Rule-based expert systems 2

In which we introduce the most popular choice for building knowledge-based systems: rule-based expert systems.

2.1 Introduction, or what is knowledge?

In the 1970s, it was finally accepted that to make a machine solve an intellectual problem one had to know the solution. In other words, one has to have knowledge, 'know-how', in some specific domain.

What is knowledge?
Knowledge is a theoretical or practical understanding of a subject or a domain. Knowledge is also the sum of what is currently known, and apparently knowledge is power. Those who possess knowledge are called experts. They are the most powerful and important people in their organisations. Any successful company has at least a few first-class experts and it cannot remain in business without them.

Who is generally acknowledged as an expert?
Anyone can be considered a domain expert if he or she has deep knowledge (of both facts and rules) and strong practical experience in a particular domain. The area of the domain may be limited. For example, experts in electrical machines may have only general knowledge about transformers, while experts in life insurance marketing might have limited understanding of a real estate insurance policy. In general, an expert is a skilful person who can do things other people cannot.

How do experts think?
The human mental process is internal, and it is too complex to be represented as an algorithm. However, most experts are capable of expressing their knowledge in the form of rules for problem solving. Consider a simple example. Imagine, you meet an alien! He wants to cross a road. Can you help him? You are an expert in crossing roads – you've been on this job for several years. Thus you are able to teach the alien. How would you do this?

You explain to the alien that he can cross the road safely when the traffic light is green, and he must stop when the traffic light is red. These are the basic rules. Your knowledge can be formulated as the following simple statements:

```
IF     the 'traffic light' is green
THEN   the action is go

IF     the 'traffic light' is red
THEN   the action is stop
```

These statements represented in the IF-THEN form are called **production rules** or just **rules**. The term 'rule' in AI, which is the most commonly used type of knowledge representation, can be defined as an IF-THEN structure that relates given information or facts in the IF part to some action in the THEN part. A rule provides some description of how to solve a problem. Rules are relatively easy to create and understand.

2.2 Rules as a knowledge representation technique

Any rule consists of two parts: the IF part, called the **antecedent** (**premise** or **condition**) and the THEN part called the **consequent** (**conclusion** or **action**).

The basic syntax of a rule is:

```
IF     <antecedent>
THEN   <consequent>
```

In general, a rule can have multiple antecedents joined by the keywords AND (conjunction), OR (disjunction) or a combination of both. However, it is a good habit to avoid mixing conjunctions and disjunctions in the same rule.

```
IF     <antecedent 1>
AND    <antecedent 2>
              .
              .
              .
AND    <antecedent n>
THEN   <consequent>

IF     <antecedent 1>
OR     <antecedent 2>
              .
              .
              .
OR     <antecedent n>
THEN   <consequent>
```

The consequent of a rule can also have multiple clauses:

```
IF      <antecedent>
THEN    <consequent 1>
        <consequent 2>
              .
              .
              .
        <consequent m>
```

The antecedent of a rule incorporates two parts: an **object** (**linguistic object**) and its **value**. In our road crossing example, the linguistic object 'traffic light' can take either the value *green* or the value *red*. The object and its value are linked by an **operator**. The operator identifies the object and assigns the value. Operators such as *is*, *are*, *is not*, *are not* are used to assign a symbolic value to a linguistic object. But expert systems can also use mathematical operators to define an object as numerical and assign it to the numerical value. For example,

```
IF      'age of the customer' < 18
AND     'cash withdrawal' > 1000
THEN    'signature of the parent' is required
```

Similar to a rule antecedent, a consequent combines an object and a value connected by an operator. The operator assigns the value to the linguistic object. In the road crossing example, if the value of *traffic light* is *green*, the first rule sets the linguistic *object* action to the value *go*. Numerical objects and even simple arithmetical expression can also be used in a rule consequent.

```
IF      'taxable income' > 16283
THEN    'Medicare levy' = 'taxable income' * 1.5 / 100
```

Rules can represent relations, recommendations, directives, strategies and heuristics (Durkin, 1994).

Relation

```
IF      the 'fuel tank' is empty
THEN    the car is dead
```

Recommendation

```
IF      the season is autumn
AND     the sky is cloudy
AND     the forecast is drizzle
THEN    the advice is 'take an umbrella'
```

Directive

```
IF      the car is dead
AND     the 'fuel tank' is empty
THEN    the action is 'refuel the car'
```

Strategy

```
IF      the car is dead
THEN    the action is 'check the fuel tank';
        step1 is complete

IF      step1 is complete
AND     the 'fuel tank' is full
THEN    the action is 'check the battery';
        step2 is complete
```

Heuristic

```
IF      the spill is liquid
AND     the 'spill pH' < 6
AND     the 'spill smell' is vinegar
THEN    the 'spill material' is 'acetic acid'
```

2.3 The main players in the expert system development team

As soon as knowledge is provided by a human expert, we can input it into a computer. We expect the computer to act as an intelligent assistant in some specific domain of expertise or to solve a problem that would otherwise have to be solved by an expert. We also would like the computer to be able to integrate new knowledge and to show its knowledge in a form that is easy to read and understand, and to deal with simple sentences in a natural language rather than an artificial programming language. Finally, we want our computer to explain how it reaches a particular conclusion. In other words, we have to build an **expert system**, a computer program capable of performing at the level of a human expert in a narrow problem area.

The most popular expert systems are rule-based systems. A great number have been built and successfully applied in such areas as business and engineering, medicine and geology, power systems and mining. A large number of companies produce and market software for rule-based expert system development – expert system shells for personal computers.

Expert system shells are becoming particularly popular for developing rule-based systems. Their main advantage is that the system builder can now concentrate on the knowledge itself rather than on learning a programming language.

What is an expert system shell?

An expert system shell can be considered as an expert system with the knowledge removed. Therefore, all the user has to do is to add the knowledge in the form of rules and provide relevant data to solve a problem.

Let us now look at who is needed to develop an expert system and what skills are needed.

In general, there are five members of the expert system development team: the domain expert, the knowledge engineer, the programmer, the project

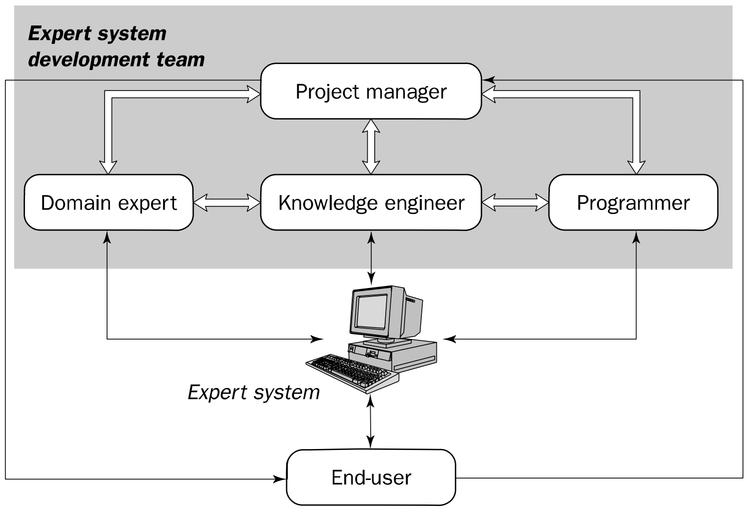

Figure 2.1 The main players of the expert system development team

manager and the end-user. The success of their expert system entirely depends on how well the members work together. The basic relations in the development team are summarised in Figure 2.1.

The **domain expert** is a knowledgeable and skilled person capable of solving problems in a specific area or **domain**. This person has the greatest expertise in a given domain. This expertise is to be captured in the expert system. Therefore, the expert must be able to communicate his or her knowledge, be willing to participate in the expert system development and commit a substantial amount of time to the project. The domain expert is the most important player in the expert system development team.

The **knowledge engineer** is someone who is capable of designing, building and testing an expert system. This person is responsible for selecting an appropriate task for the expert system. He or she interviews the domain expert to find out how a particular problem is solved. Through interaction with the expert, the knowledge engineer establishes what reasoning methods the expert uses to handle facts and rules and decides how to represent them in the expert system. The knowledge engineer then chooses some development software or an expert system shell, or looks at programming languages for encoding the knowledge (and sometimes encodes it himself). And finally, the knowledge engineer is responsible for testing, revising and integrating the expert system into the workplace. Thus, the knowledge engineer is committed to the project from the initial design stage to the final delivery of the expert system, and even after the project is completed, he or she may also be involved in maintaining the system.

The **programmer** is the person responsible for the actual programming, describing the domain knowledge in terms that a computer can understand. The programmer needs to have skills in symbolic programming in such AI languages as LISP, Prolog and OPS5 and also some experience in the application of different types of expert system shells. In addition, the programmer should know conventional programming languages like C, Pascal, FORTRAN and Basic. If an expert system shell is used, the knowledge engineer can easily encode the knowledge into the expert system and thus eliminate the need for the programmer. However, if a shell cannot be used, a programmer must develop the knowledge and data representation structures (knowledge base and database), control structure (inference engine) and dialogue structure (user interface). The programmer may also be involved in testing the expert system.

The **project manager** is the leader of the expert system development team, responsible for keeping the project on track. He or she makes sure that all deliverables and milestones are met, interacts with the expert, knowledge engineer, programmer and end-user.

The **end-user**, often called just the **user**, is a person who uses the expert system when it is developed. The user might be an analytical chemist determining the molecular structure of soil from Mars (Feigenbaum *et al.*, 1971), a junior doctor diagnosing an infectious blood disease (Shortliffe, 1976), an exploration geologist trying to discover a new mineral deposit (Duda *et al.*, 1979), or a power system operator needing advice in an emergency (Negnevitsky, 1996). Each of these users of expert systems has different needs, which the system must meet: the system's final acceptance will depend on the user's satisfaction. The user must not only be confident in the expert system performance but also feel comfortable using it. Therefore, the design of the user interface of the expert system is also vital for the project's success; the end-user's contribution here can be crucial.

The development of an expert system can be started when all five players have joined the team. However, many expert systems are now developed on personal computers using expert system shells. This can eliminate the need for the programmer and also might reduce the role of the knowledge engineer. For small expert systems, the project manager, knowledge engineer, programmer and even the expert could be the same person. But all team players are required when large expert systems are developed.

2.4 Structure of a rule-based expert system

In the early 1970s, Newell and Simon from Carnegie-Mellon University proposed a production system model, the foundation of the modern rule-based expert systems (Newell and Simon, 1972). The production model is based on the idea that humans solve problems by applying their knowledge (expressed as production rules) to a given problem represented by problem-specific information. The production rules are stored in the long-term memory and the problem-specific information or facts in the short-term memory. The production system model and the basic structure of a rule-based expert system are shown in Figure 2.2.

A rule-based expert system has five components: the knowledge base, the database, the inference engine, the explanation facilities, and the user interface.

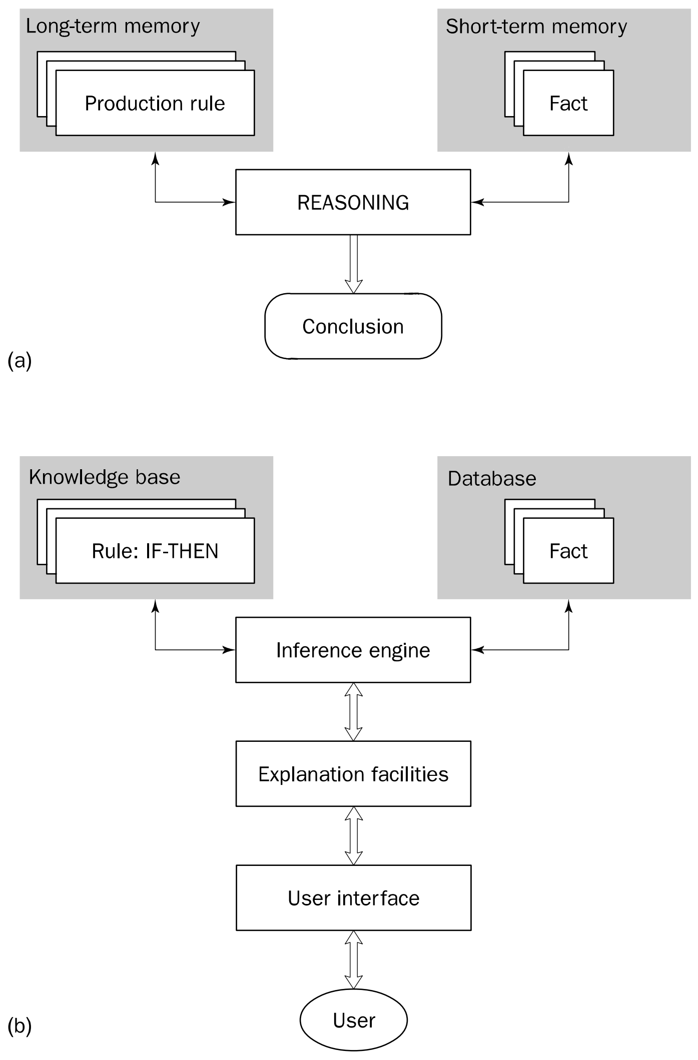

Figure 2.2 Production system and basic structure of a rule-based expert system: (a) production system model; (b) basic structure of a rule-based expert system

The **knowledge base** contains the domain knowledge useful for problem solving. In a rule-based expert system, the knowledge is represented as a set of rules. Each rule specifies a relation, recommendation, directive, strategy or heuristic and has the IF (condition) THEN (action) structure. When the condition part of a rule is satisfied, the rule is said to **fire** and the action part is executed.

The **database** includes a set of facts used to match against the IF (condition) parts of rules stored in the knowledge base.

The **inference engine** carries out the reasoning whereby the expert system reaches a solution. It links the rules given in the knowledge base with the facts provided in the database.

The **explanation facilities** enable the user to ask the expert system **how** a particular conclusion is reached and **why** a specific fact is needed. An expert system must be able to explain its reasoning and justify its advice, analysis or conclusion.

The **user interface** is the means of communication between a user seeking a solution to the problem and an expert system. The communication should be as meaningful and friendly as possible.

These five components are essential for any rule-based expert system. They constitute its core, but there may be a few additional components.

The **external interface** allows an expert system to work with external data files and programs written in conventional programming languages such as C, Pascal, FORTRAN and Basic. The complete structure of a rule-based expert system is shown in Figure 2.3.

The **developer interface** usually includes knowledge base editors, debugging aids and input/output facilities.

All expert system shells provide a simple **text editor** to input and modify rules, and to check their correct format and spelling. Many expert systems also

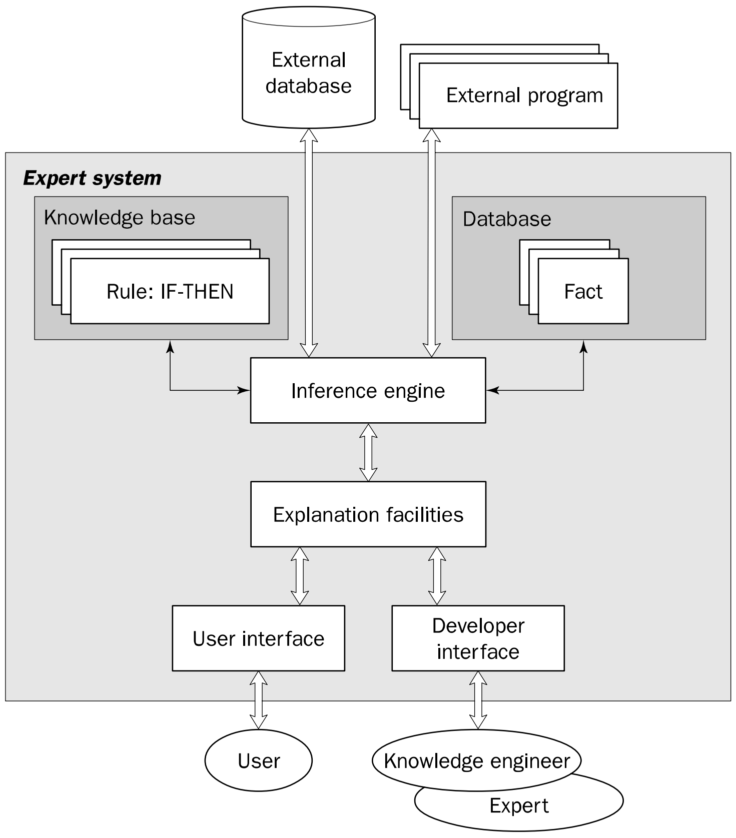

Figure 2.3 Complete structure of a rule-based expert system

include **book-keeping facilities** to monitor the changes made by the knowledge engineer or expert. If a rule is changed, the editor will automatically store the change date and the name of the person who made this change for later reference. This is very important when a number of knowledge engineers and experts have access to the knowledge base and can modify it.

Debugging aids usually consist of **tracing facilities** and **break packages**. Tracing provides a list of all rules fired during the program's execution, and a break package makes it possible to tell the expert system in advance where to stop so that the knowledge engineer or the expert can examine the current values in the database.

Most expert systems also accommodate input/output facilities such as **run-time knowledge acquisition**. This enables the running expert system to ask for needed information whenever this information is not available in the database. When the requested information is input by the knowledge engineer or the expert, the program resumes.

In general, the developer interface, and knowledge acquisition facilities in particular, are designed to enable a domain expert to input his or her knowledge directly in the expert system and thus to minimise the intervention of a knowledge engineer.

2.5 Fundamental characteristics of an expert system

An expert system is built to perform at a human expert level in a **narrow, specialised domain**. Thus, the most important characteristic of an expert system is its high-quality performance. No matter how fast the system can solve a problem, the user will not be satisfied if the result is wrong. On the other hand, the speed of reaching a solution is very important. Even the most accurate decision or diagnosis may not be useful if it is too late to apply, for instance, in an emergency, when a patient dies or a nuclear power plant explodes. Experts use their practical experience and understanding of the problem to find short cuts to a solution. Experts use rules of thumb or **heuristics**. Like their human counterparts, expert systems should apply heuristics to guide the reasoning and thus reduce the search area for a solution.

A unique feature of an expert system is its **explanation capability**. This enables the expert system to review its own reasoning and explain its decisions. An explanation in expert systems in effect traces the rules fired during a problem-solving session. This is, of course, a simplification; however a real or 'human' explanation is not yet possible because it requires basic understanding of the domain. Although a sequence of rules fired cannot be used to justify a conclusion, we can attach appropriate fundamental principles of the domain expressed as text to each rule, or at least each high-level rule, stored in the knowledge base. This is probably as far as the explanation capability can be taken. However, the ability to explain a line of reasoning may not be essential for some expert systems. For example, a scientific system built for experts may not be required to provide extensive explanations, because the conclusion it reaches can be self-explanatory to other experts; a simple rule-tracing might be quite appropriate. On the other hand, expert systems used in decision making usually demand complete and thoughtful explanations, as the cost of a wrong decision may be very high.

Expert systems employ **symbolic reasoning** when solving a problem. Symbols are used to represent different types of knowledge such as facts, concepts and rules. Unlike conventional programs written for numerical data processing, expert systems are built for knowledge processing and can easily deal with qualitative data.

Conventional programs process data using algorithms, or in other words, a series of well-defined step-by-step operations. An algorithm always performs the same operations in the same order, and it always provides an exact solution. Conventional programs do not make mistakes – but programmers sometimes do. Unlike conventional programs, expert systems do not follow a prescribed sequence of steps. They permit inexact reasoning and can deal with incomplete, uncertain and fuzzy data.

Can expert systems make mistakes?

Even a brilliant expert is only a human and thus can make mistakes. This suggests that an expert system built to perform at a human expert level also should be allowed to make mistakes. But we still trust experts, although we do recognise that their judgements are sometimes wrong. Likewise, at least in most cases, we can rely on solutions provided by expert systems, but mistakes are possible and we should be aware of this.

Does it mean that conventional programs have an advantage over expert systems?

In theory, conventional programs always provide the same 'correct' solutions. However, we must remember that conventional programs can tackle problems if, and only if, the data is complete and exact. When the data is incomplete or includes some errors, a conventional program will provide either no solution at all or an incorrect one. In contrast, expert systems recognise that the available information may be incomplete or fuzzy, but they can work in such situations and still arrive at some reasonable conclusion.

Another important feature that distinguishes expert systems from conventional programs is that **knowledge is separated from its processing** (the knowledge base and the inference engine are split up). A conventional program is a mixture of knowledge and the control structure to process this knowledge. This mixing leads to difficulties in understanding and reviewing the program code, as any change to the code affects both the knowledge and its processing. In expert systems, knowledge is clearly separated from the processing mechanism. This makes expert systems much easier to build and maintain. When an expert system shell is used, a knowledge engineer or an expert simply enters rules in the knowledge base. Each new rule adds some new knowledge and makes the expert system smarter. The system can then be easily modified by changing or subtracting rules.

Table 2.1 Comparison of expert systems with conventional systems and human experts

Human experts	Expert systems	Conventional programs
Use knowledge in the form of rules of thumb or heuristics to solve problems in a narrow domain.	Process knowledge expressed in the form of rules and use symbolic reasoning to solve problems in a **narrow domain**.	Process data and use algorithms, a series of well-defined operations, to solve general numerical problems.
In a human brain, knowledge exists in a compiled form.	Provide a **clear separation of knowledge from its processing**.	Do not separate knowledge from the control structure to process this knowledge.
Capable of explaining a line of reasoning and providing the details.	**Trace the rules fired** during a problem-solving session and **explain how** a particular conclusion was reached and **why** specific data was needed.	Do not explain how a particular result was obtained and why input data was needed.
Use inexact reasoning and can deal with incomplete, uncertain and fuzzy information.	Permit **inexact reasoning** and can deal with incomplete, uncertain and fuzzy data.	Work only on problems where data is complete and exact.
Can make mistakes when information is incomplete or fuzzy.	**Can make mistakes** when data is incomplete or fuzzy.	Provide no solution at all, or a wrong one, when data is incomplete or fuzzy.
Enhance the quality of problem solving via years of learning and practical training. This process is slow, inefficient and expensive.	Enhance the quality of problem solving by adding new rules or adjusting old ones in the knowledge base. When new knowledge is acquired, **changes are easy** to accomplish.	Enhance the quality of problem solving by changing the program code, which affects both the knowledge and its processing, making changes difficult.

The characteristics of expert systems discussed above make them different from conventional systems and human experts. A comparison is shown in Table 2.1.

2.6 Forward chaining and backward chaining inference techniques

In a rule-based expert system, the domain knowledge is represented by a set of IF-THEN production rules and data is represented by a set of facts about the current situation. The inference engine compares each rule stored in the

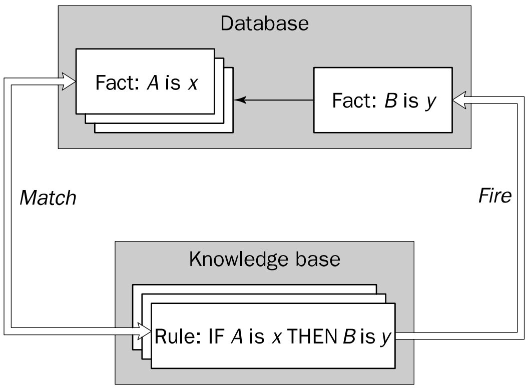

Figure 2.4 The inference engine cycles via a match-fire procedure

knowledge base with facts contained in the database. When the IF (condition) part of the rule matches a fact, the rule is **fired** and its THEN (action) part is executed. The fired rule may change the set of facts by adding a new fact, as shown in Figure 2.4. Letters in the database and the knowledge base are used to represent situations or concepts.

The matching of the rule IF parts to the facts produces **inference chains**. The inference chain indicates how an expert system applies the rules to reach a conclusion. To illustrate chaining inference techniques, consider a simple example.

Suppose the database initially includes facts *A*, *B*, *C*, *D* and *E*, and the knowledge base contains only three rules:

Rule 1:	IF	*Y* is true
	AND	*D* is true
	THEN	*Z* is true
Rule 2:	IF	*X* is true
	AND	*B* is true
	AND	*E* is true
	THEN	*Y* is true
Rule 3:	IF	*A* is true
	THEN	*X* is true

The inference chain shown in Figure 2.5 indicates how the expert system applies the rules to infer fact *Z*. First Rule 3 is fired to deduce new fact *X* from given fact *A*. Then Rule 2 is executed to infer fact *Y* from initially known facts *B* and *E*, and already known fact *X*. And finally, Rule 1 applies initially known fact *D* and just-obtained fact *Y* to arrive at conclusion *Z*.

An expert system can display its inference chain to explain how a particular conclusion was reached; this is an essential part of its explanation facilities.

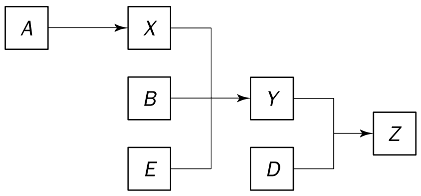

Figure 2.5 An example of an inference chain

The inference engine must decide when the rules have to be fired. There are two principal ways in which rules are executed. One is called **forward chaining** and the other **backward chaining** (Waterman and Hayes-Roth, 1978).

2.6.1 Forward chaining

The example discussed above uses forward chaining. Now consider this technique in more detail. Let us first rewrite our rules in the following form:

Rule 1: $Y \,\&\, D \rightarrow Z$

Rule 2: $X \,\&\, B \,\&\, E \rightarrow Y$

Rule 3: $A \rightarrow X$

Arrows here indicate the IF and THEN parts of the rules. Let us also add two more rules:

Rule 4: $C \rightarrow L$

Rule 5: $L \,\&\, M \rightarrow N$

Figure 2.6 shows how forward chaining works for this simple set of rules.

Forward chaining is the **data-driven** reasoning. The reasoning starts from the known data and proceeds forward with that data. Each time only the topmost rule is executed. When fired, the rule adds a new fact in the database. Any rule can be executed only once. The match-fire cycle stops when no further rules can be fired.

In the first cycle, only two rules, Rule 3: $A \rightarrow X$ and Rule 4: $C \rightarrow L$, match facts in the database. Rule 3: $A \rightarrow X$ is fired first as the topmost one. The IF part of this rule matches fact A in the database, its THEN part is executed and new fact X is added to the database. Then Rule 4: $C \rightarrow L$ is fired and fact L is also placed in the database.

In the second cycle, Rule 2: $X \& B \& E \rightarrow Y$ is fired because facts B, E and X are already in the database, and as a consequence fact Y is inferred and put in the database. This in turn causes Rule 1: $Y \& D \rightarrow Z$ to execute, placing fact Z in the database (cycle 3). Now the match-fire cycles stop because the IF part of Rule 5: $L \& M \rightarrow N$ does not match **all** facts in the database and thus Rule 5 cannot be fired.

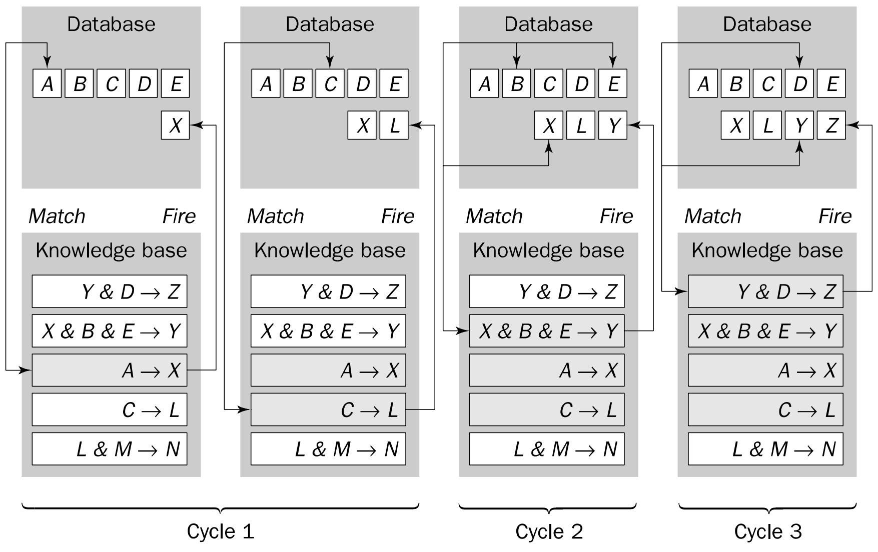

Figure 2.6 Forward chaining

Forward chaining is a technique for gathering information and then inferring from it whatever can be inferred. However, in forward chaining, many rules may be executed that have nothing to do with the established goal. Suppose, in our example, the goal was to determine fact *Z*. We had only five rules in the knowledge base and four of them were fired. But Rule 4: $C \rightarrow L$, which is unrelated to fact *Z*, was also fired among others. A real rule-based expert system can have hundreds of rules, many of which might be fired to derive new facts that are valid, but unfortunately unrelated to the goal. Therefore, if our goal is to infer only one particular fact, the forward chaining inference technique would not be efficient.

In such a situation, backward chaining is more appropriate.

2.6.2 Backward chaining

Backward chaining is the **goal-driven** reasoning. In backward chaining, an expert system has the goal (a hypothetical solution) and the inference engine attempts to find the evidence to prove it. First, the knowledge base is searched to find rules that might have the desired solution. Such rules must have the goal in their THEN (action) parts. If such a rule is found and its IF (condition) part matches data in the database, then the rule is fired and the goal is proved. However, this is rarely the case. Thus the inference engine puts aside the rule it is working with (the rule is said to **stack**) and sets up a new goal, a sub-goal, to prove the IF part of this rule. Then the knowledge base is searched again for rules that can prove the sub-goal. The inference engine repeats the process of stacking the rules until no rules are found in the knowledge base to prove the current sub-goal.

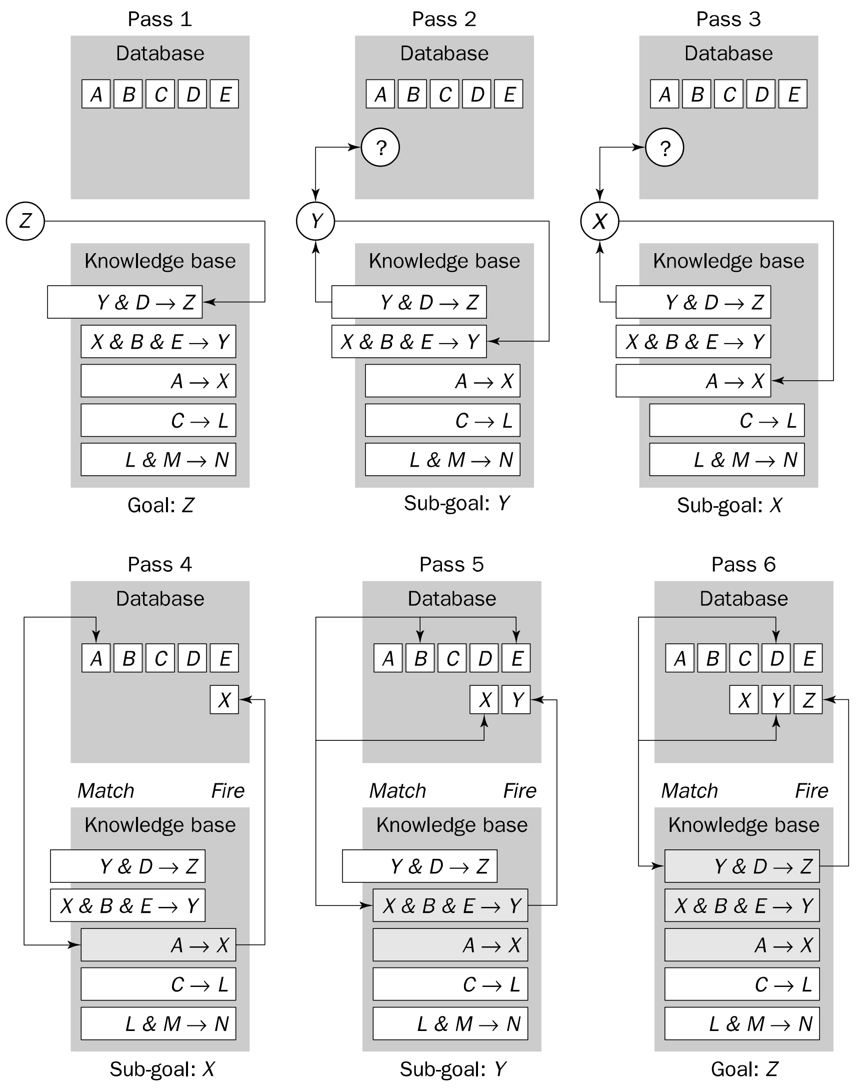

Figure 2.7 Backward chaining

Figure 2.7 shows how backward chaining works, using the rules for the forward chaining example.

In Pass 1, the inference engine attempts to infer fact *Z*. It searches the knowledge base to find the rule that has the goal, in our case fact *Z*, in its THEN part. The inference engine finds and stacks Rule 1: $Y \& D \rightarrow Z$. The IF part of Rule 1 includes facts *Y* and *D*, and thus these facts must be established.

In Pass 2, the inference engine sets up the sub-goal, fact *Y*, and tries to determine it. First it checks the database, but fact *Y* is not there. Then the knowledge base is searched again for the rule with fact *Y* in its THEN part. The inference engine locates and stacks Rule 2: $X \& B \& E \rightarrow Y$. The IF part of Rule 2 consists of facts *X*, *B* and *E*, and these facts also have to be established.

In Pass 3, the inference engine sets up a new sub-goal, fact *X*. It checks the database for fact *X*, and when that fails, searches for the rule that infers *X*. The inference engine finds and stacks Rule 3: $A \rightarrow X$. Now it must determine fact *A*.

In Pass 4, the inference engine finds fact *A* in the database, Rule 3: $A \rightarrow X$ is fired and new fact *X* is inferred.

In Pass 5, the inference engine returns to the sub-goal fact *Y* and once again tries to execute Rule 2: $X \& B \& E \rightarrow Y$. Facts *X*, *B* and *E* are in the database and thus Rule 2 is fired and a new fact, fact *Y*, is added to the database.

In Pass 6, the system returns to Rule 1: $Y \& D \rightarrow Z$ trying to establish the original goal, fact *Z*. The IF part of Rule 1 matches all facts in the database, Rule 1 is executed and thus the original goal is finally established.

Let us now compare Figure 2.6 with Figure 2.7. As you can see, four rules were fired when forward chaining was used, but just three rules when we applied backward chaining. This simple example shows that the backward chaining inference technique is more effective when we need to infer one particular fact, in our case fact *Z*. In forward chaining, the data is known at the beginning of the inference process, and the user is never asked to input additional facts. In backward chaining, the goal is set up and the only data used is the data needed to support the direct line of reasoning, and the user may be asked to input any fact that is not in the database.

How do we choose between forward and backward chaining?

The answer is to study how a domain expert solves a problem. If an expert first needs to gather some information and then tries to infer from it whatever can be inferred, choose the forward chaining inference engine. However, if your expert begins with a hypothetical solution and then attempts to find facts to prove it, choose the backward chaining inference engine.

Forward chaining is a natural way to design expert systems for analysis and interpretation. For example, DENDRAL, an expert system for determining the molecular structure of unknown soil based on its mass spectral data (Feigenbaum *et al.*, 1971), uses forward chaining. Most backward chaining expert systems are used for diagnostic purposes. For instance, MYCIN, a medical expert system for diagnosing infectious blood diseases (Shortliffe, 1976), uses backward chaining.

Can we combine forward and backward chaining?

Many expert system shells use a combination of forward and backward chaining inference techniques, so the knowledge engineer does not have to choose between them. However, the basic inference mechanism is usually backward chaining. Only when a new fact is established is forward chaining employed to maximise the use of the new data.

2.7 MEDIA ADVISOR: a demonstration rule-based expert system

To illustrate some of the ideas discussed above, we next consider a simple rule-based expert system. The Leonardo expert system shell was selected as a tool to build a decision-support system called MEDIA ADVISOR. The system provides advice on selecting a medium for delivering a training program based on the trainee's job. For example, if a trainee is a mechanical technician responsible for maintaining hydraulic systems, an appropriate medium might be a workshop, where the trainee could learn how basic hydraulic components operate, how to troubleshoot hydraulics problems and how to make simple repairs to hydraulic systems. On the other hand, if a trainee is a clerk assessing insurance applications, a training program might include lectures on specific problems of the task, as well as tutorials where the trainee could evaluate real applications. For complex tasks, where trainees are likely to make mistakes, a training program should also include feedback on the trainee's performance.

Knowledge base

```
/* MEDIA ADVISOR: a demonstration rule-based expert system

Rule: 1
if    the environment is papers
or    the environment is manuals
or    the environment is documents
or    the environment is textbooks
then  the stimulus_situation is verbal

Rule: 2
if    the environment is pictures
or    the environment is illustrations
or    the environment is photographs
or    the environment is diagrams
then  the stimulus_situation is visual

Rule: 3
if    the environment is machines
or    the environment is buildings
or    the environment is tools
then  the stimulus_situation is 'physical object'

Rule: 4
if    the environment is numbers
or    the environment is formulas
or    the environment is 'computer programs'
then  the stimulus_situation is symbolic
```

```
Rule: 5
if     the job is lecturing
or     the job is advising
or     the job is counselling
then   the stimulus_response is oral

Rule: 6
if     the job is building
or     the job is repairing
or     the job is troubleshooting
then   the stimulus_response is ‘hands-on’

Rule: 7
if     the job is writing
or     the job is typing
or     the job is drawing
then   the stimulus_response is documented

Rule: 8
if     the job is evaluating
or     the job is reasoning
or     the job is investigating
then   the stimulus_response is analytical

Rule: 9
if     the stimulus_situation is ‘physical object’
and    the stimulus_response is ‘hands-on’
and    feedback is required
then   medium is workshop

Rule: 10
if     the stimulus_situation is symbolic
and    the stimulus_response is analytical
and    feedback is required
then   medium is ‘lecture – tutorial’

Rule: 11
if     the stimulus_situation is visual
and    the stimulus_response is documented
and    feedback is not required
then   medium is videocassette

Rule: 12
if     the stimulus_situation is visual
and    the stimulus_response is oral
and    feedback is required
then   medium is ‘lecture – tutorial’
```

```
Rule: 13
if      the stimulus_situation is verbal
and     the stimulus_response is analytical
and     feedback is required
then    medium is 'lecture – tutorial'

Rule: 14
if      the stimulus_situation is verbal
and     the stimulus_response is oral
and     feedback is required
then    medium is 'role-play exercises'

/* The SEEK directive sets up the goal of the rule set

seek medium
```

Objects

MEDIA ADVISOR uses six linguistic objects: *environment, stimulus_situation, job, stimulus_response, feedback* and *medium*. Each object can take one of the allowed values (for example, object *environment* can take the value of *papers, manuals, documents, textbooks, pictures, illustrations, photographs, diagrams, machines, buildings, tools, numbers, formulas, computer programs*). An object and its value constitute a fact (for instance, the *environment* is *machines*, and the *job* is *repairing*). All facts are placed in the database.

Object	Allowed values	Object	Allowed values
environment	*papers*	*job*	*lecturing*
	manuals		*advising*
	documents		*counselling*
	textbooks		*building*
	pictures		*repairing*
	illustrations		*troubleshooting*
	photographs		*writing*
	diagrams		*typing*
	machines		*drawing*
	buildings		*evaluating*
	tools		*reasoning*
	numbers		*investigating*
	formulas		
	computer programs	*stimulus_ response*	*oral*
			hands-on
			documented
			analytical
stimulus_situation	*verbal*		
	visual		
	physical object	*feedback*	*required*
	symbolic		*not required*

Options

The final goal of the rule-based expert system is to produce a solution to the problem based on input data. In MEDIA ADVISOR, the solution is a medium selected from the list of four options:

medium is workshop
medium is 'lecture – tutorial'
medium is videocassette
medium is 'role-play exercises'

Dialogue

In the dialogue shown below, the expert system asks the user to input the data needed to solve the problem (the environment, the job and feedback). Based on the answers supplied by the user (answers are indicated by arrows), the expert system applies rules from its knowledge base to infer that the *stimulus_situation* is *physical object*, and the *stimulus_response* is *hands-on*. Rule 9 then selects one of the allowed values of *medium*.

What sort of environment is a trainee dealing with on the job?
⇒ **machines**

```
Rule: 3
if     the environment is machines
or     the environment is buildings
or     the environment is tools
then   the stimulus_situation is 'physical object'
```

In what way is a trainee expected to act or respond on the job?
⇒ **repairing**

```
Rule: 6
if     the job is building
or     the job is repairing
or     the job is troubleshooting
then   the stimulus_response is 'hands-on'
```

Is feedback on the trainee's progress required during training?
⇒ **required**

```
Rule: 9
if     the stimulus_situation is 'physical object'
and    the stimulus_response is 'hands-on'
and    feedback is required
then   medium is workshop
```

medium is workshop

Inference techniques

The standard inference technique in Leonardo is backward chaining with opportunistic forward chaining, which is the most efficient way to deal with the available information. However, Leonardo also enables the user to turn off either backward or forward chaining, and thus allows us to study each inference technique separately.

Forward chaining is data-driven reasoning, so we need first to provide some data. Assume that

the environment is **machines**
  '**environment**' instantiated by user input to '**machines**'

the job is **repairing**
  '**job**' instantiated by user input to '**repairing**'

feedback is **required**
  '**feedback**' instantiated by user input to '**required**'

The following process will then happen:

Rule: 3 fires	'**stimulus_situation**' instantiated by Rule: 3 to '**physical object**'
Rule: 6 fires	'**stimulus_response**' instantiated by Rule: 6 to '**hands-on**'
Rule: 9 fires	'**medium**' instantiated by Rule: 9 to '**workshop**'
No rules fire	stop

Backward chaining is goal-driven reasoning, so we need first to establish a hypothetical solution (the goal). Let us, for example, set up the following goal:

'medium' is 'workshop'

Pass 1	
Trying Rule: 9	Need to find object '**stimulus_situation**'
Rule: 9 stacked	Object '**stimulus_situation**' sought as '**physical object**'
Pass 2	
Trying Rule: 3	Need to find object '**environment**'
Rule: 3 stacked	Object '**environment**' sought as '**machines**'
ask environment	
⇒**machines**	'**environment**' instantiated by user input to '**machines**'
Trying Rule: 3	'**stimulus_situation**' instantiated by Rule: 3 to '**physical object**'
Pass 3	
Trying Rule: 9	Need to find object '**stimulus_response**'
Rule: 9 stacked	Object '**stimulus_response**' sought as '**hands-on**'

Pass 4	
Trying Rule: 6	Need to find object '**job**'
Rule: 6 stacked	Object '**job**' sought as '**building**'
ask job	
⇒ **repairing**	'**job**' instantiated by user input to '**repairing**'
Trying Rule: 6	'**stimulus_response**' instantiated by Rule: 6 to '**hands-on**'
Pass 5	
Trying Rule: 9	Need to find object '**feedback**'
Rule: 9 stacked	Object '**feedback**' sought as '**required**'
ask feedback	
⇒ **required**	'**feedback**' instantiated by user input to '**required**'
Trying Rule: 9	'**medium**' instantiated by Rule: 9 to '**workshop**'

medium is workshop

It is useful to have a tree diagram that maps a consultation session with an expert system. A diagram for MEDIA ADVISOR is shown in Figure 2.8. The root node is the goal; when the system is started, the inference engine seeks to determine the goal's value.

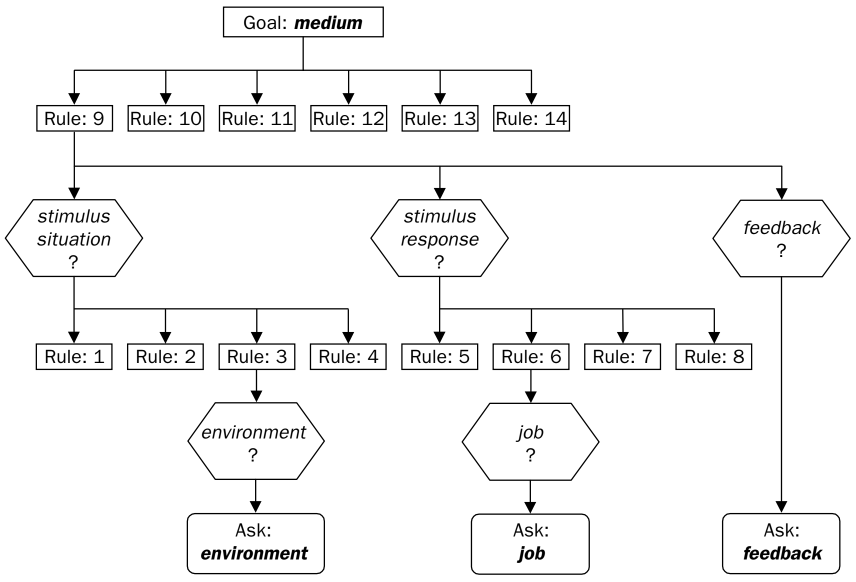

Figure 2.8 Tree diagram for the rule-based expert system MEDIA ADVISOR

Does MEDIA ADVISOR handle all possible situations?

When we start to use our expert system more often, we might find that the provided options do not cover all possible situations. For instance, the following dialogue might occur:

What sort of environment is a trainee dealing with on the job?
⇒**illustrations**

In what way is a trainee expected to act or respond on the job?
⇒**drawing**

Is feedback on the trainee's progress required during training?
⇒**required**

I am unable to draw any conclusions on the basis of the data.

Thus, MEDIA ADVISOR in its present state cannot handle this particular situation. Fortunately, the expert system can easily be expanded to accommodate more rules until it finally does what the user wants it to do.

2.8 Conflict resolution

Earlier in this chapter, we considered two simple rules for crossing a road. Let us now add a third rule. We will get the following set of rules:

```
Rule 1:
IF     the 'traffic light' is green
THEN   the action is go

Rule 2:
IF     the 'traffic light' is red
THEN   the action is stop

Rule 3:
IF     the 'traffic light' is red
THEN   the action is go
```

What will happen?

The inference engine compares IF (condition) parts of the rules with data available in the database, and when conditions are satisfied the rules are set to fire. The firing of one rule may affect the activation of other rules, and therefore the inference engine must allow only one rule to fire at a time. In our road crossing example, we have two rules, Rule 2 and Rule 3, with the same IF part. Thus both of them can be set to fire when the condition part is satisfied. These rules represent a conflict set. The inference engine must determine which rule to fire from such a set. A method for choosing a rule to fire when more than one rule can be fired in a given cycle is called **conflict resolution**.

If the traffic light is red, which rule should be executed?

In forward chaining, **both** rules would be fired. Rule 2 is fired first as the top-most one, and as a result, its THEN part is executed and linguistic object *action* obtains value *stop*. However, Rule 3 is also fired because the condition part of this rule matches the fact *'traffic light' is red*, which is still in the database. As a consequence, object *action* takes new value *go*. This simple example shows that the rule order is vital when the forward chaining inference technique is used.

How can we resolve a conflict?

The obvious strategy for resolving conflicts is to establish a goal and stop the rule execution when the goal is reached. In our problem, for example, the goal is to establish a value for linguistic object *action*. When the expert system determines a value for *action*, it has reached the goal and stops. Thus if the *traffic light* is *red*, Rule 2 is executed, object *action* attains value *stop* and the expert system stops. In the given example, the expert system makes a right decision; however if we arranged the rules in the reverse order, the conclusion would be wrong. It means that the rule order in the knowledge base is still very important.

Are there any other conflict resolution methods?

Several methods are in use (Giarratano and Riley, 1998; Shirai and Tsuji, 1982):

- Fire the rule with the highest priority. In simple applications, the priority can be established by placing the rules in an appropriate order in the knowledge base. Usually this strategy works well for expert systems with around 100 rules. However, in some applications, the data should be processed in order of importance. For example, in a medical consultation system (Durkin, 1994), the following priorities are introduced:

```
Goal 1. Prescription is? Prescription

RULE 1 Meningitis Prescription1
(Priority 100)
IF      Infection is Meningitis
AND     The Patient is a Child
THEN    Prescription is Number_1
AND     Drug Recommendation is Ampicillin
AND     Drug Recommendation is Gentamicin
AND     Display Meningitis Prescription1

RULE 2 Meningitis Prescription2
(Priority 90)
IF      Infection is Meningitis
AND     The Patient is an Adult
THEN    Prescription is Number_2
AND     Drug Recommendation is Penicillin
AND     Display Meningitis Prescription2
```

- Fire the most specific rule. This method is also known as the **longest matching strategy**. It is based on the assumption that a specific rule processes more information than a general one. For example,

Rule 1:	
IF	the season is autumn
AND	the sky is cloudy
AND	the forecast is rain
THEN	the advice is 'stay home'
Rule 2:	
IF	the season is autumn
THEN	the advice is 'take an umbrella'

  If the *season* is *autumn*, the *sky* is *cloudy* and the *forecast* is *rain*, then Rule 1 would be fired because its antecedent, the matching part, is more specific than that of Rule 2. But if it is known only that the *season* is *autumn*, then Rule 2 would be executed.

- Fire the rule that uses the **data most recently entered** in the database. This method relies on time tags attached to each fact in the database. In the conflict set, the expert system first fires the rule whose antecedent uses the data most recently added to the database. For example,

Rule 1:		
IF	the forecast is rain	[08:16 PM 11/25/96]
THEN	the advice is 'take an umbrella'	
Rule 2:		
IF	the weather is wet	[10:18 AM 11/26/96]
THEN	the advice is 'stay home'	

  Assume that the IF parts of both rules match facts in the database. In this case, Rule 2 would be fired since the fact *weather* is *wet* was entered after the fact *forecast* is *rain*. This technique is especially useful for real-time expert system applications when information in the database is constantly updated.

The conflict resolution methods considered above are simple and easily implemented. In most cases, these methods provide satisfactory solutions. However, as a program grows larger and more complex, it becomes increasingly difficult for the knowledge engineer to manage and oversee rules in the knowledge base. The expert system itself must take at least some of the burden and understand its own behaviour.

To improve the performance of an expert system, we should supply the system with some knowledge about the knowledge it possesses, or in other words, **metaknowledge**.

Metaknowledge can be simply defined as **knowledge about knowledge**. Metaknowledge is knowledge about the use and control of domain knowledge in an expert system (Waterman, 1986). In rule-based expert systems, metaknowledge is represented by metarules. A metarule determines a strategy for the use of task-specific rules in the expert system.

What is the origin of metaknowledge?

The knowledge engineer transfers the knowledge of the domain expert to the expert system, learns how problem-specific rules are used, and gradually creates in his or her own mind a new body of knowledge, knowledge about the overall behaviour of the expert system. This new knowledge, or metaknowledge, is largely domain-independent. For example,

Metarule 1:
Rules supplied by experts have higher priorities than rules supplied by novices.

Metarule 2:
Rules governing the rescue of human lives have higher priorities than rules concerned with clearing overloads on power system equipment.

Can an expert system understand and use metarules?

Some expert systems provide a separate inference engine for metarules. However, most expert systems cannot distinguish between rules and metarules. Thus metarules should be given the highest priority in the existing knowledge base. When fired, a metarule 'injects' some important information into the database that can change the priorities of some other rules.

2.9 Advantages and disadvantages of rule-based expert systems

Rule-based expert systems are generally accepted as the best option for building knowledge-based systems.

Which features make rule-based expert systems particularly attractive for knowledge engineers?

Among these features are:

- **Natural knowledge representation**. An expert usually explains the problem-solving procedure with such expressions as this: 'In such-and-such situation, I do so-and-so'. These expressions can be represented quite naturally as IF-THEN production rules.
- **Uniform structure**. Production rules have the uniform IF-THEN structure. Each rule is an independent piece of knowledge. The very syntax of production rules enables them to be self-documented.
- **Separation of knowledge from its processing**. The structure of a rule-based expert system provides an effective separation of the knowledge base from the inference engine. This makes it possible to develop different applications using the same expert system shell. It also allows a graceful and easy expansion of the expert system. To make the system smarter, a knowledge engineer simply adds some rules to the knowledge base without intervening in the control structure.

- **Dealing with incomplete and uncertain knowledge**. Most rule-based expert systems are capable of representing and reasoning with incomplete and uncertain knowledge. For example, the rule

  ```
  IF     season is autumn
  AND    sky is 'cloudy'
  AND    wind is low
  THEN   forecast is clear     { cf 0.1 };
         forecast is drizzle   { cf 1.0 };
         forecast is rain      { cf 0.9 }
  ```

  could be used to express the uncertainty of the following statement, 'If the season is autumn and it looks like drizzle, then it will probably be another wet day today'.

  The rule represents the uncertainty by numbers called **certainty factors** {cf 0.1}. The expert system uses certainty factors to establish the degree of confidence or level of belief that the rule's conclusion is true. This topic will be considered in detail in Chapter 3.

All these features of the rule-based expert systems make them highly desirable for knowledge representation in real-world problems.

Are rule-based expert systems problem-free?

There are three main shortcomings:

- **Opaque relations between rules**. Although the individual production rules tend to be relatively simple and self-documented, their logical interactions within the large set of rules may be opaque. Rule-based systems make it difficult to observe how individual rules serve the overall strategy. This problem is related to the lack of hierarchical knowledge representation in the rule-based expert systems.
- **Ineffective search strategy**. The inference engine applies an exhaustive search through all the production rules during each cycle. Expert systems with a large set of rules (over 100 rules) can be slow, and thus large rule-based systems can be unsuitable for real-time applications.
- **Inability to learn**. In general, rule-based expert systems do not have an ability to learn from the experience. Unlike a human expert, who knows when to 'break the rules', an expert system cannot automatically modify its knowledge base, or adjust existing rules or add new ones. The knowledge engineer is still responsible for revising and maintaining the system.

2.10 Summary

In this chapter, we presented an overview of rule-based expert systems. We briefly discussed what knowledge is, and how experts express their knowledge in the form of production rules. We identified the main players in the expert system development team and showed the structure of a rule-based system. We discussed fundamental characteristics of expert systems and noted that expert systems can make mistakes. Then we reviewed the forward and backward chaining inference techniques and debated conflict resolution strategies. Finally, the advantages and disadvantages of rule-based expert systems were examined.

The most important lessons learned in this chapter are:

- Knowledge is a theoretical or practical understanding of a subject. Knowledge is the sum of what is currently known.
- An expert is a person who has deep knowledge in the form of facts and rules and strong practical experience in a particular domain. An expert can do things other people cannot.
- The experts can usually express their knowledge in the form of production rules.
- Production rules are represented as IF (antecedent) THEN (consequent) statements. A production rule is the most popular type of knowledge representation. Rules can express relations, recommendations, directives, strategies and heuristics.
- A computer program capable of performing at a human-expert level in a narrow problem domain area is called an expert system. The most popular expert systems are rule-based expert systems.
- In developing rule-based expert systems, shells are becoming particularly common. An expert system shell is a skeleton expert system with the knowledge removed. To build a new expert system application, all the user has to do is to add the knowledge in the form of rules and provide relevant data. Expert system shells offer a dramatic reduction in the development time of expert systems.
- The expert system development team should include the domain expert, the knowledge engineer, the programmer, the project manager and the end-user. The knowledge engineer designs, builds and tests an expert system. He or she captures the knowledge from the domain expert, establishes reasoning methods and chooses the development software. For small expert systems based on expert system shells, the project manager, knowledge engineer, programmer and even the expert could be the same person.
- A rule-based expert system has five basic components: the knowledge base, the database, the inference engine, the explanation facilities and the user interface. The knowledge base contains the domain knowledge represented as a set of rules. The database includes a set of facts used to match against the IF parts of rules. The inference engine links the rules with the facts and carries out the reasoning whereby the expert system reaches a solution. The explanation facilities enable the user to query the expert system about **how** a particular conclusion is reached and **why** a specific fact is needed. The user interface is the means of communication between a user and an expert system.

- Expert systems separate knowledge from its processing by splitting up the knowledge base and the inference engine. This makes the task of building and maintaining an expert system much easier. When an expert system shell is used, a knowledge engineer or an expert simply enter rules in the knowledge base. Each new rule adds some new knowledge and makes the expert system smarter.
- Expert systems provide a limited explanation capability by tracing the rules fired during a problem-solving session.
- Unlike conventional programs, expert systems can deal with incomplete and uncertain data and permit inexact reasoning. However, like their human counterparts, expert systems can make mistakes when information is incomplete or fuzzy.
- There are two principal methods to direct search and reasoning: forward chaining and backward chaining inference techniques. Forward chaining is data-driven reasoning; it starts from the known data and proceeds forward until no further rules can be fired. Backward chaining is goal-driven reasoning; an expert system has a hypothetical solution (the goal), and the inference engine attempts to find the evidence to prove it.
- If more than one rule can be fired in a given cycle, the inference engine must decide which rule to fire. A method for deciding is called conflict resolution.
- Rule-based expert systems have the advantages of natural knowledge representation, uniform structure, separation of knowledge from its processing, and coping with incomplete and uncertain knowledge.
- Rule-based expert systems also have disadvantages, especially opaque relations between rules, ineffective search strategy, and inability to learn.

Questions for review

1. What is knowledge? Explain why experts usually have detailed knowledge of a limited area of a specific domain. What do we mean by heuristic?
2. What is a production rule? Give an example and define two basic parts of the production rule.
3. List and describe the five major players in the expert system development team. What is the role of the knowledge engineer?
4. What is an expert system shell? Explain why the use of an expert system shell can dramatically reduce the development time of an expert system.
5. What is a production system model? List and define the five basic components of an expert system.
6. What are the fundamental characteristics of an expert system? What are the differences between expert systems and conventional programs?

7 Can an expert system make mistakes? Why?

8 Describe the forward chaining inference process. Give an example.

9 Describe the backward chaining inference process. Give an example.

10 List problems for which the forward chaining inference technique is appropriate. Why is backward chaining used for diagnostic problems?

11 What is a conflict set of rules? How can we resolve a conflict? List and describe the basic conflict resolution methods.

12 List advantages of rule-based expert systems. What are their disadvantages?

References

Duda, R., Gaschnig, J. and Hart, P. (1979). Model design in the PROSPECTOR consultant system for mineral exploration, *Expert Systems in the Microelectronic Age*, D. Michie, ed., Edinburgh University Press, Edinburgh, Scotland, pp. 153–167.

Durkin, J. (1994). *Expert Systems Design and Development*. Prentice Hall, Englewood Cliffs, NJ.

Feigenbaum, E.A., Buchanan, B.G. and Lederberg, J. (1971). On generality and problem solving: a case study using the DENDRAL program, *Machine Intelligence 6*, B. Meltzer and D. Michie, eds, Edinburgh University Press, Edinburgh, Scotland, pp. 165–190.

Giarratano, J. and Riley, G. (1998). *Expert Systems: Principles and Programming*, 3rd edn. PWS Publishing Company, Boston.

Negnevitsky, M. (1996). Crisis management in power systems: a knowledge based approach, *Applications of Artificial Intelligence in Engineering XI*, R.A. Adey, G. Rzevski and A.K. Sunol, eds, Computational Mechanics Publications, Southampton, UK, pp. 122–141.

Newell, A. and Simon, H.A. (1972). *Human Problem Solving*. Prentice Hall, Englewood Cliffs, NJ.

Shirai, Y. and Tsuji, J. (1982). *Artificial Intelligence: Concepts, Technologies and Applications*. John Wiley, New York.

Shortliffe, E.H. (1976). *MYCIN: Computer-Based Medical Consultations*. Elsevier Press, New York.

Waterman, D.A. (1986). *A Guide to Expert Systems*. Addison-Wesley, Reading, MA.

Waterman, D.A. and Hayes-Roth, F. (1978). An overview of pattern-directed inference systems, *Pattern-Directed Inference Systems*, D.A. Waterman and F. Hayes-Roth, eds, Academic Press, New York.