15.1 INTRODUCTION

An expert system is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journal articles, and data bases. This type of knowledge usually requires much training and experience in some specialized field such as medicine, geology, system configuration, or engineering design. Once a sufficient body of expert knowledge has been acquired, it must be encoded in some form, loaded into a knowledge base, then tested, and refined continually throughout the life of the system.

Characteristic Features of Expert Systems

Expert systems differ from conventional computer systems in several important ways.

1. Expert systems use knowledge rather than data to control the solution process. "In the knowledge lies the power" is a theme repeatedly followed and supported throughout this book. Much of the knowledge used is heuristic in nature rather than algorithmic.

2. The knowledge is encoded and maintained as an entity separate from the control program. As such, it is not compiled together with the control program itself. This permits the incremental addition and modification (refinement) of the knowledge base without recompilation of the control programs. Furthermore, it is possible in some cases to use different knowledge bases with the same control programs to produce different types of expert systems. Such systems are known as expert system shells since they may be loaded with different knowledge bases.

3. Expert systems are capable of explaining how a particular conclusion was reached, and why requested information is needed during a consultation. This is important as it gives the user a chance to assess and understand the system's reasoning ability, thereby improving the user's confidence in the system.

4. Expert systems use symbolic representations for knowledge (rules, networks, or frames) and perform their inference through symbolic computations that closely resemble manipulations of natural language. (An exception to this is the expert system based on neural network architectures.)

5. Expert systems often reason with metaknowledge; that is, they reason with knowledge about themselves, and their own knowledge limits and capabilities.

Background History

Expert systems first emerged from the research laboratories of a few leading U.S. universities during the 1960s and 1970s. They were developed as specialized problem

solvers which emphasized the use of knowledge rather than algorithms and general search methods. This approach marked a significant departure from conventional AI systems architectures at the time. The accepted direction of researchers then was to use AI systems that employed general problem solving techniques such as hill-climbing or means-end analysis (Chapter 9) rather than specialized domain knowledge and heuristics. This departure from the norm proved to be a wise choice. It led to the development of a new class of successful systems and special system designs.

The first expert system to be completed was DENDRAL, developed at Stanford University in the late 1960s. This system was capable of determining the structure of chemical compounds given a specification of the compound's constituent elements and mass spectrometry data obtained from samples of the compound. DENDRAL used heuristic knowledge obtained from experienced chemists to help constrain the problem and thereby reduce the search space. During tests, DENDRAL discovered a number of structures previously unknown to expert chemists.

As researchers gained more experience with DENDRAL, they found how difficult it was to elicit expert knowledge from experts. This led to the development of Meta-DENDRAL, a learning component for DENDRAL which was able to learn rules from positive examples, a form of inductive learning described later in detail (Chapters 18 and 19).

Shortly after DENDRAL was completed, the development of MYCIN began at Stanford University. MYCIN is an expert system which diagnoses infectious blood diseases and determines a recommended list of therapies for the patient. As part of the Heuristic Programming Project at Stanford, several projects directly related to MYCIN were also completed including a knowledge acquisition component called THEIRESIUS, a tutorial component called GUIDON, and a shell component called EMYCIN (for Essential MYCIN). EMYCIN was used to build other diagnostic systems including PUFF, a diagnostic expert for pulmonary diseases. EMYCIN also became the design model for several commercial expert system building tools.

MYCIN's performance improved significantly over a several year period as additional knowledge was added. Tests indicate that MYCIN's performance now equals or exceeds that of experienced physicians. The initial MYCIN knowledge base contained about only 200 rules. This number was gradually increased to more than 600 rules by the early 1980s. The added rules significantly improved MYCIN's performance leading to a 65% success record which compared favorably with experienced physicians who demonstrated only an average 60% success rate (Lenat, 1984). (An example of MYCIN's rules is given in Section 4.9, and the treatment of uncertain knowledge by MYCIN is described in Section 6.5.)

Other early expert system projects included PROSPECTOR, a system that assists geologists in the discovery of mineral deposits, and R1 (aka XCON), a system used by the Digital Equipment Corporation to select and configure components of complex computer systems. Since the introduction of these early expert systems, numerous commercial and military versions have been completed with a high degree of success. Some of these application areas are itemized below.

Applications

Since the introduction of these early expert systems, the range and depth of applications has broadened dramatically. Applications can now be found in almost all areas of business and government. They include such areas as

Different types of medical diagnoses (internal medicine, pulmonary diseases, infectious blood diseases, and so on)
Diagnosis of complex electronic and electromechanical systems
Diagnosis of diesel electric locomotion systems
Diagnosis of software development projects
Planning experiments in biology, chemistry, and molecular genetics
Forecasting crop damage
Identification of chemical compound structures and chemical compounds
Location of faults in computer and communications systems
Scheduling of customer orders, job shop production operations, computer resources for operating systems, and various manufacturing tasks
Evaluation of loan applicants for lending institutions
Assessment of geologic structures from dip meter logs
Analysis of structural systems for design or as a result of earthquake damage
The optimal configuration of components to meet given specifications for a complex system (like computers or manufacturing facilities)
Estate planning for minimal taxation and other specified goals
Stock and bond portfolio selection and management
The design of very large scale integration (VLSI) systems
Numerous military applications ranging from battlefield assessment to ocean surveillance
Numerous applications related to space planning and exploration
Numerous areas of law including civil case evaluation, product liability, assault and battery, and general assistance in locating different law precedents
Planning curricula for students
Teaching students specialized tasks (like trouble shooting equipment faults)

Importance of Expert Systems

The value of expert systems was well established by the early 1980s. A number of successful applications had been completed by then and they proved to be cost effective. An example which illustrates this point well is the diagnostic system developed by the Campbell Soup Company.

Campbell Soup uses large sterilizers or cookers to cook soups and other canned

products at eight plants located throughout the country. Some of the larger cookers hold up to 68,000 cans of food for short periods of cooking time. When difficult maintenance problems occur with the cookers, the fault must be found and corrected quickly or the batch of foods being prepared will spoil. Until recently, the company had been depending on a single expert to diagnose and cure the more difficult problems, flying him to the site when necessary. Since this individual will retire in a few years taking his expertise with him, the company decided to develop an expert system to diagnose these difficult problems.

After some months of development with assistance from Texas Instruments, the company developed an expert system which ran on a PC. The system has about 150 rules in its knowledge base with which to diagnose the more complex cooker problems. The system has also been used to provide training to new maintenance personnel. Cloning multiple copies for each of the eight locations cost the company only a few pennies per copy. Furthermore, the system cannot retire, and its performance can continue to be improved with the addition of more rules. It has already proven to be a real asset to the company. Similar cases now abound in many diverse organizations.

15.2 RULE-BASED SYSTEM ARCHITECTURES

The most common form of architecture used in expert and other types of knowledge-based systems is the production system, also called the rule-based system. This type of system uses knowledge encoded in the form of production rules, that is, if . . . then rules. We may remember from Chapter 4 that rules have an antecedent or condition part, the left-hand side, and a conclusion or action part, the right-hand side.

```
IF: Condition-1 and Condition-2 and Condition-3
THEN: Take Action-4

IF: The temperature is greater than 200 degrees, and
    The water level is low
THEN: Open the safety valve.

A & B & C & D → E & F
```

Each rule represents a small chunk of knowledge relating to the given domain of expertise. A number of related rules collectively may correspond to a chain of inferences which lead from some initially known facts to some useful conclusions. When the known facts support the conditions in the rule's left side, the conclusion or action part of the rule is then accepted as known (or at least known with some degree of certainty). Examples of some typical expert system rules were described in earlier sections (for example see Sections 4.9, 6.5, and 10.6).

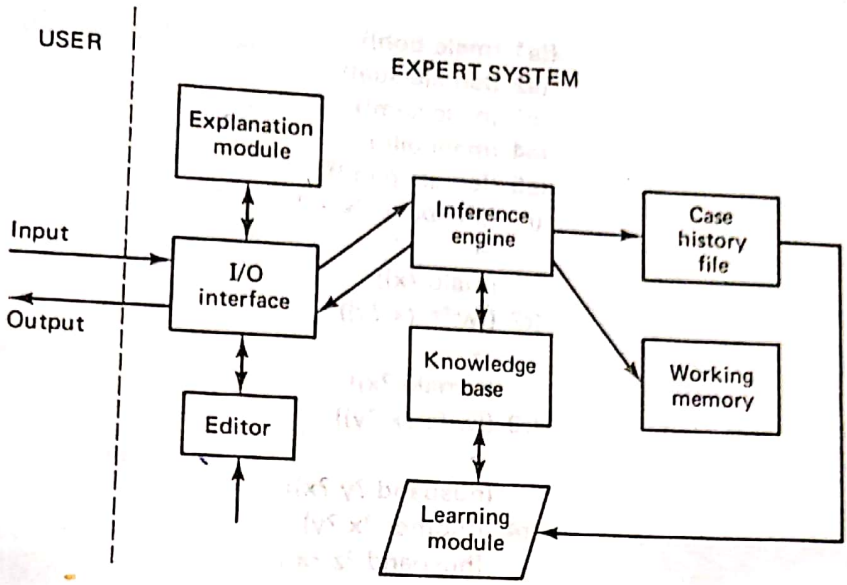

Figure 15.1 Components of a typical expert system.

Inference in production systems is accomplished by a process of chaining through the rules recursively, either in a forward or backward direction, until a conclusion is reached or until failure occurs. The selection of rules used in the chaining process is determined by matching current facts against the domain knowledge or variables in rules and choosing among a candidate set of rules the ones that meet some given criteria, such as specificity. The inference process is typically carried out in an interactive mode with the user providing input parameters needed to complete the rule chaining process.

The main components of a typical expert system are depicted in Figure 15.1. The solid lined boxes in the figure represent components found in most systems whereas the broken lined boxes are found in only a few such systems.

The Knowledge Base

The knowledge base contains facts and rules about some specialized knowledge domain. An example of a simple knowledge base giving family relationships is illustrated in Figure 15.2. The rules in this figure are given in the same LISP format as those of Section 10.6 which is similar to the format given in the OPS5 language as presented by Bronston, Farrell, Kant, and Martin (1985). Each fact and rule is identified with a name ($a1$, $a2$, . . . , $r1$, $r2$, . . .). For ease in reading, the left side is separated from the right by the implication symbol →. Conjuncts on the left are given within single parentheses (sublists), and one or more conclusions may follow the implication symbol. Variables are identified as a symbol preceded by a question mark. It should be noted that rules found in real working systems may have many conjuncts in the LHS. For example, as many as eight or more are not uncommon.

```
((a1 (male bob))
 (a2 (female sue))
 (a3 (male sam))
 (a4 (male bill))
 (a5 (female pam))
 (r1 ((husband ?x ?y))
     →
     (male ?x))
 (r2 ((wife ?x ?y))
     →
     (female ?x))
 (r3 ((wife ?x ?y))
     →
     (husband ?y ?x))
 (r4 ((mother ?x ?y)
      (husband ?z ?x))
     →
     (father ?z ?y))
 (r5 ((father ?x ?y))
      (wife ?z ?x))
     →
     (mother ?z ?y))
 (r6 ((husband ?x ?y))
     →
     (wife ?y ?x))
 (r7 ((father ?x ?z)
      (mother ?y ?z))
     →
     (husband ?x ?y))
 (r8 ((father ?x ?z)
      (mother ?y ?z))
     →
     (wife ?y ?z))
 (r9 ((father ?x ?y)
      (father ?y ?z))
     →
     (grandfather ?x ?z)))
```

Figure 15.2 Facts and rules in a simple knowledge base.

In PROLOG, rules are written naturally as clauses with both a head and body. For example, a rule about a patient's symptoms and the corresponding diagnosis of hepatitis might read in English as the rule

IF: The patient has a chronic disorder, and
the sex of the patient is female, and
the age of the patient is less than 30, and
the patient shows condition A, and
test B reveals biochemistry condition C
THEN: conclude the patient's diagnosis is autoimmune-chronic-hepatitis.

This rule could be written straightaway in PROLOG as

```
conclude(patient, diagnosis, autoimmune_chronic_hepatitis):-
        same(patient, disorder, chronic),
        same(patient, sex, female),
        lessthan(patient, age, 30),
        same(patient, symptom_a, value_a),
        same(patient, biochemistry, value_c).
```

Note that PROLOG rules have at most one conclusion clause.

The Inference Process

The inference engine accepts user input queries and responses to questions through the I/O interface and uses this dynamic information together with the static knowledge (the rules and facts) stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case or situation as presented by the user's input.

The inferring process is carried out recursively in three stages: (1) match, (2) select, and (3) execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When consistent matches are found, the corresponding rules are placed in a conflict set. To find an appropriate and consistent match, substitutions (instantiations) may be required. Once all the matched rules have been added to the conflict set during a given cycle, one of the rules is selected for execution. The criteria for selection may be most recent use, rule condition specificity, (the number of conjuncts on the left), or simply the smallest rule number. The selected rule is then executed and the right-hand side or action part of the rule is then carried out. Figure 15.3 illustrates this match-select-execute cycle.

As an example, suppose the working memory contains the two clauses

```
(father bob sam)
(mother sue sam)
```

When the match part of the cycle is attempted, a consistent match will be made between these two clauses and rules r7 and r8 in the knowledge base. The match is made by substituting Bob for ?x, Sam for ?z, and Sue for ?y. Consequently, since all the conditions on the left of both r7 and r8 are satisfied, these two rules will be placed in the conflict set. If there are no other working memory clauses to match, the selection step is executed next. Suppose, for one or more of the selection criteria stated above, r7 is the rule chosen to execute. The clause on the right side of r7 is instantiated and the execution step is initiated. The execution step may result in the right-hand clause (husband bob sue) being placed in working memory or it may be used to trigger a message to the user. Following the execution step, the match-select-execute cycle is repeated.

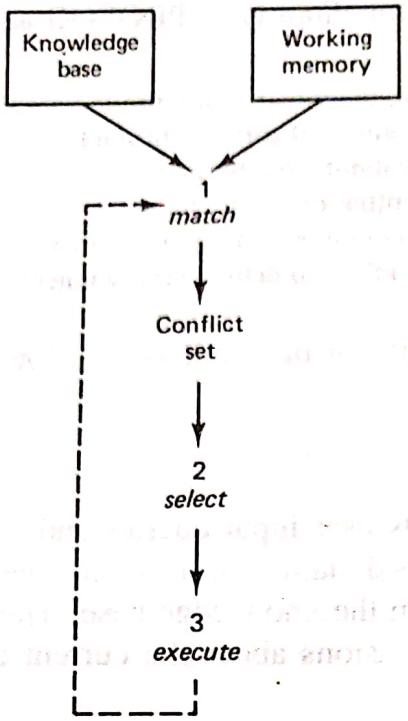

Figure 15.3 The production system inference cycle.

As another example of matching, suppose the two facts (a6 (father sam bill)) and (a7 (father bill pam)) have been added to the knowledge base and the immediate goal is a query about Pam's grandfather. When made, assume this query has resulted in placement of the clause (grandfather ?x pam) into working memory. For this goal to succeed, consistent substitutions must be made for the variables ?*x* and ?*y* in rule r9 with a6 and a7. This will be the case if Sam and Bill are substituted for ?*x* and ?*y* in the subgoal left-hand conditions of r9. The right hand side will then correctly state that Pam's grandfather is Sam.

When the left side of a sequence of rules is instantiated first and the rules are executed from left to right, the process is called forward chaining. This is also known as data-driven inference since input data are used to guide the direction of the inference process. For example, we can chain forward to show that when a student is encouraged, is healthy, and has goals, the student will succeed.

ENCOURAGED(student) → MOTIVATED(student)
MOTIVATED(student) & HEALTHY(student) → WORKHARD(student)
WORKHARD(student) & HASGOALS(student) → EXCELL(student)
EXCELL(student) → SUCCEED(student)

On the other hand, when the right side of the rules is instantiated first, the left-hand conditions become subgoals. These subgoals may in turn cause sub-subgoals to be established, and so on until facts are found to match the lowest subgoal conditions. When this form of inference takes place, we say that backward chaining is performed. This form of inference is also known as goal-driven inference since an initial goal establishes the backward direction of the inferring.

For example, in MYCIN the initial goal in a consultation is "Does the patient have a certain disease?" This causes subgoals to be established such as "are certain bacteria present in the patient?" Determining if certain bacteria are present may require such things as tests on cultures taken from the patient. This process of setting up subgoals to confirm a goal continues until all the subgoals are eventually satisfied or fail. If satisfied, the backward chain is established thereby confirming the main goal.

When rules are executed, the resulting action may be the placement of some new facts in working memory, a request for additional information from the user, or simply the stopping of the search process. If the appropriate knowledge has been stored in the knowledge base and all required parameter values have been provided by the user, conclusions will be found and will be reported to the user. The chaining continues as long as new matches can be found between clauses in the working memory and rules in the knowledge base. The process stops when no new rules can be placed in the conflict set.

Some systems use both forward and backward chaining, depending on the type of problem and the information available. Likewise, rules may be tested exhaustively or selectively, depending on the control structure. In MYCIN, rules in the KB are tested exhaustively. However, when the number of rules exceeds a few hundred, this can result in an intolerable amount of searching and matching. In such cases, techniques such as those found in the RETE algorithm (Chapter 10) may be used to limit the search.

Many expert systems must deal with uncertain information. This will be the case when the evidence supporting a conclusion is vague, incomplete, or otherwise uncertain. To accommodate uncertainties, some form of probabilities, certainty factors, fuzzy logic, heuristics, or other methods must be introduced into the inference process. These methods were introduced in Chapters 5 and 6. The reader is urged at this time to review those methods to see how they may be applied to expert systems.

Explaining How or Why

The explanation module provides the user with an explanation of the reasoning process when requested. This is done in response to a how query or a why query.

To respond to a how query, the explanation module traces the chain of rules fired during a consultation with the user. The sequence of rules that led to the conclusion is then printed for the user in an easy to understand human-language style. This permits the user to actually see the reasoning process followed by the system in arriving at the conclusion. If the user does not agree with the reasoning steps presented, they may be changed using the editor.

To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process before it can proceed. For example, in diagnosing a car that will not start, a system might be asked why it needs to know the status of the

distributor spark. In response, the system would reply that it needs this information to determine if the problem can be isolated to the ignition system. Again, this information allows the user to determine if the system's reasoning steps appear to be sound. The explanation module programs give the user the important ability to follow the inferencing steps at any time during the consultation.

Building a Knowledge Base

The editor is used by developers to create new rules for addition to the knowledge base, to delete outmoded rules, or to modify existing rules in some way. Some of the more sophisticated expert system editors provide the user with features not found in typical text editors, such as the ability to perform some types of consistency tests for newly created rules, to add missing conditions to a rule, or to reformat a newly created rule. Such systems also prompt the user for missing information, and provide other general guidance in the KB creation process.

One of the most difficult tasks in creating and maintaining production systems is the building and maintaining of a consistent but complete set of rules. This should be done without adding redundant or unnecessary rules. Building a knowledge base requires careful planning, accounting, and organization of the knowledge structures. It also requires thorough validation and verification of the completed knowledge base, operations which have yet to be perfected. An "intelligent" editor can greatly simplify the process of building a knowledge base.

TEIRESIAS (Davis, 1982) is an example of an intelligent editor developed to assist users in building a knowledge base directly without the need for an intermediary knowledge engineer. TEIRESIUS was developed to work with systems like MYCIN in providing a direct user-to-system dialog. TEIRESIUS assists the user in formulating, checking, and modifying rules for inclusion in the performance program's knowledge base. For this, TEIRESIUS uses some metaknowledge, that is, knowledge about MYCIN's knowledge. The dialog is carried out in a near English form so that the user needs to know little about the internal form of the rules.

The I/O Interface

The input-output interface permits the user to communicate with the system in a more natural way by permitting the use of simple selection menus or the use of a restricted language which is close to a natural language. This means that the system must have special prompts or a specialized vocabulary which encompasses the terminology of the given domain of expertise. For example, MYCIN can recognize many medical terms in addition to various common words needed to communicate. For this, MYCIN has a vocabulary of some 2000 words.

Personal Consultant Plus, a commercial PC version of the MYCIN architecture, uses menus and English prompts to communicate with the user. The prompts, written in standard English, are provided by the developer during the system building stage. How and why explanations are also given in natural language form.