CHAPTER 9

Rules and Expert Systems

Any problem that can be solved by your in-house expert in a 10–30 minute telephone call can be developed as an expert system.

—M. Firebaugh, *Artificial Intelligence: A Knowledge-Based Approach*

'Rule Forty-two. All persons more than a mile high to leave the court.'
'That's not a regular rule: you invented it just now.'
'It's the oldest rule in the book,' said the King.
'Then it ought to be number one,' said Alice.

—Lewis Carroll, *Alice's Adventures in Wonderland*

These so-called expert systems were often right, in the specific areas for which they had been built, but they were extremely brittle. Given even a simple problem just slightly beyond their expertise, they would usually get a wrong answer. Ask a medical program about a rusty old car, and it might blithely diagnose measles.

—Douglas B. Lenat, *Programming Artificial Intelligence*

9.1 Introduction

In this chapter, we introduce the ideas behind production systems, or expert systems, and explain how they can be built using rule-based systems, frames, or a combination of the two.

This chapter explains techniques such as forward and backward chaining, conflict resolution, and the Rete algorithm. It also explains the architecture of an expert system and describes the roles of the individuals who are involved in designing, building, and using expert systems.

9.2 Rules for Knowledge Representation

One way to represent knowledge is by using rules that express what must happen or what does happen when certain conditions are met. Rules are usually expressed in the form of IF . . . THEN . . . statements, such as:

IF A THEN B

This can be considered to have a similar logical meaning as the following:

$A \rightarrow B$

As we saw in Chapter 7, A is called the antecedent and B is the consequent in this statement. In expressing rules, the consequent usually takes the form of an **action** or a **conclusion**. In other words, the purpose of a rule is usually to tell a system (such as an **expert system**) what to do in certain circumstances, or what conclusions to draw from a set of inputs about the current situation.

In general, a rule can have more than one antecedent, usually combined either by AND or by OR (logically the same as the operators $\wedge$ and $\vee$ we saw in Chapter 7). Similarly, a rule may have more than one consequent, which usually suggests that there are multiple actions to be taken.

In general, the antecedent of a rule compares an **object** with a possible **value**, using an **operator**. For example, suitable antecedents in a rule might be

IF $x > 3$

IF name is "Bob"

IF weather is cold

Here, the objects being considered are x, name, and weather; the operators are ">" and "is", and the values are 3, "Bob," and cold. Note that an object is not necessarily an object in the real-world sense—the weather is not a real-world object, but rather a state or condition of the world. An object in this sense is simply a variable that represents some physical object or state in the real world.

An example of a rule might be

IF name is "Bob"

AND weather is cold

THEN tell Bob 'Wear a coat'

This is an example of a **recommendation** rule, which takes a set of inputs and gives advice as a result. The conclusion of the rule is actually an action, and the action takes the form of a recommendation to Bob that he should wear a coat. In some cases, the rules provide more definite actions such as "move left" or "close door," in which case the rules are being used to represent **directives**.

Rules can also be used to represent relations such as:

IF temperature is below 0

THEN weather is cold

9.3 Rule-Based Systems

Rule-based systems or **production systems** are computer systems that use rules to provide recommendations or diagnoses, or to determine a course of action in a particular situation or to solve a particular problem.

A rule-based system consists of a number of components:

- a database of rules (also called a **knowledge base**)
- a database of facts
- an **interpreter**, or **inference engine**

In a rule-based system, the knowledge base consists of a set of rules that represent the knowledge that the system has. The database of facts represents inputs to the system that are used to derive conclusions, or to cause actions.

The interpreter, or inference engine, is the part of the system that controls the process of deriving conclusions. It uses the rules and facts, and combines them together to draw conclusions.

As we will see, these conclusions are often derived using deduction, although there are other possible approaches. Using deduction to reach a conclusion from a set of antecedents is called **forward chaining**. An alternative method, **backward chaining**, starts from a conclusion and tries to show it by following a logical path backward from the conclusion to a set of antecedents that are in the database of facts.

9.3.1 Forward Chaining

Forward chaining employs the same deduction method that we saw in Chapter 7. In other words, the system starts from a set of facts, and a set of rules, and tries to find a way of using those rules and facts to deduce a conclusion or come up with a suitable course of action.

This is known as **data-driven reasoning** because the reasoning starts from a set of data and ends up at the goal, which is the conclusion.

When applying forward chaining, the first step is to take the facts in the fact database and see if any combination of these matches all the antecedents of one of the rules in the rule database. When all the antecedents of a rule are matched by facts in the database, then this rule is **triggered**. Usually, when a rule is triggered, it is then **fired**, which means its conclusion is added to the facts database. If the conclusion of the rule that has fired is an action or a recommendation, then the system may cause that action to take place or the recommendation to be made.

For example, consider the following set of rules that is used to control an elevator in a three-story building:

Rule 1

IF on first floor and button is pressed on first floor

THEN open door

Rule 2

IF on first floor

AND button is pressed on second floor

THEN go to second floor

Rule 3

IF on first floor

AND button is pressed on third floor

THEN go to third floor

Rule 4

IF on second floor

AND button is pressed on first floor

AND already going to third floor

THEN remember to go to first floor later

This represents just a subset of the rules that would be needed, but we can use it to illustrate how forward chaining works.

Let us imagine that we start with the following facts in our database:

Fact 1

At first floor

Fact 2

Button pressed on third floor

Fact 3

Today is Tuesday

Now the system examines the rules and finds that Facts 1 and 2 match the antecedents of Rule 3. Hence, Rule 3 fires, and its conclusion

Go to third floor

is added to the database of facts. Presumably, this results in the elevator heading toward the third floor. Note that Fact 3 was ignored altogether because it did not match the antecedents of any of the rules.

Now let us imagine that the elevator is on its way to the third floor and has reached the second floor, when the button is pressed on the first floor. The fact

Button pressed on first floor

Is now added to the database, which results in Rule 4 firing. Now let us imagine that later in the day the facts database contains the following information:

Fact 1

At first floor

Fact 2

Button pressed on second floor

Fact 3

Button pressed on third floor

In this case, two rules are triggered—Rules 2 and 3. In such cases where there is more than one possible conclusion, **conflict resolution** needs to be applied to decide which rule to fire.

9.3.2 Conflict Resolution

In a situation where more than one conclusion can be deduced from a set of facts, there are a number of possible ways to decide which rule to fire (i.e., which conclusion to use or which course of action to take).

For example, consider the following set of rules:

IF it is cold

THEN wear a coat

IF it is cold

THEN stay at home

IF it is cold

THEN turn on the heat

If there is a single fact in the fact database, which is "it is cold," then clearly there are three conclusions that can be derived. In some cases, it might be fine to follow all three conclusions, but in many cases the conclusions are incompatible (for example, when prescribing medicines to patients).

In one conflict resolution method, rules are given priority levels, and when a conflict occurs, the rule that has the highest priority is fired, as in the following example:

IF patient has pain

THEN prescribe painkillers priority 10

IF patient has chest pain

THEN treat for heart disease priority 100

Here, it is clear that treating possible heart problems is more important than just curing the pain.

An alternative method is the **longest-matching strategy**. This method involves firing the conclusion that was derived from the longest rule. For example:

IF patient has pain

THEN prescribe painkiller

IF patient has chest pain

AND patient is over 60

AND patient has history of heart conditions

THEN take to emergency room

Here, if all the antecedents of the second rule match, then this rule's conclusion should be fired rather than the conclusion of the first rule because it is a more specific match.

A further method for conflict resolution is to fire the rule that has matched the facts most recently added to the database.

In each case, it may be that the system fires one rule and then stops (as in medical diagnosis), but in many cases, the system simply needs to choose a suitable ordering for the rules (as when controlling an elevator) because each rule that matches the facts needs to be fired at some point.

9.3.3 Meta Rules

In designing an expert system, it is necessary to select the conflict resolution method that will be used, and quite possibly it will be necessary to use different methods to resolve different types of conflicts. For example, in some situations it may make most sense to use the method that involves firing the most recently added rules. This method makes most sense in situations in which the timeliness of data is important. It might be, for example, that as research in a particular field of medicine develops, new rules are added to the system that contradict some of the older rules. It might make most sense for the system to assume that these newer rules are more accurate than the older rules.

It might also be the case, however, that the new rules have been added by an expert whose opinion is less trusted than that of the expert who added the earlier rules. In this case, it clearly makes more sense to allow the earlier rules priority.

This kind of knowledge is called **meta knowledge**—knowledge about knowledge. The rules that define how conflict resolution will be used, and how other aspects of the system itself will run, are called **meta rules**.

The knowledge engineer who builds the expert system is responsible for building appropriate meta knowledge into the system (such as "expert A is

to be trusted more than expert B" or "any rule that involves drug X is not to be trusted as much as rules that do not involve X").

Meta rules are treated by the expert system as if they were ordinary rules but are given greater priority than the normal rules that make up the expert system. In this way, the meta rules are able to override the normal rules, if necessary, and are certainly able to control the conflict resolution process.

9.3.4 Backward Chaining

Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived, which is useful when a set of facts are present, but you do not know what conclusions you are trying to prove. In some cases, forward chaining can be inefficient because it may end up proving a number of conclusions that are not currently interesting. In such cases, where a single specific conclusion is to be proved, **backward chaining** is more appropriate.

In backward chaining, we start from a conclusion, which is the **hypothesis** we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the database.

The conclusion we are aiming to prove is called a **goal**, and so reasoning in this way is known as **goal-driven reasoning**.

As we see in Chapter 16, backward chaining is often used in formulating plans. A plan is a sequence of actions that a program (such as an intelligent agent) decides to take to solve a particular problem. Backward chaining can make the process of formulating a plan more efficient than forward chaining.

Backward chaining in this way starts with the goal state, which is the set of conditions the agent wishes to achieve in carrying out its plan. It now examines this state and sees what actions could lead to it. For example, if the goal state involves a block being on a table, then one possible action would be to place that block on the table. This action might not be possible from the start state, and so further actions need to be added before this action in order to reach it from the start state. In this way, a plan can be formulated starting from the goal and working back toward the start state.

The benefit in this method is particularly clear in situations where the first state allows a very large number of possible actions. In this kind of situation, it can be very inefficient to attempt to formulate a plan using forward chaining because it involves examining every possible action, without paying any attention to which action might be the best one to lead to the goal state. Backward chaining ensures that each action that is taken is one that will definitely lead to the goal, and in many cases this will make the planning process far more efficient.

9.3.5 Comparing Forward and Backward Chaining

Let us use an example to compare forward and backward chaining. In this case, we will revert to our use of symbols for logical statements, in order to clarify the explanation, but we could equally well be using rules about elevators or the weather.

Rules:

```
Rule 1      A ∧ B → C
Rule 2      A → D
Rule 3      C ∧ D → E
Rule 4      B ∧ E ∧ F → G
Rule 5      A ∧ E → H
Rule 6      D ∧ E ∧ H → I
```

Facts:

```
Fact 1  A
Fact 2  B
Fact 3  F
```

Goal:

Our goal is to prove H.

First let us use forward chaining. As our conflict resolution strategy, we will fire rules in the order they appear in the database, starting from Rule 1.

In the initial state, Rules 1 and 2 are both triggered. We will start by firing Rule 1, which means we add C to our fact database. Next, Rule 2 is fired, meaning we add D to our fact database.

We now have the facts A, B, C, D, F, but we have not yet reached our goal, which is G.

Now Rule 3 is triggered and fired, meaning that fact E is added to the database. As a result, Rules 4 and 5 are triggered. Rule 4 is fired first, resulting in Fact G being added to the database, and then Rule 5 is fired, and Fact H is added to the database. We have now proved our goal and do not need to go on any further.

This deduction is presented in the following table:

Facts	Rules triggered	Rule fired
A, B, F	1, 2	1
A, B, C, F	2	2
A, B, C, D, F	3	3
A, B, C, D, E, F	4, 5	4
A, B, C, D, E, F, G	5	5
A, B, C, D, E, F, G, H	6	STOP

Now we will consider the same problem using backward chaining. To do so, we will use a goals database in addition to the rule and fact databases. In this case, the goals database starts with just the conclusion, H, which we want to prove. We will now see which rules would need to fire to lead to this conclusion. Rule 5 is the only one that has H as a conclusion, so to prove H, we must prove the antecedents of Rule 5, which are A and E.

Fact A is already in the database, so we only need to prove the other antecedent, E. Therefore, E is added to the goal database. Once we have proved E, we now know that this is sufficient to prove H, so we can remove H from the goals database.

So now we attempt to prove Fact E. Rule 3 has E as its conclusion, so to prove E, we must prove the antecedents of Rule 3, which are C and D. Neither of these facts is in the fact database, so we need to prove both of them. They are both therefore added to the goals database. D is the conclusion of Rule 2 and Rule 2's antecedent, A, is already in the fact database, so we can conclude D and add it to the fact database.

Similarly, C is the conclusion of Rule 1, and Rule 1's antecedents, A and B, are both in the fact database. So, we have now proved all the goals in the goal database and have therefore proved H and can stop.

This process is represented in the table below:

Facts	Goals	Matching rules
A, B, F	H	5
A, B, F	E	3
A, B, F	C, D	1
A, B, C, F	D	2
A, B, C, D, F		STOP

In this case, backward chaining needed to use one fewer rule. If the rule database had had a large number of other rules that had A, B, and F as their antecedents, then forward chaining might well have been even more inefficient.

In many situations, forward chaining is more appropriate, particularly in a situation where a set of facts is available, but the conclusion is not already known.

In general, backward chaining is appropriate in cases where there are few possible conclusions (or even just one) and many possible facts, not very many of which are necessarily relevant to the conclusion. Forward chaining is more appropriate when there are many possible conclusions.

The way in which forward or backward chaining is usually chosen is to consider which way an expert would solve the problem. This is particularly appropriate because rule-based reasoning is often used in **expert systems**.

9.4 Rule-Based Expert Systems

An expert system is one designed to model the behavior of an expert in some field, such as medicine or geology. Rule-based expert systems are designed to be able to use the same rules that the expert would use to draw conclusions from a set of facts that are presented to the system.

9.4.1 The People Involved in an Expert System

The design, development, and use of expert systems involves a number of people. The **end-user** of the system is the person who has the need for the system. In the case of a medical diagnosis system, this may be a doctor, or it may be an individual who has a complaint that they wish to diagnose.

The **knowledge engineer** is the person who designs the rules for the system, based on either observing the expert at work or by asking the expert questions about how he or she works.

The **domain expert** is very important to the design of an expert system. In the case of a medical diagnosis system, the expert needs to be able to explain to the knowledge engineer how he or she goes about diagnosing illnesses.

9.4.2 Architecture of an Expert System

A typical expert system architecture is shown in Figure 9.1.

The knowledge base contains the specific domain knowledge that is used by an expert to derive conclusions from facts. In the case of a rule-based expert system, this domain knowledge is expressed in the form of a series of rules.

The explanation system provides information to the user about how the inference engine arrived at its conclusions. This can often be essential, particularly if the advice being given is of a critical nature, such as with a medical diagnosis system. If the system has used faulty reasoning to arrive at its

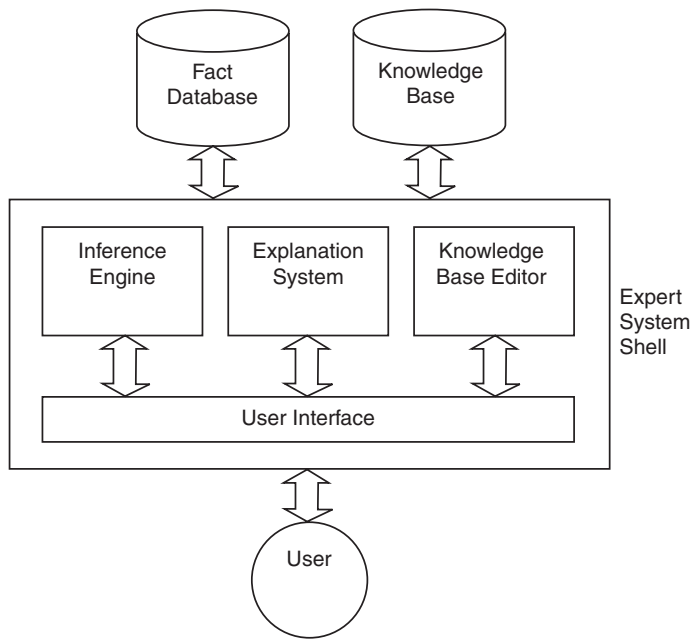

Figure 9.1
Architecture of an expert system

conclusions, then the user may be able to see this by examining the data given by the explanation system.

The fact database contains the case-specific data that are to be used in a particular case to derive a conclusion. In the case of a medical expert system, this would contain information that had been obtained about the patient's condition.

The user of the expert system interfaces with it through a user interface, which provides access to the inference engine, the explanation system, and the knowledge-base editor. The inference engine is the part of the system that uses the rules and facts to derive conclusions. The inference engine will use forward chaining, backward chaining, or a combination of the two to make inferences from the data that are available to it.

The knowledge-base editor allows the user to edit the information that is contained in the knowledge base. The knowledge-base editor is not usually made available to the end user of the system but is used by the knowledge engineer or the expert to provide and update the knowledge that is contained within the system.

9.4.3 The Expert System Shell

Note that in Figure 9.1, the parts of the expert system that do not contain domain-specific or case-specific information are contained within the **expert system shell**. This shell is a general toolkit that can be used to build a number of different expert systems, depending on which knowledge base is added to the shell.

An example of such a shell is CLIPS (C Language Integrated Production System), which is described in more detail in Section 9.4. Other examples in common use include OPS5, ART, JESS, and Eclipse.

9.4.4 The Rete Algorithm

One potential problem with expert systems is the number of comparisons that need to be made between rules and facts in the database. In some cases, where there are hundreds or even thousands of rules, running comparisons against each rule can be impractical.

The **Rete Algorithm** is an efficient method for solving this problem and is used by a number of expert system tools, including OPS5 and Eclipse.