THE LOGIC OF
ADAPTIVE BEHAVIOR

KNOWLEDGE REPRESENTATION AND ALGORITHMS FOR
THE MARKOV DECISION PROCESS FRAMEWORK
IN FIRST-ORDER DOMAINS
THE LOGIC OF
ADAPTIVE BEHAVIOR

KNOWLEDGE REPRESENTATION AND ALGORITHMS FOR
THE MARKOV DECISION PROCESS FRAMEWORK
IN FIRST-ORDER DOMAINS
DECISION MAKING IS A VERY CHALLENGING PROBLEM, both in human thinking as in artificial intelligence systems. While you are reading this text, many things take place inside your brain. For one thing, you are trying to stay focused on reading this, you are trying to keep yourself nourished, you are trying to remember to send this very important e-mail, and so on. Furthermore, you know how to ride a bicycle, you know how to make coffee and you may know how to write a report using \LaTeX, and many more such things. And, additionally, you may have knowledge about Bayesian networks, your left ear, table spoons and possibly even about ninja swords. How on earth can you possibly decide on your next action?

Apparently, humans have the ability to store many types of knowledge, operational skills, and do many types of reasoning processes, all at the same time. A complete explanation of this phenomenon, and a working computer-based implementation of such processes, counts as the Holy Grail of the field of artificial intelligence. Therefore, let us first take a look at the significantly more restricted setting of decision making in Figure 1.1. These examples were described by Tversky and Kahneman (1981), who experimented with variants of essentially the same decision problem and investigated the influence of how people interpret the problem on their decisions. The variance in the answer distribution in the two problems is explained by the authors as

"The majority choice in this problem is risk averse: the prospect of certainly saving 200 lives is more attractive than a risky prospect of equal expected value, that is, a one-in-three chance of saving 600 lives. [...] The majority choice in problem 2 is risk taking: the certain death of 400 people is less acceptable than the two-in-three chance that 600 will die. The preferences in problems 1 and 2 illustrate a common pattern: choices involving gains are often risk averse and choices involving losses are often risk taking. However, it is easy to see that the two problems are effectively identical."

Interestingly, for humans it seems to matter how a particular problem is represented. Both problems pose the same dilemma, but trigger different responses, due to a concept called decision frame that refers to the decision-maker's conception of the acts, outcomes, and contingencies associated with a particular choice.

From this example, we see that the representation of a decision problem can be just as important as the intrinsic difficulty of making the decision itself. On the contrary, for
Imagine that the U.S. is preparing for the outbreak of an unusual Asian disease, which is expected to kill 600 people. Two alternative programs to combat the disease have been proposed. Assume that the exact scientific estimate of the consequences of the programs are as follows:

<table>
<thead>
<tr>
<th>Problem 1 [N=152]</th>
<th>Problem 2 [N=155]</th>
</tr>
</thead>
<tbody>
<tr>
<td>If Program C is adopted, 400 people will die [22 percent]</td>
<td>If Program A is adopted, 200 people will be saved. [72 percent]</td>
</tr>
<tr>
<td>If Program D is adopted, there is a ( \frac{1}{3} ) probability that nobody will die, and a ( \frac{2}{3} ) probability that 600 people will die [78 percent]</td>
<td>If Program B is adopted, there is a ( \frac{1}{3} ) probability that 600 people will be saved, and a ( \frac{2}{3} ) probability that no people will be saved [28 percent]</td>
</tr>
</tbody>
</table>

**Figure 1.1:** A deceiving decision problem for humans. \( N \) is the number of people in the survey, and bracketed numbers in the answers denote what percentage of respondents chose a particular answer.

a computer, the representation of a problem is relatively meaningless. As long as all the necessary information is present, and it knows how the answer can be computed in whatever mechanical way, correct answers can be ensured, i.e. computers are rational entities (Russell, 1997). Still they are heavily dependent on representation, albeit in a different way. It does not influence the correctness of the computer’s decisions, but it does influence the range of problems they can solve, the generality of their solutions and furthermore how efficient solutions are computed. This is the main theme in this book.

Now, deciding a single thing to do may already be challenging and dependent on representation. But let us go one step further, to sequential decision making. Consider the game of Chess. Each move in a Chess game is important, but it is the complete game consisting of around 40 consecutive moves that determines winning or losing. Playing a bad move might not be too disastrous for winning the game in the end, though this also depends on the opponent. To play a game of Chess successfully requires one to plan ahead, and to cope with possibly unforeseen circumstances along the way. This is also influenced by uncertainty about your opponent’s moves, which can make your planned strategy fail and force you to adjust.

Whereas computers can be programmed to play games like Chess, or to perform other sequential decision making tasks such as navigating a robot in a factory, ultimately we would like an intelligent system to learn these things by itself. When humans learn how to play Chess, they use a variety of learning techniques to master the game. Initially, they have to be taught the rules of the game, but after that, they usually acquire increasing levels of play by practicing the game, and observing the effects of moves on the outcome of the game. People are not told the optimal moves for each possible Chess board, but they learn to evaluate moves and positions, in order to play better moves. Furthermore, they generalize what they have learned such that such knowledge can be applied in ‘similar’ situations or when playing against ‘similar’ opponents. In this book we study computer algorithms that mimic this type of learning in sequential decision making tasks. A central role is played by the representation of such problems, because these determine which types of problems can be learned (see Section 1.2 for an extended example).
1.1 Science and Engineering of Adaptive Behavior

Summarizing, this book is about learning behaviors for **sequential decision making tasks** in which there is a significant amount of **uncertainty** and **limited, delayed feedback**. The core topic is about employing **first-order knowledge representation** in such tasks, which is a particular way to 'see' the problem in terms of **objects** and **relations** between objects. The purpose of this introductory chapter is threefold. In the first place, its intention is to introduce the reader to the topic and focus of the research as described in this book. Taking a helicopter view, the location of the matter can be found by zooming in on the field of **artificial intelligence**, then on **machine learning** and finally on **reinforcement learning**. The exact focus of the research is concerned with the **representational aspects** of the reinforcement learning methodology, and in particular the use of powerful **first-order** or **relational** representational devices. The second goal of this chapter is to provide a road map through the chapters of this book. The final aim of the chapter is to highlight the contributions of this book, and their embedding in an existing body of research as reported in the literature.

### 1.1. Science and Engineering of Adaptive Behavior

We are interested in creating artificial behaviors for sequential decision making. More specifically, we are interested in artificial systems that *learn* how to *do* something. The field of **artificial intelligence** (AI) has been studying this for decades, taking inspiration from many different fields.

#### 1.1.1 Artificial Intelligence

AI is a large field of research that tries to build systems that perform tasks *in which it is understood that some form of intelligence is required*. AI is generally seen as a subfield of computer science, but its connections with and influences from other fields are much more diverse and include **cognitive science, engineering, psychology, biology, sociology, economics, philosophy** and **mathematics**. Many books exist that provide general treatments of the field (Nilsson, 1980; Görtz et al., 2003; Luger, 2002; Russell and Norvig, 2003) of which some are more **logically** oriented (Poole et al., 1998; Minker, 2000b), others deal with **embodied** (Pfeifer and Scheier, 1999) views on AI, and yet others deal with the conceptual ideas of AI (Hofstadter, 1979; Minsky, 1985; Haugeland, 1997; Baum, 2004).

AI was originally founded in 1956 and has been occupied with studying, and building **minds** (Haugeland, 1997). An exact characterization of **intelligence** is not all that important to understanding it. Whether some system is intelligent will always be debatable, and therefore, the important question is the following. Given some behavior (by e.g. a human or an animal) that we find interesting in some way, how does this behavior come about? Many sub-fields in AI have developed based on this question, studying various topics such as **memory, vision, logical and commonsense reasoning, navigation, physical movement, evolution, brain functioning**, and most importantly for this book, **decision making and learning**. Much has been achieved so far, also witnessed by widespread use of **expert systems, datamining**, and even **fuzzy controllers** in washing machines and much has still to come.

---

1 Predictions about the future of AI trigger many sorts of reactions, and are often disproved later. For example (Hofstadter, 1979, p.678)’s predictions on the possibility of a computer beating anyone with *Chess* was disproved by the victory of *Deep Blue* over the best human player Gary Kasparov (Schaeffer and Plaat, 1997). Other well-known predictions on whether a robot team will beat the human best team at soccer in...
Since the eighties, AI has developed into a strong discipline of science, embracing approaches from other fields such as control theory, statistics, mathematics and operations research, and supported by theories, rigorous experiments, and applications. Owing much to the work by Pearl (1988), AI is now dominated by probabilistic approaches. In the mid-nineties, the agent metaphor (Wooldridge and Jennings, 1995) became popular as a core object of study (Russell and Norvig, 2003) and nowadays the game industry – which dominates the movie industry in terms of financial investments – has discovered AI as a way to make their products smarter.

An important dichotomy in AI is that between general-purpose systems and performance systems (see Nilsson, 1995, 2005, for further discussion). The first is about the systems AI basically started out with; those that aim at understanding and building general, human-like intelligent systems. The second is about programs that are highly specialized and limited to a particular area of expertise. It is related to an old, yet persistent, debate in AI between strong AI in which the appropriately programmed computer is really considered a mind, and weak AI, in which the principal value of a computer is to be a very powerful tool to formulate and test hypotheses in a rigorous and precise fashion. Many of the current AI approaches belong to the latter category, causing AI to be subdivided into a large number of nearly disjoint fields, for example logical inference vs. probabilistic inference, empirical vs. purely theoretic approaches, and many more fine-grained subdivisions. It includes the work in this book, which is targeted at a very specific area, that of learning sequential decision making. Yet, we argue that the best way is to pursue research into such individual subdivisions, while keeping in mind the needs and constraints of general AI architectures. Or, so to say, keeping the eye on the prize (Nilsson, 1995).

1.1.2 Constructing Artificial Behavior

AI has produced several distinct ways to build intelligent agents that can perform well in sequential decision making problems under uncertainty. Note that we focus here on reactive behaviors in which the agent’s main task is to choose an action based on its current state. In general, we can distinguish three main types of approaches to obtain a controller for the robot’s actions, which are programming, planning or reasoning, and learning.

1.1.2.1 Programming

The first thing that comes to mind when creating an agent for a specific task is to write a program that completely drives the agent’s behavior. The advantages are that the behavior can be tested, it can be set up and programmed in a modular way, and that guarantees can be given about its performance. However, for most realistic problems this is impossible to do. There can be uncertainty about the environment’s dynamics, about possible effects of actions, about behaviors of possible other agents in the environment and so on. Furthermore, some aspects of the environment may be inaccessible to the agent, such that it misses vital information for its current decision. In addition, programmed behaviors are not robust to changes in the environment, or unforeseen circumstances. In other words, programmed systems are often brittle (Holland, 1986), and adaptive systems are preferred.

---

2050, and whether robots will dominate humans in the near future, remain to be seen.

2When graphical techniques were still developing, games would advertise with increasingly better looking graphics. Nowadays they advertise with slogans such as "Enhanced AI opponents included".
1.1.2.2 REASONING AND PLANNING

Instead of fixing the complete behavior beforehand by programming, a second option is to supply all information about the environment to the agent and let it reason about it to plan ahead a suitable course of action. In deterministic environments this is very well possible, though in environments with uncertainty about the outcomes of actions it becomes more challenging because there are no guarantees that the current plan will reach the goal. On the other hand, giving the agent the ability to plan enables it to cope with such circumstances, for example by adjusting the plan when needed.

For planning to work, the agent must first know everything about the domain. This includes facts, for example that room 1 is also known as the coffee room, but also knowledge about how certain things in the environment change either because of the agent’s actions or because of external factors. The main challenge is to make this knowledge as complete and as precise as possible. Haddon (2003) tells the story of a fifteen year old boy named Christopher Boon who has Asperger’s Syndrome, and in many ways Christopher requires the same kind of precision that is required for a computer.

"And this is because when people tell you what to do it is usually confusing and does not make sense. For example, people often say 'Be quiet', but they don't tell you how long to be quiet for. Or you see a sign which says KEEP OFF THE GRASS but it should say KEEP OFF THE GRASS AROUND THIS SIGN or KEEP OFF THE GRASS IN THIS PARK because there is lots of grass you are allowed to walk on.” (Haddon, 2003, p.38).

Although complete and precise formalizations are required, there is a delicate trade-off with the employment of this knowledge in an actual reasoning system. Because computers lack a kind of commonsense reasoning, they cannot naturally distinguish between relevant and irrelevant lines of reasoning. For example, Dennett (1998) describes a robot that spends all its time reasoning about the possible consequences of its actions, without actually doing anything anymore. Thus, in addition to knowledge, for planning to work the agent must have efficient reasoning mechanisms that use the information wisely. Otherwise it might end up thinking about (or even doing) stupid things, like Christopher.

"Stupid things are things like emptying a jar of peanut butter onto the table in the kitchen and making it level with a knife so it covers all the table right to the edges, or burning things on the gas stove to see what happened to them, like my shoes or silver foil or sugar.” (Haddon, 2003, p.60).

Planning approaches are widespread, and in Section 1.3.1.3 we will briefly outline some historical developments. Most of these approaches cannot be employed in domains with significant uncertainty, and are impossible to apply when information about the dynamics of the domain is absent.

1.1.2.3 LEARNING

In the context of uncertainty and the inability of specifying all necessary information beforehand, it would be best to supply the agent with all the information that is available, and let it learn from experience how to perform the task. In other words, "learning is more
Introduction

Machine learning (ML) (Mitchell, 1997) is a large sub-field of AI and it deals with various kinds of learning, or adaptive systems. A general definition of learning is the process or technique by which a device modifies its own behavior as the result of its past experience and performance.

Learning algorithms can be classified along several dimensions, which include the type of problem (e.g. classification, behavior), the knowledge representation used (see more on this later), and the source of the learning experiences. Examples of the latter include datasets and simulation environments, but also prior knowledge that may be available about the domain. One of the most important dimensions in ML algorithms is the amount of feedback that is available to the learning system. Basically, there are three types of amounts, ranging from full feedback to essentially none.

**Supervised learning** is the most common form of ML. Usually the desired result is a mapping from problem instances to a set of class values. A training set that contains examples of problem instances along with their desired class label are given to the system. The task now is to take the training set and use it to construct a generalized mapping that can label the instances correctly, but in addition, that can label other, unseen examples correctly too. An example of such a problem can be found in direct marketing. Let us assume a company has much information about its customers, for example buying habits, living environment, age, income and so on. Based on previous experience on which customers respond to prospects the company sends out, a learning algorithm could use a relatively small set of customers to learn a mapping that classifies customers into responsive and non-responsive. After learning, the mapping could be applied to all customers to predict whether it would make sense to send out brochures to a particular customer, thereby maximizing the efficiency of the marketing efforts.

When the class labels are discrete symbols, as in our example, then this type of learning is called classification. If the mapping is required to predict real numbers, it is called regression. Classification could be used to learn behaviors, though the problem is that one would need correct labels (i.e. actions) for all examples (i.e. states), generating problems similar to the programming setting described above. However, we will see that supervised learning algorithms are used in the process of learning behaviors, though embedded in the reinforcement learning paradigm.

**Unsupervised learning** is characterized by a complete lack of feedback. Usually the goal of learning is to find a clustering of the problem instances. For example, a company can try to find groups of customers that are 'similar', in some way. Often there is some feedback that is measured in terms of how useful the clustering is for another task. Another application is to find association rules that express regularities in customer's data; for example, people who buy chips often also buy beer. Unsupervised learning algorithms are often used in behavior learning systems to cluster the state space into regions that are in some way similar, often called state quantization. For example, one can cluster states that require the same action, or that have a similar distance to the goal state.

The setting that is most relevant for learning behaviors is the reinforcement learning setting. It is characterized by limited and often delayed feedback. Because it is the main

---

3Learning versus (genetically) prewiring refers to the learning versus programming dichotomy. Yet, artificial evolution has been used for decades as a population-based alternative to ML approaches (e.g. see the work by Holland, 1975). Such evolutionary approaches evolve complete populations of individuals from which the best functioning (i.e. the fittest) individual for some particular environment is selected.
1.1 Science and Engineering of Adaptive Behavior

The topic of this book, we describe it in somewhat more detail in the following paragraph.

1.1.3 The Reinforcement Learning Paradigm

Reinforcement Learning (RL) (Kaelbling et al., 1996; Sutton and Barto, 1998) is a learning paradigm that is – if we look at the amount of feedback that is given to the learner – positioned somewhere between supervised and unsupervised learning. In a typical RL task, an environment consists of a set of distinct states, one of which is the current state. In each of these states an agent can choose an action from a predefined set of actions. After performing an action the current state is changed to another state, based on a probabilistic transition function. In addition, the agent receives a numerical reward that is determined by a reward function. The objective of the agent now is to choose its actions in such a way that the sum of the rewards obtained by making transitions from state to state, is maximized.4

The states, actions, transition function and reward function together make up a Markov decision process (MDP). An important aspect of MDPs is the so-called Markov assumption, which states that the current state provides enough information to make an optimal decision. That is, the agent can choose its best action by looking only at the current state; no other information is needed. For example, this is true for Chess, but not for poker. A variety of problems can be modeled using MDPs. A goal-based task is one where there are one or more goal states. In this type, the agent only receives a positive reward for reaching such a state; on all other transitions it gets zero reward. An example of such environments is a maze in which the task is to find the exit. In other types of environments there is no goal state and the task simply is to maximize the total reward in the long run.

The action choices of the agent are kept in a policy that stores for each state the action that the agent will choose. An optimal policy is that policy that will gain the most reward when applied in the environment, i.e., the MDP. Now we could program the optimal policy directly into the agent, or we could use reasoning, but here we want to learn them. Whereas there exist algorithms that learn policies directly, most methods employ value functions to facilitate learning. The value function of a policy expresses for each state how much reward will be obtained in the future if we start in that state and use the policy to select all future actions. An action value function expresses for each state the expected reward in the future if that action is taken. An optimal value function is the value function of the optimal policy. If we would have the optimal value function, optimal action selection would be easy; we simply take the action that will lead us to the state with the highest (expected) value. Thus, learning an optimal policy can be achieved by learning an optimal value function and to do this there are basically two types of algorithms.

The first solution algorithm is dynamic programming (DP). A crucial assumption is that one has complete knowledge of both the transition and the reward function. DP algorithms typically start with a default value for each state, e.g., zero. Then, they iteratively recompute the value of each state as an expected value over all transitions (and rewards)

---

4 Usually, in AI, RL approaches try to maximize the rewards. However, in the context of operations research one often sees the opposite, where rewards reflect costs and the agent must try to minimize the sum of rewards (e.g., see Bertsekas and Tsitsiklis, 1996).

5 Note that an MDP behaves probabilistically. Thus, actions can always have less-than-optimal effects, and we can only choose to maximize the expected value of the next state.
to other states and their values. Because all the information about the environment is known, DP algorithms can be shown to compute optimal value functions, and thus optimal policies. Note that, although value functions are iteratively improved, DP is more similar to planning than to learning.

The second type of algorithms is generally referred to as RL. Here, the agent has no knowledge about transition probabilities or rewards. Initially, the agent starts with a random value function and a random policy, in some state \( s \). It chooses some action using its policy and sees the result, i.e. a new state \( s' \) and a reward. Now, if the reward plus the value of the new state is higher than predicted by the value function, the agent increases that value by a small amount. If it is lower, then it decreases the original value. In this way, the value function becomes an improved version of the original one, caused by real experience. And it makes sense. For example, let us assume I can normally predict that it takes me 20 minutes to get home on my bike. On some day, I start at 16:00h, and after five minutes of traveling I meet a colleague on the street and we spend 10 minutes talking. After the conversation, at 16:15h, I update my original time of arrival of 16:20h to 16:30h, because now it takes me still 20 – 5 = 15 minutes on the bike. So, I have updated my original prediction of 20 minutes to 30, based on actual experience.

RL approaches learn from experience to estimate value functions. Now there are two aspects that make RL difficult. One is the problem of delayed rewards. For example, when playing a game such as Chess, all rewards obtained during the game will be zero, except when entering one of the goal states, e.g. when winning the game. Depending on the number of actions taken in order to reach the goal state, learning the value of the initial state may take many games before the goal state reward is propagated to this initial state. A second challenge in RL is something that is called the exploration–exploitation problem. If the agent would always choose the best action based on its current value function (exploitation), it would never find out whether there are possibly better actions. So, in order to find those, it sometimes has to ‘try out’ worse actions that enable the agent to find other courses of actions (exploration) that might deliver more reward. Balancing this trade-off is vital for finding an optimal policy.

1.2. You Can Only Learn What You Can Represent

Talking about generic states and actions, like we have done when explaining RL, is useful to convey the conceptual ideas. Yet, when we want to build artificially intelligent systems that can learn from experience, we have to make these things explicit, and talk about the representation of the world (see Markman, 1999; Sowa, 1999; Brachman and Levesque, 2004, for overviews). Humans are limited by the things they can perceive using their ears, eyes, touch sensors (e.g. hands, skin), nose and mouth, which, in various forms, represent the world to them. Some things of the world are beyond our perception, such as high frequency sounds, and radio waves. Inside our heads we can form additional representations of complex concepts such as chairs, government buildings, trust and time. These representations may be built directly on top of our sensors, and additionally in terms of each other. Much is known and unknown about cognitive representations in humans (e.g. see Margolis, 1999; Claplin, 2002, for some pointers).

A general definition of what representation is and that applies to both human and artificial systems contains at least three elements (Markman, 1999). The represented world
is the domain that the representations are about. The represented world may be the world outside the (cognitive) system or some other set of representations inside the system. That is, one set of representations can be about another set of representations. The representing world is the domain that contains the representations. The set of representing rules relates the representing world to the represented world through a set of rules that map elements of the represented world to elements in the representing world. Rules induce isomorphisms when every element in the represented world is represented by a unique element in the representing world, otherwise it is called a homomorphism.

For humans and artificial systems, the lowest level of representation consists of what they perceive through their sensors. This marks a boundary between the intelligent system and the outside world, and puts a limit on what things the agent considers to be part of the real world:

**Perception is Reality**

Representations can be very complex, or very simple. In Figure 1.2 two Braitenberg vehicles (see Braitenberg, 1984, for many interesting vehicles) are depicted. The only level of representation that is present consists of two sensors that detect light. In the left vehicle the right sensor is connected to the right motor and this will make the vehicle back away from the light in the current situation. In the right vehicle the left sensor is connected to the right motor (and vice versa), which makes this vehicle to move towards the light source. Both vehicles do not introduce any more sophisticated level of representation, but still they perform a simple behavior consistently.

This shows how powerful a couple of such simple control structures and representations can be. In contrast, many other types of architectures for intelligent behavior are like the one in Figure 1.3. In this cognitive architecture (see Langley, 2006, for more examples) the control mechanism has a much more complex structure, both in terms of the representations that are used (e.g. the agent’s beliefs about the world, descriptions of goals it must achieve, and predefined plans to achieve sub-tasks), and in terms of the algorithmic structures that are needed to decide on an action based on all the constituents of the agent’s mind. In Chapter 7 we go into more detail on this kind of architectures, and more specifically we focus on how learning can be incorporated.

Before discussing which types of representations are used in AI systems, we may first raise a question on how much representation we need and how they come about. This has been the subject of many debates in the past decades. The problem of how representations relate to the real world is essentially the symbol grounding problem (Harnad, 1990), but we ignore it and only consider the situation where there is some representation of the world to

---

6Pfeifer and Scheier (1999, pp. 475–478) describe interesting experiments using a group of such robots that seem to “tidy up” a room with obstacles.
begin with. The other issue has been subject of debate during the eighties. Brooks (1991) provided the start of developments in behavior-based architectures (Arkin, 1998) such as the subsumption architecture, by proposing that intelligent behavior could be achieved by a large number of loosely coupled processes that mostly function in an asynchronous and parallel way. He argued that internal processing would have to be minimal and that sensory signals should be mapped relatively directly to motor signals, as in Braitenberg vehicles. In essence, this called for less representation and abstraction. Later, this grew into the field of embodied intelligence (Pfeifer and Scheier, 1999, or, new AI), which emphasizes the fact that behavior consists of a bodily activity in a physical world and that we must understand intelligence in terms of the interaction between the embodied system and the environment.

A central motto\(^7\) is: "The world is there, no need to remember it all". This contrasts with what is often referred to as the computer metaphor of seeing intelligence as information processing or the manipulation of abstract symbols. Other approaches in learning and evolving behaviors show the potential of such reactive approaches (e.g. see Nolfi and Floreano, 2000; Nolfi, 2002), but in this book we argue that representation is important to be able to scale up to larger problems and to insert and extract knowledge from the learner (see also Markman and Dietrich, 2000b).

### 1.2.1 Generalization, Abstraction and Representation Formation

The most important aspect of a learning process is generalization, which is the capability to use information learned from one situation in other situations that are in some way "similar". In daily life, we do it all the time. For example, when going to a conference in a country to which we have never been before, we often experience little to no problems when using public transportation at that location. We can do this because the process of using them is quite similar in many countries: you have to look at the departure schedules, find out which line you require, buy a ticket, get into the right vehicle and get out at your destination. It does not matter much that the trains have different colors, or that stations may be built and structured in various ways. However, for generalization to work, we must have some idea of whether a new situation is sufficiently similar to already experienced situations and that we transfer the right aspects of this experience.

\(^7\)We’re always learning from experience by seeing some examples and then applying them to situations that we’ve never seen before. A single frightening growl or bark may lead a baby to fear all dogs of similar size – or, even animals of

---

7This is similar to the fairy tale of Hop o’ My Thumb who dropped bread crumbs to find his way back. In this way, he would not have to remember everything about how to get back; he only needed to modify its environment and simply follow the trail of bread crumbs.
1.2 You Can Only Learn What You Can Represent

every kind. How do we make generalizations from fragmentary bits of evidence?
A dog of mine was once hit by a car, and it never went down the same street again
– but it never stopped chasing cars on other streets.”
(Minsky, 1985, Section 19.8)

A generic generalization process requires a representation space and a similarity measure that defines for each pair of representations how similar they are. A similarity measure induces a distance in the representation space. Now representations can be grouped according to the measure and generalization takes place among situations that are near in that space. This makes generalization completely dependent on the representation space.

**Similarity is Proximity in Representation Space**

More complex representations offer more opportunities to construct such similarity measures, but at the same time they introduce more choices that have to be considered.

A broader view upon generalization is that it introduces a form of dynamic representation, or representation formation. As already said, representations can be about other representations, and generalization can be seen as building higher levels of abstraction in a new representation space (see also Korf, 1980). For example, based on the representations of a mouse, a keyboard, a monitor and a system, one can build the higher-order concept of computer. Building more complex representations from simpler ones is generally called constructivism, and has its origins in the cognitive development theories in psychology (Piaget, 1950; Thornton, 2002). In AI, constructivist approaches are often based on neural networks (Elman et al., 1996; Quartz and Sejnowski, 1997; Westermann, 2000), but many other types have been described (e.g. see Drescher, 1991; Thornton, 2000).

AI has introduced many types of representations, including sub-symbolic ones as used in neural network approaches and purely symbolic representations such as propositional logic. Popular representation schemes include Bayesian networks, relational databases, rules, trees, graphs and many more. In the end, general AI systems should employ a whole range of different representations depending on their suitability in various sub-tasks in the intelligent architecture (see also Minsky, 1985). In this book we are mainly interested in a division in three fundamental classes of representation that have to do with how the intelligent system perceives the world. These are atomic, in which the environment’s state is perceived as a single symbol, propositional, in which the world is structured in terms of propositions, and first-order, in which the current situation is perceived as consisting of objects. In the following we illustrate these classes using three imaginary robots.

**1.2.2 CANTOR: Representing the World in Snapshots**

Our first robot, named CANTOR⁸, is very simple. Each possible state is represented as an atomic thing, e.g. a symbol. For ease of explanation, we assume that CANTOR stores its value function and policy in a small notebook. On each page, it stores a state with an action value table for that state, see Figure 1.4a. Here we also see that CANTOR has experienced several learning steps in which it has once decreased the value for action a and increased it twice for action b.

---

⁸This robot can only reason in terms of sets. It cannot generalize using the structure of the elements in these sets. Georg Cantor is well-known for his contributions to set theory (see Davis, 2000).
At each step in the world, CANTOR observes the current state and looks it up in its notebook. Based on the values in the figure it would choose action b in this state in case it would not explore. Depending on the total number of states, looking up the current state may be a time-consuming operation. Stored in a computer memory, this may not seem too problematic. However, let us assume the states are photos, taken by a camera mounted on the robot. In a physical world, the number of distinct photos of the robot’s surroundings is enormous. Every two photos that differ only slightly in a single pixel are completely different for the robot, and they get a different page in CANTOR’s notebook.

The same storage and retrieval problems occur when a model is available. For each of the states, CANTOR would have to keep a page such as in Figure 1.4b), in which the non-zero transition probabilities to all other states must be stored. Depending on the stochasticity in the environment, each page might end up storing all states.

**Generalization, Abstraction and Representation Formation.** The possibilities for generalization for CANTOR are very limited. Even though there could be many states that are almost identical, as can be the case with photos, CANTOR can only see whether two states are exactly identical or not. What CANTOR can do to generalize is to *group* of states once experience shows that they have similar action values. CANTOR can then replace all pages of the states in one group by just one page that contains all states in that group and one action value table, see Figure 1.4c). In this way, states *share* information about action values and each time CANTOR visits a state in that group, implicitly all action values of all the group's states are updated at once, thereby generalizing over that set of states. Note that this implies that from the moment of grouping, all states will have equal action values, and it depends on whether the grouping is 'right' how this will affect future experiences and with that, the possibility of learning an optimal behavior.

**1.2.3 Boole: Representing the World in Twenty Questions**

Although CANTOR can – in principle – learn optimal behaviors, it is not of much use for most applications. Therefore, let us introduce the more advanced robot **Boole**. For this robot, state information is *decomposed* into a small number of indicators. Each such indicator represents the presence of a relevant aspect of the state. For example, there could be an indicator for the presence of a wall in front of the robot, or it could indicate whether
1.2 You Can Only Learn What You Can Represent

The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

Now, each page of Boole's notebook contains for each state a distinct set of answers to the questions in the feature set, see Figure 1.5a. Boole's learning process is similar to that of Cantor. First it gets a list of answers, which it looks up in its notebook. Then, based on the action values for that state, the robot chooses an action and perceives the next state, i.e., set of answers, and a reward.

Storage and retrieval of states can be made easier by looking at the structure of states. For example, Boole can have a separate part in its notebook for all states in which the first question is answered yes. And then another division in these parts based on the answer to the second question and so on. In this way, looking up a state can be done more quickly than Cantor did\textsuperscript{11}. More importantly, Boole can emulate Cantor's representation by introducing one question for each state in Cantor's representation. Such a question only asks for "is it this particular state?". Thus, each state in Boole's representation would consist of a list of all no's except for one yes. In other words, Boole can do everything Cantor can, but not the other way around.

**Generalization, Abstraction and Representation Formation.** Boole's representation of states offers many opportunities for generalization and abstraction. For example, the specification of a transition model can make use of abstraction over effects of actions on separate features of the state. For example, Figure 1.5b) shows a part of such a model. Here, it specifies that in all states where the answer to the first question is yes and either

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

Boole’s representation of states offers many opportunities for generalization and abstraction. For example, the specification of a transition model can make use of abstraction over effects of actions on separate features of the state. For example, Figure 1.5b) shows a part of such a model. Here, it specifies that in all states where the answer to the first question is yes and either

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.

\begin{equation}
\begin{aligned}
q_1 & = \text{yes} \\
q_2 & = \text{no} \\
q_3 & = \text{yes} \\
q_4 & = \text{yes} \\
q_5 & = 5.7
\end{aligned}
\end{equation}

or not the robot is carrying a load. The general form of such a state representation can be seen as a list of answers to questions\textsuperscript{9}. The questions are fixed, and are part of the robot (e.g., its sensors). Each answer can be either boolean, i.e., true or false, or real-valued. The technical term for such representations is propositional\textsuperscript{10}, or feature-based, and it is the most common representation in many AI or ML systems.
the second answers no or the fifth answer is larger than 3.0, the action a can have two possible effects. In one – with probability 0.8 – we know that (and how) the answers to the first and fifth question are changed, with respect to the current state in which the action was performed, and in the other – with probability 0.2 – we can see the effects on the answers to questions three and five. Such abstractions presuppose the notion of a propositional language to describe the model in this way. Note that where CANTOR needed such a description for each state separately, BOOLE describes transitions for a whole set of states simultaneously.

A similar type of generalization can be used to store the value functions. Instead of storing such values for each state separately, as CANTOR did, BOOLE can group states that share the same answers to selected questions. Figure 1.5c) shows a state-action value function in which all states that share the same answer to question three, and additionally have an answer to the fifth question that is lower or equal to 10.0, are grouped. Depending on the number of different answers to the fifth question, such generalization may group many states. Another advantage of BOOLE’s representation is that it can generalize over states it has not seen yet. For example, the page in Figure 1.5c) generalizes over the state 〈yes, yes, no, no, 2.9〉, such that CANTOR may know which action is better in that state even though it may not have been there yet. Obviously, such state generalization is only possible if it is known from experience that states can be grouped.

One of the main advantages of BOOLE’s representation is that states can be interpreted as vectors in an n-dimensional space, where n is the total number of questions. This n-dimensional space offers many possibilities for generalization, and the similarity of states can be equated with the Euclidean distance in that space. For example, if the current state has not been visited before, but there is a nearby state (in this space) in BOOLE’s notebook for which the optimal action value is known, then it may carry over this knowledge to this new state. Similarly, the value of a state can be expressed as a linear weighting of the feature values, which is depicted in Figure 1.5d). In this way, learning a value function comes down to tuning only five weight parameters, independently of how large the complete set of states is. This use of generalization forms the basis for many types of neural network and other approximation architectures that are used in RL.

1.2.4 Frege: Representing the World in Terms of Objects and Relations

Propositional representation is useful for modeling various tasks. However, it lacks the expressive power to capture a number of important general forms of reasoning, in particular reasoning about individuals, the properties they possess and relations between individuals. Furthermore, such representations cannot quantify over individuals, i.e. say that some property holds for some individuals, or for all individuals. For this, one needs first-order logic. Many dialects exist, though the common factor is that they represent and reason over structures containing objects (i.e. individuals) and relations between them.

Representing the world in terms of objects is most natural from a number of different perspectives. For humans it seems the most intuitive way to structure the world, (see Dennett, 1987, on the intentional stance). Objects often tend to refer to coherent entities of physical material though we have no problems with mental objects such as pride, love or the square root of –4.33. One possible explanation is that evolution has endowed us with these capabilities because they can be very efficient. "The description of the world in terms of objects and simple interactions is an enormously compressed description." (Baum, 2004, p. 14
1.2 You Can Only Learn What You Can Represent

In AI systems, the power of relational representations has been recognized since the beginning. Mainly because of reasons concerning computational complexity, using these representations for challenging tasks such as RL and other types of environments where there is a significant amount of uncertainty, has begun fairly recently. But in the end, relational representations are vital for scaling up to more complex problems. "It is hard to imagine a truly intelligent agent that does not conceive of the world in terms of objects and their properties and relations to other objects" (Kaelbling et al., 2001). Fortunately, for many types of computer-based tasks, relational representations are wide-spread, for example in relational databases.

Let us take a look at our third robot, Frege. From all three robots, Frege uses the richest representation. An example is depicted in Figure 1.6a). A number of the keywords in the graphical representations are objects. These are table, mug, pot, floor and room. Solid lines between objects denote relations. For example, the objects table and floor are connected by the relation on, meaning that the table is on the floor. Dashed lines denote attributes, or unary relations. These types of relations denote certain properties of an object. For example, we can see that the color of mug is red. An interesting aspect of such a relational representation is that actions are now parameterized with objects. For example, the action pickup(mug) denotes picking up the object mug. This causes the number of actions to be dependent on the number and type of objects in the current situation. If there would be just one more mug on the table, presumably an additional action pickup(mug) would be applicable. This is another advantage of relational representations; in contrast to BOOLE’s fixed-length question list, Frege’s representation can vary in size with each state.

A relational representation provides much detailed information about the state. A natural question to ask is whether the same world could be described by BOOLE. In principle, the answer is positive, but it comes with some problems. For each possible relation (including unary ones) we have to create a question. For example, if the colors include red, blue, yellow and green, BOOLE would need – for each object in the system – four questions such as "is the color of the first mug red?", "is the color of the first mug blue?" and so on, and only one of these four questions would be answered with yes. This becomes

\[ \text{Figure 1.6: Data structures for Frege: a) A state and a set of applicable actions, b) Part of the transition model (T).} \]

---

12Gottlob Frege (Frege, 1879) is the founder of modern approaches in first-order logic (see Davis, 2000, for a historical overview).
more troublesome with more objects and relations between more objects. An additional problem is that the questions (and their order) completely depend on the exact number of objects. Frege’s representation, on the contrary, scales easily with the number of objects.

**Generalization and Abstraction and Representation Formation.** Because there is much information present in Frege’s state and action representation, there are many possibilities for generalization and abstraction. The most common way is to introduce a formal language that can express certain properties of the states and that lets Frege reason about them. For example, a logical formula such as $\forall M \text{ on}(M, \text{table}) \rightarrow \text{blue}(M)$ expresses the phrase *all things on the table are blue*. This is a false statement because there is at least one red mug on the table too. The $M$ symbol is a variable that can stand for any object that satisfies some property, in this case being on the table. The $\forall$ symbol is a quantifier that enables to express properties that must hold for all objects in the current state, and is independent of how many or which types of objects are present. This shows that Frege’s representation is a lot more general than that of Boole, that uses a fixed set of questions.

Figure 1.6b) employs a very simple language to specify the effects of the pickup action. It says that if the robot is in a room $R$, and there is some object $T$ (which presumably is the table here), and the robot tries to pickup the object $M$ that is on $T$, then two things can happen. With 0.9 probability, the robot’s intended action succeeds and it is carrying the object and it is not on $T$ anymore. Yet, with a small 0.1 probability, the action fails and the object $M$ is not on $T$ anymore, but it has fallen onto the floor and it is broken.\(^{13}\)

Formal languages such as used in the examples are very powerful in expressing generalized statements about the world, yet automatically generating such descriptions is computationally hard, especially when combined with probability and utility. Boole’s representation offers a simple similarity measure for generalization if one interprets the answers to the questions as a vector in an $n$-dimensional space. Because of the symbolic nature of Frege’s representation, this trick no longer exists. Nevertheless, relational representations offer many opportunities to generalize over objects through the use of variables combined with quantifiers, when learning value functions and policies. And in the same way actions can be parameterized, so can goals. For example, when learning how to deliver a specific mug to a specific room, the robot can generalize its policy (by using variables) to be able to deliver any mug to any room. Another example is that a policy that is learned for stacking 5 objects can be used for any number of objects, which is something that cannot be achieved by Boole.

1.2.5 The World Might be Larger than We See

So far, we have assumed that everything that might be important for a current, optimal action, can actually be perceived. Obviously, this will not be the case for most general environments, and certainly not for our own, human, real world. There are many things we cannot perceive with certainty, or maybe not at all. Think of a some card game you play with multiple people. You first have to know the rules of the games, and you might have some general strategies but your knowledge about the current state is restricted to

---

\(^{13}\)Interestingly, a full specification of the failing action would in general be impossible. This is because one would have to specify all the different ways in which it can be broken, which pieces are created in breaking the mug, and where they are. A full symbolic specification of all these possibilities combined with a proper probability distribution over all these possibilities is impossible, though Zettlemoyer et al. (2005) describe a useful trick to handle such outcomes as noise in the probability distribution.
what you can see, which are your own cards. Furthermore, there is also uncertainty about what other people will do with their cards, and which cards are still in the deck.

The partially observable MDP (POMDP) model extends the MDP. In a POMDP the agent does not perceive the actual state, but only gets an observation that probabilistically depends on this state. One general solution in this type of environments is to represent the agent’s current state by a probability distribution over all states that gets conditioned on the observations. Many types of abstraction and generalization can be employed in this setting too. POMDP solution techniques are more complex, and exact solutions can only be obtained for modest-size problems. Many recent techniques for POMDPs focus on approximate solutions. This book is about fully-observable MDPs because virtually all current first-order methods are restricted by the Markov assumption, and it is here that we draw a boundary. Still, when employing abstraction and generalization in MDPs, some of the same problems of non-Markovian environments arise. That is why, in Section 2.7, we briefly describe what happens when one goes beyond the Markov assumption and we discuss some intuitions behind POMDPs and predictive state representations.

1.3. About the Contents and Structure of this Book

Basically, we are interested in representations and algorithms that can be used to build a robot such as FREGE. But in order to do just that, we need to take a close look at techniques that were used to construct the previous two robots CANTOR and BOOLE. Presumably, a lot of experience and insights have been gained by trying to build these two robots. FREGE will operate in a much more complex, and much more structured world, but it will also share concepts with CANTOR and BOOLE such as learning evaluation functions, predicting what consequences actions will have and how to generalize experience.

Investigating how to build intelligent robots such as FREGE is our main goal, but at the same time, it is part of the current focus of development in at least three distinct sub-fields of AI. In this section we first describe the main motivations and theme of this book at the crossing of these three distinct research fields. After that we outline the book’s structure and give a short preview of the contents and structure of its chapters.

1.3.1 Main Theme of This Book

This book is about modeling and computing behavior, with a special emphasis on representational aspects. One of the reasons for pursuing this line of research was that a couple of years ago there were almost no techniques for utility-based learning in first-order domains. Yet, it is desirable because of technical reasons, e.g. scaling up to larger – and more complex – problems, but also because for possible connections to other contexts, such as natural language, humans, and general cognitive agent architectures. Conceptually, RL seems to be the most natural candidate as a learning technique for agents. Most state-of-the-art formalisms that are used for programming intelligent agents are based on first-order logic, but almost all of the work in RL was occupied with efficient means for generalization in propositional domains. For example, the use of neural networks was widespread, but such techniques cannot handle rich representations consisting of arbitrary sized domains of objects and various relations between these objects. Furthermore, the use of (symbolic) domain knowledge is usually not considered in RL, though it has been shown very useful in supervised learning approaches in first-order settings. Coming
up with general solutions requires one to look at a whole range of first-order representation, learning and generalization techniques. Therefore, the main purpose of this book is to find answers to the following research questions.

*How can MDPs be posed over first-order domains, consisting of objects and relations, how can these be solved using either model-based DP techniques, or simulation-based methods such as RL, what are the characteristics and challenges of this setting, and how is it related to existing, propositional techniques?*

In this book we emphasize the representational aspects of the first-order setting. We focus on an agent that is put in an environment that consists of objects and relations between objects, and its goal is to learn a policy for a sequential decision making problem that is guided by reward feedback. Obviously, new representations require new algorithms in general, but as we will see, most of the algorithmic knowledge and achievements in utility-based learning that already exist for propositional representations, can and will be reused. Our personal insight into the incapability of techniques such as neural networks to cope with RL in first-order domains, is paralleled by natural developments in three distinct – but related – research fields. These fields are RL, first-order ML, and (agent) planning languages, and we describe them briefly below. Doing so gives us an opportunity to place the material in this book in its proper historical context. It also shows how learning sequential decision making in first-order domains fits into the natural developments and maturation of these fields.

### 1.3.1.1 REINFORCEMENT LEARNING

The first, and main, field that is extended by the material in this book is generally referred to as RL. As do some general books in this area, such as those by Sutton and Barto (1998), Bertsekas and Tsitsiklis (1996) and Si et al. (2004), it includes sample-based, model-based and various approximate techniques, all centered around MDPs and its extensions. Sutton and Barto (1998, pp 16–23) describe at length three main threads that run up to the mid nineties, which are 1) trial-and-error learning started in the psychology of animal learning, 2) optimal control learning and solutions using value functions and DP, and 3) temporal-difference methods. These threads came together at the end of the eighties to form the modern field of RL. Treatments of the history of RL can be found in many texts, for example in Sutton and Barto (1998)’s book, in general AI books such as Russell and Norvig (2003)’s, and in other overviews, for example (Kaelbling et al., 1996; Keerthi and Ravindran, 1997; Barto and Dietterich, 2004).

Much about the model-based setting for MDPs was known before that starting from early work on efficient solution concepts and algorithms such as the Bellman equations and value iteration (Bellman, 1957) and policy iteration (Howard, 1960) and the connection with shortest path algorithms such as Dijkstra’s algorithm. Many of these approaches are described in the standard textbook by Puterman (1994). Early work that includes ideas of the model-free setting such as trial-and-error learning are Samuel (1959)’s CHECKERS playing program, Holland (1975, 1986)’s work on evolutionary adaptive algorithms and the bucket brigade algorithm, but some ideas can be traced back to early work by Minsky and even Turing in the early fifties (see Sutton and Barto, 1998, Section 1.6 for an excellent overview). The seminal paper by Sutton (1988) marks the beginning of the modern field
of RL with which we start in this book. In this paper, Sutton separated learning value functions from learning control, introducing the general concept of prediction.

In the early nineties, many developments can be found in employing value function approximation methods, for example many using neural networks (Lin, 1992) and some using decision trees (Chapman and Kaelbling, 1991). In this period algorithms and proofs for Q-learning (Watkins, 1989; Watkins and Dayan, 1992), SARSA (Rummery and Niranjan, 1994; Rummery, 1995) and other one-step algorithms (Singh et al., 1995) were developed. Much of the research was focused on applying RL algorithms and using various approximation architectures to learn generalized value functions. During the nineties there were many developments for the model-based setting too (thoroughly surveyed in Boutilier et al., 1999) in the form of decision-theoretic planning. Structured abstraction schemes for states, and algorithms that make use of this structure during computation, have been developed in the context of factored representations of MDPs, value functions and policies by, for example, Dearden (2000), Boutilier et al. (2000a) and Dean and Givan (1997). Other developments during the nineties focused on learning policies directly, in contrast with value-based methods. Early approaches include the REINFORCE algorithm (Williams, 1988, 1992), and later other approaches appeared (e.g. Sutton et al., 2000). A large class of other related methods that deviate from online value-based methods are evolutionary approaches (Moriarty et al., 1999), which use evolution to find policies. During the nineties, and especially from the second half of it to now, much progress has been reported on yet another type of models and algorithms, specifically designed for temporal abstraction and hierarchical decomposition of policies. Identifying sequences of actions as behaviors, the field of hierarchical RL has found new ways of modularization of the learning process (see Dietterich, 2000b; Barto and Mahadevan, 2003; Ryan, 2004a, for overviews). Early work by Kaelbling (1993a), (Dayan and Hinton, 1993) and Thrun and Schwartz (1995) was quickly followed by widely used approaches such as HAMQ (Parr and Russell, 1998), MAXQ (Dietterich, 1998) and OPTIONS (Sutton et al., 1999).

An alternative view on historical developments in RL was sketched by Sutton (1999), and it consists of three periods. The past encompasses the period roughly until 1985 in which the idea of trial-and-error learning was developed, and includes all approaches back to the early fifties. The present (in 1999) was about value functions that were formalized, and the construction of value function approximation schemes, and proofs of convergence. The future or RL (again, in 1999) is about constructivism, i.e. to take a further step away to focus on the structures that enable value function estimation. In this view, we are far into this period, and indeed many approaches that have been developed in the recent decade are about this constructivism. Not only have there been many new types of representation and approximation schemes developed, but also algorithms that develop their own representations, which is particularly useful for the general goal of utility-based learning in unknown environments. For example, learning structured models of MDPs (e.g. see Littman et al., 2005; Degris et al., 2006; Jonsson, 2006) is a constructivist extension of the work on factored MDPs during the nineties. Early on, for hierarchical approaches, this type of constructivism was investigated in, for example, HQ (Wiering and Schmidhuber, 1997) and HEXQ (Hengst, 2002). In the past decade, many such algorithms have been developed as extensions to MAXQ, HAMQ and OPTIONS. The automatic construction of basis functions to describe the world is a topic of much research lately in DP algorithms (e.g. see Keller et al., 2006) but also in more generic contexts in which general characteristics
of the environment are extracted (Mahadevan and Maggioni, 2007),

Much about all these developments in MDP algorithms and the employment of abstraction and generalization can be found in Chapters 2 and 3 in this book. We describe the main types of algorithms and construct a general classification based on the type of generalization (e.g. of value functions) and whether representations are constructed by the learning system. Much recent RL research has focused on more general contexts such as partially observable MDPs where there is uncertainty about the true state, though most of this is beyond the scope of this book (but see Section 2.7).

So far, we have described some important historical lines that are mainly concerned with the algorithmic side of RL approaches. If we focus on the representational dimension, we obtain another picture of the field, one that is most relevant for this book. The early work in MDP research, both model-based and model-free approaches, used atomic state representations, which is the CANTOR setting. In this setting, states have no inherent structure and most results are about the algorithmic aspect: how to learn state values and optimal policies. The second period, mostly concentrated around the nineties, is about propositional, or feature-based, representations of states, the BOOLE setting. Many representational devices can handle these types of state representations for abstraction and generalization purposes, such as neural networks, propositional rules, decision trees, support-vector machines and many more. In this setting, states have propositional structure and generalization can make use of that. Most RL approaches use some kind of generalization that is tailored to these representations, and at the end of the nineties propositional representations were the state-of-the-art in RL.

Yet, around the turn of the century, approaches started to appear that extended the state, and action, representations to the first-order case in which the world is described in terms of objects and relations; the FREGE setting. The first was an application of first-order decision trees as a value function generalization technique for Q-learning in a small BLOCKS WORLD domain by Džeroski et al. (1998). They were inspired by, as is noted in that paper, the invited talks (at IJCAI-97) by Richard Sutton and Leslie Kaelbling who both suggested the combination of first-order learning algorithms and RL. Soon followed the first DP technique for first-order representations by Boutilier et al. (2001) who defined a value iteration algorithm that operates over a first-order logical MDP specification in the situation calculus. These two approaches initiated the third, representational period in MDP research, which is the main topic of this book: MDPs defined over first-order representations of the world and efficient algorithms for solving them. First-order approaches extend the applicability of MDPs to new, and larger domains, and in addition they create new possibilities such as parameterized actions, learning for multiple environments simultaneously and learning in indefinite or infinitely large environments.

The three representational dimensions in MDPs currently coexist. But more importantly, as we argue in this book, the historical developments in each of these dimensions are heavily co-dependent. Propositional approaches are usually built on top of results in the atomic approaches, in the sense that when generalization and abstraction make use of the propositional state structure they must obey certain (algorithmic) principles that are defined in the atomic case such as formulated in the Bellman equations for individual states. In a similar way, the first-order dimension borrows from the preceding developments, in particular from existing propositional mechanisms for generalization and abstraction over propositional MDPs. At the end of Chapter 3 we make these connections
explicit after which we describe the first-order dimension in full in Chapters 4 to 7.

1.3.1.2 Machine Learning

Knowing that RL is a subfield of machine learning (ML) we might expect somewhat similar developments. However, general ML has used first-order logical representations for several decades. In fact, many of the now popular approaches such as Bayesian networks were developed much later than these logical approaches. Some ML approaches can be traced back to the early days of AI (see Mitchell, 1997, for an excellent description).

First-order learning was much developed during the early seventies. One of the main developments was the formalization of learning in first-order clausal logic by Plotkin (1970). Other earlier approaches in the context of PROLOG, clausal logic and logical rules can be found in the works by e.g. Shapiro, Michalski and Winston. The study of (supervised) first-order ML, mainly in clausal logic, became a field of its own around 1990 (see Muggleton and De Raedt, 1994; Lavrac and Džeroski, 1994; Bergadano and Gunetti, 1995, for overviews). It became known as inductive logic programming (ILP) and was aimed at learning essentially PROLOG programs from data in a supervised setting. Much emphasis in ILP is placed upon a priori knowledge that can be used in the induction process. More recently, even higher-order logical learning approaches have been developed along the same lines (Lloyd, 2003).

Much of what is now generally understood as ML is centered around propositional representations and popular techniques such as decision trees and neural networks. Neural networks are computational structures that mimic some of the functioning of the human brain. They have been around since 1943, but after some initial setbacks it took until 1986 when they were revived by a general technique called backpropagation and the work by Rumelhart and McLelland. Since then they have become very popular (Bishop, 1995; Haykin, 1999; Reed and Marks II, 1999). Other popular techniques include decision trees, fuzzy logic and genetic algorithms, also commonly referred to as computational intelligence techniques (Jang et al., 1997). Since the early nineties many probabilistic techniques have made their way into ML. Techniques such as Bayesian networks and hidden Markov models have become very popular for learning probability distributions from data. More recently, support vector machines and kernel-based approaches (Schölkopf and Smola, 2002) have become popular, general ML approaches. What all these approaches have in common is that they use propositional (feature-based) representations and that they can handle uncertainty, in both supervised and unsupervised settings. Another aspect is that most of these statistical approaches focus on parameter estimation rather than on model selection. Some textbooks focus entirely on the propositional setting (see e.g. Hastie et al., 2001; Alpaydin, 2004) whereas others treat both the propositional and first-order setting (Langley, 1996; Mitchell, 1997; Poole et al., 1998).

Coming back to first-order ML, a drawback of many of the early ILP approaches was that they could not handle noise or uncertainty well. Starting in the mid-nineties, a new wave of probabilistic extensions to ILP approaches appeared, under the general name of statistical relational learning (SRL). The foundations of this approach are based on ILP methods and a long history of approaches in combining logic with probability (Halpern, 2003; Galavotti, 2005). In the same way first-order approaches in RL can incorporate techniques from propositional RL, many popular propositional ML algorithms have been upgraded to the first-order case. Examples in the supervised learning setting include
Bayesian networks, hidden Markov models, decision trees and rules (see De Raedt and Kersting, 2003, for an overview). Unsupervised techniques involving distances, kernels and clustering techniques have made their way to the first-order case (Ramon, 2002; Gärtner, 2003), and even neural networks (Bader et al., 2006) and genetic algorithms and classifier systems (Divina, 2006; Mellor, 2007). For other general overviews see (Džeroski and Lavrac, 2001b; De Raedt and Kersting, 2004; Getoor and Taskar, 2007). All of these approaches upgrade propositional ML approaches to the first-order case, but all of them are based on either a supervised or unsupervised learning setting.

Now we have two main historical lines along which one can place the material in this book. On the one hand, first-order extensions to RL extend traditional approaches that combine ILP methods and probabilistic aspects with yet another element, namely utility. On the other hand, they extend the field of SRL, that has focused so far on supervised and unsupervised settings, with an extra paradigm, namely that of RL. In both ways, first-order RL approaches extend the field of ML with a combination of first-order logic, probability, learning and utility.

1.3.1.3 ACTION LANGUAGES, PLANNING AND AGENTS

Classical planning is one of the oldest AI subjects. Starting with the early work by McCarthy (1963) on the situation calculus, AI has been occupied with modeling and axiomatizing changing world and commonsense reasoning. Many such logics exist and many are based on first-order logic (see Gelfond and Lifschitz, 1998; Reiter, 2001; Russell and Norvig, 2003; Brachman and Levesque, 2004; Mueller, 2006). However, it turns out to be very difficult to completely specify a world such that an automated system can compute a course of action that will reach some specified goal. One of the biggest problems is that the system can be overwhelmed by irrelevant actions. The key is to have a language that is expressive enough for interesting problems, but restrictive enough to allow efficient algorithms to operate over it. The basic representation for many classical planners is the STRIPS language (Fikes and Nilsson, 1971). It can specify some simple properties of states and how things change when actions are applied. One crucial assumption is that anything that is not specified does not change, which eliminates many irrelevant effects to be dealt with. An extension of the STRIPS formalism is the ADL language by Pednault (1989). ADL is more expressive than STRIPS because, for example, it supports conditional effects and quantified variables in goals. Many other variations and extensions have been introduced afterwards and later these were systematized within a standard syntax called planning domain definition language (PDDL) (Ghallab et al., 1998). Extensions to planning such as the incorporation of hierarchies (as in hierarchical RL) (e.g. see Erol et al., 1994) are also supported by PDDL.

In addition to restricted languages that are aimed at planning domains, a wide variety of action formalisms and logics exist for more general purposes. Examples include the fluent calculus (FC) (Thielscher, 1998), the event calculus (EC) (Mueller, 2006) and the situation calculus (SC) (Reiter, 2001). Many of these languages are constantly being extended to incorporate modal constructs, such as belief and knowledge, aspects of time and duration, domain knowledge, and even emotions. Much of the research is focused on theoretical properties of the logics and their extensions. Some have evolved into full programming languages such as GLOG (Levesque et al., 1997) based on the SC and FLUX (Thielscher, 2005) based on the FC. Finzi and Lukasiewicz (2004a) extend GLOG with
game-theoretic constructs to deal with multi-agent domains. In fact, the agent literature (Weiss, 1999; Ferber, 1999; Wooldridge, 2002) has produced many agent programming languages that have much in common with the approaches based on FC, EC and SC.

A complete description of all approaches fills many books. What is important to note here are two things. One is that there are many action languages that are based on first-order logic. The other is that most are targeted at deterministic domains. Some planning approaches are able to deal with probabilistic action effects (Kushmerick et al., 1995; Blythe, 1999) but work on this is still far more limited than on the deterministic setting. The extension towards MDPs, with reward-based goals, is known as decision-theoretic planning (Boutilier et al., 1999). What a plan is for classical planning, is a policy (or, universal plan) for probabilistic environments such as MDPs. Extending first-order formalisms to deal with the specification of MDPs is relatively straightforward based on existing languages. Some formalizations appeared earlier than the work we have described in the RL historical setting (see for example Poole, 1997a; Geffner and Bonet, 1998) but it was not before the SDP approach (Boutilier et al., 2001) that solution algorithms for such MDPs were developed.

Research in planning approaches has more focused on probabilistic contexts such as MDPs in the recent years. The PDDL language was extended to probabilistic PDDL (Younes et al., 2005) to standardize syntax, and the international planning competition (IPC) was extended in 2004 with a probabilistic track to encourage tackling these domains, to facilitate cross-fertilization between approaches, and to provide a standard benchmark. The algorithms and representations in this book contribute to this field, and in fact, some of the methods we describe, have entered the IPC in recent years (e.g. Hölldobler et al., 2006; Fern et al., 2006). An additional historical connection exists with ML approaches for planning (Zimmerman and Kambhampati, 2003). As many algorithms for first-order MDPs use various kinds of ML algorithms, and learning is essentially focused on heuristics (i.e. value functions), much cross-fertilization is possible here.

Summary of Dimensions. As described, the material in this book extends three distinct fields in AI, see also Figure 1.7. First, it extends RL at a representational level by moving to first-order world representations. Second, it extends ML in two ways: one by adding a utility component to probabilistic, logical ML approaches, and another one by extending probabilistic logic learning to the RL learning paradigm. Third, it extends first order (probabilistic) planning approaches with a utility component, thereby extending propositional probabilistic planning approaches towards first-order knowledge representation.

Many questions naturally arise from the descriptions of the historical developments. For example, when is it possible to design algorithms for rich representations by reduction to traditional techniques? This is one of the leading questions when we go over from Chapters 2 and 3 to the rest of the chapters. Another question is about how RL can benefit from (or contribute to) existing models and techniques used for (decision-theoretic) planning and agents that already use richer representations, but lack learning? This is a question that will be approached in Chapter 4. Yet another one is about whether the interaction between rich representations and the (known and validated) framework of (PO)MDPs can be characterized in a theoretically rigorous way? This question is leading in many of

---

14Some of these questions played a central role in the workshop on 'rich representations for reinforcement learning' that we organized at the international conference on machine learning (ICML) in 2005.
the chapters, and in the final chapter we come back to this when identifying some of the remaining challenges.

1.3.2 A Road Map

The structure of this book closely follows the historical lines of the field of RL. In the first part we try to identify important concepts and algorithms that exist in the propositional setting. The second and third parts of the book are about the first-order setting. In the following we briefly describe each of the chapters.

1.3.2.1 PART I: ELEMENTS OF LEARNING SEQUENTIAL DECISION MAKING UNDER UNCERTAINTY

The first part of the book deals with solving MDPs. This part will introduce the standard framework, highlight important methodological directions and review various methods. An important aspect of this part of the book is that it will describe various ways in which abstraction and generalization can be employed in the framework of MDPs and algorithms that compute solutions. Abstraction and generalization are studied in the propositional setting, and important concepts and techniques that are found will play an important role when we upgrade to the first-order knowledge representation setting in the second part of this book. This part can be seen as describing the adaptive behavior part of the title and providing answers to the first series of questions.

How can adaptive behavior be modeled and computed using the MDP framework, which types of efficient solution algorithms exist and how can propositional generalization and abstraction be employed in the framework?

This part consists of two chapters:

Chapter 2: Markov Decision Processes: Concepts and Algorithms
In this chapter, the classical MDP framework is described mathematically. This is the setting CANTOR operates in. States and actions have no structure and here we deal with the algorithmic part of learning sequential decision making. We distinguish two main types of algorithms for computing optimal policies. Model-based methods assume a known transition and reward model and employ DP techniques to obtain policies. Model-free methods interact with a domain simulator and use sampled traces to compute value functions and policies. For both types of algorithms we explain various extensions to make computations more efficient. This chapter lays down the mathematical foundation for much of the rest of the book, and some of the boundaries of the material are identified in the context of non-Markovian models and algorithms.

Chapter 3: Generalization and Abstraction in Markov Decision Processes

This chapter is about employing abstraction and generalization in propositional MDPs, which correspond to BOOLE’s twenty-questions setting. Among the first things discussed is a description of what abstraction and generalization is, and why it is needed. Based on a distinction between fixed and adaptive representations, we introduce the novel PIAGET principle, as an extension of the general mechanism of generalized policy iteration (GPI) introduced by Sutton and Barto (1998). The principle provides a useful way to classify various algorithms and to see how behavior learning and representation formation interact. A large part of this chapter describes five main types of abstraction, that differ in the type of structures generalization is performed over. These are, in order, state spaces, transition and reward models (so-called factored representations), value functions, policies and hierarchical decompositions. For each of these five types, we survey important concepts, algorithms and methods, guided by the PIAGET principle. Described as a case study in value function approximation in RL we then outline a novel RL application in fingerprint recognition in which various types of concepts are combined. This chapter also includes a table which holds pointers to sections and chapters in this book where first-order versions of the algorithms that were discussed, can be found. The chapter ends with a discussion of the interplay between behavior and representation.

1.3.2.2 Part II: Learning Sequential Decision Making under Uncertainty in First-Order Domains

The second part of the book deals with problems posed as a Markov decision process over a first-order domain, which is the environment we have described for the robot FREGE. It will describe the intuitions and arguments that together form the main motivation for moving to more powerful representation languages. The chapters in this part of the book form the main contributions to the field. We will distinguish between model-free and model-based, as was done in the previous two chapters. For both types of methods new algorithms will be described, and in addition, a full survey of existing methods is provided. This part can be seen as describing the logic element of the title of this book, and providing answers to the second series of questions.

What are first-order knowledge representation, abstraction, generalization and action formalisms, how can they be used for first-order versions of MDPs and how can propositional algorithms be upgraded to solve such MDPs?
This part of the book consists of three chapters.

Chapter 4: Reasoning, Learning and Acting in Worlds with Objects

This chapter defines the setting for our most advanced robot Frege. We begin this chapter by defining why representing the world in terms of objects and relations is desired and necessary, why it is natural and what the consequences are. Based on our findings in the previous two chapters, we need three main things. First we describe first-order knowledge representation and reasoning and based on this we introduce the notion of a relational Markov decision process (RMDP). A second aspect is first-order abstraction and generalization techniques. We describe inductive ML techniques for the first-order setting and survey important concepts and algorithms. A third main component is a definition of first-order domain and action theories. We describe a number of important issues in modeling dynamic domains, and some action logics that are helpful in later chapters. The second part of this chapter introduces a general first-order represented MDP (FORM) capturing the idea of specifying a relational MDP in any particular logical formalism. Additionally, we upgrade value functions, policies and the PIaget principle to the first-order setting and discuss both the representational and algorithmic aspects in this new setting.

Placed in between Chapters 2 and 3 on the one hand, and Chapters 5 to 7 on the other, Chapter 4 functions as a bridge between propositional and first-order MDP approaches.

Chapter 5: Model-Free Algorithms for Relational MDPs

This chapter upgrades the model-free setting introduced in Chapters 2 and 3 to the first-order setting defined in Chapter 4, utilizing much of the first-order generalization techniques in that chapter. The first part of the chapter is about value function approximation methods. We introduce CARCASS, a novel representational formalism for RMDP and a model-free (Q-learning) algorithm for learning RMDP policies. Furthermore, we describe an indirect RL algorithm based on prioritized sweeping to compute value functions and policies more efficiently, by learning a transition model of the abstract RMDP. This part also contains an extensive survey of all model-free, value function approximation methods for RMDPs. The second part is about policy search methods. We introduce the first evolutionary policy search for RMDPs, named GREY. In addition, we provide an extensive survey of all other policy search techniques for RMDPs.

Chapter 6: Model-based Algorithms for Relational MDPs

Complementing the previous chapter, this chapter is about the model-based setting in which the action formalisms defined in Chapter 4 play an important role. In this chapter we show how virtually all kinds of the model-based algorithms discussed in Chapters 2 and 3 can be unified in a novel technique called intensional dynamic programming. In five steps we move from classical value iteration to dynamic programming with general knowledge representation formalisms. As an intermediate step, we outline set-based dynamic programming that provides the semantics for intensional dynamic programming. In this way, an unified approach for many existing techniques for DP in the face of abstraction and generalization is provided. It is shown that it also applies to the first-order setting, and as
1.3 About the Contents and Structure of this Book

an example we introduce **ReBEL**, the first implemented first-order dynamic programming approach. **ReBEL** uses extensions of action formalisms and deductive techniques from Chapter 4 and can be applied even in problems with infinite domains. It is shown that DP in first-order domains introduces a number of new concepts and furthermore, **ReBEL** clearly shows that in the relational setting, convergence of algorithms such as value iteration is not guaranteed anymore. We also introduce and analyze several extensions to **ReBEL**, concerning logical and RL efficiency issues, and the use of background knowledge and bottom-up generalization for policy induction. At the end of the chapter we provide a thorough survey of all existing model-based techniques for RMDPs.

### 1.3.2.3 Part III: Implications, Challenges and Conclusions

The third part of this book goes beyond the approaches described in Chapters 5 and 6 and is concerned with models and behavioral decompositions as in hierarchical RL. Furthermore, this part outlines a number of distinct areas with open research questions, gaps in our current practical and theoretical understanding of applying decision-theoretic approaches in first-order domains. The main question in this part can be stated as

*What are the implications of representations and algorithms for first-order MDPs for modular behaviors, models and knowledge transfer, and what are the main challenges in first-order MDPs?*

The third part of the book deals with the implications of the new methods for RL in relational domains. There are prominent implications and possibilities for logical agents and logical agent programming languages. Furthermore, a number of new challenges arises. On the one hand, these challenges are about extending more of traditional RL methods to the relational domains. On the other hand, these are challenges because of the new possibilities due to the powerful combination of logic, probability, utility and ML. This part can be seen as providing answers to the third question, and it contains two chapters.

**Chapter 7: Sapience, Models and Hierarchy**

In Chapter 7 we describe how the ideas within relational RL might be taken as step further, by incorporating adaptivity in logical, cognitive agent architectures, so-called sapient agents. This involves hierarchical decompositions of behaviors and dealing with transition models. Furthermore, we discuss issues in guidance to help the learning agent, and transfer of learned knowledge to other, similar, problems. This chapter contains a survey of all such approaches that have been described in the literature.

**Chapter 8: Conclusions and Future Directions**

In this chapter we reflect on what has been accomplished in this book, and it contains the main conclusions. An important aspect of this chapter is that it outlines a number of research challenges that are interesting to pursue. These include technical advances such as lifting more propositional algorithms to the first-order case, new directions for first-order representations such as POMDPs, and conceptual challenges such as dealing rigourously with varying domain sizes and knowledge transfer.
1.3.3 Other Main Themes and Contributions

Two other main contributions this book makes, are:

A Reference Guide on Knowledge Representation in Reinforcement Learning

Representation is the central issue in this book. Chapter 3 contains an extensive exposition of abstraction and generalization dimensions in the context of MDPs and solution algorithms. As far as we know, this is the first time that all current major directions in RL have been brought together in one text. Throughout the book we highlight how abstraction and generalization techniques can be (and have been) upgraded to the relational representation case.

A Complete Survey of Relational Reinforcement Learning

One of the main contributions of this book is the first complete survey of the field of relational RL. Chapter 4 starts the exposition with an outline of relational representational formalisms and their use in modeling MDPs. The next three chapters (5–7) cover the complete spectrum of all methods that have been proposed in the literature. Chapter 8, finally, covers a number of challenges and future directions for the field.

The following sections together provide a complete survey of first-order approaches in RL.

Section 4.1.3.3 From Propositional to Relational. This section contains methods that cope with first-order domains by using propositional representations and algorithms. Among these are propositionalization approaches and deictic representations.

Section 5.3 A Survey of Model-Free, Value-Based Approaches. This part of the book describes methods for relational MDPs focusing on learning value functions. We distinguish between fixed and adaptive generalization.

Section 5.5 A Survey of Policy-Based Model-Free Relational RL. This part of the book describes methods for relational MDPs focusing on learning policies directly. We distinguish between policy search techniques based on evolutionary algorithms and techniques that use classification learning.

Section 6.5 A Survey of Model-Based Approaches. In this section we survey all methods that operate under the assumption that a full model of the RMDP is available. Among these are various exact algorithms, approximate versions of exact methods, and other approximate methods that, for example, upgrade solutions obtained in small, ground problem instances. This section also describes a number of approaches that deviate from the Markov assumption.

Section 7.3 A Survey of Hierarchies, Models, Guidance and Transfer. In this part of the book we survey a number of other approaches in first-order domains. These are hierarchical approaches, model-learning approaches and transfer techniques. Furthermore, some examples of decision-theoretic (agent) programming languages are discussed.

Section 8.2 Future Challenges. We highlight several directions in which the field of relational RL can be extended.