over it doesn't matter if it is getting gas. Rule 4 asks the user to check for gas in the fuel tank before asking that the user open up the carburetor and look there. It is performing the easier check first.

In addition to the ordering of a rule's premises, the content of a rule itself may be fundamentally heuristic in nature. In the automotive example, all the rules are heuristic; consequently, the system may obtain erroneous results. For example, if the engine is getting gas and turning over, the problem may be a bad distributor rather than bad spark plugs. In the next section, we examine this problem and some ways of dealing with it.

REQUIRED READING IN EGR 434

8.3 Using Uncertainty Measures in Expert Systems

8.3.1 Introduction

Until Section 8.2, our inference procedures followed the model presented with the predicate calculus: from correct premises sound inference rules produce new, guaranteed correct, conclusions. In expert systems, we must often attempt to draw correct conclusions from poorly formed and uncertain evidence using unsound inference rules.

This is not an impossible task; Humans do it successfully in almost every aspect of our daily survival. We deliver correct medical treatment for ambiguous symptoms; we mine natural resources with little or no guarantee of success before we start; we comprehend language statements that are often ambiguous or incomplete, and so on.

The reasons for this ambiguity may be better understood by referring once again to our automotive expert system example. Consider rule 2:

```
if
    the engine does not turn over, and
    the lights do not come on
then
    the problem is battery or cables.
```

This rule is heuristic in nature; it is possible (although less likely) that the battery and cables are fine but that the car simply has a bad starter motor and burned-out headlights. This rule seems to resemble a logical implication, but it is not: failure of the engine to turn over and the lights to come on does not necessarily imply that the battery and cables are bad. What is interesting to note, however, is that the *converse* of the rule is a true implication:

```
if
    the problem is battery or cables
then
    the engine does not turn over, and
    the lights do not come on.
```

Barring the supernatural, a car with a dead battery will not light its headlamps or turn the starter motor.

This is an example of abductive reasoning. Formally, abduction states that from P -> Q and Q it is possible to infer P.

Abduction is an *unsound* rule of inference, meaning that the conclusion is not necessarily true for every interpretation in which the premises are true. For example, if someone says "If it rains then I will not go running at 3:00" and you do not see that person on the track at 3:00, does it necessarily follow that it is raining? It is possible that the individual decided not to go running because of an injury or that he needed to work late, etc.

Although abduction is unsound, it is often essential to solving problems. The "correct" version of the battery rule is not particularly useful in diagnosing car troubles since its premise (bad battery) is our goal and its conclusions are the observable symptoms we must work with. Modus ponens cannot be applied and the rule must be used in an abductive fashion. This is generally true of diagnostic (and other) expert systems. Faults or diseases cause (imply) symptoms, not the other way around, but diagnosis must work from the symptoms back to the cause.

Uncertainty results from the use of abductive inference as well as from attempts to reason with missing or unreliable data. To get around this problem, we can attach some measure of confidence to the conclusions. For example, although battery failure does not always accompany the failure of a car's lights and starters, it almost always does, and confidence in this rule is justifiably high.

Note that there are problems that do not require certainty measures. When configuring a computer, for instance, the components either go together or they do not. The idea that "a particular disk drive and bus go together with certainty 0.75" does not even make sense. Similarly, if MACSYMA is attempting to find the integral of a function, a confidence of "0.6" that a result is correct is not useful. These programs may be either data driven (Digital's XCON) or goal driven (MIT's MACSYMA), but because they do not require abductive rules of inference or do not deal with unreliable data they do not require the use of confidence measures.

In this section we discuss several ways of managing the uncertainty that results from heuristic rules: first, the Bayesian approach (8.3.2), and second, the Stanford certainty theory (8.3.3). Finally, we briefly consider Zadeh's *fuzzy set theory*, the Dempster/Shafer theory of evidential reasoning, and nonmonotonic reasoning.

8.3.2 Bayesian Probability Theory

The Bayesian approach to uncertainty is based in formal probability theory and has shown up in several areas of AI research, including pattern recognition and classification problems. The PROSPECTOR expert system, built at Stanford and SRI International and employed in mineral exploration (copper, molybdenum, and others), also uses a form of the Bayesian statistical model.

Assuming random distribution of events, probability theory allows the calculation of more complex probabilities from previously known results. In simple probability calculations we are able to conclude, for example, how cards might be distributed to a number of players.

Suppose that I am one person of a four-person card game where all the cards are equally distributed. If I do not have the queen of spades I can conclude that each of the others players has it with probability 1/3. Similarly, I can ALSO conclude that each player has the ace of hearts with probability 1/3 and that any one player has both cards at 1/3 * 1/3 or 1/9.

In the mathematical theory of probability, individual probability instances are worked out by sampling and combinations of probabilities are worked out as above, using a rule such as:

$$\boxed{\text{probability(A and B)} = \text{probability(A)} * \text{probability(B)}}$$

given that A and B are independent events.

[illegible]

One of the most important results of probability theory is Bayes's theorem. Bayes's results provide a way of computing the probability of a hypothesis following from a particular piece of evidence, given only the probabilities with which the evidence follows from actual causes (hypotheses).

Bayes's theorem states:

$$\boxed{p(H_i \mid E) = \frac{P(E \mid H_i) * P(H_i)}{\sum_{k=1}^{n} (P(E \mid H_k) * P(H_k))}}$$

where:

$P(H_i \mid E)$ is the probability that H_i is true given evidence E.
$P(H_i)$ is the probability that H_i is true overall.
$P(E \mid H_i)$ is the probability of observing evidence E when H_i is true.
n is the number of possible hypotheses.

BUT "CERTAINTY" IS DIFFERENT →

EX COPPER MINING

$P(H_i)$ = PROB. OF FINDING COPPER ANYWHERE

$P(H|E)$ = PROB OF FINDING COPPER WHEN THERE IS EVIDENCE THAT IT MIGHT BE PRESENT

$P(E|H_i)$ = PROB OF FINDING EVIDENCE THAT THERE MAY BE COPPER WHEN THERE DEFINITELY IS COPPER (i.e., H_i is TRUE)

Suppose we desire to examine the geological evidence at some location to see if the location is suited to finding copper. We must know in advance the probability of finding each of a set of minerals and the probability of certain evidence being present when each particular mineral is found. Then we can use Bayes's theorem to determine the likelihood that copper will be present using the evidence we collect at the location. This is the approach taken in the PROSPECTOR program, which has found commercially significant mineral deposits (Duda et al. 1979*a*) at several sites.

$P(H_k)$

$P(E|H_k)$

There are two major assumptions for the use of Bayes's theorem: first that all the statistical data on the relationships of the evidence with the various hypotheses are known; second, and more difficult to establish, that all relationships between evidence and hypotheses, or $P(E \mid H_k)$, are independent. Actually, this assumption of independence can be a quite tricky matter, especially when many assumptions of independence are needed to establish the validity of this approach across many rule applications. This represents the entire collected probabilities on the evidence given various hypothesis relationships in the denominator of Bayes's theorem. In general, and especially in areas like medicine, this assumption of independence cannot be justified.

A final problem, which makes keeping the statistics of the "evidence given hypotheses" relationships virtually intractable, is the need to rebuild all probability relationships when any new relationship of hypothesis to evidence is discovered. In many active research areas (again, like medicine) this is happening continuously. Bayesian reasoning requires complete and up-to-date probabilities if its conclusions are to be correct. In many domains, such extensive data collection and verification are not possible.

Where these assumptions are met, Bayesian approaches offer the benefit of a well-founded and statistically correct handling of uncertainty. Most expert system domains do not meet these requirements and must rely on more heuristic approaches. It is also felt that the human expert does not use the Bayesian model in successful problem solving. In the next section we describe certainty theory, a heuristic approach to the management of uncertainty.

8.3.3 A Theory for Certainty

DERIVED FROM, BUT A LITTLE DIFFERENT THAN PROBABILITY THEORY

Several early expert system projects (besides PROSPECTOR) attempted to adapt Bayes's theorem to their problem-solving needs. The independence assumptions, continuous updates of statistical data, and other problems mentioned in Section 8.3.2 gradually led these researchers to search for other measures of "confidence." Probably the most important alternative approach was used at Stanford in developing the MYCIN program (Buchanan and Shortliff 1984).

Certainty theory is based on a number of observations. The first is that in traditional probability theory the sum of confidence for a relationship and confidence against the same relationship must add to 1. However, it is often the case that an expert might have confidence 0.7 (say) that some relationship is true and have no feeling about it being not true.

Another assumption that underpins certainty theory is that the knowledge content of the rules is much more important than the algebra of confidences that holds the system together. Confidence measures correspond to the informal evaluations that human experts attach to their conclusions, e.g., "it is probably true" or "it is highly unlikely."

Certainty theory makes some simple assumptions for creating confidence measures and has some equally simple rules for combining these confidences as the program moves toward its conclusion. The first assumption is to split "confidence for" from "confidence against" a relationship:

Call MB(H|E) the measure of belief of a hypothesis H given evidence E.

Call MD(H|E) the measure of disbelief of a hypothesis H given evidence E.

Now either:

$$1 > MB(H|E) > 0 \text{ while } MD(H|E) = 0, \quad \text{or}$$

$$1 > MD(H|E) > 0 \text{ while } MB(H|E) = 0.$$

The two measures constrain each other in that a given piece of evidence is either for or against a particular hypothesis. This is an important difference between certainty theory and probability theory. Once the link between measures of belief and disbelief has been established, they may be tied together again with the certainty factor calculation:

CERTAINTY FACTOR

$$\boxed{CF(H|E) = MB(H|E) - MD(H|E).}$$

As the certainty factor (CF) approaches 1 the evidence is stronger for a hypothesis; as

[illegible]

FOR HELP [illegible]

approaches −1 the confidence against the hypothesis gets stronger; and a CF around 0 indicates that there is little evidence either for or against the hypothesis.

When experts put together the rule base they must agree on a CF to go with each rule. This CF reflects their confidence in the rule's reliability. Certainty measures may be adjusted to tune the system's performance, although slight variations in this confidence measure tend to have little effect on the overall running of the system (again, "the knowledge gives the power").

The premises for each rule are formed of the and and or of a number of facts. When a production rule is used, the certainty factors that are associated with each condition of the premise are combined to produce a certainty measure for the overall premise in the following manner.

For P1 and P2, premises of the rule,

LAW #A
LAW #B

CF(P1 and P2) = MIN(CF(P1), CF(P2)), and

CF(P1 or P2) = MAX(CF(P1), CF(P2)).

The combined CF of the premises, using the above combining rules, is then multiplied by the CF of the rule to get the CF for the conclusions of the rule.

EX

~~For example, consider the rule in a~~ knowledge base: IF

(P1 and P2) or P3 → R1 (.7) and R2 (.3)

WILL THEN PRINT

ie 2 possible answers

ie, results

where P1, P2, and P3 are premises and R1 and R2 are the conclusions of the rules having CFs 0.7 and 0.3, respectively. These numbers are added to the rule when it is designed and represent the expert's confidence in the conclusion if all the premises are known with complete certainty. If the running program has produced P1, P2, and P3 with CFs of 0.6, 0.4, and 0.2, respectively, then R1 and R2 may be added to the collected case-specific results with CFs 0.28 and 0.12, respectively.

Here are the calculations for this example:

CF(P1(.6) and P2(.4)) = MIN(.6,.4) = .4.

CF((.4) or P3(.2)) = MAX(.4,.2) = .4.

The CF for R1 is .7 in the rule, so R1 is added to the set of true facts with the associated CF of (.7) * (.4) = .28.

The CF for R2 is .3 in the rule, so R2 is added to the set of true facts with the associated CF of (.3) * (.4) = .12.

One further measure is required: how to combine multiple CFs when two or more rules support the same result R. This is the certainty theory analog of the probability theory procedure of multiplying the probability measures to combine independent evidence. By using this rule repeatedly one can combine the results of any number of rules that are used for determining result R. Suppose CFR1 is the present certainty factor associated with result R, and a previously unused rule produces result R (again) with CFR2; then the new CF of R is calculated by:

LAW # C

CF(R) = CF(R1) + CF(R2) − (CF(R1) * CF(R2)) ~~when CF(R1) and CF(R2) are positive~~

~~CF(R) = CF(R1) + CF(R2) + (CF(R1) * CF(R2)) when CF(R1) and CF(R2) are negative~~

LOOK AT THIS ALSO

$$\mathrm{CF(C)} = \frac{\mathrm{CF(R1)} + \mathrm{CF(R2)}}{1 - \mathrm{MIN}(|\mathrm{CF(R1)}|, |\mathrm{CF(R2)}|)} \quad \text{otherwise}$$ ← LOOK AT THIS ALSO!

where |X| is the absolute value of X.

Besides being easy to compute, these equations have other desirable properties. First, the CFs that result from applying this rule are always between 1 and −1, as are the other CFs. Second, the result of combining contradictory CFs is that they cancel each other out, as would be desired. Finally, the combined CF measure is a monotonically increasing (decreasing) function in the manner one would expect for combining evidence.

Certainty theory has been criticized as being excessively ad hoc. Although it is defined in a formal algebra, the meaning of the certainty measures is not as rigorously founded as in formal probability theory. However, certainty theory does not attempt to produce an algebra for "correct" reasoning. Rather it is the "lubrication" that lets the expert system combine confidences as it moves along through the problem at hand. Its measures are ad hoc in the same sense that a human expert's confidence in his or her results is approximate, heuristic, and informal. In Section 8.4, when MYCIN is considered, the CFs will be used in the heuristic search to give a priority for goals to be attempted and a cutoff point when a goal need not be considered further. But even though the CF is used to keep the program running and collecting information, the power of the program is in the content of the rules themselves. This is the justification for the weakness of the certainty algebra.

NOTE: "UP-FRONT" S.W. INPUT CF RANGE USES 0 TO 100 INSTEAD OF 0 TO 1 OR −1 TO 1 JUST ASSUME ALL POSITIVE.

8.3.4 Other Approaches to Uncertainty

Because of the importance of uncertain reasoning to expert level problem solving and the limitations of certainty theory, work continues in this important area. In concluding this subsection, we mention briefly three other approaches to modeling uncertainty: Zadeh's *fuzzy set theory*, the Dempster/Shafer *theory of evidence*, and *nonmonotonic reasoning*.

Zadeh's main contention (Zadeh 1983) is that, although probability theory is appropriate for measuring randomness of information, it is inappropriate for measuring the *meaning* of information. Indeed, much of the confusion surrounding the use of English words and phrases is related to lack of clarity (vagueness) rather than randomness. This is a crucial point for analyzing language structures and can also be important in creating a measure of confidence in production rules. Zadeh proposes *possibility theory* as a measure of vagueness, just as probability theory measures randomness.

Zadeh's theory expresses lack of precision in a quantitative fashion by introducing a set membership function that can take on real values between 0 and 1. This is the notion of a *fuzzy set* and can be described as follows: let S be a set and s a member of that set. A fuzzy subset F of S is defined by a membership function mF(s) that measures the "degree" to which s belongs to F.

A standard example of the fuzzy set is for S to be the set of positive integers and F to be the fuzzy subset of S called "small integers." Now various integer values can have a "possibility" distribution defining their "fuzzy membership" in the set of small integers: mF(1) = 1, mF(2) = 1, mF(3) = 0.9, mF(4) = 0.8, . . . , mF(50) = 0.001, etc. For

END

the statement that positive integer X is a "small integer," mF creates a possibility distribution across all the positive integers (S).

Fuzzy set theory is not concerned with how these possibility distributions are created, but rather with the rules for computing the combined possibilities over expressions that each contain fuzzy variables. Thus it includes rules for combining possibility measures for expressions containing fuzzy variables. The laws for the or, and, and not of these expressions are similar to those just presented for certainty factors. In fact, the approach at Stanford was modeled on some of the combination rules described by Zadeh (Buchanan and Shortliffe 1984).

Dempster and Schafer approach the problem of measuring certainty by asking us to make a fundamental distinction between uncertainty and ignorance. In probability theory we are *forced* to express the extent of our knowledge about a belief X in a single number P(X). The problem with this (say Dempster and Shafer) is that we simply cannot always know the values of prior probabilities and thus any particular choice of P(X) may not be justified.

The Dempster/Schafer approach recognizes the distinction between uncertainty and ignorance by creating "belief functions." These belief functions satisfy axioms that are weaker than those of probability theory. Thus probability theory is seen as a subclass of belief functions, and the theory of evidence can reduce to probability theory when all the probabilities are obtainable. Belief functions therefore allow us to use our knowledge to bound the assignment of probabilities to events without having to come up with exact probabilities, when these may be unavailable.

Even though the Dempster/Shafer approach gives us methods of computing these various belief parameters, their greater complexity adds to the computational cost as well. Besides, any theory that has probability theory as a special case is plagued by the assumptions that have made probability theory already quite difficult to use. Conclusions using beliefs, even though they avoid commitment to a stronger (and often unjustified) assignment of probability, produce conclusions that are necessarily weaker. But, as the Dempster/Shafer model points out quite correctly, the stronger conclusion may not be justified.

All of the methods we have examined can be criticized for using numeric approaches to the handling of uncertain reasoning. It is unlikely that humans use any of these techniques for reasoning with uncertainty, and many applications seem to require a more qualitative approach to the problem. For example, numeric approaches do not support adequate explanations of the causes of uncertainty. If we ask human experts why their conclusions are uncertain, they can answer in terms of the qualitative relationships between features of the problem instance. In a numeric model of uncertainty, this information is replaced by numeric measures. Similarly, numeric approaches do not address the problem of changing data. What should the system do if a piece of uncertain information is later found to be true or false?

Nonmonotonic reasoning addresses these problems directly. A nonmonotonic reasoning system handles uncertainty by making the most reasonable assumptions in light of uncertain information. It proceeds with its reasoning as if these assumptions were true. At a later time, it may find out that an assumption was erroneous, either through a change in problem data or by discovering that the assumption led to an impossible conclusion. When this occurs, the system must change both the assumption and all of the conclusions that depend on it.

Nonmonotonicity is an important feature of human problem solving and commonsense reasoning. When we drive to work, for example, we make numerous assumptions about the roads and traffic. If we find that one of these assumptions is violated, perhaps by construction or an accident on our usual route, we change our plans and devise an alternative route to work.

Nonmonotonic reasoning contrasts sharply with the inference strategies we have discussed so far in the text. These strategies assume that axioms do not change and the conclusions drawn from them remain true. They are called *monotonic* reasoning systems because knowledge can only be added through the reasoning process. Since information may also be retracted in a nonmonotonic reasoning system when something that was previously thought to be true is later shown to be false, it is important to record the justification for all new knowledge. When an assumed fact is withdrawn, all conclusions that depend on that fact must be reexamined and possibly withdrawn as well. This record keeping is the task of *truth maintenance systems* (Doyle 1979; de Kleer 1986).

8.4 MYCIN: A Case Study

8.4.1 Introduction

Although our small automotive diagnostic example is useful in presenting some of the concepts of expert system design, it is a toy system and ignores much of the complexity encountered in building large knowledge-based programs. For this reason, we end this chapter with a case study of an actual working expert system.

The MYCIN project was a cooperative venture by the Department of Computer Science and the Medical School at Stanford University. The main work on the project was done during the middle and late 1970s, and about 50 person-years were expended in the effort. MYCIN was written in INTERLISP, a dialect of the LISP programming language. One of the earliest expert systems to be proposed and designed, it has become an important classic in the field. Indeed, it is often presented as the archetype for the rule-based program, with many commercial systems (e.g., TI's Personal Consultant) essentially duplicating the MYCIN approach. One of the main reasons for the influence of the MYCIN program is the extensive documentation of the project by the Stanford research teams (Buchanan and Shortliffe 1984).

MYCIN was designed to solve the problem of diagnosing and recommending treatment for meningitis and bacteremia (blood infections). This particular domain was chosen largely because the program architects wanted to explore the way in which human experts reason with missing and incomplete information. Although there are diagnostic tools that can unambiguously determine the identity of the infecting organisms in a case of meningitis, these tools require on the order of 48 hours (chiefly to grow a culture of the infecting organisms) to return a diagnosis. Meningitis patients, however, are very sick, and some treatment must begin immediately. Because of this need, doctors have developed considerable expertise, based on initial symptoms and test results, for forming a diagnosis that *covers* (i.e., includes as a subset) the actual infecting organisms. Treatment begins with this diagnosis and is refined when more conclusive information becomes available. Thus, the domain of meningitis diagnosis provided a natural focus for studying how humans solve problems using incomplete or unreliable information.

Another goal of the MYCIN design team was to pattern the behavior of the program after the way in which human physicians interact in actual consultations. This was seen as an important factor in system acceptability, particularly since medical consultations tend to follow a standard set of protocols.

In the next subsections we examine the composition of the MYCIN rule and fact descriptions, including the syntax of MYCIN rules. Next, we present a trace of a MYCIN consultation, along with explanatory comments and a discussion of the design and structure of the dialogue. We then discuss the problem of evaluating expert systems and, finally, demonstrate the use of an experimental knowledge base editor, *Teiresias*. Editors such as Teiresias assist the domain experts (in this case the doctors) in correcting the MYCIN knowledge base without the need for the computer language expert. Although such tools are still mainly experimental, this is an important area of research and a potential key to increasing the range of applicability of expert system technology.

8.4.2 Representation of Rules and Facts

Facts in MYCIN are represented as *attribute-object-value triples*. The first element of this structure is an attribute of an object in the problem domain. For example, we may wish to describe the identity of a disease organism or its sensitivity to certain drugs. The name of the object and the value of the attribute follow. Since information in this domain may be uncertain, a certainty factor is associated with MYCIN facts. Recall from Section 8.3.3 that this certainty factor will be between 1 and -1, with 1 being certain, -1 indicating certainty that the attribute is not true, and values about 0 indicating that little is known.

For example, MYCIN facts may be represented in English by:

There is evidence (.25) that the identity of the organism is Klebsiella.

and

It is known that the organism is not sensitive to penicillin.

These may be translated into the LISP s-expressions:

```
(ident organism_1 klebsiella .25)
```

and

```
(sensitive organism_1 penicillin -1.0)
```

We next present an example MYCIN rule both in English and in its LISP equivalent. This sample rule is a condition-action pair in which the premise (condition) is the conjunction of three facts (attribute-object-value triples) and the action adds a new fact to the set of things known about the patient, the identity of a particular organism that is added to the "cover set" of possible infecting organisms. Because this is an abductive rule of inference (inferring the cause from the evidence), it has an attached certainty factor.

In English:

```
if (a)  the infection is primary-bacteremia, and
   (b)  the site of the culture is one of the sterile sites, and
   (c)  the suspected portal of entry is the gastrointestinal tract
then    there is suggestive evidence (.7) that infection is bacteroid
```

is rendered for MYCIN:

```
IF:  (AND (same_context infection primary_bacteremia)
          (membf_context site sterilesite)
          (same_context portal GI))

THEN:  (conclude context_ident bacteroid tally .7).
```

Syntactically, attribute-object-valve (A-O-V) triples are essentially a restriction of predicate calculus expressions to binary predicates. This restriction is not particularly limiting, since algorithms exist for translating predicates of any arity into a set of binary predicates.

There is a deeper difference between predicate calculus and the A-O-V triples used in MYCIN: predicates may be either true or false. Predicate calculus uses sound rules of inference to determine the truth value of conclusions. In MYCIN, a fact has an attached confidence rather than a truth value. When the system is deriving new facts, it fails to fire any rule whose premise has a certainty value of less than 0.2. This contrasts with logic, which causes a rule to fail if its premise is found to be false.

This comparison of attribute-object-value triples and predicate calculus expressions is included here to emphasize both the similarities and the trade-offs involved in different knowledge representations. At the highest conceptual level, the characteristics that define a rule-based expert system are independent of any commitment to a particular representational format. These characteristics have been discussed in the comparison of rule-based expert systems and the production system model of problem solving. At the level of an implementation, however, the selection of a particular representational formalism does involve a number of important trade-offs. These issues include the clarity of the representation, expressiveness (what knowledge can the formalism effectively capture?), naturalness of expression, and ease of implementing and verifying the system.

The action of a MYCIN rule can perform a number of tasks once the rule is satisfied. It may add new information about a particular patient to the working memory. It may also write information to the terminal (or other output device), change the value of an attribute or its certainty measure, look up information in a table (where this representation is more efficient), or execute any piece of LISP code.

Although the ability to execute arbitrary LISP code gives unlimited added power, it should be used judiciously, since excessive use of escapes to the underlying language can lose many of the benefits of the production system formalism. (As an absurd example of this, imagine an expert system with only one rule, whose action is to call a large and poorly structured LISP program that attempts to solve the problem!)

MYCIN rule and fact descriptions, like all knowledge representations, are a formal language and have a formal syntactic definition. This formal syntax is essential to the definition of well-behaved inference procedures. Furthermore, the formal syntax of rules can help a knowledge base editor, such as Teiresias, determine when the domain expert has produced an incorrectly formed rule and prevent that rule from entering the knowledge base. Teiresias can detect syntactic errors in a rule and, to an impressive degree, automatically remedy the situation (see Section 8.4.5). This is possible only when the entire program is built on a set of formal specifications.

The syntax of MYCIN rules is given below. It is stated in *BNF* or *Backus-Naur form* (Pratt 1984). BNF is a form of context-free grammar that is widely used to define programming languages. In the notation, keywords in the language are written in uppercase letters. These are terminals in the syntax. Nonterminals are enclosed in angle brackets: <>. The BNF grammar is a collection of rewrite rules; the nonterminal figure to the left of the ::= can be replaced with the expression on the right. If we begin with the <rule> nonterminal, anything that can be produced through a series of legal substitutions is a legal MYCIN rule.

```
<rule> ::= (IF <antecedent> THEN <action> [ELSE <action>])
<antecedent> ::= (AND {<condition>}+)
<condition> ::= (OR {<condition>}+ | <predicate> <associative-triple>)
<associative-triple> ::= (<attribute><object><value>)
<action> ::= {<consequent>}+ | {<procedure>}+
<consequent> ::= (<associative-triple> <certainty-factor>)
<certainty-factor> ranges from -1 (false) to +1 (true).
<predicate> is any LISP predicate.
<procedure> is any piece of LISP code.
```

8.4.3 MYCIN Diagnosing an Illness

MYCIN is a goal-driven expert system. Its main action is to try to determine whether a particular organism may be present in the meningitis infection in the patient. A possible infecting organism forms a goal that is either confirmed or eliminated. It is interesting to note that the decision to make MYCIN a goal-driven system was not based exclusively on the effectiveness of the strategy in pruning the search space. Early in the design of the system, the MYCIN architects considered the use of forward search. This was rejected because the questions the system asked the user seemed random and unconnected. Goal-driven search, because it is attempting to either confirm or eliminate a particular hypothesis, causes the reasoning to seem more focused and logical. This builds the user's understanding and confidence in the system's actions. An expert system needs to do more than obtain a correct answer; it must also do so in an intelligent and understandable fashion. This criterion influences the choice of search strategies as well as the structure and order of rules.

In keeping with this goal of making the program behave like a doctor, MYCIN's designers noted that doctors ask routine questions at the beginning of a consultation (e.g., "how old are you?" and "have you had any childhood diseases?") and ask more specific questions when they are needed. Collecting the general information at the beginning makes the session seem more focused in that it is not continually interrupted by trivial questions such as the name, age, sex, and race of the patient. Certain other questions are asked at other well-defined stages of a consultation (such as when beginning to consider a new hypothesis).

Eventually the questions get more specific (questions 16 and 17 in the trace below) and related to possible meningitis. When positive responses are given to these questions, MYCIN determines that the infection is meningitis, goes into full goal-driven mode, and tries to determine the actual infecting organisms. It does this by considering each infecting organism that it knows about and attempting to eliminate or confirm each hypothesis in turn. Since a patient may have more than one infecting organism, MYCIN searches exhaustively, continuing until all possible hypotheses have been considered.

MYCIN controls its search in a number of ways that have not yet been discussed. The knowledge base includes rules that restrict the hypotheses to be tested. For example, MYCIN concludes that it should test for meningitis when the patient has had headaches and other abnormal neurological signs.

Another feature of MYCIN's inference engine is the order in which it tries the backward chaining rules. After determining the general category of the infection (meningitis, bacteremia, etc.), each candidate diagnosis is examined exhaustively in a depth-first fashion.

To make the search behave more intelligently, MYCIN first examines all the premises of a rule to determine if any of them are already known to be false. This prevents the program from trying to solve several of the premises, only to discover that the rule could have been eliminated immediately by one of the later premises.

MYCIN also attempts to find a *unity path.* In trying to prove a goal, it looks for rules that can conclude that goal with a certainty of 1. If one of these rules succeeds, the system does not have to try other rules; the goal is known to be true. This increases both the efficiency and rationality of the search procedure.

When the confidence measures for a rule premise get below a certain value, the depth-first search is terminated. As mentioned above, the certainty measure for each patient's data is combined with the measures resident in the rules to give the certainty of the new conclusions for the patient. When these get below 0.2, MYCIN abandons the search and goes back up the graph to try another branch of the search space.

When the user wants MYCIN to justify its request for some information, he or she can ask why. Initially, this is answered by the phrase "because I need the information as a necessary attribute of each patient." Once MYCIN begins its backward search for the causes of the patient's problems, however, the why question is answered by printing the rule that MYCIN is currently considering.

As was also mentioned earlier in this chapter, the "Englishlike" interaction that the computer offers the user is not a full English language understanding program. Appropriate forms for English language comments and questions are attached to each rule when it is created so that the proper query can be presented to the user when information is needed for use of the rule or so that a rule can be printed in response to a user query.

The following dialogue is a simulated run of the MYCIN program adapted from transcripts of actual MYCIN consultations. It is not necessary to understand either the medical terminology or the medical reasoning to appreciate MYCIN's problem solving.

Our comments on the MYCIN trace are reproduced in italics throughout the interaction. The MYCIN questions and answers are numbered, with the user's responses in this type.

@ mycin

MYCIN 26-March. . . .

Instructions? (Y or N)

N

26-March 10:42

-----PATIENT-1-----

Mycin has created an object (PATIENT-1) in its dynamic knowledge base or working memory. Certain attributes are immediately requested and are necessary for every patient. These make up the first four questions. . . .

(1) Patient's name:

Alan Apt

(2) Age?

34

(3) Sex

Why

MYCIN restates the question before answering. . . .

[i.e., Why is it important to determine the sex of PATIENT-1?]

[1.0] The sex of the patient is one of the four standard parameters that identify a patient. This may be useful later. . . .

[back to question 3 . . .]

Male

(4) Race?

Caucasian

(5) Are there any cultures for Alan Apt that may be related to the present illness, and from which organisms have been grown successfully in the lab?

No

Mycin is a program to analyze and treat infectious diseases, so before it gets into close questioning of the patient's present condition it asks about any present or pending data (cultures) that are available. . . .

(6) Are there any pending cultures for Alan Apt?

Yes

-----CULTURE-1-----

Another object (CULTURE-1) *is created and linked to* PATIENT-1. . . .

(7) From what site was the specimen for CULTURE-1 taken?

CSF

(8) Please give the date and time when the pending CSF culture was obtained?

26-March-7:00

(9) Was a smear examined at the time the specimen was obtained?

Yes

(10) Were any cultures seen on the smear of the specimen?

No

Indentation indicates that MYCIN's attention has returned from consideration of the new object (CULTURE-1) *back to the original object* (PATIENT-1).

(11) Are there any other pending cultures?

No

.

.

.

(16) Has Alan Apt recently had symptoms of persistent headache or other abnormal neurologic symptoms (dizziness, lethargy, etc.)?

Yes

(17) Has Alan Apt recently had objective evidence of abnormal neurologic signs (nuchal rigidity, coma, seizures, etc.) documented by a physician?

Yes

With the answers of these last two questions in the affirmative, MYCIN decides to consider meningitis infections and goes into a full back chaining search to try to identify the organisms that might be causing the meningitis.

The CSF cultures will be considered to be associated with MENINGITIS.

(18) Please give the date on which clinical evidence (symptoms, signs, or laboratory tests) of the meningitis first appeared.

25 March 10:00

.

.

.

(36) Has Alan Apt had neurosurgery?

Why

[i.e., WHY is it important to determine whether PATIENT-1 has had neurosurgery?]

[6.0] This will aid in determining the organisms (other than those seen on cultures or smears) that might be causing the infection.

It has already been established that

[6.1] the culture that requires therapy is meningitis, and

[6.2] organisms were not seen on the stain of the pending csf culture (CULTURE-1), and

[6.3] the type of infection is bacterial

Therefore, if

[6.4] Alan Apt has had neurosurgery, and

[6.5] the time since the neurosurgical operation is less than two months, and

[6.5] the type of neurosurgery is not ventricular-urethral-shunt

then

there is evidence that the organisms that might be causing the infection are staphylococcus-coag-pos (.75), staphylococcus-coag-neg (.5) e.coli (.4), klebsiella. . . . .

Author: Yu

[RULE511]

[back to question 36...]

How [6.3]

The physician is concerned how MYCIN arrived at the conclusion that the infection is bacterial. MYCIN responds, after restating its interpretation of the question the doctor is asking, with the rules it used to determine that the infection was indeed bacterial. The answer to HOW is given as the rules used, as well as their (positive and negative) certainty factors, for determining this result. . . .

[i.e., HOW was it established that the type of infection is bacterial?]

The following rules concluded about: the likelihood that the type of infection is bacterial

	Rule	Yes	No
[6.7]	500	(.4)	
[6.8]	501		(.17)
[6.9]	502	(.1)	
[6.10]	524	(.42)	
[6.11]	504	(.28)	

Thus it has been established that the type of infection might be bacterial.

[back to question 36. . .]

No

.

.

.

```
(42) Is Alan Apt a burn patient?
Yes
I have found evidence (RULE507) that treatment should cover for
Diplococcus pneumoniae causing meningitis.
I have found evidence (RULE545) that treatment should cover for e.coli
causing meningitis
I have found evidence (RULE545 RULE578) that the treatment should cover
for Pseudomonas aeruginosa causing meningitis.
```

MYCIN now attempts to prescribe appropriate therapies for these infections. First, it must be determined whether the patient has any allergies to antibody therapies. The height and weight of the patient are also requested to determine proper dosages . . .

(.5)

```
(43) Does Alan Apt have focal neurological signs?
No
[Determining which dosages are desirable for use against the e.coli . . .]
[Determining which drugs are desirable for use . . .]

INFECTION-1 is MENINGITIS
+ <ITEM-1> DIPLOCOCCUS-PNEUMONIAE (from clinical evidence only)
+ <ITEM-2> E.COLI (from clinical evidence only)
+ <ITEM-3> PSEUD. . . . .
  .
  .
  .
[REC-1] My preferred therapy recommendation is as follows:
  In order to cover for items <1 2 3>:
    Give the following in combination:
    1) AMPICILLIN
    Dose: 3.85g (30.8ml) q4h IV
    2) GENTAMICIN
    Dose: 131mg (3.2ml) q8h IV
    Comments: Monitor serum concentrations
    Since high concentrations of penicillins can inactivate aminoglycosides,
    do not mix these antibiotics in the same IV bottle.
  Do you wish to see the next choice therapy?
  No
```

If the physician did not like the recommended treatment he or she could have asked for another, eliminating one of the drugs (ampicillin, for instance). MYCIN would then have attempted to find another treatment. . . .

8.4.4 Evaluation of Expert Systems

One of the difficult issues faced by expert system design teams is to determine when the program is ready for actual use. This problem is all the more difficult when the program, like MYCIN, deals with life-threatening situations; here, mistakes, such as overlooking an infecting agent, can be catastrophic. Many expert systems, including MYCIN (Buchanan and Shortliffe 1984), have been evaluated by using a form of the Turing test.

Ten randomly selected case histories of meningitis were rediagnosed by MYCIN and eight practitioners at the Stanford Medical School. These included five faculty members, one research fellow in infectious diseases, one resident physician, and one medical student. The actual therapy given by the original doctors on the case was also included, for a total of ten diagnoses.

These ten diagnoses were evaluated by eight infectious-disease experts away from Stanford. The diagnoses were "blind" in that they were uniformly coded so that the experts did not know whether they were looking at the computer's diagnosis or that of the humans. The evaluators rated each diagnosis as acceptable or unacceptable, with a practitioner being given one point for each acceptable rating. Thus, a perfect score would be 80 points. The results of this evaluation are given in Table 8.1:

TABLE 8.1 EXPERTS EVALUATE MYCIN AND NINE OTHER PRESCRIBERS

PRESCRIBER	SCORE	PERCENT
MYCIN	55	69
Faculty-5	54	68
Fellow	53	66
Faculty-3	51	64
Faculty-2	49	61
Faculty-4	47	59
Actual RX	47	59
Faculty-1	45	56
Resident	39	49
Student	28	35

Table 8.2 presents the results of asking another important question in this "life and death" analysis: In how many cases did the recommended therapy fail to cover for a treatable infection? The results in both tables indicate that MYCIN performed at least as well as the Stanford experts. This is an exciting result, but it should not surprise us as the MYCIN knowledge base represents the combined expertise of some of the best medical minds available. Finally, MYCIN gave fewer drugs than the human experts and thus did not overprescribe for the infections.

Another noteworthy aspect of this evaluation is how little agreement there was among human experts concerning the correctness of diagnoses. Even the best of the diagnosticians failed to receive a unanimous endorsement from the evaluators. This observation underscores the extent to which human expertise is still largely heuristic in nature.

TABLE 8.2 NUMBER OF CASES IN WHICH THERAPY MISSED A TREATABLE PATHOGEN

PRESCRIBER	NUMBER
MYCIN	0
Faculty-5	1
Fellow	1
Faculty-3	1
Faculty-2	0
Faculty-4	0
Actual RX	0
Faculty-1	0
Resident	1
Student	3

Note: On the average MYCIN gave fewer drugs than the human experts.

These positive evaluation results do not mean that MYCIN is now ready to set up a medical practice and take on patients. In fact, MYCIN is not used for delivering medical care. There are several important reasons. First, and most important, the rules (approximately 600 of them) did not give a speedy response in the doctor-computer interaction. Each session with the computer lasted about one-half hour and required, as can be seen from the trace presented above, a good amount of typing.

Second, the program was ''locked into'' the particular part of its graph search for response to questions. It was not able to extrapolate to other situations or previous patients, but remained strictly within the nodes and links of the graph it was evaluating.

Third, MYCIN's explanations were limited: when the doctor expected a deep medical justification for some particular result—a physiological or antibacterial justification, say—MYCIN simply returned the ''condition action plus certainty factor'' relationship contained in its rule base. It is difficult for an expert system to relate its heuristic knowledge to any deeper understanding of the problem domain. MYCIN, for example, does not really understand human physiology; it simply applies rules to case-specific data. Folklore has it that an early version of MYCIN, before recommending a particular drug, asked if the patient was pregnant, in spite of the fact that MYCIN had been told that the patient was male! Whether this actually happened or not, it does indicate the limitations of current expert system technology. Attempting to give expert systems the flexibility and deeper understanding demonstrated by human beings is an important and open area of research.

Finally, unlike humans, who are extremely flexible in the way they apply knowledge, expert systems do not ''degrade gracefully'' when confronted with situations that do not fit the knowledge in their rule base. That is, while a human can shift to reasoning from first principles or to intelligent guesses when confronted with a new situation, expert systems simply fail to get any answer at all.

All of these issues have meant that MYCIN does not enjoy common use in the medical field. Nonetheless, it does serve as an archetype of this class of expert system and many of its descendants, such as PUFF, are being used in clinical situations.

8.4.5 Knowledge Acquisition and the Teiresias Knowledge Base Editor

Another one of the long-term goals of expert systems research is to design software that will allow the human expert to interact directly with the knowledge base to correct and improve it. This is seen as a potential remedy for what has been called the *knowledge engineering bottleneck*. This bottleneck is caused by the fact that building an expert system generally requires a substantial commitment of time by both the domain expert and the knowledge engineer. This contributes to the expense and complexity of building expert systems. Both of these individuals tend to be highly paid professionals, and the loss in productivity caused by taking the domain expert away from other work during system development can add to the cost of the project. Additional cost and complexity come from the effort needed to communicate the domain expert's knowledge to the knowledge engineer, the logistics of getting these two people together, and the complexity of the programming required for an expert system.

The obvious solution to this problem is to automate as much of the process as possible. This has been done to a great extent through the development of expert system shells and environments that reduce the amount of programming required by the knowledge engineer. However, these shells still require that the programmer understand the methodologies of knowledge representation and search. A more ambitious goal is to eliminate the need for a knowledge engineer entirely. One way to accomplish this is to provide knowledge base editors that allow the domain expert to develop the program, interacting with the knowledge base in terms that come from the problem domain and letting the software handle the details of representation and knowledge organization. This approach has been explored in an experimental program called *Teiresias* (Davis 1982; Davis and Lenat 1982), a knowledge base editor developed at Stanford University as part of the MYCIN project. We discuss the Teiresias project in this section.

Another approach to the problem is to develop programs that can learn on their own by refining their problem-solving efforts in response to feedback from the outside. These approaches have been explored with varying degrees of success by the machine learning community. These issues are introduced in Chapter 15 and continue to be a promising and exciting area of artificial intelligence research.

With the assistance of a knowledge base editor, the domain expert can analyze the performance of a knowledge base, find missing or erroneous rules, and correct the problem in a language appropriate to the expert's way of thinking about the domain. The knowledge base editor designed for MYCIN is called Teiresias after the blind seer of the Greek tragedian Euripides. The name is altogether appropriate, as Teiresias was able to see and describe the relations of the world without the (literal) gift of sight; similarly, this software is able to understand relations in a knowledge base without (literally) understanding the referents of these relations.

Teiresias allows a doctor to step through MYCIN's treatment of a patient and locate problems with its performance. Teiresias also translates the doctor's corrections into the appropriate internal representations for the knowledge base. Thus Teiresias is able to maintain the syntactic consistency of the knowledge base by updating all appropriate program structures when new information is to be integrated into the knowledge base.

For example, MYCIN stores considerable knowledge in the form of tables. Certain rules include LISP code that retrieves information from these tables, such as characteristics of disease organisms. This improves the efficiency of the knowledge base, since a single table can capture information that would require numerous rules to encode. However, if a new object (for example, a disease organism) is added to the knowledge base, then all tables that contain information about disease organisms must be updated to reflect this.

Semantic consistency is a more difficult problem, requiring knowledge of how well the knowledge base actually reflects the problem domain. Although Teiresias cannot be said to truly maintain semantic consistency, it does make an effort to ensure that rules appear consistent with one another. It does this by maintaining *rule models*, or statistical profiles of all the rules. If a new rule is added to the knowledge base, Teiresias compares this new rule with the rule model for all existing rules that draw the same conclusion. The assumption is that rules that draw similar conclusions should examine similar conditions in their premises. This enables Teiresias to detect potential inconsistencies in a new rule and prompt the user for corrections.

To illustrate Teiresias, we analyze its interaction with the doctor after MYCIN has given its recommendations. The doctor disagrees with MYCIN's result, and with Teiresias's help, analyzes MYCIN's solution. The trace is a simulation taken from a session by Stanford researchers, where the analysis is in italics, Teiresias's questions are in regular type, and the doctor's responses are in this type.

When a conclusion is given with which the doctor disagrees, the rule that drew this conclusion is not necessarily wrong. A number of things could have happened, as Teiresias correctly shows: A fact of the rule could have been improperly concluded; the rule might, in fact, be wrong; or another rule might be missing from the knowledge base. Finding the cause of an error is a difficult problem that potentially requires examination of the entire line of reasoning. Teiresias assists the physician in determining the problem by maintaining an and/or graph of the solution process and working systematically through this trace.

All rules are translated into an English language syntax that assists the physician in understanding the trace. Once a problem is located, Teiresias lets the user add the missing rule in the same simplified syntax, translating the rule into an internal representation and checking the new rule for consistency with similar rules in the knowledge base.

MYCIN has reached a set of infections that it feels should be treated (as in the dialogue of Section 8.4.3). These are presented to the physician, who disagrees with MYCIN's analysis:

> My therapy recommendation will be based on the following probable infections and potential causative organisms:
>
> INFECTION-1 is PRIMARY BACTEREMIA
> <item-1> E.COLI
> <item-2> PSEUDOMONAS-AERUGINOSA

Now Teiresias steps in:

> Are all these identities correct?
>
> No
>
> Are there any identities missing?
>
> Yes

Which ones?

KLEBSIELLA_PNEUMONIAE, PROTEUS-NON-MIRABILIS

Are there any identities which shouldn't have appeared?

Yes

Which ones?

PSEUDOMONAS-AERUGINOSA

Do you want to
1) track down the problem yourself, or
2) allow me to guide the debugging, or
3) ignore it for now and fix it another time

Type 1, 2 or 3

2

If the doctor had typed 1, he or she could have tried to sort out the problems with the traditional WHY and HOW queries on the preceding solution. Note that Teiresias has already begun to categorize the errors that appeared in the conclusions. It should also be noted that the three problems could well be related (perhaps to one missing rule). Teiresias starts by trying to find out why the wrong organism appeared for treatment. . . .

We'll start with PSEUDOMONAS-AERUGINOSA first:
The following rules were used in deducing that the identity of ORGANISM-1 is PSEUDOMONAS-AERUGINOSA.
RULE 184
Since
[1.1] The category of ORGANISM-1 is not known
[1.2] The gram stain of ORGANISM-1 is gram negative
[1.3] The morphology of ORGANISM-1 is rod
[1.4] The aerobicity of ORGANISM-1 is facultative
There is weakly suggestive evidence (.3) that the identity of ORGANISM-1 is PSEUDOMONAS-AERUGINOSA.
Is this rule correct?

Yes

Should its premise have failed in this case?

Yes

Which clause of the premise should have failed?

Clause 1

The expert feels the program should have been able to establish a category and that RULE 184 should not have fired.

Should it have been possible to conclude a category?

Yes

Which one?

ENTEROBACTERIACEAE

```
There were no rules successfully called in the attempt to deduce this category.
The following (potentially applicable) rules were NOT executed because the
marked [++] clause failed:
RULE 116
  If
  1) the identity of ORGANISM-1 is not known
++ 2) the gram stain of ORGANISM-1 is not known
  3) the morphology of ORGANISM-1 is not known
  4) the site of CULTURE-1 is CSF
  5) the infection is meningitis
  6) the age of the patient is less than or equal to 2 months
  Then there is weakly suggestive evidence (.33) that the category of
  ORGANISM-1 is ENTEROBACTERIACEAE
RULE 050
  If
  1) the morphology of ORGANISM-1 is rod
  2) the gram stain of ORGANISM-1 is gramneg
  3) the aerobicity of ORGANISM-1 is facultative
++ 4) infection ORGANISM-1 was acquired while the patient was hospitalized
  Then there is suggestive evidence (.7) that the category of ORGANISM-1 is
  ENTEROBACTERIACEAE
Are all these rules correct?
Yes
Should any of them have been successfully invoked?
No
Is there a rule missing?
Yes
Would you like to give me such a rule now?
Yes
Good...
If
  1 ** THE PATIENT'S INFECTION IS PRIMARY-BACTEREMIA
  2 ** THE SITE IS ONE OF THE STERILE SITES
Then:
  ** THERE IS SUGGESTIVE EVIDENCE (.8) THAT THE CATEGORY IS
  ENTEROBACTERIACEAE
```

Teiresias parses this rule into its internal representation and then translates it back into English so the expert can check it. After the expert agrees on the parse, Teiresias attempts to integrate the new rule into the knowledge base....

```
I hate to criticize, Dr Davis, but did you know that most rules about what the
category of an organism might be that mention:
  the site of a culture, and
  the infection
```

also mention:
  the portal of entry of the organism?
Shall I try to write a clause to account for this?

Yes

How about:

  the portal of entry is gastrointestinal. OK?

Yes

Teiresias is itself a knowledge-based program with rules for reasoning about rules and objects in a MYCIN knowledge base. As the above trace illustrates, the program systematically searches back along the trace of the reasoning to find the source of the problem. Once this is done, it helps the user add to or correct the knowledge base. Using the rule models discussed at the beginning of this section, Teiresias is able to check the rule for consistency and correct the missing premise.

To help the user add new information, Teireseas keeps a model of each class of object that appears in the system. The model for a MYCIN object is called a *schema.* Each schema describes how to create new instances of a class of objects such as a new disease organism, drug, or test. The schema for a class of objects also describes how information is to be obtained for rules of that class—for example, to compute it from existing data, look it up, or ask the user.

Schemata record the interrelationships of all classes of objects in the MYCIN rule base. For example, if the addition of a new disease requires that a table of diseases and their sensitivities to various drugs also be updated, this is recorded in the disease schema. Schemata also include pointers to all current instances of each schema. This is very important if it is decided to change the form of the schema throughout the program.

Teiresias organizes its schemata into a hierarchy: A schema for describing bacterial infections may be a specification of a general class schema, which in turn is a specification of a general schema for MYCIN facts. Each schema includes pointers to its parents and children in this hierarchy. (See also the topics of frames, objects, and object-oriented programming in chapters 9 and 14.)

Schemata also contain bookkeeping information. This includes the author and date of creation and addition of each instance to the program. It includes a description of the schema used to create it. Documentation of who created rules when is critical for discussion and analysis of the knowledge base; when rules are primarily condition-action pairs representing important aspects of an application domain and not full explications of these relationships, it is important to be able to trace the rules back to their authors for more complete discussion of the factors underlying their creation.

The top-level structure in the hierarchy of Teiresias's knowledge is the *schema-schema.* This structure is a schema for creating new schemata, when, for instance, the domain expert might wish to create a new category of objects. The schema-schema allows Teiresias to reason about its own knowledge structures, including creating new ones when appropriate. A schema-schema has the same basic structure as a schema itself and provides all the bookkeeping information to reconstitute the entire knowledge base.

Although the development of commercial programs to help with knowledge acquisition is still in the future, Teiresias has shown how and/or graph search and knowledge representation techniques provide a basis for automating knowledge base refinement.

8.5 Epilogue and References

The model for the rule-based expert system is the production system. Whether the final product is data driven or goal driven, the model for the software is production system-generated graph search. This material was presented in chapters 3, 4, and 5.

We implement simple expert system shells in PROLOG and LISP in chapters 12 and 13, respectively. These shells, and some sample rules presented with them, are able to create a goal-driven search much like MYCIN's. The rules can include certainty measures in a limited form for designing a heuristically based search. The student is encouraged to add rules to fill out the knowledge base either along the lines of the analysis of a car that won't start, as in the beginning of this chapter, or to implement some other application.

A number of references complement the material presented in this chapter; especially recommended is a collection of the original MYCIN publications from Stanford entitled *Rule-Based Expert Systems* by Buchanan and Shortliffe (1984).

Other important books on general knowledge engineering include *Building Expert Systems* by Hayes-Roth et al. (1984), *A Guide to Expert Systems* by Waterman (1986), *Expert Systems: Concepts and Examples* by Alty and Coombs (1984), *Expert Systems Technology: A Guide* by Johnson and Keravnou (1985), *Expert Systems: Tools and Applications* by Harmon et al. (1988), and *Expert Systems and Fuzzy Systems* by Negoita (1984).

Because of the domain specificity of expert system solutions, case studies are an important source of knowledge in the area. Books in this category include *Expert Systems: Techniques, Tools and Applications* by Klahr and Waterman (1986), *Competent Expert Systems: A Case Study in Fault Diagnosis* by Keravnou and Johnson (1986), *The CRI Directory of Expert Systems* by Smart and Langeland-Knudsen (1986), and *Developments in Expert Systems* by Coombs (1984).

A pair of more general books that give an overview of commercial uses of AI are *The AI Business* edited by Winston and Prendergast (1984) and *Expert Systems: Artificial Intelligence in Business* by Harmon and King (1985).

8.6 Exercises

1. In Section 8.2 we introduced a set of rules for diagnosing automobile problems. Identify possible knowledge engineers, domain experts, and potential end users for such an application. Discuss the expectations, abilities, and needs of each of these groups.
2. Take Exercise 1 above. Create in English or pseudocode 15 if-then rules (other than those prescribed in Section 8.2) to describe relations within this domain. Create a graph to represent the relationships within these 15 rules.

3. Consider the graph of Exercise 2 above. Do you recommend data- or goal-driven search? breadth- or depth-first search? In what ways could heuristics assist the search? Justify your answers to these questions.
4. Pick another area of interest for designing an expert system. Answer Exercises 1–3 for this application.
5. Implement an expert system using a commercial shell program. These are widely available for personal computers as well as larger machines.
6. Critique the shell you used to do Exercise 5. What are its strengths and weaknesses? What would you do to improve it? Was it appropriate to your problem? What problems are best suited to that tool?