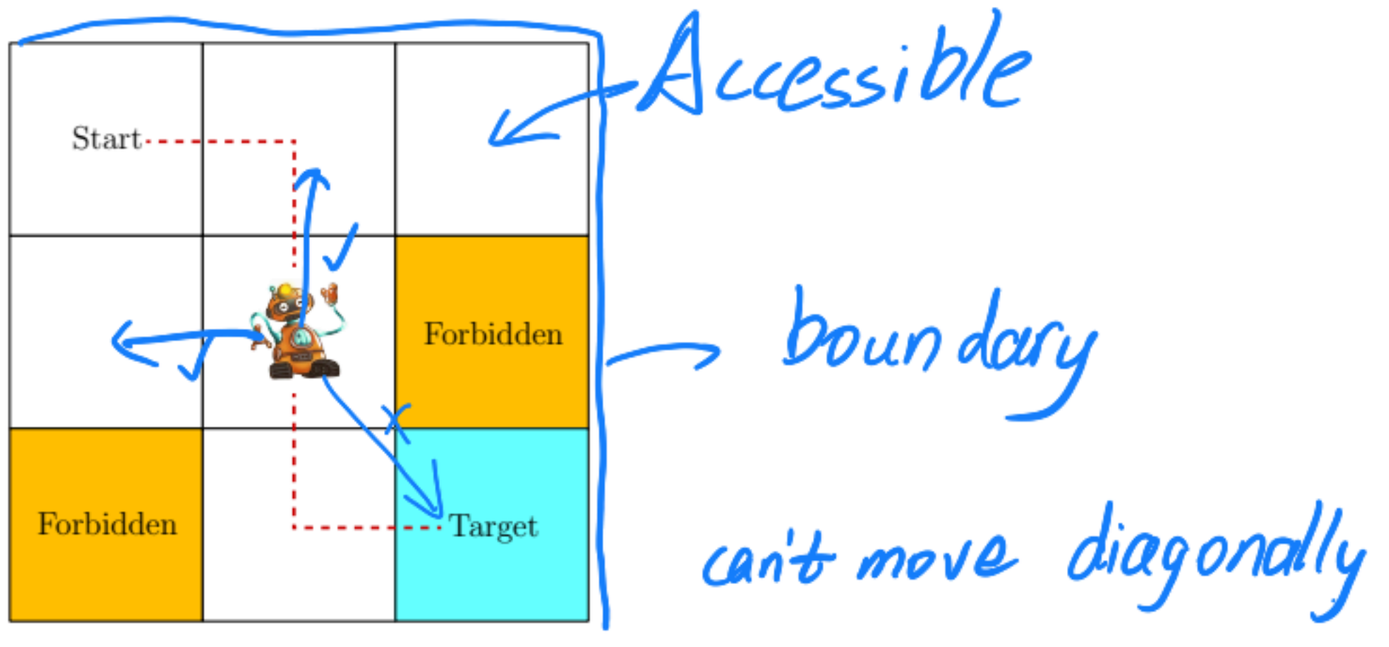


Figure 1.2: The grid world example is used throughout the book.

The ultimate goal of the agent is to find a "good" policy that enables it to reach the target cell when starting from any initial cell. How can the "goodness" of a policy be defined? The idea is that the agent should reach the target without entering any forbidden cells, taking unnecessary detours, or colliding with the boundary of the grid.

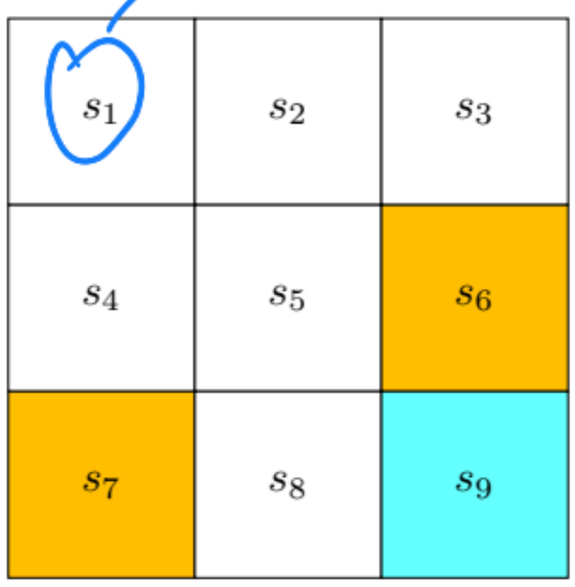
It would be trivial to plan a path to reach the target cell if the agent knew the map of the grid world. The task becomes nontrivial if the agent does not know any information about the environment in advance. Then, the agent must interact with the environment to find a good policy by trial and error. To do that, the concepts presented in the rest of the chapter are necessary.

1.2 State and action

The first concept to be introduced is the *state*, which describes the agent's status with respect to the environment. In the grid world example, the state corresponds to the agent's location. Since there are nine cells, there are nine states as well. They are indexed as $s_1, s_2, \ldots, s_9$, as shown in Figure 1.3(a). The set of all the states is called the *state space*, denoted as $\mathcal{S} = \{s_1, \ldots, s_9\}$.

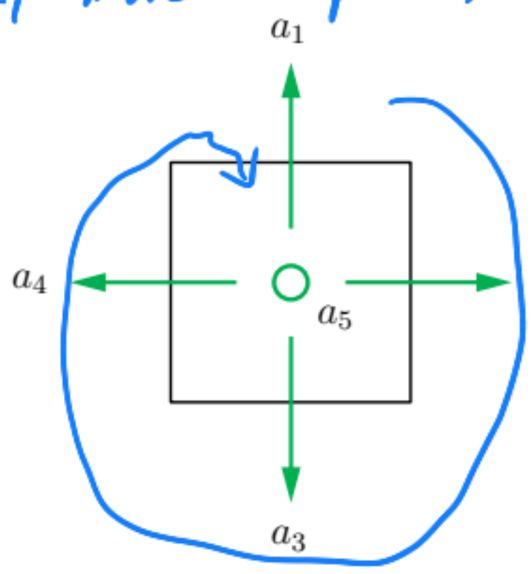
For each state, the agent can take five possible *actions*: moving upward, moving rightward, moving downward, moving leftward, and staying still. These five actions are denoted as $a_1, a_2, \ldots, a_5$, respectively (see Figure 1.3(b)). The set of all actions is called the *action space*, denoted as $\mathcal{A} = \{a_1, \ldots, a_5\}$. Different states can have different action spaces. For instance, considering that taking a_1 or a_4 in state s_1 would lead to a collision with the boundary, we can set the action space for state s_1 as $\mathcal{A}(s_1) = \{a_2, a_3, a_5\}$. In this book, we consider the most general case: $\mathcal{A}(s_i) = \mathcal{A} = \{a_1, \ldots, a_5\}$ for all i.

location of the agent is the state.



(a) States

If more complex, may V, Q.



(b) Actions

Figure 1.3: Illustrations of the state and action concepts. (a) There are nine states $\{s_1, \ldots, s_9\}$. (b) Each state has five possible actions $\{a_1, a_2, a_3, a_4, a_5\}$.

1.3 State transition

When taking an action, the agent may move from one state to another. Such a process is called *state transition*. For example, if the agent is in state s_1 and selects action a_2 (that is, moving rightward), then the agent moves to state s_2. Such a process can be expressed as

$$s_1 \xrightarrow{a_2} s_2.$$

We next examine two important examples.

- ◇ What is the next state when the agent attempts to go beyond the boundary, for example, taking action a_1 in state s_1? The answer is that the agent will be bounced back because it is impossible for the agent to exit the state space. Hence, we have $s_1 \xrightarrow{a_1} s_1$.

collide the boundary.

场景

- ◇ What is the next state when the agent attempts to enter a forbidden cell, for example, taking action a_2 in state s_5? Two different scenarios may be encountered. In the first scenario, although s_6 is forbidden, it is still *accessible*. In this case, the next state is s_6; hence, the state transition process is $s_5 \xrightarrow{a_2} s_6$. In the second scenario, s_6 is *not accessible* because, for example, it is surrounded by walls. In this case, the agent is bounced back to s_5 if it attempts to move rightward; hence, the state transition process is $s_5 \xrightarrow{a_2} s_5$.

  Which scenario should we consider? The answer depends on the physical environment. In this book, we consider the first scenario where the forbidden cells are accessible, although stepping into them may get punished. This scenario is more general and interesting. Moreover, since we are considering a simulation task, we can define the state transition process however we prefer. In real-world applications, the state transition process is determined by real-world dynamics.

可能产生进入 Forbidden 附近的情况

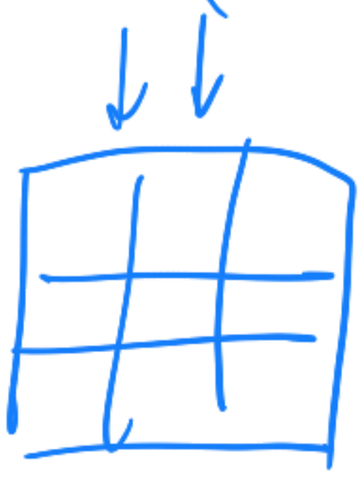
The state transition process is defined for each state and its associated actions. This process can be described by a table as shown in Table 1.1. In this table, each row

corresponds to a state, and each column corresponds to an action. Each cell indicates the next state to transition to after the agent takes an action at the corresponding state.

	a_1 (upward)	a_2 (rightward)	a_3 (downward)	a_4 (leftward)	a_5 (still)
s_1	s_1	s_2	s_4	s_1	s_1
s_2	s_2	s_3	s_5	s_1	s_2
s_3	s_3	s_3	s_6	s_2	s_3
s_4	s_1	s_5	s_7	s_4	s_4
s_5	s_2	s_6	s_8	s_4	s_5
s_6	s_3	s_6	s_9	s_5	s_6
s_7	s_4	s_8	s_7	s_7	s_7
s_8	s_5	s_9	s_8	s_7	s_8
s_9	s_6	s_9	s_9	s_8	s_9

Table 1.1: A tabular representation of the state transition process. Each cell indicates the next state to transition to after the agent takes an action at a state.

Mathematically, the state transition process can be described by conditional probabilities. For example, for s_1 and a_2, the conditional probability distribution is



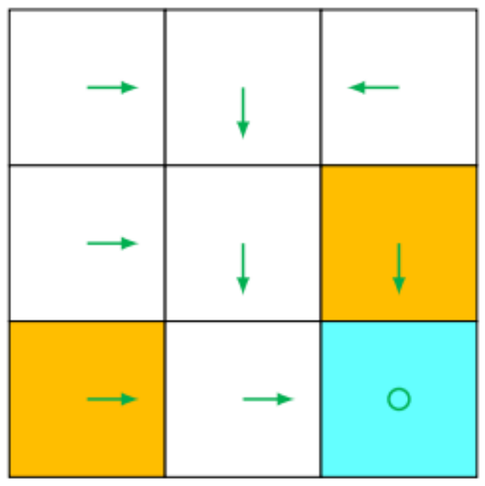
$$
\begin{aligned}
p(s_1|s_1,a_2) &= 0,\\
p(s_2|s_1,a_2) &= 1,\\
p(s_3|s_1,a_2) &= 0,\\
p(s_4|s_1,a_2) &= 0,\\
p(s_5|s_1,a_2) &= 0,
\end{aligned}
$$

which indicates that, when taking a_2 at s_1, the probability of the agent moving to s_2 is one, and the probabilities of the agent moving to other states are zero. As a result, taking action a_2 at s_1 will certainly cause the agent to transition to s_2. The preliminaries of conditional probability are given in Appendix A. Readers are strongly advised to be familiar with probability theory since it is necessary for studying reinforcement learning.

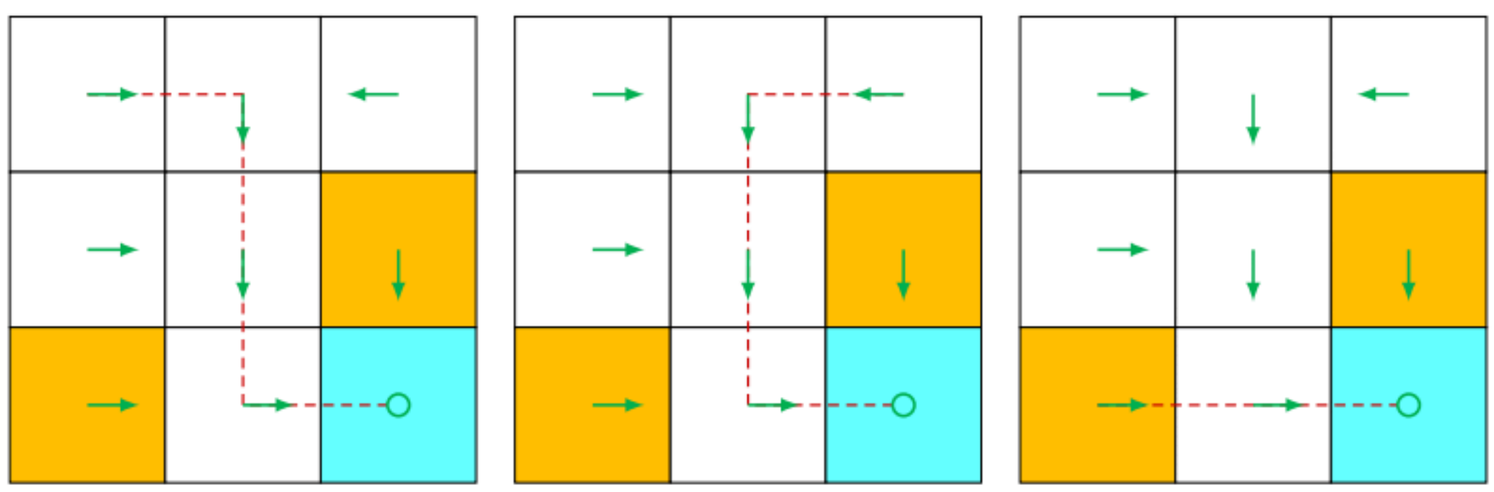
Although it is intuitive, the tabular representation is only able to describe *deterministic* state transitions. In general, state transitions can be *stochastic* and must be described by conditional probability distributions. For instance, when random wind gusts are applied across the grid, if taking action a_2 at s_1, the agent may be blown to s_5 instead of s_2. We have $p(s_5|s_1,a_2) > 0$ in this case. Nevertheless, we merely consider deterministic state transitions in the grid world examples for simplicity in this book.

1.4 Policy

A *policy* tells the agent which actions to take at every state. Intuitively, policies can be depicted as arrows (see Figure 1.4(a)). Following a policy, the agent can generate a trajectory starting from an initial state (see Figure 1.4(b)).



(a) A deterministic policy



(b) Trajectories obtained from the policy

Figure 1.4: A policy represented by arrows and some trajectories obtained by starting from different initial states.

Mathematically, policies can be described by conditional probabilities. Denote the policy in Figure 1.4 as $\pi(a|s)$, which is a conditional probability distribution function defined for *every* state. For example, the policy for s_1 is

$$\begin{aligned}
\pi(a_1|s_1) &= 0, \\
\pi(a_2|s_1) &= 1, \\
\pi(a_3|s_1) &= 0, \\
\pi(a_4|s_1) &= 0, \\
\pi(a_5|s_1) &= 0,
\end{aligned}$$

which indicates that the probability of taking action a_2 in state s_1 is one, and the probabilities of taking other actions are zero.

The above policy is *deterministic*. Policies may be *stochastic* in general. For example, the policy shown in Figure 1.5 is stochastic: in state s_1, the agent may take actions to go either rightward or downward. The probabilities of taking these two actions are the

same (both are 0.5). In this case, the policy for s_1 is

$$\pi(a_1|s_1) = 0,$$
$$\pi(a_2|s_1) = 0.5,$$
$$\pi(a_3|s_1) = 0.5,$$
$$\pi(a_4|s_1) = 0,$$
$$\pi(a_5|s_1) = 0.$$

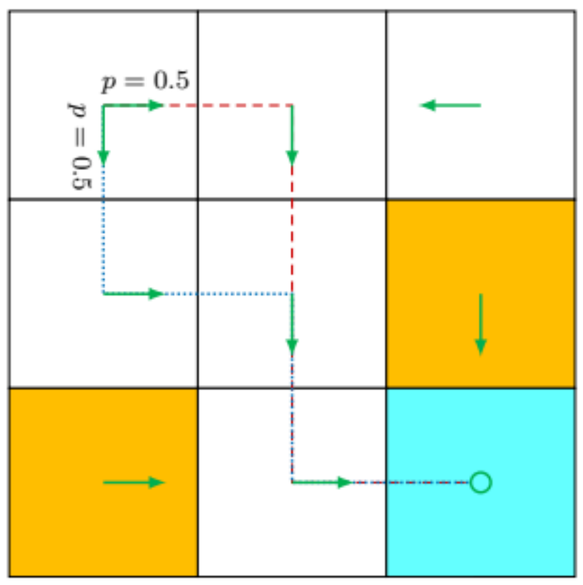


Figure 1.5: A stochastic policy. In state s_1, the agent may move rightward or downward with equal probabilities of 0.5.

Policies represented by conditional probabilities can be stored as tables. For example, Table 1.2 represents the stochastic policy depicted in Figure 1.5. The entry in the ith row and jth column is the probability of taking the jth action at the ith state. Such a representation is called a *tabular representation*. We will introduce another way to represent policies as parameterized functions in Chapter 8.

	a_1 (upward)	a_2 (rightward)	a_3 (downward)	a_4 (leftward)	a_5 (still)
s_1	0	0.5	0.5	0	0
s_2	0	0	1	0	0
s_3	0	0	0	1	0
s_4	0	1	0	0	0
s_5	0	0	1	0	0
s_6	0	0	1	0	0
s_7	0	1	0	0	0
s_8	0	1	0	0	0
s_9	0	0	0	0	1

Table 1.2: A tabular representation of a policy. Each entry indicates the probability of taking an action at a state.

1.5 Reward

Reward is one of the most unique concepts in reinforcement learning.

After executing an action at a state, the agent obtains a reward, denoted as r, as feedback from the environment. The reward is a function of the state s and action a. Hence, it is also denoted as $r(s, a)$. Its value can be a positive or negative real number or zero. Different rewards have different impacts on the policy that the agent would eventually learn. Generally speaking, with a positive reward, we encourage the agent to take the corresponding action. With a negative reward, we discourage the agent from taking that action.

In the grid world example, the rewards are designed as follows:

- ◇ If the agent attempts to exit the boundary, let $r_{\text{boundary}} = -1$.
- ◇ If the agent attempts to enter a forbidden cell, let $r_{\text{forbidden}} = -1$.
- ◇ If the agent reaches the target state, let $r_{\text{target}} = +1$.
- ◇ Otherwise, the agent obtains a reward of $r_{\text{other}} = 0$.

Special attention should be given to the target state s_9. The reward process does not have to terminate after the agent reaches s_9. If the agent takes action a_5 at s_9, the next state is again s_9, and the reward is $r_{\text{target}} = +1$. If the agent takes action a_2, the next state is also s_9, but the reward is $r_{\text{boundary}} = -1$.

A reward can be interpreted as a human-machine interface, with which we can guide the agent to behave as we expect. For example, with the rewards designed above, we can expect that the agent tends to avoid exiting the boundary or stepping into the forbidden cells. Designing appropriate rewards is an important step in reinforcement learning. This step is, however, nontrivial for complex tasks since it may require the user to understand the given problem well. Nevertheless, it may still be much easier than solving the problem with other approaches that require a professional background or a deep understanding of the given problem.

The process of getting a reward after executing an action can be intuitively represented as a table, as shown in Table 1.3. Each row of the table corresponds to a state, and each column corresponds to an action. The value in each cell of the table indicates the reward that can be obtained by taking an action at a state.

One question that beginners may ask is as follows: if given the table of rewards, can we find good policies by simply selecting the actions with the greatest rewards? The answer is no. That is because these rewards are *immediate rewards* that can be obtained after taking an action. To determine a good policy, we must consider the *total reward* obtained in the long run (see Section 1.6 for more information). An action with the greatest immediate reward may not lead to the greatest total reward.

Although intuitive, the tabular representation is only able to describe *deterministic* reward processes. A more general approach is to use conditional probabilities $p(r|s, a)$ to describe reward processes. For example, for state s_1, we have

$$p(r = -1|s_1, a_1) = 1, \quad p(r \neq -1|s_1, a_1) = 0.$$

only

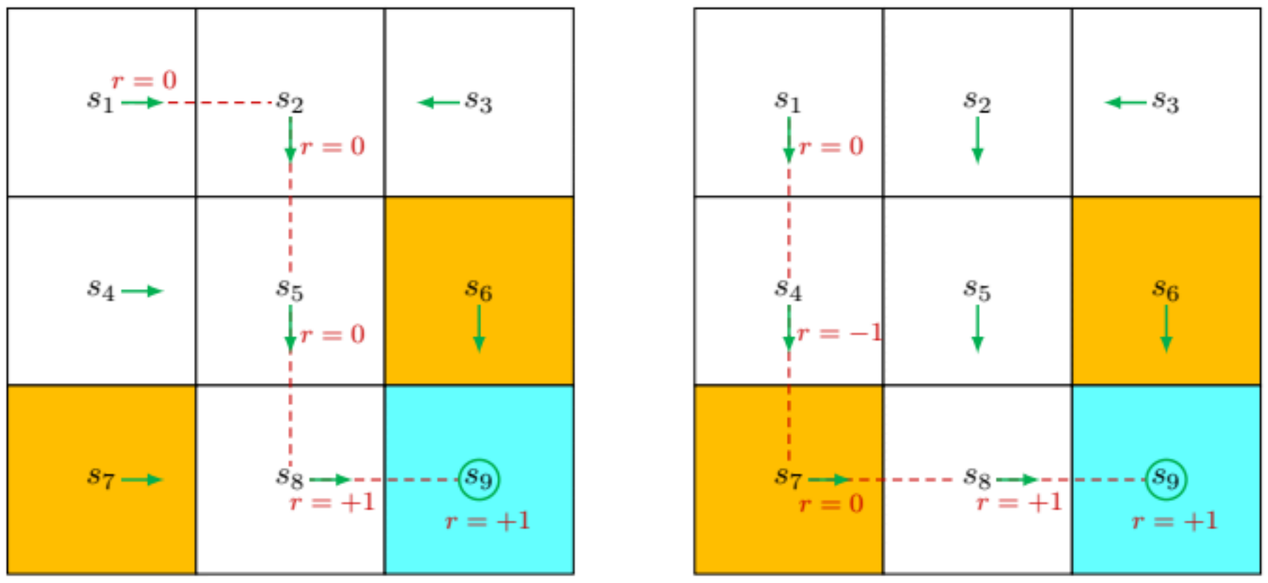
deterministic

	a_1 (upward)	a_2 (rightward)	a_3 (downward)	a_4 (leftward)	a_5 (still)
s_1	r_{boundary}	0	0	r_{boundary}	0
s_2	r_{boundary}	0	0	0	0
s_3	r_{boundary}	r_{boundary}	$r_{\text{forbidden}}$	0	0
s_4	0	0	$r_{\text{forbidden}}$	r_{boundary}	0
s_5	0	$r_{\text{forbidden}}$	0	0	0
s_6	0	r_{boundary}	r_{target}	0	$r_{\text{forbidden}}$
s_7	0	0	r_{boundary}	r_{boundary}	$r_{\text{forbidden}}$
s_8	0	r_{target}	r_{boundary}	$r_{\text{forbidden}}$	0
s_9	$r_{\text{forbidden}}$	r_{boundary}	r_{boundary}	0	r_{target}

Table 1.3: A tabular representation of the process of obtaining rewards. Here, the process is deterministic. Each cell indicates how much reward can be obtained after the agent takes an action at a given state.

This indicates that, when taking a_1 at s_1, the agent obtains $r = -1$ with certainty. In this example, the reward process is deterministic. In general, it can be stochastic. For example, if a student studies hard, he or she would receive a positive reward (e.g., higher grades on exams), but the specific value of the reward may be uncertain.

1.6 Trajectories, returns, and episodes



(a) Policy 1 and the trajectory (b) Policy 2 and the trajectory

Figure 1.6: Trajectories obtained by following two policies. The trajectories are indicated by red dashed lines.

A *trajectory* is a state-action-reward chain. For example, given the policy shown in Figure 1.6(a), the agent can move along a trajectory as follows:

$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9.$$

The *return* of this trajectory is defined as the sum of all the rewards collected along the trajectory:

$$\text{return} = 0 + 0 + 0 + 1 = 1. \tag{1.1}$$

Returns are also called *total rewards* or *cumulative rewards*.

Returns can be used to *evaluate* policies. For example, we can evaluate the two policies in Figure 1.6 by comparing their returns. In particular, starting from s_1, the return obtained by the left policy is 1 as calculated above. For the right policy, starting from s_1, the following trajectory is generated:

$$s_1 \xrightarrow[r=0]{a_3} s_4 \xrightarrow[r=-1]{a_3} s_7 \xrightarrow[r=0]{a_2} s_8 \xrightarrow[r=+1]{a_2} s_9.$$

The corresponding return is

$$\text{return} = 0 - 1 + 0 + 1 = 0. \tag{1.2}$$

The returns in (1.1) and (1.2) indicate that the left policy is better than the right one since its return is greater. This mathematical conclusion is consistent with the intuition that the right policy is worse since it passes through a forbidden cell.

A return consists of an *immediate reward* and *future rewards*. Here, the immediate reward is the reward obtained after taking an action at the initial state; the future rewards refer to the rewards obtained after leaving the initial state. It is possible that the immediate reward is negative while the future reward is positive. Thus, which actions to take should be determined by the return (i.e., the total reward) rather than the immediate reward to avoid short-sighted decisions.

The return in (1.1) is defined for a finite-length trajectory. Return can also be defined for infinitely long trajectories. For example, the trajectory in Figure 1.6 stops after reaching s_9. Since the policy is well defined for s_9, the process does not have to stop after the agent reaches s_9. We can design a policy so that the agent stays still after reaching s_9. Then, the policy would generate the following infinitely long trajectory:

$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_3} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9 \xrightarrow[r=1]{a_5} s_9 \xrightarrow[r=1]{a_5} s_9 \ldots$$

The direct sum of the rewards along this trajectory is

$$\text{return} = 0 + 0 + 0 + 1 + 1 + 1 + \cdots = \infty,$$

which unfortunately diverges. Therefore, we must introduce the *discounted return* concept for infinitely long trajectories. In particular, the discounted return is the sum of the discounted rewards:

$$\text{discounted return} = 0 + \gamma 0 + \gamma^2 0 + \gamma^3 1 + \gamma^4 1 + \gamma^5 1 + \ldots, \tag{1.3}$$

where $\gamma \in (0, 1)$ is called the *discount rate*. When $\gamma \in (0, 1)$, the value of (1.3) can be

calculated as

$$\text{discounted return} = \gamma^3(1+\gamma+\gamma^2+\dots) = \gamma^3\frac{1}{1-\gamma}.$$

The introduction of the discount rate is useful for the following reasons. First, it removes the stop criterion and allows for infinitely long trajectories. Second, the discount rate can be used to adjust the emphasis placed on near- or far-future rewards. In particular, if γ is close to 0, then the agent places more emphasis on rewards obtained in the near future. The resulting policy would be short-sighted. If γ is close to 1, then the agent places more emphasis on the far future rewards. The resulting policy is far-sighted and dares to take risks of obtaining negative rewards in the near future. These points will be demonstrated in Section 3.5.

One important notion that was not explicitly mentioned in the above discussion is the *episode*. When interacting with the environment by following a policy, the agent may stop at some *terminal states*. The resulting trajectory is called an *episode* (or a *trial*). If the environment or policy is stochastic, we obtain different episodes when starting from the same state. However, if everything is deterministic, we always obtain the same episode when starting from the same state.

An episode is usually assumed to be a finite trajectory. Tasks with episodes are called *episodic tasks*. However, some tasks may have no terminal states, meaning that the process of interacting with the environment will never end. Such tasks are called *continuing tasks*. In fact, we can treat episodic and continuing tasks in a unified mathematical manner by converting episodic tasks to continuing ones. To do that, we need well define the process after the agent reaches the terminal state. Specifically, after reaching the terminal state in an episodic task, the agent can continue taking actions in the following two ways.

- ◇ First, if we treat the terminal state as a special state, we can specifically design its action space or state transition so that the agent stays in this state forever. Such states are called *absorbing states*, meaning that the agent never leaves a state once reached. For example, for the target state s_9, we can specify $\mathcal{A}(s_9) = \{a_5\}$ or set $\mathcal{A}(s_9) = \{a_1, \dots, a_5\}$ with $p(s_9|s_9, a_i) = 1$ for all $i = 1, \dots, 5$.
- ◇ Second, if we treat the terminal state as a normal state, we can simply set its action space to the same as the other states, and the agent may leave the state and come back again. Since a positive reward of $r = 1$ can be obtained every time s_9 is reached, the agent will eventually learn to stay at s_9 forever to collect more rewards. Notably, when an episode is infinitely long and the reward received for staying at s_9 is positive, a discount rate must be used to calculate the discounted return to avoid divergence.

In this book, we consider the second scenario where the target state is treated as a normal state whose action space is $\mathcal{A}(s_9) = \{a_1, \dots, a_5\}$.

1.7 Markov decision processes

The previous sections of this chapter illustrated some fundamental concepts in reinforcement learning through examples. This section presents these concepts in a more formal way under the framework of Markov decision processes (MDPs).

An MDP is a general framework for describing stochastic dynamical systems. The key ingredients of an MDP are listed below.

- ◇ Sets:
  - State space: the set of all states, denoted as $\mathcal{S}$.
  - Action space: a set of actions, denoted as $\mathcal{A}(s)$, associated with each state $s \in \mathcal{S}$.
  - Reward set: a set of rewards, denoted as $\mathcal{R}(s,a)$, associated with each state-action pair (s,a).
- ◇ Model:
  - State transition probability: In state s, when taking action a, the probability of transitioning to state s' is $p(s'|s,a)$. It holds that $\sum_{s' \in \mathcal{S}} p(s'|s,a) = 1$ for any (s,a).
  - Reward probability: In state s, when taking action a, the probability of obtaining reward r is $p(r|s,a)$. It holds that $\sum_{r \in \mathcal{R}(s,a)} p(r|s,a) = 1$ for any (s,a).
- ◇ Policy: In state s, the probability of choosing action a is $\pi(a|s)$. It holds that $\sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1$ for any $s \in \mathcal{S}$.
- ◇ Markov property: The *Markov property* refers to the memoryless property of a stochastic process. Mathematically, it means that

$$\begin{aligned} p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) &= p(s_{t+1}|s_t, a_t), \\ p(r_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) &= p(r_{t+1}|s_t, a_t), \end{aligned} \tag{1.4}$$

where t represents the current time step and $t+1$ represents the next time step. Equation (1.4) indicates that the next state or reward depends merely on the current state and action and is independent of the previous ones. The Markov property is important for deriving the fundamental Bellman equation of MDPs, as shown in the next chapter.

Here, $p(s'|s,a)$ and $p(r|s,a)$ for all (s,a) are called the *model* or *dynamics*. The model can be either *stationary* or *nonstationary* (or in other words, time-invariant or time-variant). A stationary model does not change over time; a nonstationary model may vary over time. For instance, in the grid world example, if a forbidden area may pop up or disappear sometimes, the model is nonstationary. In this book, we only consider stationary models.

One may have heard about the Markov processes (MPs). What is the difference between an MDP and an MP? The answer is that, once the policy in an MDP is fixed, the MDP degenerates into an MP. For example, the grid world example in Figure 1.7 can be abstracted as a Markov process. In the literature on stochastic processes, a Markov process is also called a Markov chain if it is a discrete-time process and the number of states is finite or countable [1]. In this book, the terms "Markov process" and "Markov chain" are used interchangeably when the context is clear. Moreover, this book mainly considers *finite* MDPs where the numbers of states and actions are finite. This is the simplest case that should be fully understood.

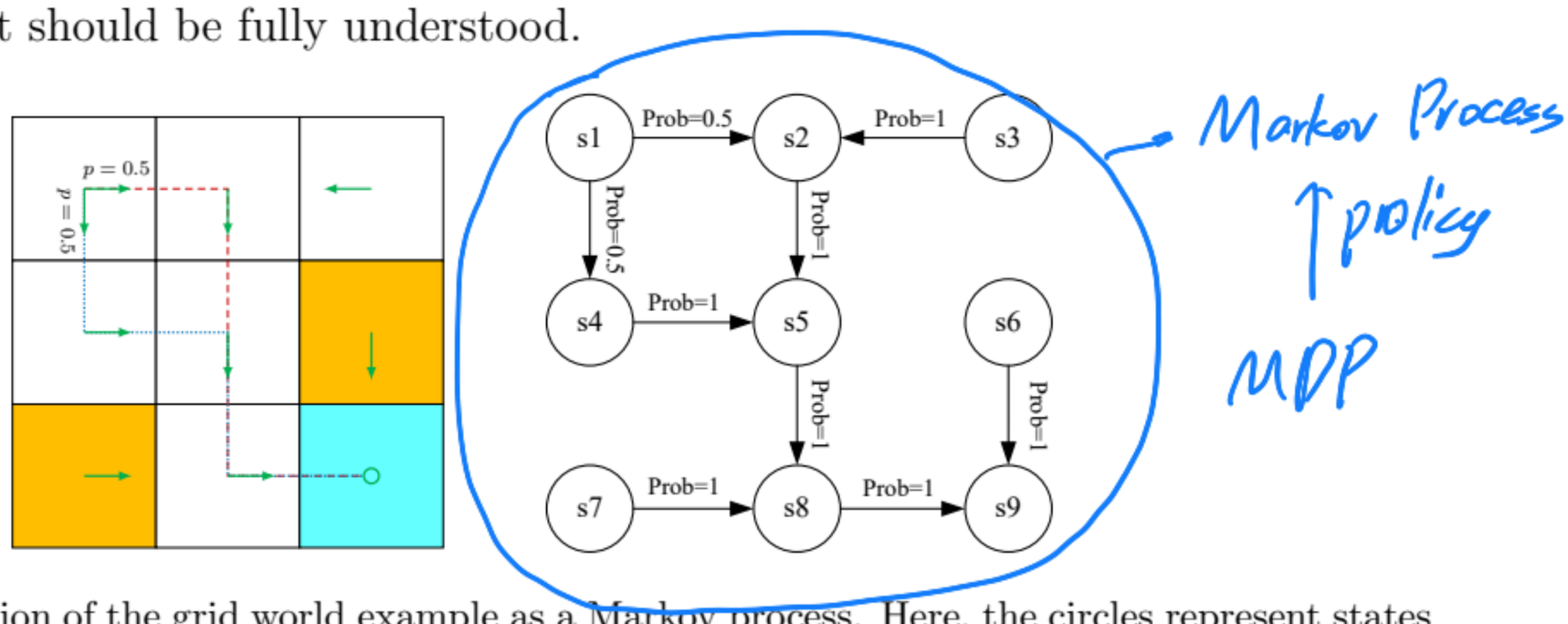


Figure 1.7: Abstraction of the grid world example as a Markov process. Here, the circles represent states and the links with arrows represent state transitions.

Finally, reinforcement learning can be described as an agent-environment interaction process. The *agent* is a decision-maker that can sense its state, maintain policies, and execute actions. Everything outside of the agent is regarded as the *environment*. In the grid world examples, the agent and environment correspond to the robot and grid world, respectively. After the agent decides to take an action, the actuator executes such a decision. Then, the state of the agent would be changed and a reward can be obtained. By using interpreters, the agent can interpret the new state and the reward. Thus, a closed loop can be formed.

1.8 Summary

This chapter introduced the basic concepts that will be widely used in the remainder of the book. We used intuitive grid world examples to demonstrate these concepts and then formalized them in the framework of MDPs. For more information about MDPs, readers can see [1, 2].

1.9 Q&A

◇ Q: Can we set all the rewards as negative or positive?

A: In this chapter, we mentioned that a positive reward would encourage the agent to take an action and that a negative reward would discourage the agent from taking

the action. In fact, it is the *relative* reward values instead of the *absolute* values that determine encouragement or discouragement.

More specifically, we set $r_{\text{boundary}} = -1$, $r_{\text{forbidden}} = -1$, $r_{\text{target}} = +1$, and $r_{\text{other}} = 0$ in this chapter. We can also add a common value to all these values without changing the resulting optimal policy. For example, we can add -2 to all the rewards to obtain $r_{\text{boundary}} = -3$, $r_{\text{forbidden}} = -3$, $r_{\text{target}} = -1$, and $r_{\text{other}} = -2$. Although the rewards are all negative, the resulting optimal policy is unchanged. That is because optimal policies are invariant to affine transformations of the rewards. Details will be given in Chapter 3.5.

- Q: Is the reward a function of the next state?

  A: We mentioned that the reward r depends only on s and a but not the next state s'. However, this may be counterintuitive since it is the next state that determines the reward in many cases. For example, the reward is positive when the next state is the target state. As a result, a question that naturally follows is whether a reward should depend on the next state. A mathematical rephrasing of this question is whether we should use $p(r|s, a, s')$ where s' is the next state rather than $p(r|s, a)$. The answer is that r depends on s, a, and s'. However, since s' also depends on s and a, we can equivalently write r as a function of s and a: $p(r|s, a) = \sum_{s'} p(r|s, a, s')p(s'|s, a)$. In this way, the Bellman equation can be easily established as shown in Chapter 2.