1
Logic-Based Approach to Agent Design

The goal of artificial intelligence is to learn how to build software components of intelligent agents capable of reasoning and acting in a changing environment. To exhibit intelligent behavior, an agent should have a mathematical model of its environment and its own capabilities and goals, as well as algorithms for achieving these goals. Our aim is to discover such models and algorithms and to learn how to use them to build practical intelligent systems. Why is this effort important? There are philosophical, scientific, and practical reasons. Scientists are getting closer to understanding ancient enigmas such as the origins and the physical and chemical structure of the universe, and the basic laws of development of living organisms, but still know comparatively little about the enigma of thinking. Now, however, we have the computer – a new tool that gives us the ability to test our theories of thought by designing intelligent software agents. In the short time that this tool has been applied to the study of reasoning, it has yielded a greater understanding of cognitive processes and continues to produce new insights on a regular basis, giving us much hope for the future. On the software engineering front, mathematical models of intelligent agents and the corresponding reasoning algorithms help develop the paradigm of declarative programming, which may lead to a simpler and more reliable programming methodology. And, of course, knowledge-intensive software systems, including decision support systems, intelligent search engines, and robots, are of great practical value. In addition, attempts to solve the problems of AI illuminate connections between different areas of computer science (CS) and between CS and other areas of science, including mathematical logic, philosophy, and linguistics.

1.1 Modeling an Intelligent Agent

In this book when we talk about an **agent**, we mean an entity that observes and acts on an environment and directs its activity toward achieving goals.

Note that this definition allows us to view even the simplest programs as agents. A program usually gets information from the outside world, performs an appointed reasoning task, and acts on the outside world, say, by printing the output, making the next chess move, starting a car, or giving advice. If the reasoning tasks that an agent performs are complex and lead to nontrivial behavior, we call it intelligent. If the agent readily adapts its behavior to changes in its environment, it is called adaptive. If it performs tasks independent of human control, we call it autonomous. An agent can possess some or all of these qualities in varying degrees. For example, consider a program in charge of controlling a large system for paper production. Among its many complex tasks, it can make decisions based on temperature readings or thickness measurements and adjust the speed of a conveyer belt and alert the operator. Clearly this program is supposed to observe, think, and act on the environment to achieve certain goals. It is intelligent, as well as adaptive in response to its sensors. And it is certainly autonomous to a certain degree, although its decisions can be overridden by an operator. Of course, the scope of the program and hence its ability for intelligent behavior are very limited.

As in many AI and computer science texts, we ignore engineering tasks related to the agent's physical interaction with the world and concentrate on modeling the agent and designing software components responsible for its decision making.

A mathematical model of an intelligent agent normally consists of the following elements:

- a *language(s)* for representing the agent's knowledge
- *reasoning algorithms* that use this knowledge to perform intelligent tasks, including planning, diagnostics, and learning; most such algorithms are based on sophisticated search and are capable of solving problems of non-polynomial complexity
- an *agent architecture*, which is the structure combining different submodels of an agent (normally related to different reasoning tasks) in one coherent whole.

In this book we consider the following typical agent architecture. An agent's memory contains knowledge both about the world and that entity's capabilities and goals. The agent follows these four steps:

1. observes the world, checks that its observations are consistent with its expectations, and updates its knowledge base
2. selects an appropriate goal G

3. searches for a plan (a sequence of actions) to achieve G
4. executes some initial part of the plan, updates the knowledge base, and goes back to step 1.

Sometimes we refer to these four steps as the **agent loop**. In step 1, notice that the agent does not assume that it is the sole manipulator of its environment. This allows for incorporation of external events and for failure analysis; for example, the agent can compare its expected world model to the one it observes and attempt to *explain* the possible discrepancies between the two. In step 2, goal selection can be implemented in many different ways, including prioritization, real-time user input, or random selection. Step 3, planning, is an art in itself, requiring reasoning skills, efficient search, and, possibly, weights on actions. Finally, in step 4, performing an action can involve hardware robots or, in other domains, "softbots," capable, say, of cleaning up directories or searching the Web. There are a myriad of issues involved in this simple outline.

This architecture is simple. A more sophisticated architecture may involve a complex structuring of various intelligent tasks and communication between modules performing these tasks in parallel, powerful learning and vision components, the ability to communicate and cooperate or compete with other agents, and the like. In this book, however, we limit ourselves to a sequential architecture presented in the agent loop and show that even this simple approach can lead to important insights and systems.

To implement this architecture we need to meet a number of difficult challenges. For example, how can we create a knowledge base for our agent? How can we make it function in a changing environment and maintain focus on its current goal without losing the ability to change goals in response to important events in its environment? How can we make it capable of explaining some of these events and using the explanations to fill gaps in its knowledge base? How can we get it to use its knowledge to do intelligent planning?

To answer these and other related questions, we chose to use the **logic-based approach**. The main idea is as follows. To make machines smart, we need to teach them how to reason and how to learn. Since most teaching comes from instruction, and most learning comes from the same, we need an efficient means of communication. As with any problem in computer science, the choice of language has a large impact on the elegance and efficiency of the solution. Languages differ according to the type of information their designers want to communicate to computers. There are two basic types: algorithmic and declarative. **Algorithmic languages** describe sequences of actions for a computer to perform. **Declarative languages** describe properties of objects in a given domain and relations between them. The **logic-based** approach proposes to

- use a declarative language to describe the domain,
- express various tasks (which may include requests to find plans or explanations of unexpected observations) as queries to the resulting program, and
- use an inference engine (i.e., a collection of reasoning algorithms) to answer these queries.

1.2 Simple Family Knowledge Base – An Example

To illustrate the logic-based approach used in this book, we teach our computer basic facts about families. For simplicity we consider a small family consisting of three people – John, Alice, and their son Sam. The domain is structured in terms of binary relations $father$, $mother$, and $gender\_of$. In these terms the family can be described by statements:

$$\begin{array}{l} father(john, sam). \\ mother(alice, sam). \\ gender\_of(john, male). \\ gender\_of(sam, male). \\ gender\_of(alice, female). \end{array}$$

read as “John is the father of Sam,” etc. (We are keeping things simple here and assuming that first names are enough to uniquely identify a person. Of course, if Sam had a father also named Sam, we would need to give each his own unique identifier.)

These basic relations can be used to teach a computer new notions. For instance, the following two rules can be viewed as a definition of relation $parent$. We say that X is a parent of Y if X is a father of Y or X is a mother of Y.

$$parent(X, Y) \leftarrow father(X, Y).$$
$$parent(X, Y) \leftarrow mother(X, Y).$$

Note that identifiers starting with capital letters denote variables, whereas those starting with lowercase letters denote names of objects (e.g., sam, john) and relations (e.g., father, parent).

The next rule defines the meaning of relation “X is a child of Y.”

$$child(X, Y) \leftarrow parent(Y, X).$$

This program is written in a variant of a declarative language called **Answer Set Prolog (ASP)**. Replacing symbol $\leftarrow$ by `:-` turns these statements into an executable program. We discuss the full syntax and semantics of the language in Chapter 2.

To make sure that the program allows the computer to "understand" the material, we test it by asking a number of questions. For example, "Is Sam a child of John?" or "Who are Sam's parents?" If the answers are satisfactory, the program has learned. (Notice that this is exactly the method we use to check human understanding.) In what follows we refer to an agent answering our questions as STUDENT. We can view STUDENT as a *theoretical* question-answering system. Doing this allows us to avoid the discussion of details of actual systems that we later use to automate STUDENT's reasoning.

Since STUDENT is not capable of understanding even basic English sentences, we express the questions in its own language. To ask if Sam is a child of John, we type

$$?\ child(sam, john)$$

STUDENT will use the available knowledge and its reasoning mechanism to answer this question in the affirmative.

To ask "Who are Sam's parents?" we use variables. Statement

$$?\ parent(X, sam)$$

is read as "Find X such that X is a parent of Sam." This time the reasoning mechanism of STUDENT will look for X satisfying this query and return the names of Sam's parents: John and Alice.

Note that the real syntax for queries varies slightly with the implementation used. Here we simply put a question mark to distinguish the query from a fact.

The system still does not know that Sam is Alice's son. This is not surprising because it does not know the meaning of the word "son." For practice you can define relation $son(X, Y)$ – X is a son of Y.

The family example exhibits typical features of logic-based programming. The knowledge about the domain is stated in precise mathematical language, and various search problems are expressed as queries, which are answered by the reasoning mechanism of STUDENT. The program is **elaboration tolerant**, which means that small changes in specifications do not cause global program changes. For instance, in the process of developing the family knowledge base, we expanded the original collection of facts to accommodate new relationships between family members. In each case, the new definitions were natural and did not require changes to the rest of the program. It is equally easy to modify existing definitions. Suppose, for instance, that we would like to incorporate facts from a Spanish-language database that has statements $padre(a, b)$ instead of $father(a, b)$. These statements can be simply added to our knowledge base. The Spanish-English translation will be given by the rule

$$father(X, Y) \leftarrow padre(X, Y)$$

which modifies our previous definition of $father$. Other statements from the Spanish database can be incorporated in the same manner. Note that all relations based on $father$ remain unchanged.

The intelligence of this system can be significantly improved and its vocabulary expanded. It can be supplied with a natural-language interface allowing the user to state questions in English. It can be taught to plan with the information it has been given, as well as to find explanations for possible discrepancies between its knowledge and its observations of the real world. In this book we discuss some of these enhancements. We also address models of intelligent agents whose knowledge contains probabilistic information about the agent's domain. Usually such agents are studied within the probabilistic approach to AI, which is frequently viewed as an alternative to the logic-based approach. We do not share this view. Instead *we view probabilistic reasoning as commonsense reasoning about degrees of belief of a rational agent*. We illustrate this view and its ramifications toward the end of the book.

1.3 A Historical Comment

The roots of the logic-based approach to agent design are very deep. Interested readers are encouraged to look for a serious discussion of these roots in the history of logic. The goal of this section is to give a brief introduction to the development of the basic ideas that formed the foundations of this approach. More information on the history of the subject and on various approaches to combining logic and artificial intelligence is found in Chapter 3.

1.3.1 The Axiomatic Method

The need to structure existing mathematical knowledge and to improve its reliability and coherence led mathematicians of ancient Greece to the development of the axiomatic method. The classical exposition of this method is given in Euclid's *Elements* in which all geometric knowledge available to

Euclid is logically derived from a small collection of *axioms* – geometric propositions accepted without proof. For more than two thousand years that book served as a model of rigorous argument. It was used to teach geometry from the time of its publication to the late 19th and early 20th centuries and long remained the second most read book in Europe after the Bible. Increased interest in the enigma of thinking, the 18th-century shift of emphasis from geometry to calculus with its notion of continuum, inclusion of infinity and infinite sets as one of the main subjects of mathematics, and the centuries-long quest for the development of reliable reasoning methods for these newly created mathematical notions led to substantial progress in the development of logic and the axiomatic method. At the beginning of the 20th century, logicians developed the general idea of *formal language* and used it to axiomatize set theory. Basic mathematical notions such as natural and real number, function, and geometric figure, were defined in terms of sets and their membership relations. As a result (almost) all of the mathematical knowledge of the early 20th century could be viewed as logical consequences of a collection of axioms that could fit on a medium-sized blackboard. This exceptional scientific achievement demonstrated the high degree of maturity of logic and is similar to the development of Mendeleev's periodic table in chemistry or the understanding of the structure of DNA in biology. Another important achievement of logic was the creation of the mathematical notion of a correct mathematical argument – the notion of proof. All these notions not only deepened our understanding of mathematical reasoning but also had a strong influence on mathematics.

1.3.2 Logic and Design of Intelligent Agents

For a long time the influence of the axiomatic method has been much weaker outside of mathematics. Most likely, Gottfried Wilhelm Leibniz was the first to suggest that the method can have a much broader applicability, adding to our understanding of other areas of science and even substantially changing human behavior. In his *The Art of Discovery* written in 1685, Leibniz wrote,

> The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right.

He was hoping that

> humanity would have a new kind of instrument increasing the power of reason far more than any optical instrument has ever aided the power of vision.

The idea, often referred to as the *Leibniz Dream*, greatly contributed to the development of computing science.

In the 20th century, Alfred Tarski and others, partly influenced by the Leibniz Dream, developed a research program whose goal was to investigate if the axiomatic method could be successfully applied outside of mathematics. In the 1950s John McCarthy came up with a program of applying this method to artificial intelligence, which gave birth to what is now called the **logic-based approach to AI**. The original idea was to supply computer programs with a substantial amount of knowledge about their problem domain represented in the language of mathematical logic and to use logical inference to decide what actions are appropriate to achieve their goals. This idea served as the foundation of declarative logical languages. For instance, the logic programming language Prolog, developed in the late 1970s by Robert Kowalski, Alain Colmerauer, Philippe Roussel, and others, allows a programmer to supply the program with knowledge about its domain represented by so-called definite clauses – a small subset of the language of classical mathematical logic. A computation problem is then reduced to proving that objects in the domain have a given property. This is achieved by an inference mechanism called SLD resolution. The language is Turing complete, which means that it can be viewed as a universal programming language. The simplicity of the class of definite clauses and the effectiveness of its inference mechanism allow for efficient implementations. Unfortunately, Prolog has some nondeclarative features. For instance, a simple modification of a program that preserves its logical equivalence may cause the program to go into an infinite loop. Some other substantial limitations prevent Prolog from being used as a full-scale knowledge representation language suitable for the design and implementation of intelligent agents, but we do not discuss them in this chapter.

Another interesting declarative language, Datalog, significantly expands the more traditional query-answering languages of relational databases. It can be viewed as a subset of Prolog, but it is limited to representing domains with a finite number of objects. The inference mechanisms of Datalog, however, are quite different from those of Prolog and are tailored toward query answering. In recent years there has been a substantial resurgence of Datalog, leading to more research and more practical applications of the language.

Summary

Inspired by the Leibnitz Dream of applying the axiomatic method to understanding methods of correct reasoning, researchers have been applying the logic-based approach to the design of intelligent agents. This approach consists of using a declarative language for representation, defining reasoning tasks as queries to a program, and computing the results using an inference engine. One such branch of research involves Answer Set Prolog, a simple yet powerful language useful for creating formal representations of various kinds of knowledge. When coupled with an inference engine, these representations can yield answers to various nontrivial queries; in fact, complex reasoning tasks such as planning and diagnostics, as well as other important parts of the agent architecture, can be reduced to querying knowledge bases. It is important to note that the separation of knowledge representation from the reasoning algorithm allows for a high degree of elaboration tolerance and clarity. We used a query-answering system, generically termed STUDENT, to illustrate the knowledge gained by the computer given a specific representation. Based on what the system knows is true, false, or unknown, we can evaluate the worth of such a representation, elicit unspoken assumptions, and learn about our own view of a domain.

References and Further Reading

The ideas of the logic-based approach to artificial intelligence and declarative programming were advocated by Cordell Green (1969), Robert Kowalski (1979), and John McCarthy (1990), among others. The importance of elaboration tolerance for knowledge representation was stressed by John McCarthy (1998). The simple agent architecture used in this book was introduced in Baral and Gelfond (2000) and Balduccini and Gelfond (2008). It is similar to earlier architectures (Kowalski 1995) and can be viewed as a special case of a more general belief-desire-intentions (BDI) architecture from Rao (1991). A popular account of this architecture that includes historical references can be found in Wooldridge (2000).

The first account of the axiomatic method is given in Euclid's *Elements* (Fitzpatrick 2007). A new, simplified approach to the axiomatization of geometry was suggested by David Hilbert (1899/1980). The discovery of paradoxes of set theory in the beginning of the 20th century led to further advances in axiomatic methods and attempts to axiomatize all of mathematics. (See, for instance, the famous formalization of a large part of mathematics by Alfred Whitehead and Bertrand Russell (1910, 1912, 1913). Later work (Godel 1932/1991) demonstrated that the full axiomatization of set theory, if at all possible, would require substantial new advances in our understanding of the axiomatic method. The possibility of applying the axiomatic method to realms outside of mathematics was suggested by Gottfried Wilhelm

Leibniz (1951) and later expanded by Alfred Tarski (1941/1995) and others.

Valuable information about the early history of Prolog and logic programming can be found in Kowalski (1988), Colmerauer and Roussel (1996), and Cohen (1988). For more information on Datalog see, for instance, Abiteboul, Hull, and Vianu (1995); Ullman (1988); and Zaniolo (1999).

Exercises

1. Compare and contrast the words *intelligent* and *rational*.
2. How can intelligent systems improve our ability to reason about a specific question?
3. Do some independent reading on Leibniz and explain why some people might consider him to be the first computer scientist.
4. In an address titled *Under the Spell of Leibniz's Dream*, Edsger Dijkstra (2001) said,

   > I think it absolutely astounding that he [Leibniz] foresaw how "the symbols would direct the reasoning," for how strongly they would do so was one of the most delightful discoveries of my professional life.

   Describe one way in which, in computer science, symbols direct reasoning.
5. Read John McCarthy's 1959 paper titled "Programs with Common Sense."
   (a) How does he define a program with common sense?
   (b) Compare and contrast STUDENT and the advice taker.