

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 maximize 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 |
|---------------------------|--------------------------|--------------------------|------------------------|
| <i>environment</i> | <i>papers</i> | <i>job</i> | <i>lecturing</i> |
| | <i>manuals</i> | | <i>advising</i> |
| | <i>documents</i> | | <i>counselling</i> |
| | <i>textbooks</i> | | <i>building</i> |
| | <i>pictures</i> | | <i>repairing</i> |
| | <i>illustrations</i> | | <i>troubleshooting</i> |
| | <i>photographs</i> | | <i>writing</i> |
| | <i>diagrams</i> | | <i>typing</i> |
| | <i>machines</i> | | <i>drawing</i> |
| | <i>buildings</i> | | <i>evaluating</i> |
| | <i>tools</i> | | <i>reasoning</i> |
| | <i>numbers</i> | | <i>investigating</i> |
| | <i>formulas</i> | <i>stimulus_response</i> | <i>oral</i> |
| | <i>computer programs</i> | | <i>hands-on</i> |
| <i>stimulus_situation</i> | <i>verbal</i> | | <i>documented</i> |
| | <i>visual</i> | | <i>analytical</i> |
| | <i>physical object</i> | <i>feedback</i> | <i>required</i> |
| | <i>symbolic</i> | | <i>not required</i> |

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

Rule: 9 stacked

Need to find object 'stimulus_situation'

Object 'stimulus_situation' sought as 'physical object'

Pass 2

Trying Rule: 3

Rule: 3 stacked

Need to find object 'environment'

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

Rule: 9 stacked

Need to find object 'stimulus_response'

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.

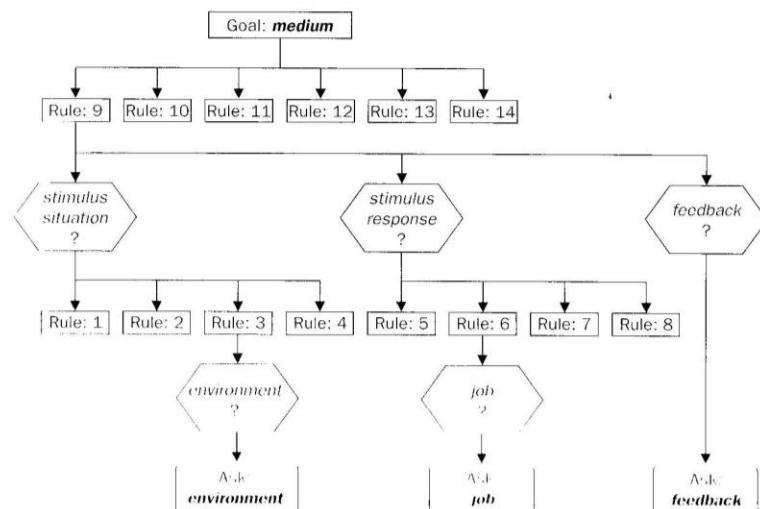


Figure 2.8 Tree diagram for the rule-based expert system MEDIA ADVISOR

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 Prescription_2

- 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

team 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?