Chapter Two: Rule-Based Expert Systems

By:
Dr Muhanad Tahrir Younis

1. Introduction

In which we introduce the most popular choice for building knowledge-based systems: Rule-Based Expert Systems(RBESs).

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.

2. 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 organizations. Any successful company has at least a few first-class experts and it cannot remain in business without them.

3. 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 skillful person who can do things other people cannot.

4. 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.

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,

5. What is an Expert System?

In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, like an expert, and not by following the procedure of a developer as is the case in conventional programming. The first expert systems were created in the 1970s and then proliferated in the 1980s. Expert systems were among the first truly successful forms of AI software.

As soon as knowledge is provided by a human expert, we can input it into a computer. We expect the computer to

1- 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.
2- 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.
3- 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.

THE APPLICATIONS OF EXPERT SYSTEMS

The spectrum of applications of expert systems technology to industrial and commercial problems is so wide as to defy easy characterization. The applications find their way into most areas of knowledge work. They are as varied as helping salespersons sell modular factory-built homes to helping NASA plan the maintenance of a space shuttle in preparation for its next flight.

Applications tend to cluster into seven major classes.

1- **Diagnosis and Troubleshooting of Devices and Systems of All Kinds:** This class comprises systems that deduce faults and suggest corrective actions for a malfunctioning device or process. Medical diagnosis was one of the first knowledge areas to which ES technology was applied, but diagnosis of engineered systems quickly surpassed medical diagnosis. There are probably

more diagnostic applications of ES than any other type. The diagnostic problem can be stated in the abstract as: given the evidence presenting itself, what is the underlying problem/reason/cause?

2- **Planning and Scheduling:** Systems that fall into this class analyze a set of one or more potentially complex and interacting goals in order to determine a set of actions to achieve those goals, and/or provide a detailed temporal ordering of those actions, taking into account personnel, materiel, and other constraints. This class has great commercial potential, which has been recognized. Examples involve airline scheduling of flights, personnel, and gates; manufacturing job-shop scheduling; and manufacturing process planning.

3- **Configuration of Manufactured Objects from Subassemblies:** Configuration, whereby a solution to a problem is synthesized from a given set of elements related by a set of constraints, is historically one of the most important of expert system applications. Configuration applications were pioneered by computer companies as a means of facilitating the manufacture of semi-custom minicomputers. The technique has found its way into use in many different industries, for example, modular home building, manufacturing, and other problems involving complex engineering design and manufacturing.

4- **Financial Decision Making:** The financial services industry has been a vigorous user of expert system techniques. Advisory programs have been created to assist bankers in determining whether to make loans to businesses and individuals. Insurance companies have used expert systems to assess the risk presented by the customer and to determine a price for the insurance. A typical application in the financial markets is in foreign exchange trading.

5- **Knowledge Publishing** This is a relatively new, but also potentially explosive area. The primary function of the expert system is to deliver knowledge that is

relevant to the user's problem, in the context of the user's problem. The two most widely distributed expert systems in the world are in this category. The first is an advisor which counsels a user on appropriate grammatical usage in a text. The second is a tax advisor that accompanies a tax preparation program and advises the user on tax strategy, tactics, and individual tax policy.

6- **Process Monitoring and Control:** Systems falling in this class analyze real-time data from physical devices with the goal of noticing anomalies, predicting trends, and controlling for both optimality and failure correction. Examples of real-time systems that actively monitor processes can be found in the steel making and oil refining industries.

7- **Design and Manufacturing:** These systems assist in the design of physical devices and processes, ranging from high-level conceptual design of abstract entities all the way to factory floor configuration of manufacturing processes.

6. 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.

7. The main players in the expert system development team

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 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 summarized in Figure (1).

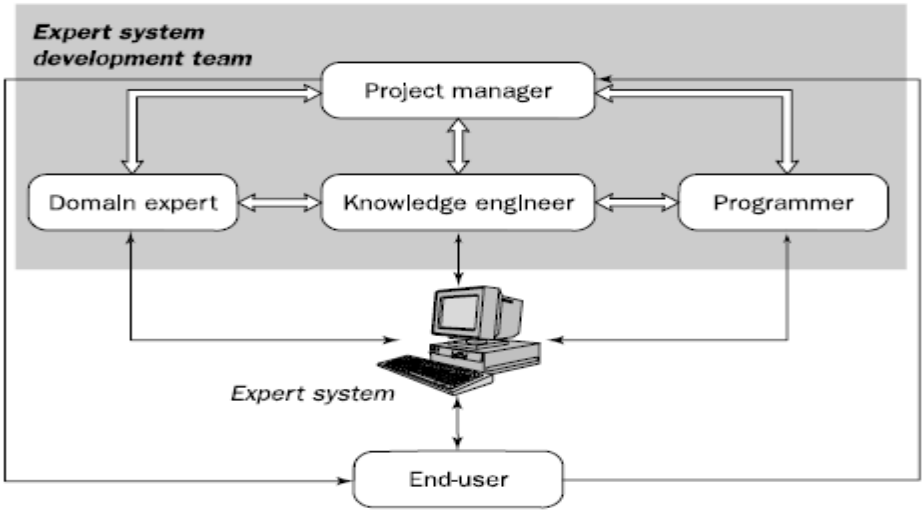

Figure (1): The main players of the expert system development team

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.

2- 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.

3- 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, and Prolog 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.

4- 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.

5- 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, a junior doctor diagnosing an infectious blood disease, an exploration geologist trying to discover a new mineral deposit, or a power system operator needing advice in an emergency. 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.

8. 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. 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).

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

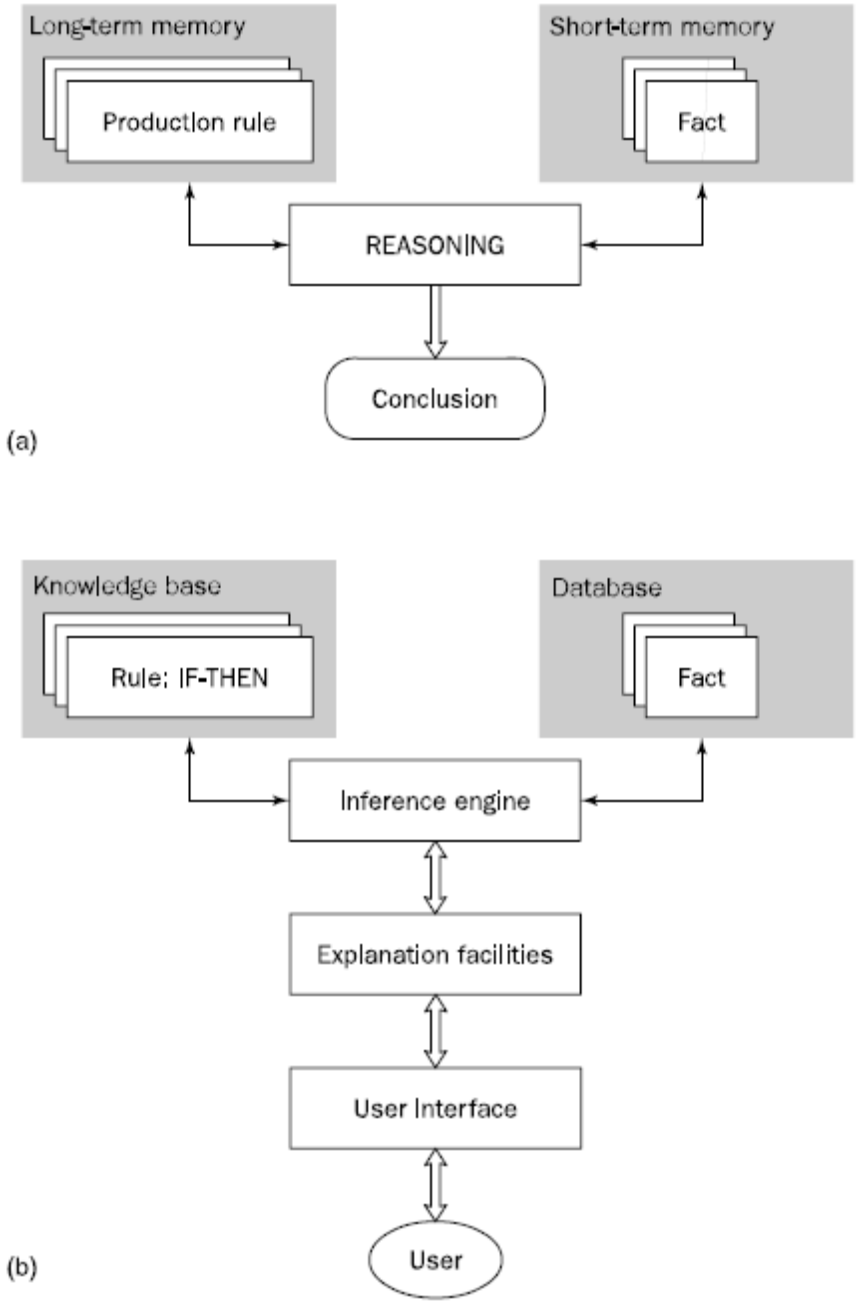

Figure (2): Production system and basic structure of a rule-based expert system: (a) production system model; (b) basic structure of a rule-based expert system

1- 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.
2- The **database** includes a set of facts used to match against the IF (condition) parts of rules stored in the knowledge base.
3- 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.
4- 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.
5- 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.

6- 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 (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.

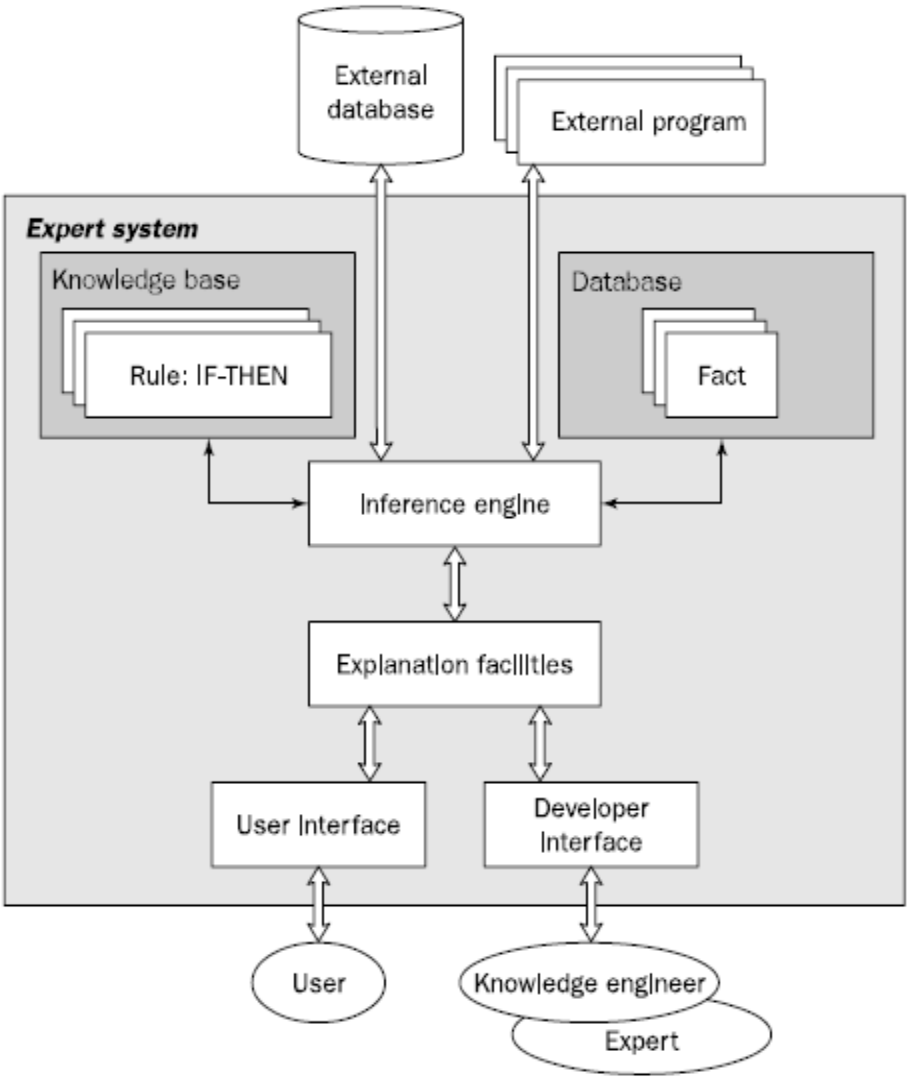

Figure (3): Complete structure of a rule-based expert system

Many expert systems also 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 runtime 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 minimize the intervention of a knowledge engineer.

9. Fundamental characteristics of an expert system

An expert system is built to perform at a human expert level in a narrow, specialized 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.

10. 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

recognize that their judgments 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.

11. 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 recognize 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.

The characteristics of expert systems discussed above make them different from conventional systems and human experts. A comparison is shown in Table (1).

Table (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.

12. 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 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 (4). Letters in the database and the knowledge base are used to represent situations or concepts.

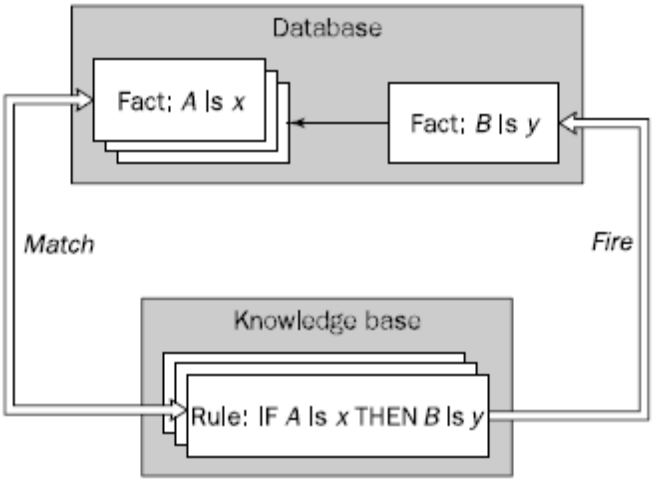

Figure (4): The inference engine cycles via a match-fire procedure

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 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.

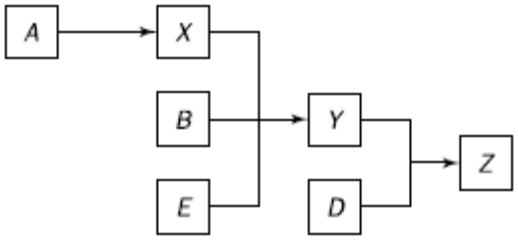

Figure (5): An example of an inference chain

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.

12.1 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 → Z
Rule 2: X & B & E → Y
Rule 3: A → X

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

Rule 4: C → L
Rule 5: L & M → N

Figure (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 → X and Rule 4: C → L, match facts in the database. Rule 3: A → 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 → L is fired and fact L is also placed in the database.

In the second cycle, Rule 2: X&B&E → 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 → 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 → N does not match all facts in the database and thus Rule 5 cannot be fired.

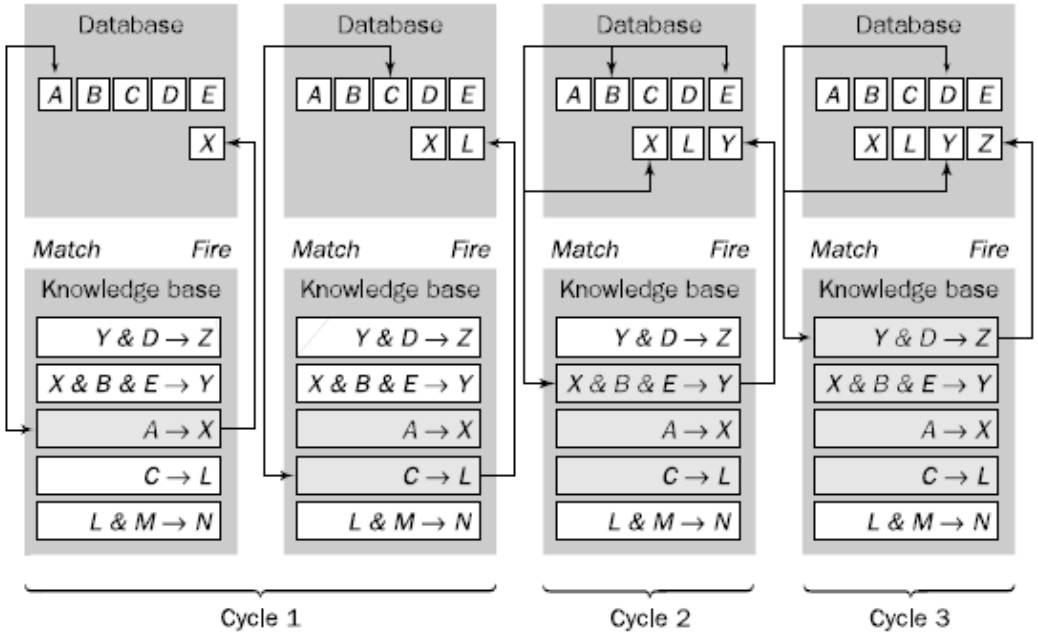

Figure (6): Forward chaining

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 →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.

12.2 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 (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 → 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 → 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 → X. Now it must determine fact A. In Pass 4, the inference engine finds fact A in the database, Rule 3: A → 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 → 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 → 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.

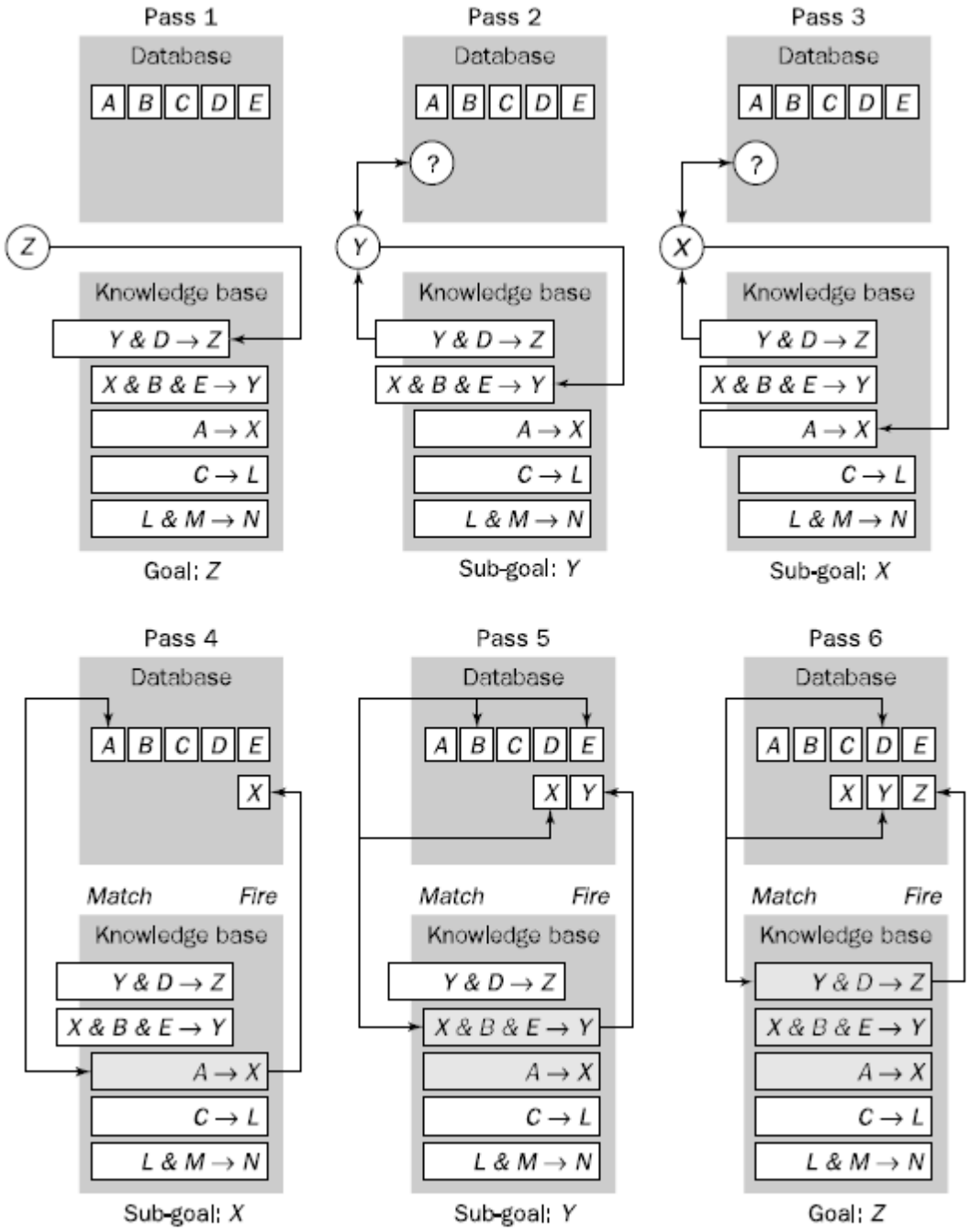

Figure (7): Backward chaining

Let us now compare Figure (6) with Figure (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.

12.3 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, 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, uses backward chaining.

12.4 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.

13. 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:

1- **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.

2- **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.

3- **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.

4- **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 0.1g}. The expert system uses certainty factors to establish the degree of confidence or level of belief that the rule's conclusion is true.

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:

1- **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.

2- **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.

3- **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.